**Green Pace Developer: Security Policy Guide – Michael Zietz**



# 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 input into a system from a user should not be able to cause errors or exits. One of the most common ways this occurs is when the incorrect data type is entered in a specific prompt. These errors are avoidable by checking the user’s input data and validating it is acceptable for the system before using it in any capacity. |
| 1. Heed Compiler Warnings | A development environment contains a compiler that will notify the programmer when an issue is present with specific details surrounding the topic. It is a programmer’s duty to listen to these warnings to avoid issues at any chance to avoid compromising the stability of the system. |
| 1. Architect and Design for Security Policies | This principle demands that a secure system be analyzed and structured in a way that inherently deploys secure tactics. The infrastructure of the secure system should be created in a beneficial way that makes secure implementation possible. |
| 1. Keep It Simple | Over complicating code can not only make it difficult for coworkers to understand and build work with, but it can increase the likelihood of errors to occur in code and make it more difficult to catch them. |
| 1. Default Deny | By defaulting to deny access or permissions rather than include allows a minimum line of defense to be established. This reduces the amount of work required to establishing a new item into the system because less needs to be specified for it to still remain safe. This reduces the likelihood that permissions are easily accessible be reducing permissions granted to only those that are specified. |
| 1. Adhere to the Principle of Least Privilege | The Principle of Least Privilege encourages only providing permissions as needed and nothing more than is necessary. This prevents unknown parties from unknowingly having greater access than they should and allows a team to lock down security on the few with higher permissions even better. |
| 1. Sanitize Data Sent to Other Systems | This principle highlights the important of adhering to the data requirements of a subsystem intended to receive from the sender. Data |
| 1. Practice Defense in Depth | Defense in Depth is the practice of utilizing multiple layers of security to guarantee the maximum amount of security possible. This method deploys various tactics intended to all work incongruence for achieving a higher level of security than possible with a single element. |
| 1. Use Effective Quality Assurance Techniques | This principle ensures a review process during a system’s development. It ensures that a product being developed meets the specified quality and security assurance. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard implores being proactive about taking a secure mindset when developing. With each step forward in a project, vulnerability mitigation is taken into account. This prevents necessary, expensive, and overwhelming reworking or redesign of a system if security is not considered until later in the development process. |

### 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** | **Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Data Type** | INT30-C | Integer data type has a maximum number it can store because of data storage limitations. If an integer that exceeds the maximum is attempted to be created it will result in an incorrect mathematical operation. This is known as an overflow. Underflows are also possible when subtraction is performed that mimics the lower boundary of the integer data type. |

| **Noncompliant Code** |
| --- |
| The following code is noncompliant because it allows any integers to be summed and does not take into account the maximum data possibly stored by the integer data type. If this error happens, erroneous mathematics we will be produced. |
| void func(unsigned int a, unsigned int b) {  unsigned int sum = a + b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In the following example, a conditional statement outputs a warning when the coding standard is violated and does not proceed with the sum. |
| #include <limits.h>    void func(unsigned int a, unsigned int b) {  unsigned int usum;  if (UINT\_MAX - a < b) {  /\* Handle error \*/  } else {  sum = a + b;  }  /\* ... \*/  } |

