**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 | Data from untrusted data sources can lead to improper input. Through proper input validation, we can circumvent a majority of software vulnerability exploits. |
| 1. Heed Compiler Warnings | Compiler warnings give a great indication of issues that are found in your code. Going through and eliminating these compiler warnings means that your code is more secure than if the warning were there. |
| 1. Architect and Design for Security Policies | Design and build your software with security in mind. Make sure that the proper variables are used for the incoming/outgoing data. One can also create subsystems that communicate with each other depending on the level of privileges needed. |
| 1. Keep It Simple | The more complex and convoluted the design of the software, the higher the chance for vulnerabilities to be present. Keeping things simple makes it easier to implement security changes or perform security maintenance on the system. |
| 1. Default Deny | Design the software to automatically deny access to information unless certain parameters are met. |
| 1. Adhere to the Principle of Least Privilege | Only permit needed information to each level of permission. If someone only needs access to a certain subset of data, they only get access to that subset of data. |
| 1. Sanitize Data Sent to Other Systems | If a certain functionality is unused in a system, remove it. Attackers can evoke unused functionality as a means to gather extra information. Removing these functions and sanitizing the information transmitted means that only the essentials are used or are present. |
| 1. Practice Defense in Depth | Having multiple layers of security with overlaps in functionality means that it will be much harder for hackers to gain information. This also allows the system to remain safe if one of the layers of defense are breached. |
| 1. Use Effective Quality Assurance Techniques | Utilizing security testing and other QA techniques helps to identify vulnerabilities and issues before changes/programs go live. This helps to prevent data leaks and successful attacks in the future through proactive steps now. |
| 1. Adopt a Secure Coding Standard | Develop/apply a secure coding standard that will be used. This allows for a common format that is proven to be effective., that also allows for security changes/upgrades to be implemented across the board in a quicker/simpler fashion. |

### 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-CLG] | Implement abstract data types using opaque types. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example is based on the managed string library developed by CERT [[Burch 2006](https://wiki.sei.cmu.edu/confluence/display/c/AA.+Bibliography#AA.Bibliography-Burch06)]. In this example, the managed string type and the functions that operate on this type are defined in the string\_m.h header file as follows: |
| **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;  unsigned **char** strtype;  **char** \*cstr;  };    **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| This compliant solution reimplements the string\_mx type as a private type, hiding the implementation of the data type from the user of the managed string library. To accomplish this, the developer of the private data type creates two header files: an external string\_m.h header file that is included by the user of the data type and an internal file that is included only in files that implement the managed string abstract data type.  In the external string\_m.h file, the string\_mx type is defined to be an instance of struct string\_mx, which in turn is declared as an [incomplete type](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-incompletetype): |
| **struct** string\_mx;  **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/ |

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

| **Principles(s):** *Architect and Design for Security Policies*.  The use of opaque abstract data types, though not essential to secure programming, can significantly reduce the number of defects and [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability) introduced in code, particularly during ongoing maintenance. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL12 |  |
| LDRA tool suite | 9.7.1 | 104 D | Partially Implemented |
| Polyspace Bug Finder | R2021a | Cert C: Rec. DCL-C | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced (rule partially covered) |
| Parasoft C/C++ test | 2021.2 | CERT\_C-DCL-a | If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-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](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** |
| --- |
| 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** **class** EnumType {  First,  Second,  Third  };    **void** f(**int** intVar) {  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/Architect and Design for Security Policies.*  It is possible for unspecified values to result in a buffer overflow, leading to the execution of arbitrary code by an attacker. However, because enumerators are rarely used for indexing into arrays or other forms of pointer arithmetic, it is more likely that this scenario will result in data integrity violations rather than arbitrary code execution. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ -INT50 |  |
| CodeSonar | 6.2p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion alters value  Cast alters value |
| Helix QAC | 2022.1 | C++ 3013 |  |
| Parasoft  C/C++ test | 2021.2 | 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 | 7.18 | V1016 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CLG] | 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):** *Architect and Design for Security Policies/Heed Compiler Warnings.*  Modern compilers recognize the difference between a char \* and a wchar\_t \*, so compiling code that violates this rule will generate warnings. It is feasible to have automated software that recognizes functions of improper width and replaces them with functions of proper width (that is, software that uses wcsncpy() when it recognizes that the parameters are of type wchar\_t \*). |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | P27 | L1 |

