**Green Pace Developer: Security Policy Guide Template**



Green Pace Secure Development Policy

Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

# 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](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

# 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. ValidateInput Data | Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including line arguments, network interfaces, environmental variables, and user-controlled files. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set. |
| 1. Keep It Simple | Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied, and the protection scheme identifies conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

## 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** | **Overload allocation and deallocation functions as a pair in the same scope** |
| --- | --- | --- |
| **Data Type** | DCL54-CPP | Allocation and deallocation functions can be overloaded at both global and class scopes. If an allocation function is overloaded in a given scope, the corresponding deallocation function must also be overloaded in the same scope (and vice versa). |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an allocation function is overloaded at global scope. However, the corresponding deallocation function is not declared. Were an object to be allocated with the overloaded allocation function, any attempt to delete the object would result in undefined behavior in violation of MEM51-CPP: Properly deallocate dynamically allocated resources. |
| #include <Windows.h>  #include <new>  void \*operator new(std::size\_t size) noexcept(false) {  static HANDLE h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  }  // No corresponding global delete operator defined. |

| **Compliant Code** |
| --- |
| In this compliant solution, the corresponding deallocation function is also defined at global scope. |
| #include <Windows.h>  #include <new>  class HeapAllocator {  static HANDLE h;  static bool init;  public:  static void \*alloc(std::size\_t size) noexcept(false) {  if (!init) {  h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  init = true;  }  if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  }  static void dealloc(void \*ptr) noexcept {  if (h) {  (void) ::HeapFree(h, 0, ptr);  }  }  };  HANDLE HeapAllocator::h = nullptr;  bool HeapAllocator::init = false;  void \*operator new(std::size\_t size) noexcept (false) {  return HeapAllocator::alloc(size);  }  void operator delete(void \*ptr) noexcept {  return HeapAllocator::dealloc(ptr);  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data |
| Validate Input Data is directly connected to DCL54-CPP because the allocation and deallocation normally would occur in times when data is inputted, either from the user, the developer, or from the actual software. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | New-delete-pairwise | Partially checked |
| Clang | 3.9 | misc-new-delete-overloads | Check with clang-tidy |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-DCL54-a | Always provide new and delete together |
| Polyspace Bug Finder | R2020a | CERT C++: DCL54-CPP | Checks for mismatch between overloaded operator new and operator delete (rule fully covered) |
| PRQA QA-C++ | 4.4 | 2160 |  |
| RuleChecker | 20.10 | New-delete-pairwise | Partially checked |
| SonarQube C/C++ Plugin | 4.10 | S1265 |  |

### Coding Standard 2

| **Coding Standard** | **Label** | **Do not cast to an out-of-range enumeration value** |
| --- | --- | --- |
| **Data Value** | INT50-CPP | Enumerations in C++ come in two forms: scoped enumerations in which the underlying type is fixed and unscoped enumerations in which the underlying type may or may not be fixed. The range of values that can be represented by either form of enumeration may include enumerator values not specified by the enumeration itself. |

| **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 ot 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):** Keep It Simple |
| Keep It Simple maps to INT50-CPP because the principle basically says to keep it simple and the coding standard says that the arithmetic value being cast must be within the range of values the enumeration can represent. By keeping the range of values simple, it will reduce the risk of bugs or errors. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-INT50 |  |
| Parasoft C/C++test | 2020.2 | 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 | 7.07 | V1016 |  |

### Coding Standard 3

| **Coding Standard** | **Label** | **Use valid references, pointers, and iterators to reference elements of a basic\_string** |
| --- | --- | --- |
| **String Correctness** | STR52-CPP | References, pointers, and iterators referring to the elements of a basic\_string sequence may be invalidated by the following uses of that basic\_string object:   * As an argument to any standard library function taking a reference to non-const basic\_string as an argument * Calling non-const member functions, except operator[], at, front, back, being, rbegin, end, and rend.   Do not use an invalidated reference, pointer, or iterator because doing so results in undefined behavior. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example copies input into a std::string, replacing semicolon (;) characters with spaces. This example is noncompliant because the iterator loc is invalidated after the first call to insert(). The behavior of subsequent calls to insert() is undefined. |
| #include <string>  void f(const std::string &input) {  std::string email;  // Copy input into email converting “;” to “ “  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  email.insert(loc, \*i != ‘;’ ? \*i : ‘ ‘);  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the value of the iterator loc is updated as a result of each call to insert() so that the invalidated iterator is never accessed. The updated iterator is then incremented at the end of the loop. |
| #include <string>  void f(const std::string &input) {  std::string email;  // Copy input into email converting “;” to “ “  std::string::iterator loc = email.begin();  for (auto i = input.being(), e = input.end(); i != e; ++i, ++loc) {  loc = email.insert(loc, \*i != ‘;’ ? \*i : ‘ ‘);  }  } |

