Green Pace

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	Make sure that protocols are in place that prevent users from entering incorrect data, either by type or value, so that errors are avoided. Provide confirmation notifications to reiterate that input is either valid or invalid.
2. Heed Compiler Warnings	Use the error and bug messages from your complier to tell you where the problems are that need to be fixed. Compiler warnings should be used as a guide to improve the code and make it free from errors.
3. Architect and Design for Security Policies	Design all aspects of an IT infrastructure with defense and protection in mind. Build systems and then try to break them to identify weak points to be fixed.
4. Keep It Simple	Do not overcomplicate code and write functions and methods as modularly as possible so that errors may be more easily pinpointed and addressed within a large program.
5. Default Deny	Always set default permissions to deny access without the correct approval in order to prevent unauthorized users from making changes that would damage the code or program.
6. Adhere to the Principle of Least Privilege	Provide all users with the least amount of privileges needed to complete their authorized tasks. Do not provide additional access or administrative privileges to those who do not need it in order to maintain better accountability of what each user has access to.
7. Sanitize Data Sent to Other Systems	Ensure that data is correct, validated, and bug free before distributing it to another system to reduce the chance of transmitting errors.
8. Practice Defense in Depth	All IT infrastructure setups should contain multiple layers of defense in order to avoid single points of failure. IT systems should be protected physically in addition to internally in both software and hardware.

Principles	Write a short paragraph explaining each of the 10 principles of security.
9. Use Effective Quality Assurance Techniques	Set up multiple points of data validation throughout the coding process. Have a group of different people like testers and non-developers review the quality of your code and programs for a thorough assessment.
10. Adopt a Secure Coding Standard	Use industry known coding standards that are transferable and thoroughly developed. Become familiar with guides and exemplars to help better and improve your own 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 Value	[INT50-CPP]	Do not cast to an out-of-range enumeration value.

Noncompliant Code

This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two's complement system, the valid range of values that can be represented by `EnumType` are [0..3], so if a value outside of that range were passed to `f()`, the cast to `EnumType` would result in an unspecified value, and using that value within the `if` statement results in unspecified behavior.

```
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 compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator 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	Unlikely	Medium	4	3

Automation

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	7.2.0	**CertC++-INT50**	
Helix QAC	2021.1	**C++3013**	
Parasoft C/C++test	2021.1	**CERT_CPP-INT50-a**	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration
PRQA QA-C++	4.4	**3013**	
PVS-Studio	7.13	**V1016**	

Coding Standard 2

Coding Standard	Label	Name of Standard
Data Type	[EXP40-C]	Do not modify constant objects.

Noncompliant Code

This noncompliant code example allows a constant object to be modified.

```
const int **ipp;
int *ip;
const int i = 42;

void func(void) {
  ipp = &ip; /* Constraint violation */
  *ipp = &i; /* Valid */
  *ip = 0;   /* Modifies constant i (was 42) */
}
```

Compliant Code

The compliant solution depends on the intent of the programmer. If the intent is that the value of `i` is modifiable, then it should not be declared as a constant, as in this compliant solution.

```
int **ipp;
int *ip;
int i = 42;

void func(void) {
  ipp = &ip; /* Valid */
  *ipp = &i; /* Valid */
  *ip = 0; /* Valid */
}
```

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

Principles(s): Heed compiler warnings, validate input data

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Medium	2	3

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	**assignment-to-non-modifiable-lvalue** **pointer-qualifier-cast-const** **pointer-qualifier-cast-const-implicit** **write-to-constant-memory**	Fully checked
Axivion Bauhaus Suite	7.2.0	**CertC-EXP40**	
Coverity	2017.07	**PW** **MISRA C 2004 Rule 11.5**	Implemented
Helix QAC	2021.1	**C0563**	
LDRA tool suite	9.7.1	**582 S**	Fully implemented
Parasoft C/C++test	2021.1	**CERT_C-EXP40-a**	A cast shall not remove any 'const' or 'volatile' qualification from the type of a pointer or reference
Polyspace Bug Finder	R2021a	CERT C: Rule EXP40-C	Checks for write operations on const qualified objects (rule fully covered)
PRQA QA-C	9.7	**0563**	Partially implemented
RuleChecker	20.10	**assignment-to-non-modifiable-lvalue** **pointer-qualifier-cast-const** **pointer-qualifier-cast-const-implicit**	Partially checked
TrustInSoft Analyzer	1.38	**mem_access**	Exhaustively verified (see the compliant and the non-compliant example).

Coding Standard 3

Coding Standard	Label	Name of Standard
String Correctness	[STR30-C]	Do not attempt to modify string literals.

Noncompliant Code

In this noncompliant code example, the `char` pointer `str` is initialized to the address of a string literal. Attempting to modify the string literal is undefined behavior.

