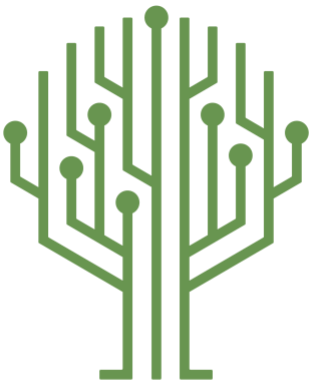


Green Pace Secure Development Policy

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	[Insert text.]
2. Heed Compiler Warnings	[Insert text.]
3. Architect and Design for Security Policies	[Insert text.]
4. Keep It Simple	[Insert text.]
5. Default Deny	[Insert text.]
6. Adhere to the Principle of Least Privilege	[Insert text.]
7. Sanitize Data Sent to Other Systems	[Insert text.]
8. Practice Defense in Depth	[Insert text.]
9. Use Effective Quality Assurance Techniques	[Insert text.]
10. Adopt a Secure Coding Standard	[Insert text.]

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	[DCL50-CPP]	Do not define a C-style variadic function

Noncompliant Code

```
The function reads arguments until there are no more. It will result as a
undefined behavior if you pass 0 or any type other than int as the value.
```

```
#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 compliant solution does not result in undefined behavior if the list
of parameters is not terminated with 0.
```

```
#include <type_traits>

template <typename Arg, typename std::enable_if<std::is_integral<Arg>::va
lue>::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): [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
Astrée	20.10	function-ellipsis	Fully checked
Axivion Bauhaus Suite	7.2.0	CertC++-DCL50	
Clang	3.9	cert-dcl50-cpp	Checked by clang-tidy.
CodeSonar	6.2p0	LANG.STRUCT.ELLIPSIS	Ellipsis

Coding Standard 2

Coding Standard	Label	Name of Standard
Data Value	EXP50-CPP	Do not depend on the order of evaluation for sides effects

Noncompliant Code

```
The first argument expression reads the value of i and then changes it. J reads
the value of I, this has undefined behavior.
```

```
extern void func(int i, int j);

void f(int i) {
  func(i++, i);
}
```

Compliant Code

```
This compliant solution is appropriate when the programmer intends for
both arguments to func() to be equivalent.
```

```
extern void func(int i, int j);

void f(int i) {
  i++;
  func(i, i);
}
```

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	Probable	Medium	P8	L2

Automation

Tool	Version	Checker	Description Tool

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	7.2.0	CertC++-EXP50	
Clang	3.9	-Wunsequenced	Can detect simple violations of this rule where path-sensitive analysis is not required
Compass/ROSE			Can detect simple violations of this rule. It needs to examine each expression and make sure that no variable is modified twice in the expression. It also must check that no variable is modified once, then read elsewhere, with the single exception that a variable may appear on both the left and right of an assignment operator
Coverity	v7.5.0	EVALUATION_ORDER	Can detect the specific instance where a statement contains multiple side effects on the same value with an undefined evaluation order because, with different compiler flags or different compilers or platforms, the statement may behave differently

Coding Standard 3

Coding Standard	Label	Name of Standard
String Correctness	STR51-CPP	Do not attempt to create a std::string from a null pointer

Noncompliant Code

Since `std::getenv()` returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist

```
#include <cstdlib>
#include <string>

void f() {
  std::string tmp(std::getenv("TMP"));
  if (!tmp.empty()) {
    // ...
  }
}
```

Compliant Code

In this compliant solution, the results from the call to `std::getenv()` are checked for null before the `std::string` object is constructed.