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

|  |
| --- |
| **Principles(s):** Adopt a Secure Coding Standard |
| The reason that the last of the 10 coding principles maps to this coding standard |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |

### Coding Standard 4

| **Coding Standard** | **Label** | **Guarantee that library functions do not overflow** |
| --- | --- | --- |
| **SQL Injection** | CTR52-CPP | Copying data into a container that is not large enough to hold that data results in a buffer overflow. To prevent such errors, data copied to the destination container must be restricted on the basis of the destination container’s size, or preferably, the destination container must be guaranteed to be large enough to hold the data to be copied. |

| **Noncompliant Code** |
| --- |
| STL containers can be subject to the same vulnerabilities as array data types. The std::copy() algorithm provides no inherent bounds checking and can lead to a buffer overflow. In this noncompliant code example, a vector of integers is copied from src to dest using std::copy(). Because std::copy() does nothing to expand the dest vector, the program will overflow the buffer on copying the first element. |
| #include <alogorithm>  #include <vector>  void f(const std::vector<int> &src) {  std::vector<int> dest;  std:copy(src.begin(), src.end(), dest.begin());  // …  } |

| **Compliant Code** |
| --- |
| The proper way to use std::copy() is to ensure the destination container can hold all of the elements being copied to it. This compliant solution enlarges the capacity of the vector prior to the copy operation. |
| #include <algorithm>  #include <vector>  void f(const std::vector<int> &src) {  // Initialize dest with src.size() default-inserted elements  std::vector<int> dest(src.size());  std::copy(src.begin(), src.end(), dest.begin());  // …  } |

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

|  |
| --- |
| **Principles(s):** Sanitize Data Sent to Other Systems |
| When copying the data into a container, that data should be sanitized and made sure that the size of the copied data is not larger than the container it is being transferred into otherwise it’ll cause a buffer overflow. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | invalid\_pointer\_deference |  |
| CodeSonar | 6.0p0 | BADFUNC.BO.\*  LANG.MEM.BO | A collection of warning classes that report uses of library functions prone to internal buffer overflows. |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-CTR52-a | Do not pass empty container iterators to std algorithms as destinations |

### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | MEM50-CPP | Evaluating a pointer – including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment – into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities. It is at the memory manager’s discretion when to reallocate or cycle the freed memory. When memory is freed, all pointers into it become invalid, and its contents might either be returned to the operating system, making the free space inaccessible, or remain intact and accessible. As a result, the data at the freed location can appear to be valid but change unexpectedly. Consequently, memory must not be written to or read form once it is freed. |

| **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):** Adopt a Secure Coding Standard |
| This coding standard basically tells us that memory that has been freed should not be accessed, so nothing should be written to or read from memory once it has been freed. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 6.9.0 | 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 | 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 | 2020.2 | 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 |  |
| PVS-Studio | 7.07 | V586, V774 |  |
| Splint | 5.0 |  |  |

### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the termination behavior of assert() and abort()** |
| --- | --- | --- |
| **Assertions** | ERR06-C | The assert macro puts diagnostic tests into programs; it expands to a void expression. When it is executed, if expression (which shall have a scalar type) is false (that is, compares equal to 0), the assert macro writes information about the particular call that failed (including the text of the argument, the name of the source file, the source line number, and the name of the enclosing function – the latter are respectively the values of the pre-processing macros \_\_FILE\_\_ and \_\_LINE\_\_ and of the identifier \_\_func\_\_) on the standard error stream in an implementation-defined format. It then calls the abort function. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example defines a function that is called before the program exits to clean up. |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }  int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle Error \*/  }  /\* … \*/  assert(/\* Something bad didn’t happen \*?);  /\* … \*/  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the call to assert() is replaced with an if statement that calls exit() to ensure that the proper termination routines are run. |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }  int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }  /\* … \*/  if (/\* something bad happened \*/) {  exit(EXIT\_FAILURE);  }  /\* … \*/  } |

