**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 the data being input into the program. Always verify external data being brought into the program to make sure the external data is not corrupt or damaged to prevent damage to the program or the operating system. |
| 1. Heed Compiler Warnings | Follow the instructions of warning messages when compiling the code to prevent any open doors which can allow hackers to enter a program. Modify the program’s code to satisfy the warnings found during a debug of the program. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. |
| 1. Keep It Simple | Keep the program code design short and simple to prevent any errors or warning, which are usually seen within complex design of coding for a program. |
| 1. Default Deny | Instead of excluding data, make data only permissible to those who are allowed access to the data and deny access permission to everyone until they can prove they have the security keys to gain access to the data. |
| 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. |
| 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. |
| 1. Practice Defense in Depth | Use multi-layer defenses when managing the safety of the computer. The multi-layer defense allows the system and/or device to stay protected even if one of the layer defenses gets compromised. |
| 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. |
| 1. Adopt a Secure Coding Standard | Use a secured code standard whether you apply or develop the standard for your choice of 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** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | [Do not cast to an out-of-range enumeration value](https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value) |

| **Noncompliant Code** |
| --- |
| 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 complement system, the valid range of values that can be represented by Enum Type are [0..3], so if a value outside of that range were passed to f(), the cast to Enum Type would result in an unspecified value, and using that value within the if statement results in [unspecified behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-unspecifiedbehavior). |
| **enum** EnumType {  First,  Second,  Third  };    **void** f(**int** intVar) {  EnumType enumVar = **static\_cast**<EnumType>(intVar);    **if** (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| 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. |
| **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):** Heed Compiler Warnings, Keep it Simple  Most compilers will generate a warning regarding integer conversion and properly addressing these warnings will create more resilient code. This also helps avoid unnecessary data type conversions and can simplify the development process |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| **Medium** | **Likely** | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.6 | Type Checks - signed integer overflow (only enabled when --platform is used)    Type Checks - dangerous sign conversion, when signed value can be negative | Open source utility which checks for several types of errors, including proper usage of the STL |
| Klocwork | 2021.4 | CXX.CAST.SIGNED\_CHAR\_TO\_INTEGER | Partial coverage pertaining to signed char to integer casts |
| Coverity | 15.0.0 | MISRA\_CAST (CWE-195) |  |
| Clangtidy | 2021.12.0 | bugprone-signed-char-misuse  bugprone-misplaced-widening-cast  abseil-duration-conversion-cast |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | [Use valid references, pointers, and iterators to reference elements of a container](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR51-CPP.+Use+valid+references,+pointers,+and+iterators+to+reference+elements+of+a+container) |

| **Noncompliant Code** |
| --- |
| pos is invalidated after the first call to insert(), and subsequent loop iterations have [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) |
| #include <deque>    **void** f(**const** **double** \*items, std::**size\_t** count) {  std::deque<**double**> d;  auto pos = d.begin();  **for** (std::**size\_t** i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| pos is assigned a valid iterator on each insertion, preventing undefined behavior |
| #include <deque>    **void** f(**const** **double** \*items, std::**size\_t** count) {  std::deque<**double**> d;  auto pos = d.begin();  **for** (std::**size\_t** i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

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

| **Principles(s):** Heed Compiler Warnings  Any warning presented by the compiler regarding the memory buffer should be immediately addressed. Developers who are mindful of the memory limitations of data types deliver the most robust and successful code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klockwork | 2021.4 | ABV.ANY\_SIZE\_ARRAY  ABV.GENERAL  ABV.ITERATOR  ABV.MEMBER  ABV.STACK  ABV.TAINTED  ABV.UNICODE.BOUND\_MAP  ABV.UNICODE.FAILED\_MAP  ABV.UNICODE.NNTS\_MAP  ABV.UNICODE.SELF\_MAP  ABV.UNKNOWN\_SIZE  NNTS.MIGHT  NNTS.MUST  NNTS.TAINTED  RABV.CHECK  RN.INDEX  SV.FMT\_STR.BAD\_SCAN\_FORMAT  SV.STRBO.BOUND\_COPY.OVERFLOW  SV.STRBO.BOUND\_COPY.UNTERM  SV.STRBO.BOUND\_SPRINTF  SV.STRBO.UNBOUND\_COPY  SV.STRBO.UNBOUND\_SPRINTF  SV.UNBOUND\_STRING\_INPUT.CIN  SV.UNBOUND\_STRING\_INPUT.FUNC | Complete coverage including file stream buffer overflows |
| SonarQube | 9.3 | Memory access should be explicitly bounded to prevent buffer overflows (CWE-131, CWE-119, STR50-CPP) |  |
| CPPCheck | 2.6 | Bounds Checking - Partial string write that leads to buffer that is not zero terminated. |  |
| Coverity | 2021.12.0 | BUFFER\_SIZE |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | [Guarantee that storage for strings has sufficient space for character data and the null terminator](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) |