```
#include <cstdlib>
#include <string>

void f() {
  const char *tmpPtrVal = std::getenv("TMP");
  std::string tmp(tmpPtrVal ? tmpPtrVal : "");
  if (!tmp.empty()) {
    // ...
  }
}
```

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

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

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	P18	L1

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	assert_failure	
Helix QAC	2022.1	C++4770, C++4771, C++4772, C++4773, C++4774	
Klocwork	2022.1	NPD.CHECK.CALL.MIGHT NPD.CHECK.CALL.MUST NPD.CHECK.MIGHT NPD.CHECK.MUST NPD.CONST.CALL NPD.CONST.DEREF NPD.FUNC.CALL.MIGHT NPD.FUNC.CALL.MUST NPD.FUNC.MIGHT NPD.FUNC.MUST NPD.GEN.CALL.MIGHT NPD.GEN.CALL.MUST NPD.GEN.MIGHT NPD.GEN.MUST RNPD.CALL RNPD.DEREF	
Parasoft C/C++test	2021.2	CERT_CPP-STR51-a	Avoid null pointer dereferencing

Coding Standard 4

Coding Standard	Label	Name of Standard
SQL Injection	IDS00-J	Prevent SQL Injection

Noncompliant Code

Permits SQL injection attack by incorporating `username` into the SQL command, allowing an attacker to inject `validuser' OR '1'='1`. The `password` argument cannot be used to attack this program because it is passed to the `hashPassword()` function.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

class Login {
  public Connection getConnection() throws SQLException {
    DriverManager.registerDriver(new
            com.microsoft.sqlserver.jdbc.SQLServerDriver());
    String dbConnection =
      PropertyManager.getProperty("db.connection");
    // Can hold some value like
    // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"
    return DriverManager.getConnection(dbConnection);
  }

  String hashPassword(char[] password) {
    // Create hash of password
  }

  public void doPrivilegedAction(String username, char[] password)
                                 throws SQLException {
    Connection connection = getConnection();
    if (connection == null) {
      // Handle error
    }
    try {
      String pwd = hashPassword(password);

      String sqlString = "SELECT * FROM db_user WHERE username = '"
                         + username +
```

Noncompliant Code

```
                          "' AND password = '" + pwd + "'";
      Statement stmt = connection.createStatement();
      ResultSet rs = stmt.executeQuery(sqlString);

      if (!rs.next()) {
        throw new SecurityException(
          "User name or password incorrect"
        );
      }

      // Authenticated; proceed
    } finally {
      try {
        connection.close();
      } catch (SQLException x) {
        // Forward to handler
      }
    }
  }
}
```

Compliant Code

Uses a question mark as a placeholder for the argument. Confirms the length of the username argument, preventing an attacker from submitting an arbitrarily long user name.

```
public void doPrivilegedAction(
  String username, char[] password
) throws SQLException {
  Connection connection = getConnection();
  if (connection == null) {
    // Handle error
  }
  try {
    String pwd = hashPassword(password);

    // Validate username length
    if (username.length() > 8) {
      // Handle error
    }

    String sqlString =
      "select * from db_user where username=? and password=?";
    PreparedStatement stmt = connection.prepareStatement(sqlString);
```

Compliant Code

```
    stmt.setString(1, username);
    stmt.setString(2, pwd);
    ResultSet rs = stmt.executeQuery();
    if (!rs.next()) {
      throw new SecurityException("User name or password incorrect");
    }

    // Authenticated; proceed
  } finally {
    try {
      connection.close();
    } catch (SQLException x) {
      // Forward to handler
    }
  }
}
```

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
The Checker Framework	2.1.3	Tainting Checker	Trust and security errors (see Chapter 8)
CodeSonar	6.2p0	JAVA.IO.INJ.SQL	SQL Injection (Java)
Coverity	7.5	SQLI FB.SQL_PREPARED_STATEMENT_GENERATED_ FB.SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	Implemented
Findbugs	1.0	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	Implemented

Coding Standard 5

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

Noncompliant Code

Allocates zero bytes of memory through a call to operator new(). If this is true, new() returns a non-null pointer value. However, attempting to dereference memory through such a pointer results in undefined behavior.

```
#include <new>

void f() noexcept(false) {
  unsigned char *ptr = static_cast<unsigned char *>(::operator new(0));
  *ptr = 0;
  // ...
  ::operator delete(ptr);
}
```

Compliant Code

If you allocate a single unsigned char object, use new instead of a direct call to operator new().

```
void f() noexcept(false) {
  unsigned char *ptr = new unsigned char;
  *ptr = 0;
  // ...
  delete 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
High	Likely	Medium	P18	L1

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	dangling_pointer_use	
Axivion Bauhaus Suite	7.2.0	CertC++-MEM50	
Clang	3.9	clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2	Checked by clang-tidy, but does not catch all violations of this rule.
CodeSonar	6.2p0	ALLOC.UAF	Use after free

Coding Standard 6

Coding Standard	Label	Name of Standard
Assertions	CTR-58-CPP	Predicate function objects should not be mutable.

Noncompliant Code

Removes the third item in a container using a predicate lambda function that returns true only on its third call. This lambda carries local state information, which it attempts to mutate within its function call operator.

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <vector>

template <typename Iter>
void print_container(Iter b, Iter e) {
  std::cout << "Contains: ";
  std::copy(b, e, std::ostream_iterator<decltype(*b)>(std::cout, " "));
  std::cout << std::endl;
}

void f() {
  std::vector<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  print_container(v.begin(), v.end());

  int timesCalled = 0;
  v.erase(std::remove_if(v.begin(), v.end(), [timesCalled](const int &)
mutable { return ++timesCalled == 3; }), v.end());
  print_container(v.begin(), v.end());
}
```

