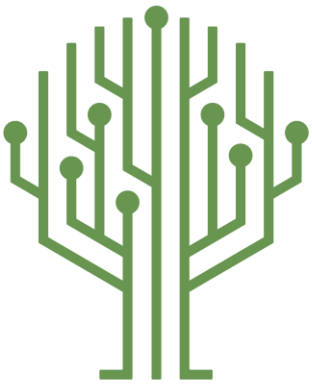


Green Pace Secure Development Policy

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	Test inputs to make sure only proper data enters an information system, and improper or malicious data is prevented.
2. Heed Compiler Warnings	Such warnings exist to notify a developer of a potential error or issue in code. An error prevents code from compiling while warnings do not, but both are equally important to consider code modification in preventing potential security risks.
3. Architect and Design for Security Policies	Consider software architecture and design when implementing security policies, such as separating a system into sub systems with different authorization or privilege levels.
4. Keep It Simple	Keeping design simple and small reduces likelihood of errors in both coding and use. This principle can also potentially minimize the complexity of security required.
5. Default Deny	By default, access is denied, and access is permitted through the conditions of the protection scheme used.
6. Adhere to the Principle of Least Privilege	Processes should execute with minimal required privileges needed to complete the job, and elevated privileges should be used as minimally as possible and with as little time as needed. This lowers the chance that an attacker has to execute code with elevated privileges
7. Sanitize Data Sent to Other Systems	Unused functions, or calls made out of context, may pass and cause damage, such as SQL injection attacks. Sanitizing data before passing the data to other systems checks these potential issues prior to invoking these systems.
8. Practice Defense in Depth	Have multiple layers to defense, which can mitigate possible exploits or damage should one layer of defense be made vulnerable.

9. Use Effective Quality Assurance Techniques	Proper testing, such as fuzz and penetration testing, as well as audits to code, can be part of an effective QA program. Security reviews, both internal and external, can help identify and correct possible issues.
10. Adopt a Secure Coding Standard	Apply coding standards in your language and platform of choice to be secure from the start.

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]	Obey the one-definition rule

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule

Noncompliant Code

Two different translation units define a class of the same name with differing definitions.

```
// a.cpp
struct S {
  int a;
};

// b.cpp
class S {
public:
  int a;
};
```

Compliant Code

Use of a header file to introduce the object into both translation units.

```
// S.h
struct S {
  int a;
};

// a.cpp
#include "S.h"

// b.cpp
#include "S.h"
```

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

Principle(s):
3: Architect and Design for Security Policies
4: Keep It Simple
10: Adopt a Secure Coding Standard

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Unlikely	High	P3	L3

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading	Partially checked
Axivion Bauhaus Suite	7.2.0	CertC++-DCL60	-
Parasoft C/C++test	2021.1	CERT_CPP-DCL60-a	A class, union or enum name (including qualification, if any) shall be a unique identifier
LDRA tool suite	9.7.1	286 S, 287 S	Fully Implemented

Coding Standard 2

Coding Standard	Label	Name of Standard
Data Value	[STD-002-CPP]	Do not read uninitialized memory

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory

Noncompliant Code

Uninitialized local variable is evaluated as part of an expression to print its value, resulting in undefined behavior.

```
#include <iostream>

void f() {
  int i;
  std::cout << i;
}
```

Compliant Code

Object is initialized prior to printing its value.

```
#include <iostream>

void f() {
  int i = 0;
  std::cout << i;
}
```

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

Principles(s):
1: Validate Input Data
4: Keep It Simple
10: Adopt a Secure Coding Standard

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Probable	Medium	P12	L1

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	uninitialized-read	Partially checked
Helix QAC	2021.2	C++2726, C++2727, C++2728, C++2961, C++2962, C++2963, C++2966, C++2967, C++2968, C++2971, C++2972, C++2973, C++2976, C++2977, C++2978	-
LDRA tool suite	9.7.1	53 D, 69 D, 631 S, 652 S	Partially implemented
Polyspace Bug Finder	R2021a	CERT C++: EXP53-CPP	Checks for: • Non-initialized variable • Non-initialized pointer Rule partially covered.

Coding Standard 3

Coding Standard	Label	Name of Standard
String Correctness	[STD-003-CPP]	Do not attempt to create a std::string from a null pointer

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer

Noncompliant Code

A std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs).

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

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

Compliant Code

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):
2: Heed Compiler Warnings

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	2021.2	C++4770, C++4771, C++4772, C++4773, C++4774	-
Klocwork	2021.1	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	2021.1	CERT_CPP-STR51-a	Avoid null pointer dereferencing

Coding Standard 4

Coding Standard	Label	Name of Standard
SQL Injection	[STD-004-CPP]	Prevent SQL injection

Java version source (for reference): https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection

Noncompliant Code

Without precautions, the untrusted data may maliciously alter the query.