```
char *str  = "string literal";
str[0] = 'S';
```

Compliant Code

As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. This code creates a copy of the string literal in the space allocated to the character array `str`. The string stored in `str` can be modified safely.

```
char str[] = "string literal";
str[0] = 'S';
```

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	9	2

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	**string-literal-modfication** **write-to-string-literal**	Fully checked
Axivion Bauhaus Suite	7.2.0	**CertC-STR30**	Fully implemented
Compass/ROSE			Can detect simple violations of this rule
Coverity	2017.07	**PW**	Deprecates conversion from a string literal to "char *"
Klocwork	2021.1	**CXX.OVERWRITE_CONST_CHAR**	

Tool	Version	Checker	Description Tool
LDRA tool suite	9.7.1	**157 S**	Partially implemented
Parasoft C/C++test	2021.1	**CERT_C-STR30-a** **CERT_C-STR30-b**	A string literal shall not be modified Do not modify string literals
PC-lint Plus	1.4	**489, 1776**	Partially supported
Polyspace Bug Finder	R2021a	CERT C: Rule STR30-C	Checks for writing to const qualified object (rule fully covered)
PRQA QA-C	9.7	**0556, 0752, 0753, 0754**	Partially implemented
PRQA QA-C++	4.4	**3063, 3064, 3605, 3606, 3607, 3842**	
PVS-Studio	7.13	**V675**	
RuleChecker	20.10	**string-literal-modfication**	Partially checked
Splint	3.1.1		
TrustInSoft Analyzer	1.38	mem_access	Exhaustively verified (see one compliant and one non-compliant example).

Coding Standard 4

Coding Standard	Label	Name of Standard
SQL Injection	[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 output 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;
}
```

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

Principles(s): Architect and design for security policies, sanitize data sent to other systems

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Likely	Medium	6	2

Automation

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	7.2.0	**CertC++-FIO50**	
Helix QAC	2021.1	**C++4711, C++4712, C++4713**	
Parasoft C/C++test	2021.1	**CERT_CPP-FIO50-a**	Do not alternately input and output from a stream without an intervening flush or positioning call
Polyspace Bug Finder	R2020a	CERT C++: FIO50-CPP	Checks for alternating input and output from a stream without flush or positioning call (rule fully covered)

Coding Standard 5

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

Noncompliant Code

In this noncompliant code example, `s` is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems.

```
#include <new>

struct S {
  void f();
};

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

Compliant Code

In this compliant solution, 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): Keep it simple, sanitize data sent to other systems

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	18	1

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	6.0p0	**ALLOC.UAF**	Use after free
Compass/ROSE			
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
Helix QAC	2021.1	**C++4303, C++4304**	
Klocwork	2021.1	**UFM.DEREF.MIGHT** **UFM.DEREF.MUST** **UFM.FFM.MIGHT** **UFM.FFM.MUST** **UFM.RETURN.MIGHT** **UFM.RETURN.MUST** **UFM.USE.MIGHT** **UFM.USE.MUST**	
LDRA tool suite	9.7.1	**483 S, 484 S**	Partially implemented
Parasoft C/C++test	2021.1	**CERT_CPP-MEM50-a**	Do not use resources that have been freed
Parasoft Insure++			Runtime detection
Polyspace Bug Finder	R2020a	CERT C++: MEM50-CPP	Checks for: • Pointer access out of bounds • Deallocation of previously deallocated pointer • Use of previously freed pointer Rule partially covered.
PRQA QA-C++	4.4	**4303, 4304**	

Tool	Version	Checker	Description Tool
PVS-Studio	7.13	**V586**, **V774**	
Splint	5.0		

Coding Standard 6

Coding Standard	Label	Name of Standard
Assertions	[DCL31-C]	Declare identifiers before using them.

Noncompliant Code

This noncompliant code example omits the type specifier.

