**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 | All data input from external sources must be validated to meet the system’s standards. External input, whether from another system, a user, an API, etc. should never be taken at face value and must be scrutinized to fall within the appropriate, non-dangerous standards of a system. |
| 1. Heed Compiler Warnings | While compiler errors halt a system, warnings do not, and they could indicate dangerous errors in a system. Be sure to investigate all warnings for code errors that could result in vulnerabilities and do not disregard warnings outright. |
| 1. Architect and Design for Security Policies | All functions, methods, variables, and the system design as a whole should be approached with a security mindset. All proposals should be scrutinized for potential security breaches and abuses, whether malicious or accidental. |
| 1. Keep It Simple | All functions should be as simple as possible. If there are sub-functions within a function that extend the parent functions complexity, then those sub-functions should be considered to be their own functions, especially if that functionality is reused elsewhere in the system. |
| 1. Default Deny | White-listing should be used over black-listing. When accepting user input, login credentials, or any external input or function calls, the default should be to deny the request unless said request meets the explicit criteria laid out by the system. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege should be followed. No user or request should be granted any further permission to system resources other than what is absolutely necessary. If a user just needs to be able to read documents, then they should not be allowed to modify them. |
| 1. Sanitize Data Sent to Other Systems | Ensure that no sensitive information is included in data sent to other systems. Data passed to other systems should else be sanitized for potentially harmful code or commands to other systems, like SQL injections. |
| 1. Practice Defense in Depth | Utilize multiple layers and methods of defense such as firewalls, intrusion detection software, role-based access-control, and more can help prevent or at least mitigate attacks if one system is bypassed. Multiple defense layers provide a necessary redundancy. |
| 1. Use Effective Quality Assurance Techniques | Both individual and group code reviews can help detect potential vulnerabilities in code. Individual reviews can work because the developer who wrote the code is most familiar with it, but other programmers may see logics errors that the writer did not. Penetration software is another good assurance technique for testing typical security faults. |
| 1. Adopt a Secure Coding Standard | Following a consistent, secure coding standard can prevent vulnerabilities from materializing by reducing novelty in code. Keeping the coding standard rigid puts developers in a security mindset from start to finish. |

### 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** | **One Definition Rule** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Collisions in definitions could result in undefined behavior. Follow the One Definition Rule to ensure deterministic behavior. |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule*](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule)

| **Noncompliant Code** |
| --- |
| While the code files define the same class, they use different definitions, resulting in undefined behavior when this code is compiled into an executable. |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| Violations can be avoided by ensuring the context in each code file is enforced within their scope and avoid collisions outside of their scope. |
| // a.cpp  namespace {  struct S {  int a;  };  }    // b.cpp  namespace {  class S {  public:  int a;  };  } |

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

| **Principles(s):** This standard maps to principles 3, 4, and 10. It is important to design securely which means avoiding confusing definitions, so this standard aligns with principle 3. This standard aligns with principle 4 because the standard keeps code simple and establishes a consistent coding standard which aligns with principle 10. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier.  Can help ensure the one-definition rule is not violated and is well integrated with C++ integrated development environments like Visual Studio. |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | This automated suite can detect this code error. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Enforce Enumeration Value Range** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Improper bounds checking on enumerations could result in data integrity violations or even a buffer overflow. Ensure given values are within the range of acceptable enumerations. |

| **Noncompliant Code** |
| --- |
| Following code checks the bounds after it has cast the enumeration type which could lead to a security violation. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| The following code not only performs the check prior to casting the type, but also implicitly converts the variable to the desired value of int. |
| enum EnumType : int {  First,  Second,  Third  };  void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value*](https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value)

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

| **Principles(s):** This standard maps to principle 3, as it designs for security policy. The standard maps to principle 4, as it keeps the code simple. The standard maps to principle 5, as it defaults to denying any value not within the acceptable range. The standard maps to standard 10 by following a consistent standard for checking enumeration ranges. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| Polyspace Bug Finder | R2024b | CERT C++: INT50-CPP | Checks for casting to out-of-range enumeration value (rule fully covered). |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not specify the bound of a character array initialized with a string literal** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | When initializing a char array with a string literal, it is important to not specify the bounds as it will exclude the null terminating character which could lead to a vulnerability. |

| **Noncompliant Code** |
| --- |
| The code below specifies the bounds of the array. This excludes the null terminator character which could result in a vulnerability where the null terminator is expected. |
| const char[5] = “Hello”; |

| **Compliant Code** |
| --- |
| Do not specify the bounds and let the compiler allocate sufficient memory to accommodate the string and the null terminator character. |
| const char[] = “Hello”; |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/display/c/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)