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = “SELECT * FROM Users WHERE Name = " + uName + " AND Pass = " +
       uPass + ”
```

Compliant Code

The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures.

```
PreparedStatement pStmt = PreparedStatement();

std::cin >> username;
std::cin >> userpassword;

sql = “SELECT * FROM Users WHERE Name = %s AND Pass = %s;”, username,
       userpassword};
```

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

Principles(s):
1: Validate Input Data
7: Sanitize Data Sent to Other Systems
10: Adopt a Secure Coding Standard

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Probable	Medium	P12	L1

Automation

Tool	Version	Checker	Description Tool
Coverity	7.5	SQLI FB.SQL_PREPARED_STATEMENT_GENERATED_ FB.SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	Implemented
The Checker Framework	2.1.3	Tainting Checker	Trust and security errors (see Chapter 8)
Fortify	1.0	HTTP_Response_Splitting SQL_Injection__Persistence SQL_Injection	Implemented
Parasoft Jtest	2021.1	CERT.IDS00.TDSQL	Protect against SQL injection

Coding Standard 5

Coding Standard	Label	Name of Standard
Memory Protection	[STD-005-CPP]	Do not access freed memory

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory

Noncompliant Code

s is dereferenced after it has been deallocated. If this access results in a write-after-free, this can be exploited to run arbitrary code 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

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):
2: Heed Compiler Warnings
5: Default Deny
6: Adhere to the Principle of Least Privilege
9: Use Effective Quality Assurance Techniques

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	P18	L1

Automation

Tool	Version	Checker	Description Tool
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.
Coverity	V7.5.0	USE_AFTER_FREE	Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer
Parasoft C/C++test	2021.1	CERT_CPP-MEM50-a	Do not use resources that have been freed
Parasoft Insure++	-	-	Runtime detection

Coding Standard 6

Coding Standard	Label	Name of Standard
Assertions	[STD-006-CLG]	Use a static assertion to test the value of a constant expression

Source: https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression

Noncompliant Code

Uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly.

```
#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

For constant expressions, a preprocessor conditional statement may be used.

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

#if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) +
sizeof(unsigned int)))
  #error "Structure must not have any padding"
#endif
```

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

Principles(s):
2: Heed Compiler Warnings
10: Adopt a Secure Coding Standard

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	High	P1	L3

Automation

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	7.2.0	CertC-DCL03	-
Clang	3.9	misc-static-assert	Checked by clang-tidy
ECLAIR	1.2	CC2.DCL03	Fully implemented
LDRA tool suite	9.7.1	44 S	Fully implemented

Coding Standard 7

Coding Standard	Label	Name of Standard
Exceptions	[STD-007-CPP]	Do not abruptly terminate the program

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/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
  }
}
```

Compliant Code

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

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

Principles(s):
9: Use Effective Quality Assurance Techniques
10: Adopt a Secure Coding Standard

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Probable	Medium	P4	L3

Automation

Tool	Version	Checker	Description Tool
CodeSonar	6.1p0	BADFUNC.ABORT BADFUNC.EXIT	Use of abort Use of exit
Klocwork	2021.1	MISRA.CATCH.ALL CERT.ERR.ABRUPT_TERM	-
LDRA tool suite	9.7.1	122 S	Enhanced Enforcement
Polyspace Bug Finder	R2021a	CERT C++: ERR50-CPP	Checks for implicit call to terminate() function (rule partially covered)

Coding Standard 8

Coding Standard	Label	Name of Standard
Object Oriented Programming	[STD-008-CPP]	Write constructor member initializers in the canonical order

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order

Noncompliant Code

The member initializer list for C::C() attempts to initialize someVal first and then to initialize dependsOnSomeVal to a value dependent on someVal. Because the declaration order of the member variables does not match the member initializer order, attempting to read the value of someVal results in an unspecified value being stored into dependsOnSomeVal.

```
class C {
  int dependsOnSomeVal;
  int someVal;

public:
  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}
};
```

Compliant Code

Change the declaration order of the class member variables so that the dependency can be ordered properly in the constructor's member initializer list.

```
class C {
  int someVal;
  int dependsOnSomeVal;

public:
  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}
};
```

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

Principles(s):
4: Keep It Simple

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Unlikely	Medium	P4	L3

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	initializer-list-order	Fully checked
Axivion Bauhaus Suite	7.2.0	CertC++-OOP53	-
LDRA tool suite	9.7.1	206 S	Fully implemented
Parasoft C/C++test	2021.1	CERT_CPP-OOP53-a	List members in an initialization list in the order in which they are declared

Coding Standard 9

Coding Standard	Label	Name of Standard
Containers	[STD-009-CPP]	Use valid iterator ranges

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterator+ranges

Noncompliant Code

On each iteration of its internal loop, std::for_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in undefined behavior.