Compliant Code

Uses std::ref to wrap the predicate in a std::reference_wrapper<T> object. The wrapper object all refer to the same underlying predicate object.

```
#include <algorithm>
#include <functional>
#include <iostream>
#include <iterator>
#include <vector>

class MutablePredicate : public std::unary_function<int, bool> {
  size_t timesCalled;
```

Compliant Code

```
public:
  MutablePredicate() : timesCalled(0) {}

  bool operator()(const int &) {
    return ++timesCalled == 3;
  }
};

template <typename Iter>
void print_container(Iter b, Iter e) {
  std::cout << "Contains: ";
  std::copy(b, e, std::ostream_iterator<decltype(*b)>(std::cout, " "));
  std::cout << std::endl;
}

void f() {
  std::vector<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  print_container(v.begin(), v.end());

  MutablePredicate mp;
  v.erase(std::remove_if(v.begin(), v.end(), std::ref(mp)), v.end());
  print_container(v.begin(), v.end());
}
```

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	High	P3	L3

Automation

Tool	Version	Checker	Description Tool
Helix QAC	2022.1	C++3225, C++3226, C++3227, C++3228, C++3229, C++3230, C++3231, C++3232, C++3233, C++3234	[Insert text.]
Parasoft C/C++test	2021.2	CERT_CPP-CTR58-a	Make predicates const pure functions
PRQA QA-C++	4.4	3225, 3226, 3227, 3228, 3229, 3230, 3231, 3232, 3233, 3234	

Coding Standard 7

Coding Standard	Label	Name of Standard
Exceptions	ERR50-CPP	Do not abruptly terminate the program

Noncompliant Code

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

f() handles all exceptions thrown by throwing_func() and does not rethrow.

```
#include <cstdlib>

void throwing_func() noexcept(false);

void f() { // Not invoked by the program except as an exit handler.
  try {
    throwing_func();
  } catch (...) {
    // Handle error
  }
}

int main() {
```

Compliant Code

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

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	Medium	P4	L3

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	stdlib-use	Partially checked
CodeSonar	6.2p0	BADFUNC.ABORT BADFUNC.EXIT	Use of abort Use of exit
Helix QAC	2022.1	C++5014	
Klocwork	2022.1	MISRA.TERMINATE CERT.ERR.ABRUPT_TERM	

Coding Standard 8

Coding Standard	Label	Name of Standard
[Student Choice]	MSC50-CPP	Do not use std::rand() for generating pseudorandom numbers

Noncompliant Code

This noncompliant code generates an ID by calling the rand() function. The IDs produced are predictable and have limited randomness. Depending on the value of RAND_MAX, the resulting value can have modulo bias.

```
#include <cstdlib>
#include <string>

void f() {
  std::string id("ID"); // Holds the ID, starting with the characters
"ID" followed
                        // by a random integer in the range [0-10000].
  id += std::to_string(std::rand() % 10000);
  // ...
}
```

Compliant Code

This code breaks random number generation into two parts: one provides random values and the other provides random values. The distribution object ensures that values are properly distributed within a given range instead of improperly distributed.

```
#include <random>
#include <string>

void f() {
  std::string id("ID"); // Holds the ID, starting with the characters
"ID" followed
                        // by a random integer in the range [0-10000].
  std::uniform_int_distribution<int> distribution(0, 10000);
  std::random_device rd;
  std::mt19937 engine(rd());
  id += std::to_string(distribution(engine));
  // ...
}
```

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	Low	P6	L2

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	bad-function (AUTOSAR.26.5.1A)	Fully checked
Axivion Bauhaus Suite	7.2.0	CertC++-MSC50	
Clang	4.0 (prerelease)	cert-msc50-cpp	Checked by clang-tidy
CodeSonar	6.2p0	BADFUNC.RANDOM.RAND	Use of rand

Coding Standard 9

Coding Standard	Label	Name of Standard
[Student Choice]	OOP52-CPP	Do not delete a polymorphic object without a virtual destructor.