**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 | 2022.1 | C0432  C++ 0403 |  |
| Klocwork | 2022.1 | MISRA.ETYPE.ASSIGN.2012 |  |
| PARASOFT  C/C++ test | 2021.2 | 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. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CLG] | 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](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior): |
| |  | | --- | | **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. (See [STR11-C. Do not specify the bound of a character array initialized with a string literal](https://wiki.sei.cmu.edu/confluence/display/c/STR11-C.+Do+not+specify+the+bound+of+a+character+array+initialized+with+a+string+literal).) 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 | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | String-literal-modifiaction  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\*” |
| Helix QAC | 2022.1 | C0556, C0752, C0753, C0754  C++ 3063, C++3064, C++3605, C++3606, C++3607 |  |
| Klockwork | 2022.1 | CERT.STR.ARG.CONST-TO-NONCONST  CERT.STR.ASSIGN.CONST-TO-NOCONST |  |
| LDRA tool suite | 9.7.1 | 157S | Partially implemented |
| Parasoft  C/C++ test | 2021.2 | 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 | 7.18 | V675 |  |
| RuleChecker | 20.10 | String-literal-modification | Partially checked |
| Splint | 3.1.1 |  |  |
| TrustInSoft Analyzer | 1.38 | Mem-access | Exhaustively verified (see one compliant and one non-compliant example). |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CLG] | Do not subtract or compare two pointers that do not refer to the same array |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pointer subtraction is used to determine how many free elements are left in the nums array: |
| #include <stddef.h>    **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** end;  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;    /\* Increment next\_num\_ptr as array fills \*/    free\_elements = &end - next\_num\_ptr;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the number of free elements is computed by subtracting next\_num\_ptr from the address of the pointer past the nums array. While this pointer may not be dereferenced, it may be used in pointer arithmetic. |
| #include <stddef.h>  **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;    /\* Increment next\_num\_ptr as array fills \*/    free\_elements = &(nums[SIZE]) - next\_num\_ptr;  } |

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

| **Principles(s):** *Architect and Design for Security Policies/Adopt a Secure Coding Standard.*  By designing the software to only point at pointers referring to the same array, we prevent potential data leakage. We are also setting this up as a standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | String-literal-modifiaction  Write-to-string-literal | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| CodeSonar | 6.2p0 | LANG.STRUCT.CUP  LANG.STRUCT.SUP | Comparison of unrelated pointers  Subtraction of unrelated pointers |
| Coverity | 2017.17 | MISRA C 2004 17.2  MISRA C 2004 17.3  MISRA C 2012 18.2  MISRA C 2012 18.3 | IMPLEMENTED |
| Helix QAC | 2022.1 | C0487, C0513, C2668, C2669, C2761, C2762, C2763, C2766, C2767, C2768, C2771, C2772, C2773 |  |
| Klockwork | 2022.1 | MISRA.PTR.ARITH |  |
| LDRA tool suite | 9.7.1 | 437 S, 438 s | Fully implemented |
| Parasoft  C/C++ test | 2021.2 | CERT\_C-ARR36-a  CERT\_C-ARR36-b | Do not subtract two pointers that do not address elements of the same array.  Do not compare two unrelated pointers. |
| Polyspace Bug Finder | R2021a | CERT C: Rule ARR36-C | Checks for subtraction or comparison between pointes to different arrays (rule partially covered) |
| PRQA QA-C | 9.7 | 0487, 0513, 2668, 2669, 2761, 2762, 2763, 2766, 2767, 2768, 2771, 2772, 2773 | Fully implemented |
| PVS-Studio | 7.18 | V736, V782 |  |
| RuleChecker | 20.10 | Pointer-subtraction | Partially checked |
| TrustInSoft  Analyzer | 1.38 | Differing\_blocks | Exhaustively verified (see the compliant and the non-compliant example). |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Understand the termination behavior of assert() and abort() |

| **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):** *Architect and Design for Security Policies/Keep it Simple.*  Understanding the usage of these functions means that we are able to develop the code properly without having “surprise features”. By keeping the design simple, and limiting the usage of these, we are able to better protect ourselves from vulnerabilities. |
| --- |