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

| **Principles(s):** This standard aligns with principle 3 as it follows an architect and design policy based on security. The standard aligns with principle 4 because it keeps things simple by letting the compiler handle null terminator character and removes human error. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_C-STR11-a | Do not specify the bound of a character array initialized with a string literal |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prepared SQL Statements** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Use prepared statements for SQL queries to prevent SQL injections. Do not concatenate strings but use formatted strings to prevent security breaches. |

| **Noncompliant Code** |
| --- |
| The code below simply concatenates the strings passed by the user and could result in a successful SQL injection. |
| std::string sqlStatement = "select \* from db\_user where username=" + username + " and password =" + password; |

| **Compliant Code** |
| --- |
| The code below prevents an SQL injection by formatting the string to create a parametric query. |
| std::string sqlStatement = format( "select \* from db\_user where username={0} and password ={1}", username, password); |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection*](https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection)

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

| **Principles(s):** This standard aligns with principle 1, as it validates input from an external source. This standard aligns with 6, as it ensures the SQL command is executed with the appropriate authority level by preventing an injection that could escalate privileges. This standard adheres to principle 7, as it sanitizes data it sends to another system. This standard adheres to principle 8, as it is one used in a series of defensive mechanisms for accessing databases. This standard follows principle 10, as it follows a consistent and secure coding standard with all SQL queries are constructed with prepared statements. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2024.2 | CWE-89-a | Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Allocate Sufficient Memory for an Object** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | When assigning memory to an object, ensure the object is allocated sufficient memory to store its contents in order to avoid a buffer overflow. |

| **Noncompliant Code** |
| --- |
| In the example below, memory is allocated based on the size of the pointer instead of the object in question. This causes less memory than is needed to be assigned to the object causing a overflow. |
| struct tm \*tmb;  tmb = (struct tm \*)malloc(sizeof(tmb));  if (tmb == NULL) {  return NULL;  }  /\* Code to assign variable \*/ |

| **Compliant Code** |
| --- |
| Code below dereferences variable ‘tmb’ and uses malloc to assign memory based on the size of the object and not the pointer. |
| struct tm \*tmb;  tmb = (struct tm \*)malloc(sizeof(\*tmb));  if (tmb == NULL) {  return NULL;  }  /\* Code to assign variable \*/ |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/display/c/MEM35-C.+Allocate+sufficient+memory+for+an+object*](https://wiki.sei.cmu.edu/confluence/display/c/MEM35-C.+Allocate+sufficient+memory+for+an+object)

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

| **Principles(s):** This standard aligns with principle 6, as it prevents a buffer overflow and escalation of privileges**.** This standard aligns with principle 9, as these types of errors are logical in nature and often caught in code reviews and with automated software. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_C-MEM35-a | Do not use sizeof operator on pointer type to specify the size of the memory to be allocated via 'malloc', 'calloc' or 'realloc' function |
| Astrée | 24.04 | malloc-size-insufficient | Partially checked  Besides direct rule violations, all undefined behavior resulting from invalid memory accesses is reported by Astrée. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the Termination Behavior of assert() and abort()** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use assertions consciously. Understand that assert() calls the abort() function, necessary cleanup functions could be bypassed and cause errors such as leaving sensitive temporary files on system or files written in an inconsistent state. |

| **Noncompliant Code** |
| --- |
| Below, the code defines cleanup operations if the exit code is not 0 but uses assert() which will call abort() and bypass that exit function call. |
| if (atexit(cleanup) != 0) {  /\* Handle error \*/  }  /\* ... \*/  assert(/\* Something bad didn't happen \*/); |

| **Compliant Code** |
| --- |
| The code below uses an if statement to check for the error instead and explicitly calls the exit function so that the cleanup operations can run. |
| if (atexit(cleanup) != 0) {  /\* Handle error \*/  }  /\* ... \*/  if (/\* Something bad happened \*/) {  exit(EXIT\_FAILURE); |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152296*](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152296)