```
extern foo;
```

Compliant Code

This compliant solution explicitly includes a type specifier.

```
extern int foo;
```

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

Principles(s): Validate input data, heed compiler warnings

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Low	3	3

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	**type-specifier** **function-return-type** **implicit-function-declaration** **undeclared-parameter**	Fully checked
Axivion Bauhaus Suite	7.2.0	**CertC-DCL31**	Fully implemented
Clang	3.9	`-Wimplicit-int`	
Compass/ROSE			

Tool	Version	Checker	Description Tool
Coverity	2017.07	**MISRA C 2012 Rule 8.1**	Implemented
ECLAIR	1.2	**CC2.DCL31**	Fully implemented
GCC	4.3.5		Can detect violations of this rule when the `-Wimplicit` and `-Wreturn-type` flags are used
Helix QAC	2021.1	**C0434, C2050, C2051, C3335**	
Klocwork	2021.1	**CWARN.IMPLICITINT** **FUNCRET.IMPLICIT** **MISRA.DECL.NO_TYPE** **MISRA.FUNC.NOPROT.CALL** **RETVOID.IMPLICIT**	
LDRA tool suite	9.7.1	**24 D, 41 D, 20 S, 326 S, 496 S**	Fully implemented
Parasoft C/C++test	2021.1	**CERT_C-DCL31-a**	All functions shall be declared before use
PC-lint Plus	1.4	**601, 718, 746, 808**	Fully supported
Polyspace Bug Finder	R2021a	CERT C: Rule DCL31-C	Checks for: • Types not explicitly specified • Implicit function declaration Rule fully covered.
PRQA QA-C	9.7	**0434 (C)** **2050** **2051** **3335**	Fully implemented
PVS-Studio	7.13	**V1031**	
SonarQube C/C++ Plugin	3.11	**S819**, **S820**	Partially implemented; implicit return type not covered.
RuleChecker	20.10	**type-specifier** **function-return-type** **implicit-function-declaration** **undeclared-parameter**	Fully checked
TrustInSoft Analyzer	1.38	**type specifier missing**	Partially verified (exhaustively detects undefined behavior).

Coding Standard 7

Coding Standard	Label	Name of Standard
Exceptions	[DCL57-CPP]	Do not let exceptions escape from destructors or deallocation functions.

Noncompliant Code

In this noncompliant code example, the class destructor does not meet the implicit `noexcept` guarantee because it may throw an exception even if it was called as the result of an exception being thrown. Consequently, it is declared as `noexcept(false)` but still can trigger undefined behavior.

```
#include <stdexcept>

class S {
  bool has_error() const;

public:
  ~S() noexcept(false) {
    // Normal processing
    if (has_error()) {
      throw std::logic_error("Something bad");
    }
  }
};
```

Compliant Code

A destructor should perform the same way whether or not there is an active exception. Typically, this means that it should invoke only operations that do not throw exceptions, or it should handle all exceptions and not rethrow them (even implicitly). This compliant solution differs from the previous noncompliant code example by having an explicit `return` statement in the `SomeClass` destructor. This statement prevents control from reaching the end of the exception handler. Consequently, this handler will catch the exception thrown by `Bad::~Bad()` when `bad_member` is destroyed. It will also catch any exceptions thrown within the compound statement of the *function-try-block*, but the `SomeClass` destructor will not terminate by throwing an exception.

```
class SomeClass {
  Bad bad_member;
public:
  ~SomeClass()
  try {
    // ...
  } catch(...) {
    // Catch exceptions thrown from noncompliant destructors of
    // member objects or base class subobjects.