**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 the 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 | 2021.2 | CERT\_C-ERR06-a | Do not use assertions |
| PC-lint Plus | 1.4 | 586 | Fully supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Guarantee exception safety |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. 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. As a result, if the new expression throws an exception, the function will have modified the state of both member variables in a way that violates the implicit invariants of the class. Consequently, such an object is in an indeterminate state and any operation on it, including its destruction, results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #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** |
| --- |
| In this compliant solution, 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. Only after the allocation succeeds does the function proceed to change the state of the object. In addition, by copying the array to the newly allocated storage before deallocating the existing array, the function avoids the test for self-assignment, which improves the performance of the code in the common case [[Sutter 2004](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-Sutter04)]. |
| #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):** *Architect and Design for Security Polices/Default Deny.*  By ensuring we maintain exceptions properly, we will be better securing the code from hackers. In addition to this, if we have unforeseen errors beyond what the exceptions can handle, this gives a chance to revisit the code, and make modifications to account for the new findings. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2022.1 | C++ 4075, C++ 4076 |  |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| Parasoft  C/C++ test | 2021.2 | CERT\_CPP-ERR56-a  CERT\_CPP-ERR56-b | Always catch exceptions  Do not leave ‘catch’ blocks empty |
| PRQA QA-C++ | 4.4 | 4075, 4076 |  |
| PVS-Studio | 7.18 | V565, V1023, V5002 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output (FIO) | [STD-008-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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #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/Sanitize Data Sent to Other Systems/Utilize Quality Assurance Tech.*  By developing the code in a way that scrubs the input/output streams, we prevent data leaks to other systems that could be picked up and cause errors/exceptions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ -FIO50 |  |
| CodeSonar | 6.2p0 | IO.IOWOP  IO.OIWOP | Input after output without positioning  Output after input without positioning |
| Helix QAC | 2022.1 | C++4711  C++4712  C++4713 |  |
| Parasoft  C/C++test | 2021.2 | CERT\_CPP-VIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2021b | CERT C++: VIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | [STD-009-CPP] | Explicitly construct and destruct objects when manually managing object lifetime. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a class with nontrivial initialization (due to the presence of a user-provided constructor) is created with a call to std::malloc(). However, the constructor for the object is never called, resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) when the class is later accessed by calling s->f(). |
| #include <cstdlib>    **struct** S {  S();    **void** f();  };    **void** g() {  S \*s = **static\_cast**<S \*>(std::**malloc**(**sizeof**(S)));    s->f();    std::**free**(s);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the constructor and destructor are both explicitly called. Further, to reduce the possibility of the object being used outside of its lifetime, the underlying storage is a separate variable from the live object. |
| #include <cstdlib>  #include <new>    **struct** S {  S();    **void** f();  };    **void** g() {  **void** \*ptr = std::**malloc**(**sizeof**(S));  S \*s = **new** (ptr) S;    s->f();    s->~S();  std::**free**(ptr);  } |

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

| **Principles(s):** *Architect and Design for Security Policies/Use Quality Assurance Techniques.*  By explicitly constructing/destructing objects, we are ensuring that the data held within those objects will not be accessed through unauthorized means either maliciously or inadvertently. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2022.1 | C++4761, C++4762, C++4766, C++4767 |  |
| Parasoft  C/C++ test | 2021.2 | CERT\_CPP—MEM53-a | Do not invoke malloc/realloc for objects having constructors |
| PVS-Studio | 7.18 | V630, V749 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions and Error Handling | [STD-010-CPP] | Do not leak resources when handling exceptions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pst is not properly released when process\_item throws an exception, causing a resource leak. |
| #include <new>    **struct** SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  **void** process\_item() noexcept(**false**);  };    **void** f() {  SomeType \*pst = **new** (std::**nothrow**) SomeType();  **if** (!pst) {  // Handle error  **return**;  }    **try** {  pst->process\_item();  } **catch** (...) {  // Process error, but do not recover from it; rethrow.  **throw**;  }  **delete** pst;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the exception handler frees pst by calling delete. |
| #include <new>    **struct** SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.    **void** process\_item() noexcept(**false**);  };    **void** f() {  SomeType \*pst = **new** (std::**nothrow**) SomeType();  **if** (!pst) {  // Handle error  **return**;  }  **try** {  pst->process\_item();  } **catch** (...) {  // Process error, but do not recover from it; rethrow.  **delete** pst;  **throw**;  }  **delete** pst;  } |