Noncompliant Code

The explicit pointer is swapped with a smart pointer object, demonstrating that smart pointers suffer from the same problem as other pointers. The default deleter for std::unique_ptr calls delete on the internal pointer value.

```
#include <memory>

struct Base {
  virtual void f();
};

struct Derived : Base {};

void f() {
  std::unique_ptr<Base> b = std::make_unique<Derived()>();
}
```

Compliant Code

The destructor for Base has declared virtual destructor, ensuring that the delete operation results in well-defined behavior.

```
struct Base {
  virtual ~Base() = default;
  virtual void f();
};

struct Derived : Base {};

void f() {
  Base *b = new Derived();
  // ...
  delete b;
}
```

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	Low	P9	L2

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	non-virtual-public-destructor-in-non-final-class	Partially checked
Axivion Bauhaus Suite	7.2.0	CertC++-OOP52	
Clang	3.9	-Wdelete-non-virtual-dtor	
CodeSonar	6.2p0	LANG.STRUCT.DNVD	delete with Non-Virtual Destructor

Coding Standard 10

Coding Standard	Label	Name of Standard
[Student Choice]	CON52-CPP	Prevent data races when accessing bit-fields from multiple threads

Noncompliant Code

Adjacent bit-fields may be stored in a single memory location. Consequently, modifying adjacent bit-fields in different threads is undefined behavior.

```
struct MultiThreadedFlags {
  unsigned int flag1 : 2;
  unsigned int flag2 : 2;
};

MultiThreadedFlags flags;

void thread1() {
  flags.flag1 = 1;
}

void thread2() {
  flags.flag2 = 2;
}
```

Compliant Code

This compliant solution protects all accesses of the flags with a mutex, thereby preventing any data races.

```
#include <mutex>

struct MultiThreadedFlags {
  unsigned int flag1 : 2;
  unsigned int flag2 : 2;
};

struct MtfMutex {
  MultiThreadedFlags s;
  std::mutex mutex;
};

MtfMutex flags;

void thread1() {
```

Compliant Code

```
  std::lock_guard<std::mutex> lk(flags.mutex);
  flags.s.flag1 = 1;
}

void thread2() {
  std::lock_guard<std::mutex> lk(flags.mutex);
  flags.s.flag2 = 2;
}
```

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

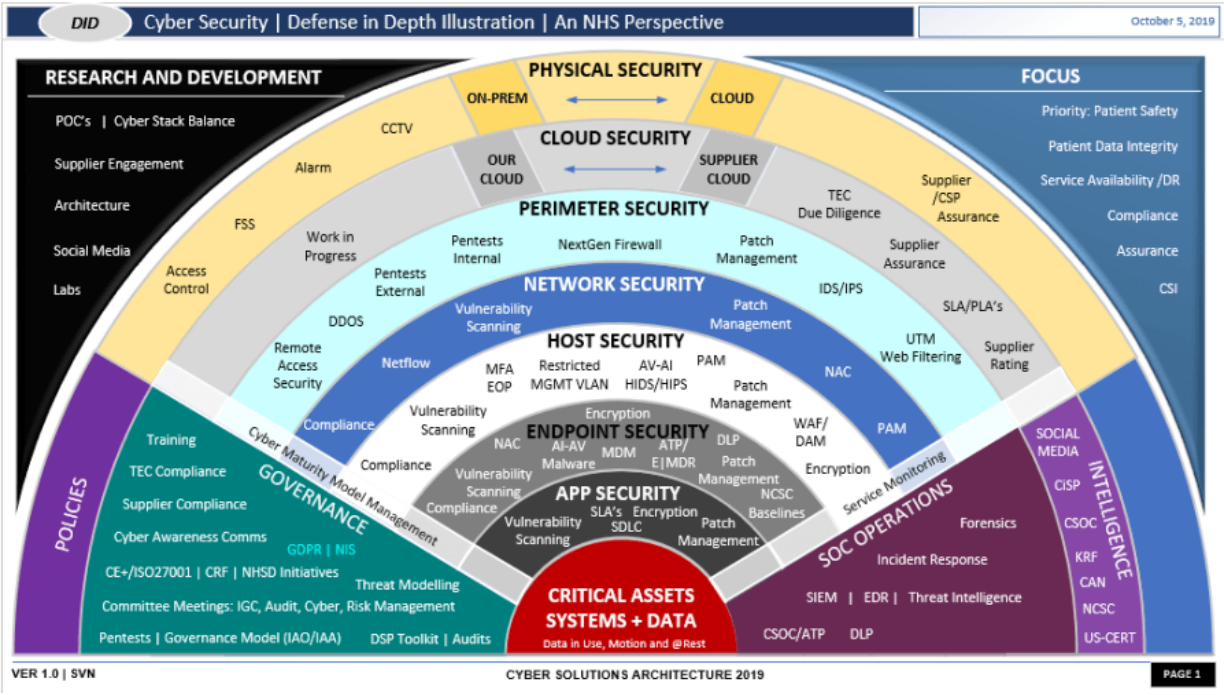
Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Probable	Medium	P8	L2