| **Noncompliant Code** |
| --- |
| the input is unbounded; the following code could lead to a buffer overflow. |
| #include <iostream>    **void** f() {  **char** buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array |
| #include <iostream>  #include <string>    **void** f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard, ValidateInput Data  While some compilers can handle this issue, one can be certain of a code’s integrity by avoiding this entirely. As such, Green Pace can more successfully maintain a secure coding standard by ensuring developers are following this practice. This will also help ensure input data is validated through additional checks for null values. |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Medium | Unlikely | Medium | Low | 2 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Cppcheck | 2.6 | Null pointers – null pointer dereferencing |  |
| SonarQube | 9.3 | Null pointers should not be dereferenced |  |
| Coverity | 2021.12.0 | STRING\_NULL |  |
| Klocwork | 2021.4 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| clang-tidy | 15.0.0 | unix.cstrisng.NullArg (C) | Specific check for coding standard |

#### 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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM56-CPP.+Do+not+store+an+already-owned+pointer+value+in+an+unrelated+smart+pointer) |

| **Noncompliant Code** |
| --- |
| two unrelated smart pointers are constructed from the same underlying pointer value. When the local, automatic variable p2 is destroyed, it deletes the pointer value it manages. Then, when the local, automatic variable p1 is destroyed, it deletes the same pointer value, resulting in a double-free [vulnerability](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-vulnerability). |
| #include <memory>    **void** f() {  **int** \*i = **new** **int**;  std::shared\_ptr<**int**> p1(i);  std::shared\_ptr<**int**> p2(i);  } |

| **Compliant Code** |
| --- |
| the std::shared\_ptr objects are related to one another through copy construction. When the local, automatic variable p2 is destroyed, the use count for the shared pointer value is decremented but still nonzero. Then, when the local, automatic variable p1 is destroyed, the use count for the shared pointer value is decremented to zero, and the managed pointer is destroyed. This compliant solution also calls std::make\_shared() instead of allocating a raw pointer and storing its value in a local variable. |
| #include <memory>    **void** f() {  std::shared\_ptr<**int**> p1 = std::make\_shared<**int**>();  std::shared\_ptr<**int**> p2(p1);  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems, Architect and Design for Security Policies, Adhere to the Principle of Least Privilege  Preventing SQL injection attacks requires that data sent and received by our system is properly sanitized and filtered for unexpected values. Developers must plan for this vector of attack when planning the structure of the system. By limiting the channels of which one can access data, Green Pace can better employ the Principle of Least Privilege. |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Very High | Likely | High | Very High | 5 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Coverity | 2021.12.0 | TAINTED\_SCALAR  TAINTED\_STRING  USER\_POINTER | Partial SQL injection coverage pertaining to string values |
| Klocwork | 2021.4 | SV.STR\_PAR.UNDESIRED\_STRING\_PARAMETER  SV.TAINTED.ALLOC\_SIZE  SV.TAINTED.BINOP  SV.TAINTED.CALL.BINOP  SV.TAINTED.CALL.DEREF  SV.TAINTED.CALL.GLOBAL  SV.TAINTED.CALL.INDEX\_ACCESS  SV.TAINTED.CALL.LOOP\_BOUND  SV.TAINTED.DEREF  SV.TAINTED.FMTSTR  SV.TAINTED.GLOBAL  SV.TAINTED.INDEX\_ACCESS  SV.TAINTED.INJECTION  SV.TAINTED.LOOP\_BOUND  SV.TAINTED.PATH\_TRAVERSAL  SV.TAINTED.SECURITY\_DECISION  SV.TAINTED.XSS.REFLECTED | SV.TAINTED checks for all un-sanitized input data. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | [Properly deallocate dynamically allocated resources](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources) |

