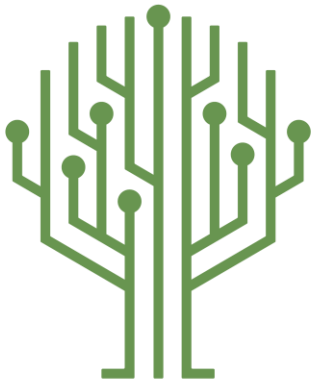
Green Pace Developer: Security Policy Guide



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	Ensure we are using the correct locale and byte character width's to prevent buffer overflow's and specifying to users a limit to the amount of characters entered. We need to also ensure integers don't overflow by checking the current resultant.
2. Heed Compiler Warnings	Using compiler flags such as –Wall and –Wextra(GCC) or /W1 to /W4 (MSVC) will enable a broad variety of warnings that must be resolved if possible.
3. Architect and Design for Security Policies	Architectural decisions need to be principle driven for design security. From this we can generate a security checklist of actionable items. This focuses on confidentiality of data, authentication, and integrity of data while in rest, transit or motion.
4. Keep It Simple	Instead of making long functions, keep them short with comments to emphasize readability. Make sure all security items stem from actionable lists that can mutate in order to better serve their purposes.
5. Default Deny	Users of any system need to explicitly ask for access to certain networks or data.
6. Adhere to the Principle of Least Privilege	Give the least number of privileges to users accessing data or other systems.
7. Sanitize Data Sent to Other Systems	Parameterize strings from other strings using well known libraries. Perform proper input validation and generalize strings from user input to not take in random strings but use programmer defined options put together after reading.
8. Practice Defense in Depth	Ensuring we have multiple layers of security controls in software to validate input, give least privileges and make sure to use a strong encryption.

Principles	Write a short paragraph explaining each of the 10 principles of security.
9. Use Effective Quality Assurance Techniques	Having linters and automated code scanners such as IDE's and dependency checks will ensure quality is high.
10. Adopt a Secure Coding Standard	Always reference several resources to determine the tradeoffs and pro's of different security options.

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 Values	[STD-DCL50-CPP]	Declarations and initializations(DCL)- Do not define a C-style variadic function. There are no safety mechanisms for the C language to check the type safety of the variable amount of arguments passed to it.

Noncompliant Code

Adding a series of integers together until a value of '0' is found. Calling the function without the '0' results in undefined behavior which guarantees errors or an error in data.

```
#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

A variadic function parameter pack is used to make the add function compliant. The compliant solution uses 'std::enable_if' to ensure that any nonintegral argument value results in an error being raised.

```
#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; }
```

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

Principles(s): Validate input data.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Probable	Not Detectable and Repairable	Medium (CERT Priority 8)	Level 2

Automation

Tool	Version	Checker	Description Tool
clang-tidy	17.x	cert-dcl50-cpp	Enforces CERT rule to avoid C-style variadic functions
Astree	7.2.0	function-ellipsis	Fully checked
CodeSonar	9.1p0	LANG.STRUCT.ELLIPSIS	Ellipsis
Parasoft C/C++test	2025.2	CERT_CPP-DCL50-a	Functions shall not be defined with a variable number of arguments

Coding Standard 2

Coding Standard	Label	Name of Standard
Data Types	[STD-EXP50-CPP]	Depending on the order of evaluation while mutating variables in code creates undefined behavior, where left to right guarantees are not maintained in compilation.

Noncompliant Code

The variable 'i' is modified twice in the same full expression

```
i = ++i + 1;
a[i++] = i;
```

Compliant Code

The variables being evaluated are independent of the order of evaluation.

