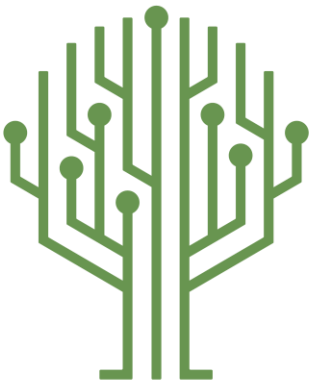


Green Pace Secure Development Policy

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	Input validation is when input from an untrusted source is validated against sets of standards for the expected input. Input from untrusted data sources should be validated on the client-side. Validating input might include checking the data entered is the right size, length, type, range, format, and content.
2. Heed Compiler Warnings	Compilers will identify errors that will prevent code from compiling and will give warnings for potential problems that will not prevent code from compiling but may cause unwanted results. Code should be compiled using the highest level of warning available. When warnings appear, code should be modified to get rid of the warnings.
3. Architect and Design for Security Policies	The architecture and design of software should be designed with the security policies in mind. Considering the security policy when creating the architecture and design of software can allow for the software to be built in such a way that is inherently more secure.
4. Keep It Simple	The more complex the software is, the more likely there are to be errors that can compromise the security of the software. It will also take more time and resources to make more complex software secure. This is why it is best to keep software simple.
5. Default Deny	Often, there are limitations to the level of access actors have. Access should be denied by default and granted only with explicit permission rather than excluding certain actors from certain access privileges. This will decrease the likelihood of an actor gaining unintended access.
6. Adhere to the Principle of Least Privilege	Access to privileges should be limited to the least set of privileges needed for the task. Privileges should also only be accessible for the least amount of time needed for the task.
7. Sanitize Data Sent to Other Systems	It is important to sanitize data from untrusted actors. Input sanitization is when you remove or modify data that could cause a vulnerability. This can be done through

Principles	Write a short paragraph explaining each of the 10 principles of security.
	blacklist sanitizing, blocking a list of malicious inputs, or whitelist sanitizing, accepting a pre-approved input only.
8. Practice Defense in Depth	It is important to have layers of defense. By implementing several defensive strategies, the software should be more secure. Layers of security mean layers attackers must get through.
9. Use Effective Quality Assurance Techniques	It is better to prevent an attack from happening rather than defend against an active attack. Implementing excellent quality assurance techniques can aid in preventing vulnerabilities. An effective quality assurance program should include penetration testing, source code audits, and fuzz testing.
10. Adopt a Secure Coding Standard	Create coding standards that encourage best practices for secure coding. Having established standards to adhere to that were created with secure coding make it easier to write secure code.

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	Do not cast to an out-of-range enumeration value

Noncompliant Code

This example checks if the value passed to the function is in the range of enumeration values after the integer is cast to enumeration type. The code below might yield undesired results due to the integer being cast to the enumeration type before verifying that the integer was within the range. This can result in unspecified behavior from the if statement.

```
Enum EnumType
{

   One,
   Two,
   Three

};

void aFunction(int myInt)
{

     EnumType myEnum = static_case<EnumType>(myInt);

   if( myEnum < One || myEnum > Three )
   {
       // error
   }

}
```

Compliant Code

The code below checks that the integer value passed into the function is within the range of the enumeration type (if it is not, it will notify user of an error) and then the integer is cast to the enumeration type.

```
Enum EnumType
{

   One,
```

Compliant Code

```
   Two,
   Three

};

void aFunction(int myInt)
{

   if( myInt < One || myInt> Three )
   {
      // error
   }

    EnumType myEnum = static_cast<EnumType>(myInt);

}
```

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

Principles(s): Validating Input – This standard relates to validating input because the best way to ensure that you do not cast an out-of-range value to an enum type would be to validate that the input being received is an acceptable value.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Unlikely	Medium	P4	L3

Automation

Tool	Version	Checker	Description Tool
Astrée	22.10	Cast-integer-to-enum	Partially checked
CodeSonar	8.1p0	LANG.CAST.COERCE LANG.CAST.VALUE	Coercion Alters Value Cast Alters Value
Parasoft C/C++ test	2023.1	CERT_CPP-INT50-a	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration
Polyspace Bug Finder	R2024a	CERT C++: INT50-CPP	Checkers for casting to out-of-range enumeration value (rule fully covered)

Coding Standard 2

Coding Standard	Label	Name of Standard
Data Value	STD-002-CPP	Ensure that container indices and iterators are within the valid range.