    // NOTE: Flowing off the end of a destructor function-try-block causes
    // the caught exception to be implicitly rethrown, but an explicit
    // return statement will prevent that from happening.
    return;
  }
```

Compliant Code

```
};
```

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

Principles(s): Use effective quality assurance techniques

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Likely	Medium	6	2

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	**destructor-without-noexcept delete-without-noexcept**	Fully checked
Axivion Bauhaus Suite	7.2.0	**CertC++-DCL57**	
Helix QAC	2021.1	**C++2045, C++2047, C++4032, C++4631**	
LDRA tool suite	9.7.1	**453 S**	Partially implemented
Parasoft C/C++test	2021.1	**CERT_CPP-DCL57-a CERT_CPP-DCL57-b**	Never allow an exception to be thrown from a destructor, deallocation, and swap Always catch exceptions
Polyspace Bug Finder	R2020a	CERT C++: DCL57-CPP	Checks for class destructors exiting with an exception (rule partially covered)
PVS-Studio	7.13	**V509, V1045**	
RuleChecker	20.10	**destructor-without-noexcept delete-without-noexcept**	Fully checked

Coding Standard 8

Coding Standard	Label	Name of Standard
Memory Protection	[MEM51-CPP]	Properly deallocate dynamically allocated resources.

Noncompliant Code

In this noncompliant code example, the local variable `space` is passed as the expression to the placement `new` operator. The resulting pointer of that call is then passed to `::operator delete()`, resulting in undefined behavior due to `::operator delete()` attempting to free memory that was not returned by `::operator new()`.

```
#include <iostream>

struct S {
  S() { std::cout << "S::S()" << std::endl; }
  ~S() { std::cout << "S::~S()" << std::endl; }
};

void f() {
  alignas(struct S) char space[sizeof(struct S)];
  S *s1 = new (&space) S;

  // ...

  delete s1;
}
```

Compliant Code

This compliant solution removes the call to `::operator delete()`, instead explicitly calling `s1`'s destructor. This is one of the few times when explicitly invoking a destructor is warranted.

```
#include <iostream>

struct S {
  S() { std::cout << "S::S()" << std::endl; }
  ~S() { std::cout << "S::~S()" << std::endl; }
};

void f() {
  alignas(struct S) char space[sizeof(struct S)];
  S *s1 = new (&space) S;

  // ...

  s1->~S();
}
```

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

Principles(s): Keep it simple, adhere to the principle of least privilege

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	18	1

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	**invalid_dynamic_memory_allocation** **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

CodeSonar	6.0p0	**ALLOC.FNH** **ALLOC.DF** **ALLOC.TM**	Free non-heap variable Double free Type mismatch
Helix QAC	2021.1	**C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339, C++4262, C++4263, C++4264**	
Klocwork	2021.1	**CL.FFM.ASSIGN** **CL.FFM.COPY** **CL.FMM** **FMM.MIGHT** **FMM.MUST** **FNH.MIGHT** **FNH.MUST** **FUM.GEN.MIGHT** **FUM.GEN.MUST** **UNINIT.CTOR.MIGHT** **UNINIT.CTOR.MUST** **UNINIT.HEAP.MIGHT** **UNINIT.HEAP.MUST** **UNINIT.STACK.ARRAY.MIGHT** **UNINIT.STACK.ARRAY.PARTIAL.MUST**	

Tool	Version	Checker	Description Tool
		UNINIT.STACK.ARRAY.MUST **UNINIT.STACK.MIGHT** **UNINIT.STACK.MUST**	
LDRA tool suite	9.7.1	**232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D**	Partially implemented
Parasoft C/C++test	2021.1	**CERT_CPP-MEM51-a** **CERT_CPP-MEM51-b** **CERT_CPP-MEM51-c** **CERT_CPP-MEM51-d**	Use the same form in corresponding calls to new/malloc and delete/free Always provide empty brackets ([]) for delete when deallocating arrays Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor Properly deallocate dynamically allocated resources
Parasoft Insure++			Runtime detection
Polyspace Bug Finder	R2020a	CERT C++: MEM51-CPP	Checks for: • Invalid deletion of pointer • Invalid free of pointer • Deallocation of previously deallocated pointer Rule partially covered.
PRQA QA-C++	4.4	**2110, 2111, 2112, 2113, 2118,** **3337, 3339, 4262, 4263, 4264**	
PVS-Studio	7.13	**V515**, **V554**, **V611**, **V701**, **V748**, **V773**, **V1066**	
SonarQube C/C++ Plugin	4.10	**S1232**	

Coding Standard 9

Coding Standard	Label	Name of Standard
Exceptions	[ERR55-CPP]	Honor exception specifications.

Noncompliant Code

In this noncompliant code example, a function is declared as nonthrowing, but it is possible for `std::vector::resize()` to throw an exception when the requested memory cannot be allocated.