**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** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced enforcement |
| Parasoft C/C++test | 2020.2 | CERT\_C-ERR06-a | Do not use assertions |
| PC-lint Plus | 1.4 | 586 | Fully supported |

### Coding Standard 7

| **Coding Standard** | **Label** | **Handle All Exceptions** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  try {  f();  } catch (…) {  // Handle errora  }  } |

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

|  |
| --- |
| **Principles(s):** Adhere to the Principle of Least Privilege |
|  |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | main-function-catch-all  early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-ERR51 |  |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| Polyspace Bug Finder | R2020a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| PRQA QA-C++ | 4.4 | 4035, 4036, 4037 |  |
| RuleChecker | 20.10 | main-function-catch-all  early-catch-all | Partially checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Do not attempt to create a std::string from a null pointer** |
| --- | --- | --- |
| **String Correctness** | STR51-CPP | The std::basic\_string type uses the traits design pattern to handle implementation details of the various string types, resulting in a series of string-like classes with a common, underlying implementation. Specifically, the std::basic\_string class is paired with std::char\_traits to create the std::string, std::wstring, std::u16string, and std::u32string classes. The std::char\_traits class explicitly specialized to provide policy-based implementation details to the std::basic\_string type. One such implementation detail is the std::char\_traits::length() function, which is frequently used to determine the number of characters in a null-terminated string. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, 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 <cstlib>  #include <string>  void f() {  std::string tmp(std::getenv(“TMP”));  if (!tmp.empty()) {  // …  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstlib>  #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):** Heed Compiler Warnings |
|  |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | assert\_failure |  |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

### Coding Standard 9

| **Coding Standard** | **Label** | **Detect and handle memory allocation errors** |
| --- | --- | --- |
| **Memory Protection** | MEM52-CPP | The default memory allocation operator, ::operator new(std::size\_t), throws a std::alloc exception if the allocation fails. Therefore, you need not check whether calling ::operator new(std::size\_t) results in nullptr. The nonthrowing form, ::operator new(std::size\_t, const std::nothrow\_t &), does not throw an exception if the allocation fails but instead returns nullptr. The same behaviors apply for the operator new[] version of both allocation functions .Additionally, the default allocator object (std::allocator) uses ::operator new(std::size\_t) to perform allocation and should be treated similarly. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of int is created using ::operator new[](std::size\_t) and ;the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program. |
| #include <cstring>  void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new int[size];  std::memcpy(copy, array, size \* sizeof(\*copy));  // …  delete [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>  void f(const int \*array, std::size\_t size) nexcept {  int \*copy = new (std::nothrow) int[size];  if (!copy) {  // Handle error  return;  }  std::memcpy(copy, array, size \* sizeof(\*copy));  // …  delete [] copy;  } |

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

|  |
| --- |
| **Principles(s):** Practice Defense in Depth |
|  |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  |  |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function’s parameters is undefined |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2020a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| PRQA QA-C++ | 4.4 | 3225, 3226, 3227, 3228, 3229, 4632 |  |
| PVS-Studio | 7.07 | V522 |  |

### Coding Standard 10

| **Coding Standard** | **Label** | **Do not define a C-style variadic function** |
| --- | --- | --- |
| **Data Type** | DCL50-CPP | Functions can be defined to accept more formal arguments at the call site than are specified by the parameter declaration clause. Such functions are called variadic functions because they can accept a variable number of arguments from a caller. C++ provides two mechanism by which a variadic function can be defined: function parameter packs and use of a C-style ellipsis as the final parameter declaration. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses a C-style variadic function to add a series of integers together. The function reads arguments until the value 0 is found. Calling this function without passing the value 0 as an argument (after the first two arguments) results in undefined behavior. Furthermore, passing any type other than an int also results in undefined behavior. |
| #include <cstdarg>  int add(int first, int second, …) {  int r = first + second;  va\_list va;  va\_start(va, second);  while (int v = va\_arg(va, int)) {  r += v;  }  va\_end(va);  return r;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. Unlike the C-style variadic function used in the noncompliant code example, this compliant solution does not result in undefined behavior if the list of parameters is not terminated with 0. Additionally, if any of the values passed to the function are not integers, the code is ill-formed rather than producing undefined behavior. |
| #include <type\_traits>  template <typename Arg, typename std::enable\_if<std:P:is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Arg s) { return f + s; }  template <typename Arg, typename… Ts, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Ts… rest) {  return f + add(rest…);  } |