Noncompliant Code

The code below iterates through the elements of a vector. However, it will iterate beyond the end the of range of the vector. This will lead to an error.

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

int main()
{

   std::vector<int> myVec = {1, 2, 3, 4, 5};

   for( int i = 0;  i <= myVec.size(); ++i )
   {

      std::cout << myVec.at(i) << std::endl;

   }

}
```

Compliant Code

The below code loops through the entire vector without iterating past the last element.

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

int main()
{

   std::vector<int> myVec = {1, 2, 3, 4, 5};

   for( int i = 0;  i < myVec.size(); ++i )
   {
```

Compliant Code

```
    std::cout << myVec.at(i) << std::endl;

  }

}
```

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

Principles(s): Adopt a Secure Coding Standard – This secure coding standard maps to the coding principle Adopt a Secure Coding Standard because ensuring that indices and iterators are within range is a general best practice for coding.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	High	P9	L2

Automation

Tool	Version	Checker	Description Tool
CodeSonar	8.1p0	LANG.MEM.BO LANG.MEM.BU LANG.MEM.TO LANG.MEM.TU LANG.MEM.TBA LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.STRUCT.PARITH	Buffer overrun Buffer underrun Type overrun Type underrun Tainted buffer access Pointer before beginning of object Pointer past end of object Pointer Arithmetic
LDRA tool suite	9.7.1	45 D, 46 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X	Partially implemented
Parasoft C/C++ test	2023.1	CERT_CPP-CTR50-a	Guarantee that container indices are within the valid range
Polyspace Bug Finer	R2024a	CERT C++: CTR50-CPP	Checks for: . Array access out of bounds . Array access with tainted index . Pointer dereference with tainted offset Rule partially covered.

Coding Standard 3

Coding Standard	Label	Name of Standard
String Correctness	STD-003-CPP	Make sure storage for strings has enough space for character data and the null terminator.

Noncompliant Code

The below code declares a character array with seven elements and then accepts input into the character array. However, since the input is unbounded, there is risk of buffer overflow.

```
#include <iostream>

void aFunction()
{

   char myChar[7];
   std::cin >> myChar;

}
```

Compliant Code

The below code declares a string variable and then accepts input into the string variable. Using a string instead of a character array ensures buffer overflow or truncation will not occur.

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

void aFunction()
{

   std::string myString;
   std::cin >> myString;

}
```

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

Principles(s): Validate Input Data, Sanitize Data Sent to Other Systems , and Adopt a Secure Coding Standard– Validate input data maps to this standard because you can ensure that there is adequate storage by validating the string input. Sanitizing Data Sent to Other Systems maps to this standard because you can also ensure that there is adequate storage for all the characters and the null terminator when you sanitize the data.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	P18	L1

Automation

Tool	Version	Checker	Description Tool
Astrée	22.10	Stream-input-char-array	Partially checked + soundly supported
CodeSonar	8.1p0	MISC.MEM.NTERM LANG.MEM.BO LANG.MEM.TO	No space for null terminator Buffer overrun Type overrun
LDRA tool suite	9.7.1	489 S, 66 X, 70 X, 71 X	Partially implemented
RuleChecker	22.10	Stream-input-char-array	Partially checked

Coding Standard 4

Coding Standard	Label	Name of Standard
SQL Injection	STD-004-CPP	Do not store an already-owned pointer value in an unrelated smart pointer

Noncompliant Code

The code below contains two smart pointers that are constructed from the same pointer value. The pointer value managed by pointer2 is deleted when pointer2 is destroyed and the same pointer value is deleted when pointer1 is destroyed which leads to a double-free vulnerability.

```
#include <memory>

void aFunction()
{

   int *myInt = new int;

   std::shared_ptr<int> pointer1(myInt);

   std::shared_ptr<int> pointer2(myInt);

}
```

Compliant Code

The code below has two smart pointers that are related through copy construction. The shared pointer value is decremented (still nonzero) when pointer2 is destroyed and when pointer1 is destroyed, the shared pointer is decremented to zero.

```
#include <memory>

void aFunction()
{

   int *myInt = new int;

   std::shared_ptr<int> pointer1 = std::make_shared<int>();

   std::shared_ptr<int> pointer2(pointer1) ;

}
```

Compliant Code

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

Principles(s): Adopt a Secure Coding Standard – This principle maps to the this standard because proper memory management is apart of general best practices for coding.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	P18	L1

Automation

Tool	Version	Checker	Description Tool
Astrée	22.10	Dangling_pointer_use	-----
Axivion Bauhaus	7.2.0	CertC++-MEM56	-----
Helix QAC	2024.1	DF4721, DF4722, DF4723	-----
Parasoft C/C++ test	2023.1	CERT_CPP-MEM6-a	Do not store an already-owned pointer value in an unrelated smart pointer

Coding Standard 5

Coding Standard	Label	Name of Standard
Memory Protection	STD-005-CPP	Properly deallocate dynamically allocated resources

Noncompliant Code

In the code below a pointer to an integer array is declared. The array allocated with array new[] is deallocated with a scalar delete call. This can lead to undefined behavior.