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

| **Principles:** ValidateInput Data – Validation of input data eliminates this error as screening for specific values that would cause this issue can be caught when specified.  Sanitize Data Sent – Preventing data sent that will knowingly cause this error is avoidable by sanitizing data sent that would cause overflow or underflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **integer-overflow** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-INT30** | Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule by ensuring that operations are checked for overflow before being performed (Be mindful of exception INT30-EX2 because it excuses many operations from requiring [validation](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validation), including all the operations that would validate a potentially dangerous operation. For instance, adding two unsigned ints together requires validation involving subtracting one of the numbers from UINT\_MAX, which itself requires no validation because it cannot wrap.) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that integer conversions do not result in lost or misinterpreted data** |
| --- | --- | --- |
| **Data Value** | INT31-C | Data conversions between integers and other data types allow data to be misinterpreted or lost due to truncation, loss of sign, or range errors. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code allows data to be misinterpreted or lost when converting between two different data types |
| #include <limits.h>    void func(void) {  unsigned long int a = ULONG\_MAX;  signed char sc;  sc = (signed char)a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The following compliant code prevents erroneous conversions between integers and characters by checking if the integer exceeds the signed character’s maximum stored value before proceeding. |
| #include <limits.h>    void func(void) {  unsigned long int a = ULONG\_MAX;  signed char sc;  if (a <= SCHAR\_MAX) {  sc = (signed char)a; /\* Cast eliminates warning \*/  } else {  /\* Handle error \*/  }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – Effective quality assurance will review code that is likely to produce errors. In this instance code that coverts integers with other data types is correctly implemented or prevented from being released.  Keep it Simple – There are likely better conversions available than to convert between integers and other data types, so doing so only opens up a larger margin for error to occur. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT**  **LANG.CAST.COERCE LANG.CAST.VALUE**  **ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC**  **LANG.MEM.TBA** | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value  Truncation of allocation size Truncation of size  Tainted buffer access |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity)\* | 2017.07 | **NEGATIVE\_RETURNS**  **REVERSE\_NEGATIVE**  **MISRA\_CAST** | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior  Can find instances where a negativity check occurs after the negative value has been used for something else  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to modify string literals** |
| --- | --- | --- |
| **String Correctness** | STR30-C | The behavior is undefined if a program attempts to modify any portion of a string literal. Modifying a string literal frequently results in an access violation because string literals are typically stored in read-only memory. |

| **Noncompliant Code** |
| --- |
| The following code is noncompliant because it attempts to modify the string literal which is undefined behavior. This will result in an error. |
| char \*str = "example String";  str[0] = 'E'; |

| **Compliant Code** |
| --- |
| Because a character array is used in this case, the first letter of the string can be accessed because each character is its own, accessible object. |
| char str[] = "example String";  str[0] = 'E'; |

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

| **Principles(s):** Heed Compiler Warnings – It is likely that attempting to modify a string literal, due to it being an undefined behavior, will produce a compiler error or warning for the developer to notice. This should be followed and corrected accordingly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **signal-handler-unsafe-call** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-SIG30** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **BADFUNC.SIGNAL** | Use of signal |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of the rule for single-file programs |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | STR02-C | Sanitizing data when passing to complex subsystems, like SQL queries, can prevent abuse of data transfer between endpoints. |

| **Noncompliant Code** |
| --- |
| The following code is noncompliant because it allows placement a matching quote inside the data, closing off your opening quote, and then the rest of the parameter data can contain malicious instructions. |
| String injectedSQL = “SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred' or 'hi'='hi';”;  runQuery(injectedSQL); |

| **Compliant Code** |
| --- |
| The following code is compliant because it searches the submitted string for signs of injection before submitting it into the SQL subsystem. |
| Std::string injectedSQL = “SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred' or 'hi'='hi';”;  if (!injectedSQL.find(' or ') != std::string::npos )  {  runQuery(injectedSQL);  } |

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

| **Principles(s):** Sanitize Data Sent – This principle is rather self-explanatory. By reviewing and sanitizing data sent in general, we can also ensure that data is sanitized when passed to complex subsystems as well. This is important to avoid an abuse of transfer between endpoints. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **IO.INJ.COMMAND IO.INJ.FMT IO.INJ.LDAP IO.INJ.LIB IO.INJ.SQL IO.UT.LIB IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | MEM30-C | Accessing a pointer to memory that have been deallocated can result in exploitable vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The following code is noncompliant because it attempts to access and read memory that has already been freed. |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  for (struct node \*p = head; p != NULL; p = p->next) {  free(p);  }  } |