```
#include <cstddef>
#include <vector>

void f(std::vector<int> &v, size_t s) noexcept(true) {
  v.resize(s); // May throw
}
```

Compliant Code

In this compliant solution, the function's *noexcept-specification* is removed, signifying that the function allows all exceptions.

```
#include <cstddef>
#include <vector>

void f(std::vector<int> &v, size_t s) {
  v.resize(s); // May throw, but that is okay
}
```

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

Principles(s): Keep it simple, sanitize data before sending to other systems

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Likely	Low	9	2

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	**unhandled-throw-noexcept**	Partially checked
Axivion Bauhaus Suite	7.2.0	**CertC++-ERR55**	
Helix QAC	2021.1	**C++4035, C++4036, C++4632**	

Tool	Version	Checker	Description Tool
LDRA tool suite	9.7.1	**56 D**	Partially implemented
Parasoft C/C++Test	2021.1	**CERT_CPP-ERR55-a**	Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s)
PRQA QA-C++	4.4	**4035, 4036, 4632**	
RuleChecker	20.10	**unhandled-throw-noexcept**	Partially checked

Coding Standard 10

Coding Standard	Label	Name of Standard
String Correctness	[STR38-C]	Do not confuse narrow and wide character strings and functions.

Noncompliant Code

This noncompliant code example incorrectly uses the `strncpy()` function in an attempt to copy up to 10 wide characters. However, because wide characters can contain null bytes, the copy operation may end earlier than anticipated, resulting in the truncation of the wide string.

```
#include <stddef.h>
#include <string.h>

void func(void) {
  wchar_t wide_str1[]  = L"0123456789";
  wchar_t wide_str2[] =  L"0000000000";

  strncpy(wide_str2, wide_str1, 10);
}
```

Compliant Code

This compliant solution uses the proper-width functions. Using `wcsncpy()` for wide character strings and `strncpy()` for narrow character strings ensures that data is not truncated and buffer overflow does not occur.

```
#include <string.h>
#include <wchar.h>

void func(void) {
  wchar_t wide_str1[] = L"0123456789";
  wchar_t wide_str2[] = L"0000000000";
  /* Use of proper-width function */
  wcsncpy(wide_str2, wide_str1, 10);