| **Noncompliant Code** |
| --- |
| 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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) 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** |
| --- |
| 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):** Architect and Design for Security Policies, Heed Compiler Warnings  Data structures must be designed with memory limitations under consideration and any issues with storage capacity can lead to compiler warnings. |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Medium | Likely | Medium | High | 4 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Cppcheck | 2.6 | Bounds Checking – Buffer Overflow |  |
| SonarQube | 9.3 | Memory access should be explicitly bounded to prevent buffer overflows |  |
| Coverity | 2021.12.0 | BUFFER\_SIZE  READLINK  SIZECHECK | Targeted check for destination size |
| Klocwork | 2021.4 | ABV.ANY\_SIZE\_ARRAY  ABV.GENERAL  ABV.ITERATOR  ABV.MEMBER  ABV.STACK  ABV.TAINTED  ABV.UNICODE.BOUND\_MAP  ABV.UNICODE.FAILED\_MAP  ABV.UNICODE.NNTS\_MAP  ABV.UNICODE.SELF\_MAP  ABV.UNKNOWN\_SIZE  NNTS.MIGHT  NNTS.MUST  NNTS.TAINTED  RABV.CHECK  RN.INDEX  SV.FMT\_STR.BAD\_SCAN\_FORMAT  SV.STRBO.BOUND\_COPY.OVERFLOW  SV.STRBO.BOUND\_COPY.UNTERM  SV.STRBO.BOUND\_SPRINTF  SV.STRBO.UNBOUND\_COPY  SV.STRBO.UNBOUND\_SPRINTF  SV.UNBOUND\_STRING\_INPUT.CIN  SV.UNBOUND\_STRING\_INPUT.FUNC | Complete coverage including file stream buffer overflows |

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

| **Noncompliant Code** |
| --- |
| noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. The use of the runtime assertion is better than nothing, it needs to be placed in a function and executed. |
| #include <assert.h>    **struct** timer {  unsigned **char** MODE;  unsigned **int** DATA;  unsigned **int** COUNT;  };    **int** func(**void**) {  **assert**(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**));  } |

| **Compliant Code** |
| --- |
| Static assertions allow incorrect assumptions to be diagnosed at compile time instead of resulting in a silent malfunction or runtime error. |
| #include <assert.h>    **struct** timer {  unsigned **char** MODE;  unsigned **int** DATA;  unsigned **int** COUNT;  };    static\_assert(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**),  "Structure must not have any padding"); |

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

| **Principles(s):** Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard, Architect and Design for Security Policies  Assertions can provide a powerful tool for developers to test the effectiveness and behavior of their code. But it is important that the debug and release versions are separate and accounted for in the initial design of the system. |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Low | Unlikely | Medium | Low | 1 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Cppcheck | 2.6 | Warn if there are side effects in assert statements (since this cause different behavior in debug/release builds). |  |
| clang-tidy | 15.0.0 | bugprone-assert-side-effect  misc-static-assert | Comprehensive checks for assert in release builds |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | [Guarantee exception safety](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR56-CPP.+Guarantee+exception+safety) |

| **Noncompliant Code** |
| --- |
| The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. The function deallocates array and assigns the element counter, nElems, before allocating a new block of memory for the copy. |
| #include <cstring>    **class** IntArray {  **int** \*array;  std::**size\_t** nElems;  **public**:  // ...    ~IntArray() {  **delete**[] array;  }      IntArray(**const** IntArray& that); // nontrivial copy constructor  IntArray& operator=(**const** IntArray &rhs) {  **if** (**this** != &rhs) {  **delete**[] array;  array = nullptr;  nElems = rhs.nElems;  **if** (nElems) {  array = **new** **int**[nElems];  std::**memcpy**(array, rhs.array, nElems \* **sizeof**(\*array));  }  }  **return** \***this**;  }    // ...  }; |

| **Compliant Code** |
| --- |
| the copy assignment operator provides the [strong exception safety](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-strongexceptionsafety) guarantee. The function allocates new storage for the copy before changing the state of the object. |
| #include <cstring>    **class** IntArray {  **int** \*array;  std::**size\_t** nElems;  **public**:  // ...    ~IntArray() {  **delete**[] array;  }    IntArray(**const** IntArray& that); // nontrivial copy constructor    IntArray& operator=(**const** IntArray &rhs) {  **int** \*tmp = nullptr;  **if** (rhs.nElems) {  tmp = **new** **int**[rhs.nElems];  std::**memcpy**(tmp, rhs.array, rhs.nElems \* **sizeof**(\*array));  }  **delete**[] array;  array = tmp;  nElems = rhs.nElems;  **return** \***this**;  }    // ...  }; |