```
void f(int i, const int *b){
     ++i;
     int a = i + b[i];
     // or
     /*
     int a = i + b[i +1];
     ++i;
     */
}
```

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

Principles(s): Keep it secure and simple. Eliminate complex, ambiguous logic.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Probable	1 or 2	Medium (8)	L2

Automation

Tool	Version	Checker	Description Tool
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	7.2.0	CertC++-EXP50	Detected by this tool.
Clang	3.9	Compiler flag '-Wunsequenced'	Can detect simple violations of this rule where path-sensitive analysis is not required

Coding Standard 3

Coding Standard	Label	Name of Standard
Assertions	[STD-DCL03-C]	Use a static assertion to test the value of a constant expression.

Noncompliant Code

This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. Although the use of the runtime assertion is better than nothing, it needs to be placed in a function and executed. This means that it is usually far away from the definition of the actual structure to which it refers. The diagnostic occurs only at runtime and only if the code path containing the assertion is executed.

```
#include <assert.h>

struct timer {
  unsigned char MODE;
  unsigned int DATA;
  unsigned int COUNT;
};

int func(void) {
  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned
int) + sizeof(unsigned int));
}
```

Compliant Code

This portable compliant solution uses static_assert, which allow incorrect assumptions to be diagnosed at compile time instead of resulting in a silent malfunction or runtime error. Because the assertion is performed at compile time, no runtime cost in space or time is incurred. An assertion can be used at file or block scope, and failure results in meaningful and informative diagnostic error message.

```
#include <assert.h>

struct timer {
  unsigned char MODE;
  unsigned int DATA;
  unsigned int COUNT;
};

static_assert(sizeof(struct timer) == sizeof(unsigned char) +
sizeof(unsigned int) + sizeof(unsigned int),
              "Structure must not have any padding");
```

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

Principles(s): Use effective quality assurance techniques, because static assertions enforce correctness at compile time, improving code quality and preventing runtime faults.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Repairable and detectable	3	L3

Automation

Tool	Version	Checker	Description Tool
Clang	3.9	misc-static-assert	Checked.
CodeSonar	9.1p0	(customizable)	Users can implement a custom check that reports uses of the assert() macro
ECLAIR	1.2	CC2.DCL03	Fully implemented check
Axivion Bauhaus	7.2.0	CertC-DCL03	Statically checked

Coding Standard 4

Coding Standard	Label	Name of Standard
Data Types	[STD-INT50-CPP]	Do not cast to an out-of-range enumeration value. Setting up enumerations with a value becomes an optimization that uses the minimum value, so checking it beforehand would-be best practice. The value check must be within the range of the enumerator-list.

Noncompliant Code

This bit of code attempts to check whether a given value is within the range of acceptable enum values. However it is doing so after casting it to an enum type which will not be able to represent the given integer value.

```
enum EnumType {
      First,
      Second,
      Third
};
void f(int intVar) {
      EnumType enumVar = static_cast<EnumType>(intVar);
      if (enumVar < First || enumVar > Third) {
      // Handle error

      }
}
```

Compliant Code

This code checks that the value can be represented by the enumeration specifier before performing the conversion to guarantee the conversion does not result in an unspecified value. It is restricted by the enum key in a comparison to guarantee it doesn't result in an 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): Validate input data

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium, Enum undefined behavior can corrupt logic but rarely gives direct remote code execution.	Probable	Not detectable and Repairable	Medium	L2

Automation

Tool	Version	Checker	Description Tool
Astree	25.10	cast-integer-to-enum	Partially checked
Axivion Bauhaus Suite	7.2.0	CertC++-INT50	Software research project.
CodeSonar	9.1p0	LANG.CAST.COERCE LANG.CAST.VALUE	Coercion alters value, or cast alters value
Helix QAC	2025.2=	C++3013	Quality Assurance and Control.

Coding Standard 5

Coding Standard	Label	Name of Standard
SQL Injection	[STD-IDS00-CPP]	Prevent SQL injection.

Noncompliant Code

An attacker could append '1=1' from a database table substituting arbitrary strings for username and password and bypassed through this method. That snippet would always return true.

```
SELECT * FROM db_user WHERE username='validuser' OR '1'='1' AND
password='<PASSWORD>'
```

Compliant Code

You create a prepared statement of the user supplied data for username and password instead of appending to the query the password or any value that can introduce new strings such as '1'='1'.