| **Compliant Code** |
| --- |
| The following code is compliant because it attempts to store a reference to the pointer in memory before freeing it. |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  struct node \*q;  for (struct node \*p = head; p != NULL; p = q) {  q = p->next;  free(p);  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard – Accessing freed memory can cause issues or allow others to exploit this vulnerability. Adopting secure coding standards will prevent creation of code that can potentially be abused or used maliciously. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **dangling\_pointer\_use** | Supported  Astrée reports all accesses to freed allocated memory. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM30** | Detects memory accesses after its deallocation and double memory deallocations |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.UAF** | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **USE\_AFTER\_FREE** | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | DCL03-C | Assertions are typically used for diagnostics and testing to eliminate software defects, but the assert() function is very limited in its uses and can provide compromised results. It is recommended to use static\_assert() function instead because it can provide compile time feedback. |

| **Noncompliant Code** |
| --- |
| The following code is noncompliant because it attempts to assert a property in regard to existing memory-mapped structures. It will result in a runtime error or a difficult to noticeable malfunction. |
| #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** |
| --- |
| The following code is compliant because it uses the static\_assert() function which allow incorrect assumptions to be diagnosed at compile time rather than terminating due to a 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 – Learning the difference in ineffective and effective testing techniques is crucial for ensuring a greater quality for a product. Ineffective strategies for diagnostics should be avoided. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Exception objects must be nothrow copy constructible** |
| --- | --- | --- |
| **Exceptions** | ERR60-CPP | If the copy constructor for the exception object type throws during the copy initialization, std::terminate() is called, which can result in possibly unexpected implementation-defined behavior. The copy constructor for an object thrown as an exception must be declared noexcept, including any implicitly-defined copy constructors. The C++ Standard allows the copy constructor to be elided when initializing the exception object to perform the initialization if a temporary is thrown. Many modern compiler implementations make use of both optimization techniques. |

| **Noncompliant Code** |
| --- |
| The following code is noncompliant because S has a std::string data member, and the copy constructor for std::string is not declared noexcept, the implicitly-defined copy constructor for S is also not declared to be noexcept. An std::bad\_alloc exception may occur instead if there is insufficient memory to complete the copy operation for std::string. |
| #include <exception>  #include <string>    class S : public std::exception {  std::string m;  public:  S(const char \*msg) : m(msg) {}    const char \*what() const noexcept override {  return m.c\_str();  }  };    void g() {  // If some condition doesn't hold...  throw S("Condition did not hold");  }    void f() {  try {  g();  } catch (S &s) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| The following code is compliant because it assumes that the type of the exception object can inherit from std::runtime\_error, or that type can be used directly. This is in contrast to std::string because the former type is required for a length error message. |
| #include <stdexcept>  #include <type\_traits>    struct S : std::runtime\_error {  S(const char \*msg) : std::runtime\_error(msg) {}  };    static\_assert(std::is\_nothrow\_copy\_constructible<S>::value,  "S must be nothrow copy constructible");    void g() {  // If some condition doesn't hold...  throw S("Condition did not hold");  }    void f() {  try {  g();  } catch (S &s) {  // Handle error  }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – Exception handling is a portion of development that attempts to roughen out the endless possibilities a program can interface with. These exceptions attempt to cause new solutions where errors have been introduced to a program rather than crashing, breaking, or proceeding faultily. It is the backbone of quality assurance for a program and utilizing these exception throws correctly will determine if a program is robust or frail in a given environment. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err60-cpp | Checked by clang-tidy |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3508** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR60-a** **CERT\_CPP-ERR60-b** | Exception objects must be nothrow copy constructible An explicitly declared copy constructor for a class that inherits from 'std::exception' should have a non-throwing exception specification |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR60-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr60cpp.html) | Checks for throwing exception object in copy constructor (rule fully covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not abruptly terminate the program** |
| --- | --- | --- |
| [Student Choice] | ERR50-CPP | The std::abort(), std::quick\_exit(), and std::\_Exit() functions are used to terminate the program in an immediate fashion. They do so without calling exit handlers registered with std::atexit() and without executing destructors for objects with automatic, thread, or static storage duration. Open streams with unwritten buffered data may or may not be flushed, open streams may or may not be closed, and temporary files may or may not be removed. Because these functions can leave external resources, such as files and network communications, in an indeterminate state, they should be called explicitly only in direct response to a critical error in the application. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() may throw an exception. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  throwing\_func();  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