  char narrow_str1[] = "0123456789";
  char narrow_str2[] = "0000000000";
  /* Use of proper-width function */
  strncpy(narrow_str2, narrow_str1, 10);
}
```

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

Principles(s): Keep it simple, use effective quality assurance techniques

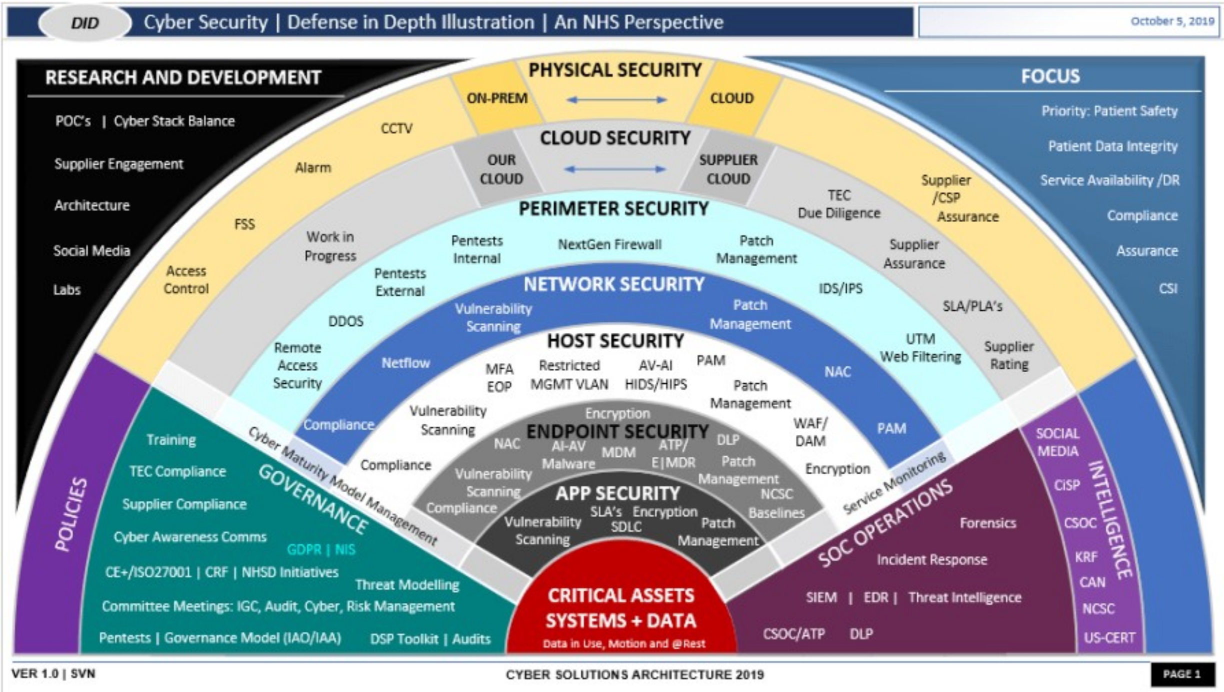
Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Low	27	1

Automation

Tool	Version	Checker	Description Tool
Astrée	20.10	**wide-narrow-string-cast** **wide-narrow-string-cast-implicit**	Partially checked
Axivion Bauhaus Suite	7.2.0	**CertC-STR38**	Fully implemented
Clang	3.9	`-Wincompatible-pointer-types`	
Coverity	2017.07	**PW**	Implemented
Helix QAC	2021.1	**C0432** **C++0403**	
Parasoft C/C++test	2021.1	**CERT_C-STR38-a**	Do not confuse narrow and wide character strings and functions
PC-lint Plus	1.4	**2454, 2480, 2481**	Partially supported: reports illegal conversions involving pointers to char or wchar_t as well as byte/wide-oriented stream inconsistencies
Polyspace Bug Finder	R2021a	CERT C: Rule STR38-C	Checks for misuse of narrow or wide character string (rule fully covered)
PRQA QA-C	9.7	**0432**	
PRQA QA-C++	4.4	**0403**	
RuleChecker	20.10	**wide-narrow-string-cast** **wide-narrow-string-cast-implicit**	Partially checked
TrustInSoft Analyzer	1.38	**pointer arithmetic**	Partially verified.

Defense-in-Depth Illustration
This illustration provides a visual representation of the defense-in-depth best practice of layered security.



Project One

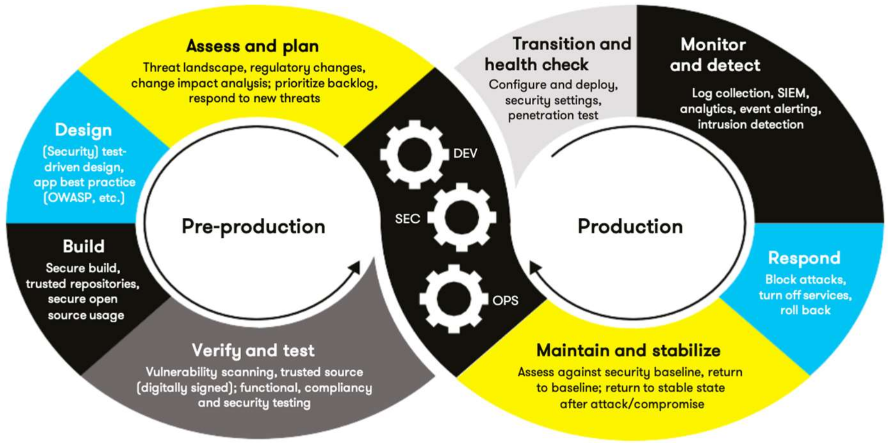
There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

1. **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.

2. **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.

3. **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.

4. **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.

DevSecOps adopts the philosophy of project management tenants as well as security standards. The infinity symbol demonstrates that you organization should be rotating through bot aspect during a project continuously. There must be continuous iterations of both the security components and operational components of the project while it is developed so that each team can learn from the others success.

5. **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
DCL31-C	Low	Unlikely	Low	3	3
DCL57-CPP	Low	Likely	Medium	6	2
ERR55-CPP	Low	Likely	Low	9	2
EXP40-C	Low	Unlikely	Medium	2	3
FIO50-CPP	Low	Likely	Medium	6	2
INT50-CPP	Medium	Unlikely	Medium	4	3
MEM50-CPP	High	Likely	Medium	18	1
MEM51-CPP	High	Likely	Medium	18	1
STR30-C	Low	Likely	Low	9	2
STR38-C	High	Likely	Low	27	1

6. **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	This means to safeguard your data while it is stored on whatever drive, server, or cloud hosts it. While data is unused, it must be encrypted to protect against any attacks trying to harvest the data. Hacking data at rest is a popular tactic in black-hat hacking, corporate espionage, and enabling ransomware. The higher the level of encryption means the higher degree of confidentiality you can assure the stakeholders within your organization. Encryption at rest not only refers to the data while it is being unused, it also refers to the physical components used in data storage, which is why it is essential to password protect the functionality of physical drives.
Encryption at flight	This means when the data is being accessed from its storage and initially sent to another point, whether that be a sever, application, program, or end user for review. Throughout the trip that the data makes though different platforms, languages, and Oss it must remain encrypted to safeguard against hackers that target data in transit. There are routers and transmissive devices that may have weak TCP/IP protocols, if any at all. Therefore, since data is susceptible in transmission by devices like sniffers or others that capture data while moving, encryption at flight is just as important as any other tenant of cybersecurity.