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

| **Principles(s):** This standard follows principle 3, as it designs its assertions with security in mind. This standard algins with principle 9 since it follows quality assurance techniques. This standard adheres to principle 10 by adopting a secure coding standard for handling assertions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_C-ERR06-a | Do not use assertions. |
| Astrée | 24.04 | bad-function  bad-macro-use | Astrée also supports detecting this error. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle All Exceptions Thrown Before main() Begins Executing** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Throwing an exception that cannot be caught in the main() function, because it occurred during object construction prior main(), could result in abrupt termination and facilitate denial-of-service attacks. All exceptions prior to the start of main() should be caught and handled, and a catch-all branch should be considered where there could be unexpected errors. |

| **Noncompliant Code** |
| --- |
| The code below implicitly calls the constructor and has no method to catch exceptions. This could result in abrupt termination of the program. |
| struct S {  S() noexcept(false);  };    static S globalS; |

| **Compliant Code** |
| --- |
| The code below places the global variable in its own function to implement a try/except block. The variable can be accessed by the rest of the program by calling the function. |
| struct S {  S() noexcept(false);  };    S &globalS() {  try {  static S s;  return s;  } catch (...) {  // Handle error and gracefully terminating the application.  }  // Unreachable.  } |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR58-CPP.+Handle+all+exceptions+thrown+before+main%28%29+begins+executing*](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR58-CPP.+Handle+all+exceptions+thrown+before+main%28%29+begins+executing)

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

| **Principles(s):** This standard adheres to principle 2, as ‘noexcept’ with throw statements often creates compiler warnings and this standard requires those warnings to be heeded. This standard adheres to principle 3 by architecting and designing for security. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-ERR58-a | Exceptions shall be raised only after start-up and before termination of the program. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do Not Store an Already-Owned Pointer Value in an Unrelated Smart Pointer** |
| --- | --- | --- |
| Smart Pointers | STD-008-CPP | Smart pointers should be utilized carefully and mindfully. Do not haphazardly try to assign a pointer value to a smart pointer. This could lead to a double free memory vulnerability and cause undefined behavior. |

| **Noncompliant Code** |
| --- |
| The code below takes an existing pointer and assigns a value to smart pointers that remain unrelated to one another. This could lead to undefined behavior as the same memory will be freed if p2 is freed after p1 and vice versa. |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| This code block creates the first smart pointer from scratch and constructs the second shared\_ptr based on the first. This creates and maintains the relation between these pointers and prevents a double-free vulnerability. |
| #include <memory>  void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1);  } |

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

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

| **Principles(s):** This standard adheres to principle 3 by architecting its design for security. This standard follows principle 4 by keeping things simple and only using smart pointers instead of trying to mix them with regular pointers. This standard aligns with principle 10 by adopting a coding standard for dealing with pointers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MEM56-a | Do not store an already-owned pointer value in an unrelated smart pointer |
| Polyspace Bug Finder | R2024b | CERT C++: MEM56-CPP | Checks for use of already-owned pointers (rule fully covered). |

#### Coding Standard 9

| **Coding Standard** | **Label** | Do Not Depend on the Order of Evaluation for Side Effects |
| --- | --- | --- |
| Order of Evaluations | STD-009-CPP | When coding, be sure to explicitly define the order of evaluations for predictive side effects, changes in the state of execution. Do not rely on the implicit order of operations and ensure the correct side effects are achieved. |

| **Noncompliant Code** |
| --- |
| Below, the variable ‘i’ is evaluated more than once in the same expression with a non-explicit order and could lead to unexpected behavior. The lack of explicit sequencing leads to undefined behavior. |
| void f(int i, const int \*b) {  int a = i + b[++i];  // ...  } |

| **Compliant Code** |
| --- |
| This examine explicitly defines the order of evaluation and leaves nothing to interpretation. It explicitly declares the order in which objects should be evaluated. |
| void f(int i, const int \*b) {  ++i;  int a = i + b[i];  // ...  } |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP50-CPP.+Do+not+depend+on+the+order+of+evaluation+for+side+effects*](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP50-CPP.+Do+not+depend+on+the+order+of+evaluation+for+side+effects)