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

| **Principles(s):** ValidateInput Data – Ideally, a program should validate input data to avoid the possibility a program would abruptly terminate. Properly handling and ensuring proper data input will inherently resolve the need for the abort(), quick\_exit(), and \_Exit() functions from being needed as they loop the user back into reentering data until an acceptable data has been entered.  Architect and Design for Security Policies – Programs should not be designed to self terminate during usage until the program has finished all proper activities and the user is finished accessing it. Abruptly terminating a program can not only terminate the environment and intended program experience but also leave important data, streams and resources open. The program must be designed to securely close these avenues before termination and disallowing these processes can create a poor structure. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stdlib-use** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++5014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.TERMINATE** **CERT.ERR.ABRUPT\_TERM** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Close files when they are no longer needed** |
| --- | --- | --- |
| [Student Choice] | FIO42-C | When utilizing an external file in a program, it must be opened in order to be read properly. It must also be close however before exiting or returning because it does not properly clear used memory during runtime, and the program may run out of file descriptors or handles available. This eventually causes any newly attempted files to fail. |

| **Noncompliant Code** |
| --- |
| The following code is noncompliant because it does not successfully close the opened file before exiting the current function. |
| int func(const char \*filename) {  FILE \*f = fopen(filename, "r");  /\* code utilizing file \*/  return 0;  } |

| **Compliant Code** |
| --- |
| The following code is compliant because it properly closes the read file before returning at the end of the function. |
| #include <stdio.h>    int func(const char \*filename) {  FILE \*f = fopen(filename, "r");  /\* code utilizing file \*/  fclose(f);  return 0;  } |

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

| **Principles(s):** Architect and Design for Security Policies – After a program is finished with an opened stream, it must be closed in order to free up memory and resources as well as prevent future errors occurring. A well structured and designed program will account for this and design the program to close these avenues before continuing as this is the security policy.  Adhere to the Principle of Least Privilege – Once a stream is no longer needed from a program, it is unwise to leave that avenue open longer than needed potentially granting it more access than it absolutely needs at one point in time. Adhering to the principle of lease privilege will recognize that this stream is not longer necessary and to be terminated. If future access is needed it can always traverse the needed avenues to be granted access properly for the intended task. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported, but no explicit checker |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.LEAK** | Leak |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **RESOURCE\_LEAK (partial)** | Partially implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not form or use out-of-bounds pointers or array subscripts** |
| --- | --- | --- |
| [Student Choice] | ARR30-C | Arrays cannot possess tables with less than zero indexes as they start at zero and increase by one. However, since a pointer can access the array as an integer, negative numbers can mistakenly be used and cause errors. |

| **Noncompliant Code** |
| --- |
| The following code is noncompliant because it fails to reject negative integers used as index values. In the event a negative integer is used, it will result in an error message due to undefined behavior. |
| enum { TABLESIZE = 100 };    static int table[TABLESIZE];    int \*f(int index) {  if (index < TABLESIZE) {  return table + index;  }  return NULL;  } |

| **Compliant Code** |
| --- |
| The following code is compliant because it rejects negative integers in addition to those larger than the array’s limit. |
| enum { TABLESIZE = 100 };    static int table[TABLESIZE];    int \*f(int index) {  if (index >= 0 && index < TABLESIZE) {  return table + index;  }  return NULL;  } |