Encryption in use	While in use, this is the optimal time for data to be altered in some way to meet the demands of the end user. When data is being constantly used in calculations and by multiple programs and platforms make this the prime time for hackers to change both its contents, and also its metadata. Encryption in use is probably the most important of the three because this is when the attacker also has the means of the native program to modify it to their means. Altering any part of a database's fields or records or schema part can cause catastrophic change.

b. Triple-A Framework*	Explain what it is and how and why the policy applies.
Authentication	Authentication is the first wall of defense in the triple-A framework. This is the most outer layer that uses methods to ensure that the user logging on for the desired access is in fact that use. Authentication has developed much since the 90's and no MFA, or even just 2FA, is always required to safeguard against potential identify theft and corporate espionage. This enforces users to have access only to the predesignated drives that the DBA approves them for. MFA and 2FA are added security against people who might have ascertained someone's password so that there are "back-ups" in place when that happens.
Authorization	Authorization is the system of managing permissions and access control for a particular organization. You need to implement the least level of access across the organization in order to help prevent users who are authorize, but unwanted in certain areas. Once the system knows who a user is, then they present them with all the viable options in searching your organization's data that they are restricted to. In order to enforce this, you need to have a secure and well architected network and hierarchy of administrators, managers, and users.
Accounting	This piece is the aftermath of what has occurred in your system in order to gauge whether it is running successfully, or with errors. This logs and records all actions made by both users and

b. Triple-A Framework*	Explain what it is and how and why the policy applies.
	the system that is being accessed. This is the internal audit of how well security is working, therefore pointing out all opportunities for your current system.

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

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

7. **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.

c. Encryption	Explain what it is and how and why the policy applies.
Encryption in rest	Heed Compiler Warnings, Architect and Design for Security Policies, Keep It Simple, Adhere to the Principle of Least Privilege, Sanitize Data Sent to Other Systems, Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard
Encryption at flight	Validate Input Data, Architect and Design for Security Policies, Sanitize Data Sent to Other Systems, Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard
Encryption in use	Validate Input Data, Architect and Design for Security Policies, Sanitize Data Sent to Other Systems, Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard

d. Triple-A Framework*	Explain what it is and how and why the policy applies.
Authentication	Validate Input Data, Architect and Design for Security Policies, Keep It Simple, Default Deny, Adhere to the Principle of Least Privilege, Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard
Authorization	Heed Compiler Warnings, Architect and Design for Security Policies, Default Deny, Adhere to the Principle of Least Privilege, Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard
Accounting	Architect and Design for Security Policies, Default Deny, Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard

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	
1.2	05/20/2021	Coding standards updates	Michael Palatta	
1.3	08/15/2021	Tools, summaries, and explanations updated	Michael Palatta	

Appendix A Lookups

Approved C/C++ Language Acronyms

Language	Acronym
C++	CPP
C	CLG
Java	JAV