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

| **Principles(s):** Practice Defense in Depth, Adopt a Secure Coding Standard, ValidateInput Data  This standard prevents any unexpected outcomes or values from running within the main() portion of the code. As such, this gives depth to the code by ensuring no statement blindly outputs data and it is instead validated prior to finalization. |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| High | Likely | Low | High | 3 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Coverity | 2021.12.0 | UNCAUGHT\_EXCEPT |  |
| SonarQube | 9.3 | "std::uncaught\_exception" should not be used |  |
| clang-tidy | 15.0.0 | bugprone-unhandled-exception-at-new  bugprone-exception-escape  hicpp-exception-baseclass  modernize-use-uncaught-exceptions | Coverage of unhandled exceptions and proper throw type |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | [Range check element access](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access) |

| **Noncompliant Code** |
| --- |
| the value returned by the call to get\_index() may be greater than the number of elements stored in the string, resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {  std::string s("01234567");  s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| solution uses the std::basic\_string::at() function, which behaves in a similar fashion to the index operator[] but throws a std::out\_of\_range exception if pos >= size() |
| #include <stdexcept>  #include <string>  **extern** std::**size\_t** get\_index();    **void** f() {  std::string s("01234567");  **try** {  s.at(get\_index()) = '1';  } **catch** (std::out\_of\_range &) {  // Handle error  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard, Keep It Simple, Heed Compiler Warnings  Developers following this standard avoid unnecessarily complicated code and use std string to get possible out of range exceptions |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Low | Unlikely | Medium | Low | 1 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Coverity | 2021.12.0 | UNCAUGHT\_EXCEPT |  |
| SonarQube | 9.3 | "std::uncaught\_exception" should not be used |  |
| clang-tidy | 15.0.0 | bugprone-unhandled-exception-at-new  bugprone-exception-escape  hicpp-exception-baseclass  modernize-use-uncaught-exceptions | Coverage of unhandled exceptions and proper throw type |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | [Do not attempt to create a std::string from a null pointer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std::string+from+a+null+pointer) |

| **Noncompliant Code** |
| --- |
| a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) when the environment variable does not exist (or some other error occurs) |
| #include <cstdlib>  #include <string>    **void** f() {  std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** Heed Compiler Warnings  Any warning presented by the compiler regarding the getenv() should be immediately addressed. Developers who are mindful of the pointer limitations deliver the most robust and successful code. |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| High | Likely | Low | High | 4 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Klockwork | 2021.4 | ABV.ANY\_SIZE\_ARRAY  ABV.GENERAL  ABV.ITERATOR  ABV.MEMBER  ABV.STACK  ABV.TAINTED  ABV.UNICODE.BOUND\_MAP  ABV.UNICODE.FAILED\_MAP  ABV.UNICODE.NNTS\_MAP  ABV.UNICODE.SELF\_MAP  ABV.UNKNOWN\_SIZE  NNTS.MIGHT  NNTS.MUST  NNTS.TAINTED  RABV.CHECK  RN.INDEX  SV.FMT\_STR.BAD\_SCAN\_FORMAT  SV.STRBO.BOUND\_COPY.OVERFLOW  SV.STRBO.BOUND\_COPY.UNTERM  SV.STRBO.BOUND\_SPRINTF  SV.STRBO.UNBOUND\_COPY  SV.STRBO.UNBOUND\_SPRINTF  SV.UNBOUND\_STRING\_INPUT.CIN  SV.UNBOUND\_STRING\_INPUT.FUNC | Complete coverage including file stream buffer overflows |
| SonarQube | 9.3 | Memory access should be explicitly bounded to prevent buffer overflows (CWE-131, CWE-119, STR50-CPP) |  |
| CPPCheck | 2.6 | Bounds Checking - Partial string write that leads to buffer that is not zero terminated. |  |
| Coverity | 2021.12.0 | BUFFER\_SIZE |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | [Do not define a C-style variadic function](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL50-CPP.+Do+not+define+a+C-style+variadic+function) |