```
void aFunction()
{

   int *myArray = new int[4];

   delete myArray;

}
```

Compliant Code

In the code below a pointer to an integer array is declared and then the memory is properly deallocated with delete [] instead of a scalar delete call.

```
void aFunction()
{

   int *myArray = new int[4];

   delete[] myArray;

}
```

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

Principles(s): Adopt a Secure Coding Standard – This principle maps to the this standard because proper memory management is apart of general best practices for coding.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	P18	L1

Automation

Tool	Version	Checker	Description Tool
Astrée	22.10	Invalid_dynamic_memory_allocaion Dangling_pointer_use	-----
Axivion Bauhaus Suite	7.2.0	CertC++-MEM51	-----
Clang	3.9	clang-analyzer-cplusplus.NewDeleteLeaks -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator	Checked by clang-tidy, but does not catch all violations of this rule
SonarQube C/C++ Plugin	4.10	S1232	-----

Coding Standard 6

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

Noncompliant Code

The code below contains an assertion test. However, it is inside a function that may or may not be executed. The diagnostic will only occur if the code path containing the assertion is executed.

```
#include <assert.h>

Struct timer
{

   Unsigned char MODE;
   Unsigned int DATA;
   Unsigned int COUNT;

};

Int myFunction(void)
{

   Assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));

}
```

Compliant Code

In this code, assertion is performed at compile time, not runtime which will result in a meaningful and informative diagnostic error message.

```
#include <assert.h>

Struct timer
{

   Unsigned char MODE;
   Unsigned int DATA;
   Unsigned int COUNT;
```

Compliant Code

```
};

Static_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int), “Structure must not have padding”);
```

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

Principles(s): Defense in Depth – This principle maps back to the standard above because assertions are one layer of security for your code.

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
CodeSonar	8.1p0	customization	Users can implement a custom ch3ck that reports uses of the asert () macro
ECLAIR	1.2	CC2.DCL03	Fully implemented

Coding Standard 7

Coding Standard	Label	Name of Standard
Exceptions	STD-007-CPP	Handle all exceptions

Noncompliant Code

In the code below, neither function catches the exception thrown. Std:: terminate() is called since no matching handler is found.

```
void throwing_function() noexcept(false);

void aFunction()
{

   throwing_function();

}

int main()
{

   aFunction();

}
```

Compliant Code

In the code below, exceptions are handled.

```
void throwing_function() noexcept(false);

void aFunction()
{

   throwing_function();

}

int main()
{
```

Compliant Code

```
  try
  {

     aFunction();

  } catch (...) {

     // handle error

  }

}
```

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

Principles(s): Defense in Depth – This principle maps back to the standard above because assertions are one layer of security for your code.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Probable	Medium	P4	L3

Automation

Tool	Version	Checker	Description Tool
Astrée	22.10	Main-function-catch-all-early-catch-all	Partially checked
Axivion Bauhaus Suite	7.2.0	CertC++-ERR51	-----
CodeSonar	8.1p0	LANG.STRUCT.UCTCH	Unreachable Catch
Helix QAC	2024.1	C++4035, C++4036, C++4037	-----

Coding Standard 8

Coding Standard	Label	Name of Standard
Error Handling	STD-008-CPP	Choose an appropriate termination strategy

Noncompliant Code

Abort() is called after data is sent to an open file which may affect if data is written to the file.