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

void f(const std::vector<int> &c) {
  std::for_each(c.end(), c.begin(), [](int i) { std::cout << i; });
}
```

Compliant Code

The iterator values passed to std::for_each() are passed in the proper order.

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

void f(const std::vector<int> &c) {
  std::for_each(c.begin(), c.end(), [](int i) { std::cout << i; });
}
```

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

Principles(s):
3: Architect and Design for Security Policies
4: Keep It Simple
10: Adopt a Secure Coding Standard

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Probable	High	P6	L2

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	overflow_upon_dereference	-
Helix QAC	2021.2	C++3802	-
Parasoft C/C++test	2021.1	CERT_CPP-CTR53-a CERT_CPP-CTR53-b	Do not use an iterator range that isn't really a range Do not compare iterators from different containers
PVS-Studio	7.14	V539, V662, V789	-

Coding Standard 10

Coding Standard	Label	Name of Standard
Expressions	[STD-010-CPP]	Do not access an object outside of its lifetime

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP54-CPP.+Do+not+access+an+object+outside+of+its+lifetime

Noncompliant Code

A pointer to an object is used to call a non-static member function of the object prior to the beginning of the pointer's lifetime, resulting in undefined behavior.

```
struct S {
  void mem_fn();
};

void f() {
  S *s;
  s->mem_fn();
}
```

Compliant Code

Storage is obtained for the pointer prior to calling S::mem_fn().

```
struct S {
  void mem_fn();
};

void f() {
  S *s = new S;
  s->mem_fn();
  delete s;
}
```

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

Principles(s):
2: Heed Compiler Warnings
10: Adopt a Secure Coding Standard

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level

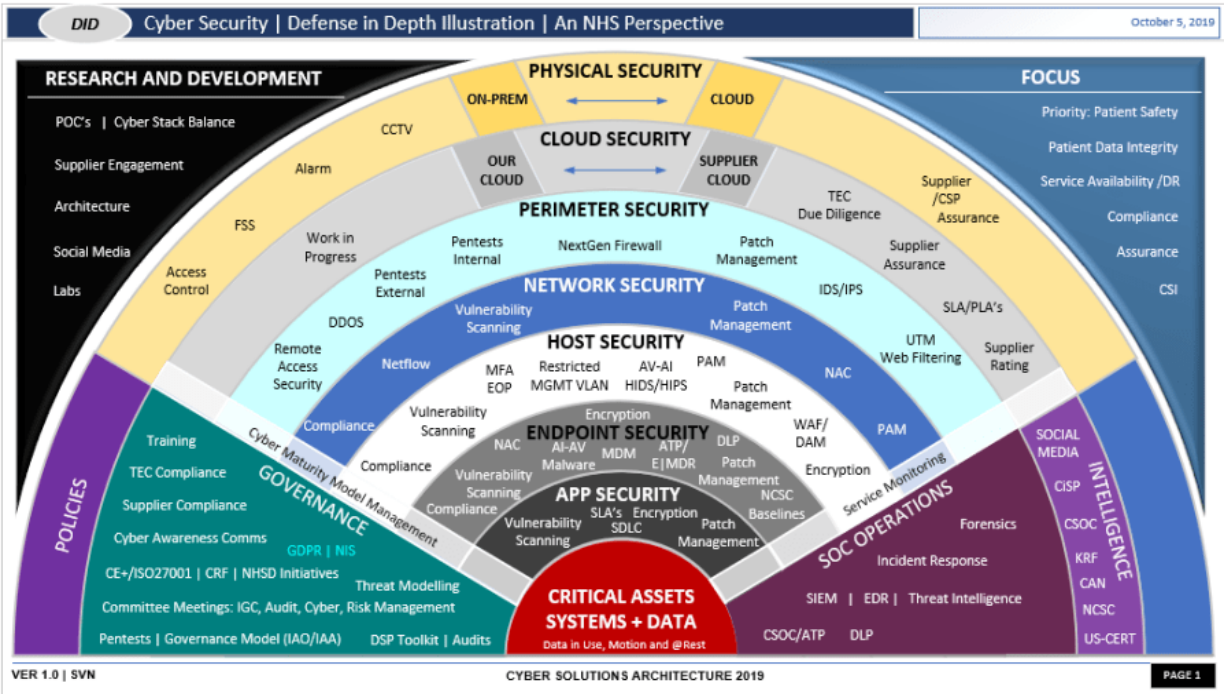
Severity	Likelihood	Remediation Cost	Priority	Level
High	Probable	High	P6	L2