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

|  |
| --- |
| **Principles(s):** Keep It Simple |
|  |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | function-ellipsis | Fully checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-DCL50 |  |
| Clang | 3.9 | cert-dcl50-cpp | Check by clang-tidy |
| CodeSonar | 6.0p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |
| LDRA tool suite | 9.7.1 | 41 S | Fully implemented |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-DCL50-a | Functions shall not be defined with a variable number of arguments |
| Polyspace Bug Finder | R2020a | CERT C++: DCL50-CPP | Checks for function definition with ellipsis notation (rule fully covered) |
| PRQA QA-C++ | 4.4 | 2012, 2625 |  |
| RuleChecker | 20.10 | function-ellipsis | Fully checked |
| SonarQube C/C++ Plugin;l | 4.10 | FunctionEllipsis |  |

## 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.

As with any development cycle, there will be a need for automation when it comes to detection of any bugs during the development. The automation will be done with using tools that detect certain threats and there is some tools that can be used to detect a wider range of threats such as Astree which can be used to find threats covered in all of my coding standards.

## 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 |
| --- | --- | --- | --- | --- | --- |
| CTR52-CPP | High | Likely | Medium | P18 | L1 |
| DCL50-CPP | High | Probable | Medium | P12 | L1 |
| DCL54-CPP | Low | Probable | Low | P6 | L2 |
| ERR06-C | Medium | Unlikely | Medium | P4 | L3 |
| ERR51-CPP | Low | Probable | Medium | P4 | L3 |
| INT50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| MEM50-CPP | High | Likely | Medium | P18 | L1 |
| MEM52-CPP | High | Likely | Medium | P18 | L1 |
| STR51-CPP | High | Likely | Medium | P18 | L1 |
| STR52-CPP | High | Probable | High | P6 | 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***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. 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.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption in rest is used for data that is at rest or data that is being stored. The way it works is that it takes the data and changes it to another form of data, and it is done with an algorithm. By changing the data into another form of data, it can’t be converted back by anyone unless they have access to the encryption key. |
| Encryption at flight | Encryption in flight is when data is being protected while it is in transit from one area to another. Basically, what happens is that the data is encrypted prior to transmission, and the two computer endpoints are authenticated and when the data arrives at its destination, it is decrypted and verified. |
| Encryption in use | Encryption in use is when the data you are currently using is being encrypted at the same time. It is not a very common process, and it is usually hardware or software based, with software being the newest way of encryption in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authenticating users is the first step in a secure identification system. The system needs to make sure the person accessing a system is who they say they are. The method of authenticating a person can fall into three main categories:   * **What They Know:** Someone’s password can authenticate what they know. Security questions also accomplish the same function. * **Who They Are:** A fingerprint or other bio metric tests can authenticate people. * **What They Have:** Access cards to enter a building can be used to authenticate a person. Mobile devices providing 2-factor authentication also use what a person has to verify identity.   Often the methods of authentication can be combined to ensure people are accurately authenticated. |
| Authorization | Determining the type of authorization employees have within a network is the next step. The right people should the right access level to areas of a network. There are different frameworks to handle this within a company.   * **Mandatory Access Control (MAC):** The level of security a person is granted is related to the security of the content being accessed. This is common in military use-cases. * **Discretionary Access Control (DAC):** Access to a file or area is given by the owner of that area. An example of this framework is a Google doc where access can shared by the creator with whoever they desire. * **Role-Based Access Control (RBAC):** Access is determined by the role within an organization. For example, the shipping department might have access to inventory but not marketing collateral.   In general, giving users the least number of privileges needed to accomplish their job is the goal. Limiting access to sensitive areas makes there more secure. |
| Accounting | After a person begins logging into a network and working, their usage should be monitored. This can be accomplished with a Security Information and Event Management (SIEM) or other auditing and monitoring tool. Knowing what files a person is accessing or attempting to access can inform whether more or less authorization is needed. Suspicious activity can prompt questions as to whether the person accessing the network was authenticated correctly. |

**\***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 |