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

| **Principles(s):** *Architect and Design for Security Policies/Adhere to Principle of Least Prvilege/Sanitize Data Sent.*  Building the software to prevent leaking of data/resources when handling exceptions allows us to ensure our code is secure. If there are artifacts left over from other functions/objects, and an exception occurs, the data could be leaked due to the mishandled data. This would be especially bad if someone of a higher privelage had generated the objects/artifacts which would mean more sensitive data would be at risk. Therefore, if we sanitize the data and destroy objects when we are capable, this will help mitigate the issue. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2022.1 | C++4756, C++ 4757, C++ 4758 |  |
| Klockwork | 2022.1 | CL.MLK  MLK.MIGHT  MLK.MUST  MLK.RET.MIGHT  MLK.RET.MUST  RH.LEAK |  |
| Ldra TOOL SUITE | 9.7.1 | 50 D | Partially implemented |
| Parasoft  C/C++ test | 2021.2 | CERT\_CPP-ERR57-a | Ensure resources are freed |
| Polyspace Bug Finder | R2021b | CERT C++: ERR57-CPP | Checks for:  Resource leak cause by exception.  Bad allocation in constructor.  Object left in partially initialized state. |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

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

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

**[Insert your written explanations here.]**

Our automation process will start from auditing the authorization levels for our employees and determining if there are adjustments that need to be made. Furthermore, we will set up/modify our authentication procedures by not only utilizing login credentials, but also utilizing a 2FA system that can be set up on a specific USB like key or through an employee’s work phone.

Another way we will automate our DevSecOps plan will be to incorporate unit testing throughout the software development process and utilizing the mitigation tools that were talked about in the different code standard sections. This will help to create sound and secure code that we will be able to deliver to our customers.

After deployment, code will be monitored systematically and all error/exceptions will be logged and reviewed. We will also have an access/change log that will be time stamped with the corresponding employee’s information for transparity and accountability reasons.

### 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-CLG | Low | Unlikely | High | P1 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CLG | High | Likely | Low | P27 | L1 |
| STD-004-CLG | Low | Likely | Low | P9 | L2 |
| STD-005-CLG | Medium | Probable | Medium | P8 | L2 |
| STD-006-CLG | Medium | Unlikely | Medium | P4 | L3 |
| STD-007-CPP | High | Likely | High | P9 | L2 |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | High | Likely | Medium | P18 | L1 |
| STD-010-CPP | Low | Probable | High | P2 | L3 |

### 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 securing stored data by converting the data from one form to another through the use of an algorithm that can only be understood if the user has the associated key. This policy applies to our implementation of DevSecOps because we will be storing sensitive data on our servers/setting up systems that will hold sensitive data, and we need to secure this data from potential breaches. |
| Encryption at flight | Encryption at flight is the concept of encrypting the data when it is being transmitted. Data is going to be transmitted either internally or externally with the different software packages we will be creating; so, encrypting the data for transmission gives us the benefits we saw in the previous section for when the data is in a more “vulnerable” state. |
| Encryption in use | Encryption in use is never leaving the data unsecured despite stage, source, or location. Again, data is going to be housed in multiple locations between us, our clients, or their customers. Providing this level of security for data despite the stage/location is paramount to our success in DevSecOps. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication provides a method for identifying users based on each user having a unique set of login credentials to access a network. |
| Authorization | Authorization is giving people permissions to perform certain actions like accessing data, altering data sets, or making system changes. In the same idea as adhereing to the principle of least privilege, giving authorized users only enough access to the data to complete their job helps to mitigate how many hands are in the cookie jar, which in turn makes the data more secured. |
| Accounting | Accounting is the idea of monitoring system and data usage during sessions of access/utilization and tabulating the data to see the different statistics and trends that are happening with said data. This is used to see who needs what levels of access, and how we can adjust the authorization levels throughout the business. This can also be used to see abnormalities in data access/usage that can help denote that there is mischief afoot. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.5 | 08/05/2021 | Spell Check (IT JUST WORKS) | Joe Schmoe | Todd Howard |
| 2.0 | 03/20/2022 | Initial Risks Established | Scott Baker | Trevor Hodde |
| 2.1 | 04/10/2022 | Expand and Finalize Risk Assessments | Scott Baker | Trevor Hodde |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

| Language | Acronym |
| --- | --- |
| C++ | CPP |
| C | CLG |
| Java | JAV |