| **Noncompliant Code** |
| --- |
| 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. |
| #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** |
| --- |
| a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. |
| #include <type\_traits>    **template** <**typename** Arg, **typename** std::enable\_if<std::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):** Heed Compiler Warnings  Any warning presented by the compiler regarding this function type not passing through after functions could result in unexpected undefined behavior. |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| High | Likely | High | High | 4 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| SonarQube | 9.3 | Appropriate arguments should be passed to stream functions | Verifies correct values are passed to functions |
| Cppcheck | 2.6 | IO Using Format String - File input/output without positioning results in undefined behavior | Partial coverage of file IO preventing undefined behavior |

### 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.

The diagram above represents the standard DevSecOps security pipeline. It gives us a path to follow during the software development lifecycle. However, the automation tools discussed in this document will prove most useful and relevant during the span between the Build stage and the Respond stage. The developers at Green Pace should actively build data structures and code with integration of static testing tools. These tools will then be used extensively to verify and validate the code before deployment. Once the code has been deployed, some of the automation tools can be transitioned into monitoring tools that your team will use to detect any issues which may arise. They will also be beneficial in the future should other security risks arise.

### 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 | Likely | Low | Medium | 3 |
| **STD-002-CPP** | High | Likely | Low | High | 4 |
| **STD-003-CPP** | Medium | Unlikely | Medium | Low | 2 |
| **STD-004-CPP** | Very High | Likely | High | Very High | 5 |
| **STD-005-CPP** | Medium | Likely | Medium | High | 4 |
| **STD-006-CPP** | Low | Unlikely | Medium | Low | 1 |
| **STD-007-CPP** | High | Likely | Low | High | 3 |
| **STD-008-CPP** | Low | Unlikely | Medium | Low | 1 |
| **STD-009-CPP** | High | Likely | Low | High | 4 |
| **STD-010-CPP** | High | Likely | High | High | 4 |

### 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 | Data being stored will be considered at rest. Sensitive data logins, or financials should always be encrypted when at rest. Data which is stored as plaintext is at an extreme risk of becoming compromised leading to legal, financial, and moral liabilities as well as a loss of trust. AES encryption would be my recommendation for the task of encrypting this data. |
| Encryption at flight | Data in motion being sent from one system to another can also be extremely vulnerable and if intercepted by malicious parties could cause a lot of problems so a secure defense must be in place to prevent this from happening. One must not only consider data in transit within their system, but also data which is leaving/arriving to or from their system. Secure transmission channels must be established to protect the integrity of the data and several encryption protocols exist for this purpose. I would recommend a protocol like TLS to encrypt our data in motion or flight. |
| Encryption in use | This may be the hardest of the three to follow because it refers to data being created, edited or viewed; it is still a risk due to things like human error an unlocked computer could be the largest security breach a company has ever had to prevent this using real time encryption of data could help thwart would be villains from using this attack vector along with a strong user training plan to help prevent avoidable exposures. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is our first A and is the process of ensuring that only those who are authorized specific data are allowed to access this data. It is commonly implemented with usernames and passwords to verify the identity of the user but in modern time we have added things like MFA and tokens This added layer of security allows us to be safer this paired with an active and robust monitoring program should be enacted, where all logons are tracked, and suspicious attempts are reported. Will help secure this network. |
| Authorization | While similar to authentication, the next step of authorization refers to granting users access to different categories of data. One must ensure that only those who are authorized to view and modify data have those permissions. It is critical that the principle of Least Privilege is followed here which translates to giving users access to only the bare minimum data that is necessary for their job. An efficient system will categorize data based on levels of sensitivity and an Administrator will personally review, authorize, and assign users the correct level of access. |
| Accounting | The last A in our triple A helps reinforce the effectiveness of our previous two. For accounting we monitor our system activities and verify that policies are followed to the letter of the law. This responsibility will fall on the systems administrators and should be generating logs and reviewing these logs. They will use these logs to monitor all changes on the systems including user information databases and system policies. |

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

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

### Map the Principles

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

**NOTE:** Green Pace has already successfully implemented the following:

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 04/10/2022 | Completed form | Alex Grimes | TBD |
| [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 |