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	return-reference-local dangling_pointer_use	Partially checked
Clang	3.9	-Wdangling-initializer-list	Catches some lifetime issues related to incorrect use of std::initializer_list<>
CodeSonar	6.1p0	IO.UAC ALLOC.UAF	Use after close Use after free
Parasoft C/C++test	2021.1	CERT_CPP-EXP54-a CERT_CPP-EXP54-b CERT_CPP-EXP54-c	Do not use resources that have been freed The address of an object with automatic storage shall not be returned from a function The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist

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

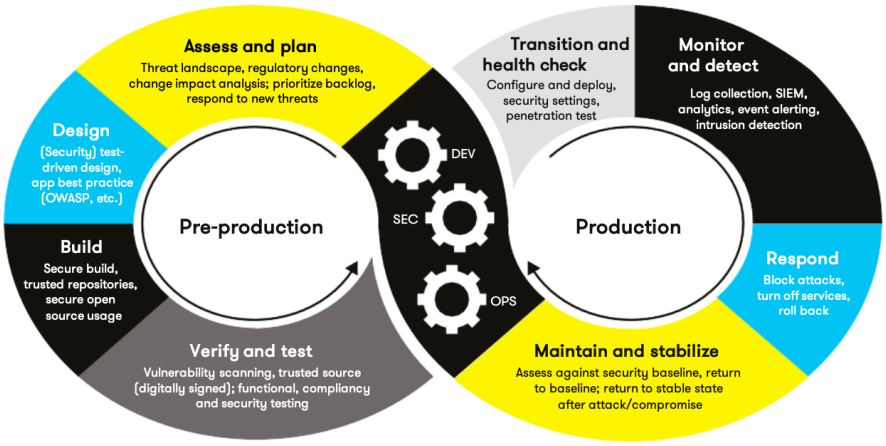
Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority,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

By integrating security measures into each step of the DevOps toolchain, DevOps becomes DevSecOps. In the "Assess and Plan" phase, threat modeling In the "Design" and "Build" phases, IDE security is addressed. Static application testing and automated security scans are added to the "Verify & Test" phase along with unit, integration and other tests.

Once in production, the automated testing continues with prevention by using integrity checks and defense-in-depth measures. Network monitoring, penetration testing, network monitoring and performance logs are some methods of continuous threat detection. Just as with testing in the QA sense, testing in the security sense should also be performed early and often.

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-002-CPP	High	Probable	Medium	High (12)	1
STD-003-CPP	High	Likely	Medium	High (18)	1
STD-004-CPP	High	Probable	Medium	High (12)	1
STD-005-CPP	High	Likely	Medium	High (18)	1
STD-010-CPP	High	Probable	High	Medium (6)	2
STD-006-CLG	Low	Unlikely	High	Low (1)	3
STD-007-CPP	Low	Probable	Medium	Low (4)	3
STD-008-CPP	Medium	Unlikely	Medium	Low (4)	3
STD-001-CPP	High	Unlikely	High	Low (3)	3

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 protects stored data. This may include hard drives, phones, computers, and cloud assets, among others. Protection of this data can be done through encryption tools, disk encryption and security for mobile devices and computers.
Encryption at flight	Encryption at flight is about protecting data that is moving. This can be between two devices within a network, or moving outside of a network. This can be protected through examples such as email encryption, DLP solutions, and solid network security features, such as firewalls and authentication. It is also important to consider the path data may be taking, and the security of this path.
Encryption in use	Encryption in use protects data that is created, edited, or otherwise defined as in-use. Protection of this data can be done by ensuring data control and protection exists prior to use, and in place in the first place. Managing access rights and identity will also minimize risk to this data.

Source: https://www.mimecast.com/blog/data-in-transit-vs-motion-vs-rest/

b. Triple-A Framework*	**Explain what it is and how and why the policy applies.**
Authentication	Authentication is the act of confirming one's identity. This can cover a range of types, but often are examples such as static passwords, one-time passwords, certifications, and biometric credentials. These forms of identification work to ensure a person is who they claim to be.
Authorization	Authorization specifies the access rights and privileges of a user, and are an important part of information and computer security. Where authentication confirms an identity, authorization determines what a user can and cannot access in the first place, limiting possible vulnerabilities when someone may interact with sensitive data they may not need to access, or the permissions one may have during access.
Accounting	Accounting is the process of keeping track of activity while interacting with a system, showing timestamps, accessed resources, and data transfer information. This is valuable in both creating a "bread crumb trail" in user activity, and also for the purposes of forensic analysis and investigation, should it be required.

Source: https://www.ccsinet.com/blog/aaa-identity-management/

*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	
1.1	05/26/2024	Module 3 Milestone	Tonessa Chatten	
1.2	06/16/2024	Module 6 Milestone	Tonessa Chatten	

Appendix A Lookups

Approved C/C++ Language Acronyms

Language	Acronym
C++	CPP
C	CLG
Java	JAV