```
#include <stdlib.h>
#include <stdio.h>

Int write_data(void)
{

   Const char *fileName = “file.txt”;
   FILE *myFile = fopen(fileName,  “w”);

   If(myFile == NULL)
   {

   /* handle error*/

   }

   Fprintf(myFile, “some text\n”);

   Abort();

   Return 0;

}

Int main(void)
{

   Write_data():
   Return EXIT_SUCCESS;

}
```

Compliant Code

Calling exit is a typical way to end a program.

#include <stdlib.h>
 	
If(/* a condition */)
{
 	
  Exit(EXIT_FAILURE)
 	
}

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

Principles(s): Adopt a Secure Coding Standard – This principle maps back to the standard above because choosing an appropriate termination strategy is a general best coding practice. By adoptint this as a standard, code will be more secure.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Probable	High	P4	L3

Automation

Tool	Version	Checker	Description Tool
Parasoft C/C++test	2023.1	CERT_C-ERR04-a CERT_C-ERR04-b VART_C-ERR04-c	The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used The 'exit()' function from the 'stdlib.h' or 'cstdlib' library shall not be used The 'quick_exit()' and '_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used
PC-lint Plus	1.4	586	Fully supported
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]

Coding Standard 9

Coding Standard	Label	Name of Standard
Input Output	STD-009-CPP	Take care when calling remove() on an open file

Noncompliant Code

File is removed while still open

```
Char *fileName;
FILE *myFile;

/* initaialize fileName */

MyFile = fopen(fileName, “w+”);
If(myFile == NULL)
{

   /* handle error */

}

If (remove(fileName) != 0)
{

   /* handle error  */

}

Fclose(myFile);
```

Compliant Code

Unlink() unlinks the file from the file system hierarchy while keeping the file on disk until it is closed.

```
Char *fileName;
FILE *myFile;

/* initaialize fileName */

MyFile = fopen(fileName, “w+”);
If(myFile == NULL)
{
```

Compliant Code

```
   /* handle error */

}

If (unlink(fileName) != 0)
{

   /* handle error  */

}

Fclose(myFile);
```

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

Principles(s): Adopt a Secure Coding Standard – This principle maps back to the standard above because managing files is a general best coding practice. By adopting this as a standard, code will be more secure.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Probable	High	P4	L3

Automation

Tool	Version	Checker	Description Tool
CodeSonar	8.1p0	customization	Users can implement a custom check for calls to remove() on a file that is currently open.
Compass/ROSE	-----	-----	-----
Helix QAC	2024.1	C5014	-----
LDRA tool suite	9.7.1	81 D	Fully implemented

Coding Standard 10

Coding Standard	Label	Name of Standard
Strings	STD-010-CPP	Do not inadvertently truncate a string

Noncompliant Code

The below code leads to truncation

```
Char *myString;
Char a[16];

Strncpy(a, myString, sizeof(a));
```

Compliant Code

The below code prevents truncation

```
Char *myString = NULL;
Char a[16];

If(myString == NULL)
{

   /* handle error */

}
Else
{

   Strcpy(a, myString);