```
# String sqlString = "select * from db_user where username=? and
password=?"; PreparedStatement stmt =
connection.prepareStatement(sqlString); stmt.setString(1, username);
stmt.setString(2, pwd); ResultSet rs = stmt.executeQuery();
```

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

Principles(s): Sanitize data sent to other systems.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High, unparameterized atabase queries allow attacker-controlled input to alter SQL execution, enabling unauthorized data access or modification	Likely	Detectable and Repairable	High	L1

Automation

Tool	Version	Checker	Description Tool
The Checker Framework	2.1.3	Tainting checker	Protect against SQL injection
Fortify	1.0	HTTP_Response_Splitting SQL_Injection_Persistence	Implemented
SonarQube	9.0p0	S2077	Executing SQL queries is security-sensitive SQL queries should not be vulnerable to injection attacks.
Findbugs	1.0	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	Implemented

Coding Standard 6

Coding Standard	Label	Name of Standard
String Correctness	[STD-STR50-CPP]	Guarantee that storage strings have sufficient space for character data and the null terminator.

Noncompliant Code

This code creates a buffer overflow vulnerability.

```
#include <iostream>

void f(){
      char buff[12];
      std::cin >> buff;
}
```

Compliant Code

Using string we can avoid this vulnerability.

```
#include <iostream>
#include <string>

void f() {
      std::string input;
      std::string stringOne, stringTwo;
      std::cin >> stringOne >> stringTwo;
}
```

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

Principles(s): Validate Input Data

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	High	Not Detectable and Repairable	High	L1

Automation

Tool	Version	Checker	Description Tool
Astree	25.10	stream-input-char-array	Partially checked + soundly supported
CodeSonar	9.1p0	MISC.MEM.NTERM LANG.MEM.BO LANG.MEM.TO	No space for null terminator Buffer overrun Type overrun
Parasoft C/C++test	2025.2	CERT-CPP-STR50-*	Avoid overflow due to reading a not zero terminated string
Polyspace Bug Finder	R2025b	CERT C++: STR50-CPP	Checks for: use of dangerous function, missing null in string array, buffer overflow, or insufficient destination buffer size.

Coding Standard 7

Coding Standard	Label	Name of Standard
Memory Protection	[STD-MEM50-CPP]	Do not access freed memory

Noncompliant Code

In this example, S is dereferenced after it has been deallocated and could lead to arbitrary code being run with the permissions of the vulnerable process.

```
#include <new>
struct S{
      void f();
};

void g() noexcept(false) {
      S *s = new S;
      // ...
      delete s;
      // ...
      s->f();
}
```

Compliant Code

This example shows that the dynamically allocated memory is not deallocated until it is no longer required.

```
#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): Practice Defense in depth, because it prevents use-after-free exploitation.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High, because it could cause RCE.	Likely	Not detectable and not repairable.	P9	L2

Automation

Tool	Version	Checker	Description Tool
Astree	25.10	csa-double-free	partially checked + soundly supported
Axivion Bauhaus Suite	7.2.0	CertC++-MEM50	
Clang	3.9	clang-analyzer-cplusplus.NewDelete	Checked by clang-tidy, but does not catch all violations of this rule.
Parasoft C/C++test	2025.2	CERT_CPP-MEM50-a	Do not use resources that have been freed.

Coding Standard 8

Coding Standard	Label	Name of Standard
Input Output (FIO)	[STD-FIO50-CPP]	Do not alternately input and output from a file stream without an intervening positioning call.