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	read_write_data_race write_write_data_race	Supported
Axivion Bauhaus Suite	7.2.0	CertC++-CON52	[Insert text.]
Coverity	6.5	RACE_CONDITION	Fully implemented
Helix QAC	2022.1	C++1774, C++1775	

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

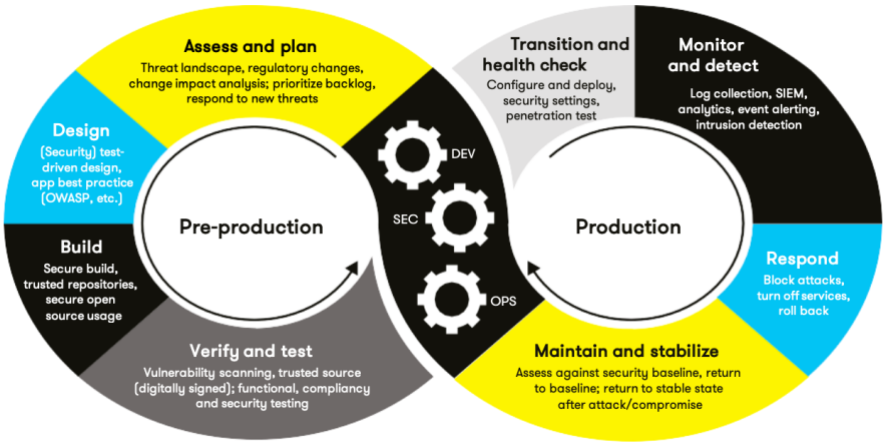
Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority,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

[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
STR51-CPP	High	Likely	Medium	P18	L1
IDS00-J	High	Probable	Medium	P12	L1
MEM50-CPP	High	Likely	Medium	P18	L1
[DCL50-CPP]	High	Probable	Medium	P12	L1
MSC50-CPP	Medium	Unlikely	Low	P6	L2
OOP52-CPP	Low	Likely	Low	P9	L2
CON52-CPP	Medium	Probable	Medium	P8	L2
STD-001-CPP	High	Unlikely	Medium	High	L2
EXP50-CPP	Medium	Probable	Medium	P8	L2
CTR-58-CPP	Low	Likely	High	P3	L3
ERR50-CPP	Low	Probable	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 in rest confirms that the attacker cannot access the unencrypted data on disk by ensuring that the data is encrypted.
Encryption at flight	Encryption at flight means that you ensure that data is being encrypted over transition to its destination.
Encryption in use	The best way to ensure that data is being encrypted while in use is to limit access to certain information. You want to ensure that only certain people can access information only necessary to their eyes.

b. Triple-A Framework*	Explain what it is and how and why the policy applies.
Authentication	This could be confirming that the username/email and password are correct before allowing further access. It could even look like confirming that the user is signing in from a certain digital certificate.
Authorization	Authorization can look like, only allowing users to access certain information through a specific I.P address. An example of this could be, allowing users to view their paystubs from home, but be able to change personal information until they are using their work desktop.
Accounting	This means to ensure that the resources employees use have certain limits. An example of this can mean, if a certain coworker forgets to sign out of their email address at the end of the day, the software should automatically time out after no use in 5 minutes, this ensures the safety of their as well as client's information.

*Use this checklist for the Triple A to be sure you include these elements in your policy:

- User logins
- Changes to the database
- Addition of new users
- User level of access
- Files accessed by users

Map the Principles

Map the principles to each of the standards and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it's time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

NOTE: Green Pace has already successfully implemented the following:

- Operating system logs
- Firewall logs
- Anti-malware logs

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

Audit Controls and Management

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

Evidence will include the following:

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

Enforcement

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

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

Exceptions Process

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

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

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

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

Distribution

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

Policy Change Control

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

Policy Version History

Version	Date	Description	Edited By	Approved By
1.0	08/05/2020	Initial Template	David Buksbaum	
2.0	03/20/2022	Rough Draft	Esi McAllen	
3.0	04/10/2022	Final Project	Esi McAllen	

Appendix A Lookups

Approved C/C++ Language Acronyms

Language	Acronym
C++	CPP
C	CLG
Java	JAV