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

| **Principles(s):** ValidateInput Data – Pointers or array subscripts should not be allowed to be created from user input and should be accounted for before a program attempts to initialize and use these out of bounds characters.  Architect and Design for Security Policies – Proper design and architect for security policies will have the errors surrounding the limitations of pointers and arrays and the ability for one to input acceptable data that has devastating consequences on the programs machinations. Taking this into account and coding around it will assist in preventing this issue. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **array-index-range array-index-range-constant null-dereferencing pointered-deallocation return-reference-local** | Partially checked  Can detect all accesses to invalid pointers as well as array index out-of-bounds accesses and prove their absence.  This rule is only partially checked as invalid but unused pointers may not be reported. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-ARR30** | Can detect out-of-bound access to array / buffer |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.MEM.BO LANG.MEM.BU LANG.MEM.TBA LANG.MEM.TO** **LANG.MEM.TU LANG.STRUCT.PARITH LANG.STRUCT.PBB** **LANG.STRUCT.PPE BADFUNC.BO.\*** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun Pointer Arithmetic Pointer before beginning of object Pointer past end of object A collection of warning classes that report uses of library functions prone to internal buffer overflows. |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **array-index-range array-index-range-constant null-dereferencing pointered-deallocation return-reference-local** | Partially checked  Can detect all accesses to invalid pointers as well as array index out-of-bounds accesses and prove their absence.  This rule is only partially checked as invalid but unused pointers may not be reported. |

### 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.

Incorporating security protocols at every stage of the DevOps process transforms it into DevSecOps. During the initial “Assess and Plan” stage, the focus is on identifying potential threats. As we move to the “Design” and “Build” stages, the emphasis shifts to securing the Integrated Development Environment (IDE). The “Verify & Test” phase is enhanced by integrating static application security testing and automated security assessments, in addition to the standard unit and integration tests.

After deployment, the security measures persist with ongoing automated tests that include integrity verifications and layered security strategies. Techniques such as network surveillance, ethical hacking, and the analysis of network and performance logs contribute to the continuous identification of threats. Similar to quality assurance testing, security evaluations should be conducted frequently and initiated early in the development cycle.

### 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 |
| --- | --- | --- | --- | --- | --- |
| INT30-C | High | Likely | High | **P9** | **L2** |
| ARR30-C | High | Likely | High | **P9** | **L2** |
| INT31-C | High | Probable | High | **P6** | **L2** |
| STR02-C | High | Likely | Medium | **P18** | **L1** |
| MEM30-C | High | Likely | Medium | **P18** | **L1** |
| FIO42-C | Medium | Unlikely | Medium | **P4** | **L3** |
| STR30-C | Low | Likely | Low | **P9** | **L2** |
| ERR50-CPP | Low | Probable | Medium | **P4** | **L3** |
| ERR60-CPP | Low | Probable | Medium | **P4** | **L3** |
| DCL03-C | Low | Unlikely | High | **P1** | **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 at rest | Encryption at rest safeguards data that is not actively moving from device to device or network to network. It encompasses data stored on various devices such as hard drives, smartphones, laptops, and cloud storage. To secure this data, methods like encryption software, full disk encryption, and security measures for portable devices and computers are utilized. |
| Encryption in flight | Encryption in flight refers to the security measures applied to data as it travels across networks. This includes data being transferred between devices internally or to external networks. Protection during transit can be achieved with methods like email encryption, Data Loss Prevention (DLP) systems, and robust network security tools including firewalls and authentication mechanisms. Additionally, it’s crucial to evaluate the security of the routes that data traverses. |
| Encryption in use | Encryption in use is aimed at securing data during its active phase, which includes when it’s being created, modified, or actively processed. To protect such data, it’s essential to establish control and security measures before the data is utilized. Additionally, regulating who has access and verifying identities are key steps in reducing potential threats to this data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication involves verifying an individual’s identity. It encompasses various methods, including static passwords, temporary passwords, digital certificates, and biometric verification. These identification techniques are utilized to signal to the system that the individual is who they say they are and to allow access. |
| Authorization | Authorization defines a user’s access level and permissions within a system, playing a crucial role in information and cybersecurity. While authentication establishes a user’s identity, authorization sets the boundaries of the user’s access, restricting their ability to interact with sensitive information that isn’t necessary for their role, or defining the extent of their permissions while accessing data. |
| Accounting | Accounting refers to the systematic recording of user activities within a system, including details like time stamps, resources accessed, and data transfer details. This documentation is crucial for constructing a detailed log of user actions and is instrumental for forensic examinations and investigative purposes if needed. |

**\***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 | 06/16/2024 | Project 1 Submission | Michael Zietz | [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 |