Noncompliant Code

This noncompliant code example appends data to the end of a file and then reads from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is undefined.

```
#include <fstream>
#include <string>

void f(const std::string &fileName) {
  std::fstream file(fileName);
  if (!file.is_open()) {
    // Handle error
    return;
  }

  file << "Output some data";
  std::string str;
  file >> str;
}
```

Compliant Code

In this compliant solution, the 'std::basic_istream<T>::seekg() function is called between the ouput and input, eliminating the undefined behavior.

```
#include <fstream>
#include <string>

void f(const std::string &fileName) {
  std::fstream file(fileName);
  if (!file.is_open()) {
    // Handle error
    return;
  }

  file << "Output some data";

  std::string str;
  file.seekg(0, std::ios::beg);
  file >> str;
}
```

Compliant Code

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

Principles(s): Architect secure and safe coding practices.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Likely	Detectable and not repairable	P6	L2

Automation

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	7.2.0	CertC++FIO50	
CodeSonar	9.1p0	IO.IOWOP	Input after output without positioning
Parasoft C/C++ test	2025.2	CERT_CPP-FIO50-a	Do not alternately input and output from a stream without an intervening flush or positioning call
Security Reviewer- Static reviewer	6.02	C17	Fully implemented

Coding Standard 9

Coding Standard	Label	Name of Standard
Exceptions and Error Handling	[STD-ERR50-CPP]	Do not abruptly terminate the program. The 'std::abort()', 'std::quick_exit()', and 'std::_Exit()' functions are used to terminate the program in an immediate fashion. They do so without calling exit handlers registered with 'std::atexit() and without executing destructors for objects with automatic, thread, or static storage duration.

Noncompliant Code

In this noncompliant code example, the call to f(), which was registered as an exit handler with 'std::at_exit()', may result in a call to 'std::terminate()' because 'throwing_func()' may throw an exception.

```
#include <cstdlib>
void throwing_func() noexcept(false);

void f() {

     // Not invoked by the program except as an exit handler.
     throwing_func();

}

int main() {
     if (0 != std::atexit(f)) { // Handle error }
}
```

Compliant Code

In this compliant solution, f() handles all exceptions thrown by 'throwing_func()' and does not rethrow.

```
#include <cstdlib>

void throwing_func() noexcept(false);

void f(){
     try {
          throwing_func();
     } catch (...) {
          // handle error
     }
}

int main() {
     if (0 != std::atexit(f)) {
```

Compliant Code

```
                // handle error
            }
            // ...
        }
```

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

Principles(s): Practice Defense in Depth, because it bypasses layered error handling which prevents logging and containment controls. It also weakens resilience and recovery safeguards.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low, can cause denial of service events due to resources being consumed and not freed.	Probable	Not detectable and Not repairable	P2	L3

Automation

Tool	Version	Checker	Description Tool
Astree	25.10	stdlib-use unhandled-throw-noexcept	Partially checked
CodeSonar	9.1p0	BADFUNC.ABORT BADFUNC.EXIT	Finds the use of abort and exit.
Helix QAC	2025.2	C++5014	
Parasoft C/C++ test	2025.2	CERT_CPP-ERR50-*	Functions registered with std::atexit() or std::at_quick_exit() shall not exit by throwing exceptions. The quick_exit() and _Exit() functions from <stdlib.h> or <cstdlib> shall not be used. The abort() function from <stdlib.h> or <cstdlib> shall not be used.

Coding Standard 10

Coding Standard	Label	Name of Standard
Object Oriented Programming	[STD-OOP50-CPP]	Do not invoke virtual functions from constructors or destructors

Noncompliant Code

In this noncompliant code example, the base class attempts to seize and release an object's resources through calls to virtual functions from the constructor and destructor. However, the 'B::B()' constructor calls 'B::seize()' rather than 'D::seize()'. Likewise, the B::~B() destructor calls 'B::release() rather than 'D::release()'.

The result of running this code is that no derived class resources will be seized or released during the initialization and destruction of object of type 'D'.

```
struct B{
      B() { seize(); }
      virtual ~B() { release(); }
      protected:
            virtual void seize();
            virtual void release();
};

struct D : B {
      virtual ~D() = default;
      protected:
            void seize() override {
                  B::seize();
                  // Get derived resources...
            }

            void release() override {
                  // Release derived resources...
                  B::release();
            }
};
```

Compliant Code

In this compliant solution, the constructors and destructors call a nonvirtual, private member function (suffixed with min) instead of calling a virtual function. The result is that each class is responsibile for seizing and releasing its own resources.

```
class B {
      void seize_mine();
      void release_mine();

      public:
```