}
```

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

Principles(s): Adopt a Secure Coding Standard – This principle maps back to the standard above because preventing truncation is a general best coding practice. By adopting this as a standard, code will be less likely to have unintended behavior and be more secure.

Threat Level

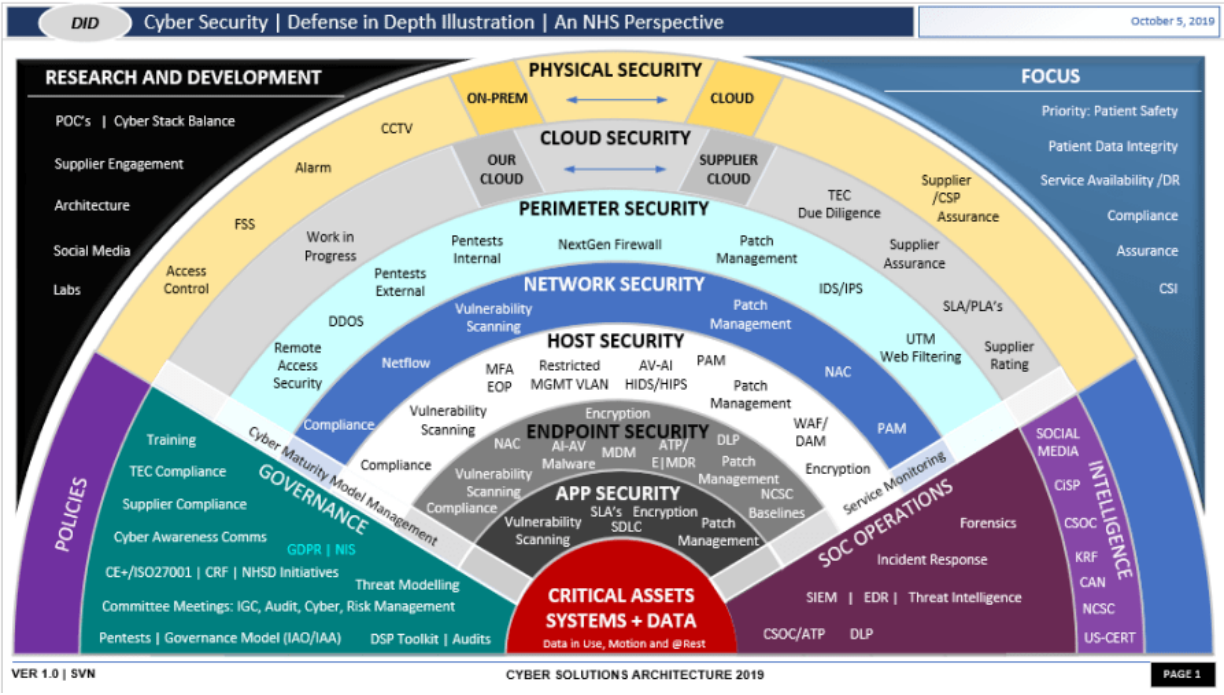
Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Portable	Medium	P8	L2

Automation

Tool	Version	Checker	Description Tool
CodeSonar	8.1p0	MISC.MEM.NTERM	No Space For Null Terminator
GCC	8.1	-Wstringop-truncation	Detects string truncation by strncat and strncpy.
LDRA tool suite	9.7.1	115 S, 44 S	Partially implemented
Klocwork	2024.1	NNTS.MIGHT NNTS.MUST	-----

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

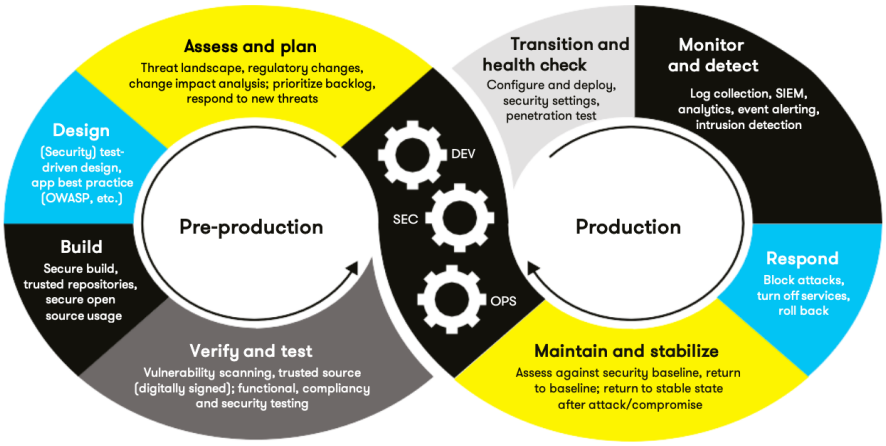
Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, 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
STD-001-CPP	Medium	Unlikely	Medium	P4	L3
STD-002-CPP	High	Likely	High	P9	L2
STD-003-CPP	High	Likely	Medium	P18	L1
STD-004-CPP	High	Likely	Medium	P18	L1
STD-005-CPP	High	Likely	Medium	P18	L1
STD-006-CPP	Low	Unlikely	High	P1	L3
STD-007-CPP	Low	Probable	Medium	P4	L3
STD-008-CPP	Medium	Probable	High	P4	L3
STD-009-CPP	Medium	Probable	High	P4	L3

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
STD-010-CPP	Medium	Probable	Medium	P8	L2

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 at rest	Encryption at rest means data is encrypted while stored. This protects it from unauthorized access because only those with the decryption key can understand the encrypted data, which is transformed using an algorithm. This policy ensures data in the system remains secure.
Encryption in flight	Encryption in flight occurs when data is encrypted while it travels through a flightnetwork. This prevents data theft during transmission. Authorized senders and receivers each have a key to decrypt the data. This policy is important for safeguarding data during its transition across networks.
Encryption in use	Encryption in use is data at rest, data in use, and data in transit. This data should remain secure at all times. This policy guarantees that sensitive data remains protected, providing assurance to customers, partners, and businesses that their information is safe.

b. Triple-A Framework*	**Explain what it is and how and why the policy applies.**
Authentication	Authentication is important because it allows businesses to control who can access their systems. This is achieved by creating unique usernames, passwords, or signatures. It ensures that only authorized users can access the system, preventing unauthorized access and data exploitation.

b. Triple-A Framework*	Explain what it is and how and why the policy applies.
Authorization	Authorization comes right after authentication because the system must check if these users are indeed authenticated. If data is accessed by too many users who lack clearance, it can lead to security problems.
Accounting	Accounting is a system of checks and balances that works by monitoring the system and tracking the usage and collection of data.

*****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	
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]
[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