Compliant Code

```
            B() { seize_mine(); }
            virtual ~B() { release_mine(); }
      protected:
            virtual void seize() { seize_mine(); }
            virtual void release() { release_mine(); }

};

class D : public B {
      void seize_mine();
      void release_mine();

      public:
            D() { seize_mine(); }
            virtual ~D() { release_mine(); }
      protected:
            void seize() override {
                  B::seize();
                  seize_mine();
            }

            void release() override {
                  release_mine();
                  B::release();
            }
};
```

Principles(s): Keep it simple by adopting a secure coding standard

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium, because it causes logic/resource errors, not usually direct RCE.	Low	Detectable and not repairable	P2	L3

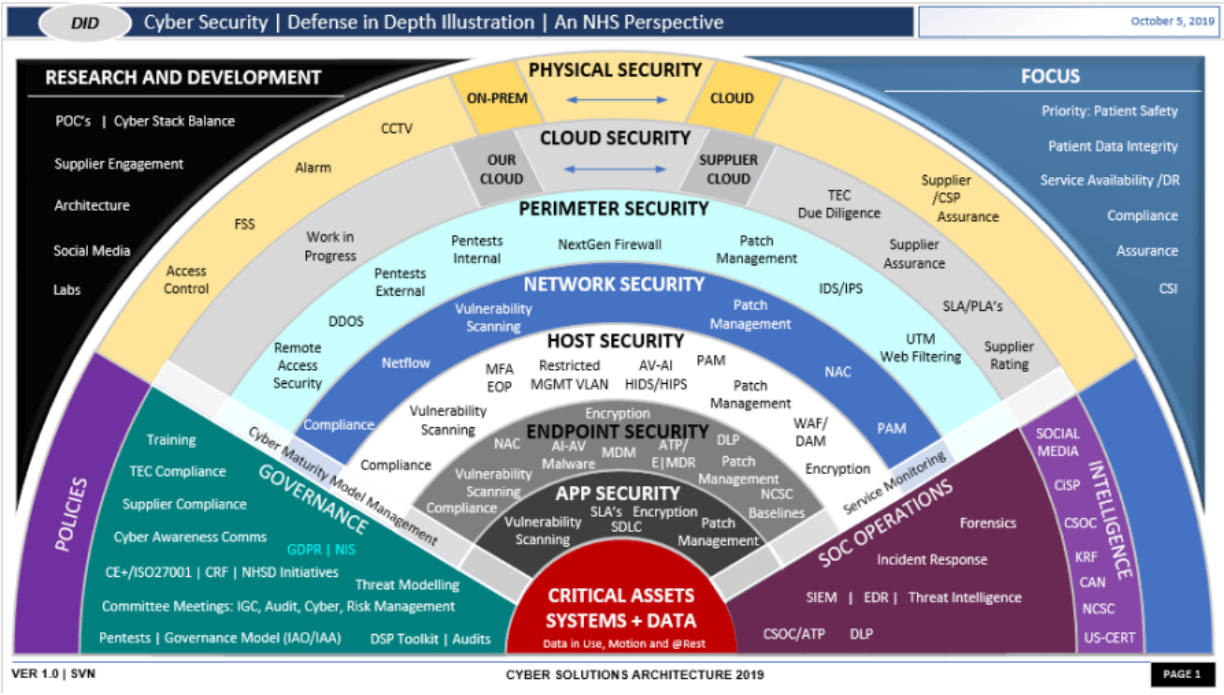
Automation

Tool	Version	Checker	Description Tool
Astree	25.10	virtual-call-in-constructor invalid_function_pointer	Fully checked

Tool	Version	Checker	Description Tool
Axivion Bauhaus	7.2.0	CertC++-OOP50	
Clang	3.9	clang-analyzer-alpha.cplusplus.VirtualCall	Checked by clang-tidy
CodeSonar	9.1p0	LANG.STRUCT.VCALL_IN_CTOR LANG.STRUCT.VCALL_IN_DTOR	Virtual Call in Constructor Virtual Call in Destructor

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.

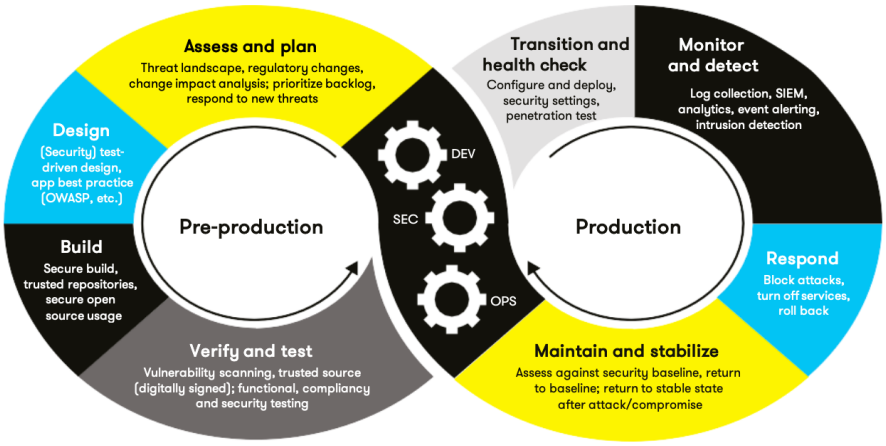
Risk Assessment

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



Summary of Risk Assessments

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

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
STD-001-CPP	High	Unlikely	Medium	High	2
STD-DCL50-CPP	Medium	Probable	Not Detectable & Repairable	Medium	L2
STD-EXP50-CPP	Medium	Probable	Not Detectable & Repairable	Medium	L2
STD-DCL03-C	Low	Unlikely	Detectable & Repairable	Low	L3
STD-INT50-CPP	Medium	Probable	Not Detectable & Repairable	Medium	L2
STD-IDS00-CPP	High	Likely	Detectable & Repairable	High	L1
STD-STR50-CPP	High	Likely	Not Detectable & Repairable	High	L1
STD-MEM50-CPP	High	Likely	Not Detectable & Not Repairable	High	L1
STD-FIO50-CPP	Low	Likely	Detectable & Repairable	Medium	L2
STD-ERR50-CPP	Medium	Probable	Detectable & Repairable	Medium	L2
STD-OOP50-CPP	Medium	Probable	Detectable & Repairable	Medium	L2

Encryption	
Encryption at rest	Data that is stored on disks, databases, backups and removable media using strong AES-256 cryptographic standards. Usually personally identifiable information, financial records, or system configuration files.
Encryption in flight	Data that is transmitted across networks using secure protocols such as TLS 1.2+ and HTTPS. This policy applies to all communications between clients, servers, APIs, databases, and third-party services.
Encryption in use	This is data that is protected by encryption of operating system methods such as AMD SME or Intel TME. Secure enclaves (Intel SGX, and AMD SEV). Encryption in use is enforced through operating system and hardware-assisted memory encryption, secure enclave execution, and protected runtime environments that prevent plain text exposure during active computation.

Triple-A Framework	
Authentication	This part of the framework verifies the identity of users and systems before granting access. This policy requires unique user logins, strong password standards, and multi-factor authentication (MFA) for all privileged and remote access accounts. Authentication controls apply during system login, database access, administrative actions, and API usage.
Authorization	This piece of the framework determines what resources an authenticated user can access. It enforces role-based access control and the principle of least privilege to restrict user capabilities based on job function.
Accounting	This framework tool logs user and system activity for monitoring and forensic purposes. This policy requires logging login attempts, database modifications, file access, permission changes, and new user creation. Logs must be timestamped, protected from tampering, and retained according to organizational policy.

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	
[Insert text.]	02/15/2026	Finalizing policy	Andres Trujillo	Green Pace Board of Directors

Appendix A Lookups

Approved C/C++ Language Acronyms

Language	Acronym
C++	CPP
C	CLG
Java	JAV