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

| **Principles(s):** This standard adheres to principle 4 by keeping things simple and explicitly defining the order of evaluation instead of relying on inference. This standard aligns with principle 10 by following a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2024b | CERT C++: EXP50-CPP | Checks for situations where expression value depends on order of evaluation (rule fully covered). |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-EXP50-a  CERT\_CPP-EXP50-b  CERT\_CPP-EXP50-c  CERT\_CPP-EXP50-d  CERT\_CPP-EXP50-e  CERT\_CPP-EXP50-f | The value of an expression shall be the same under any order of evaluation that the standard permits.  Don't write code that depends on the order of evaluation of function arguments.  Don't write code that depends on the order of evaluation of function designator and function arguments.  Don't write code that depends on the order of evaluation of expression that involves a function call.  Between sequence points an object shall have its stored value modified at most once by the evaluation of an expression.  Don't write code that depends on the order of evaluation of function calls. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do Not Leak Resources When Handling Exceptions** |
| --- | --- | --- |
| Exceptions and Errors | STD-010-CPP | Constructors do not automatically free resources if they terminate prematurely. Exception blocks should be sure to free any resources allocated by constructors in the event an error is thrown to avoid leaking resources. |

| **Noncompliant Code** |
| --- |
| In this example, the code fails to delete the object it attempted to construct, which causes a resource leak. |
| try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  throw;  } |

| **Compliant Code** |
| --- |
| The code here does delete the object with the catch block and releases its resources by calling delete. |
| try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  delete pst;  throw;  } |

*Reference:* [*https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR57-CPP.+Do+not+leak+resources+when+handling+exceptions*](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR57-CPP.+Do+not+leak+resources+when+handling+exceptions)

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

| **Principles(s):** This standard adheres to principle 3 by structure its exception catches with security in mind. This standard follows principle 10 by creating following a secure coding standard for handling resources in exception branches. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-ERR57-a | Ensure that resources are freed. |
| Polyspace Bug Finder | R2024b | CERT C++: ERR57-CPP | This checker checks for:  Resource leak caused by exception.  Object left in partially initialized state.  Bad allocation in constructor. |

### 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.

To transform DevOps to DevSecOps, we must shift security to the left. That is, security implementation must be part of every phase of the software development life cycle. During the “Assess and plan” stage, we must research the relevant regulatory requirements, model potential threats, and analyze their impact. The standards outlined in this document help us address these issues by establishing a secure coding standard for handling SQL queries, pointer assignment, order of evaluation processes, and more. This will enable developers to design a system during the “Design” phase that addresses security as well as business requirements by following our best practices and using a test-driven design that automates security tests.

Automated tools will test the for vulnerabilities and lapses in logic in the “Build” and “Verify and test” stages. Here, static testing tools like “Parasoft C/C++ test” or “Cppcheck” can be used in the “Build” phase to ensure our security standards are being followed. Unit tests can be automated for the “Verify and test” phase for testing the functionality and security of software to ensure the expected behavior is performed and security is maintained. Automated testing continues in the production phases with penetration testing, logging of activities and intrusion detection, responding to attacks, and assessing those attacks against the security baseline. This continuous testing and constant monitoring in addition to Green Pace’s existing firewall logs, malware logs, and operating system logs will create a defense-in-depth strategy for combating malicious actors.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | High | High | 4 |
| STD-002-CPP | Medium | Unlikely | Medium | Medium | 2 |
| STD-003-CPP | High | Probable | Medium | Medium | 3 |
| STD-004-CPP | High | Likely | Medium | High | 5 |
| STD-005-CPP | High | Probable | Medium | High | 4 |
| STD-006-CPP | Medium | Unlikely | Medium | Low | 2 |
| STD-007-CPP | Low | Likely | Low | Low | 1 |
| STD-008-CPP | High | Likely | Medium | High | 4 |
| STD-009-CPP | Medium | Probable | Medium | Medium | 3 |
| STD-010-CPP | Low | Probable | High | Medium | 3 |

### 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 is the process of encrypting data that is stored in a database while it is not being used or transported to another system or part of a system. Sensitive information such as passwords or credit card numbers should always be stored as hashes and not in plaintext. While this data is less at risk than data in transit or in use, it is still important to protect data at rest with firewalls, passwords, and encryption while in storage. |
| Encryption in flight | Encryption in flight is the encryption of data that is being transported to another part of the system or to another system. An example of this would be Transport Layer Security, or TLS, which is used to secure web traffic sent over HTTPS. Encrypting data in flight ensures outsiders prevents from easily reading the plaintext, which is unencrypted data. Data should always be transferred using encrypted protocols like TLS or Secure Shell, also known as SSH. |
| Encryption in use | Data being accessed by a user should be protected. Controlling access to data can prevent leaks or abuse while it is in use. Encryption in use can be enforced with the help of encryption in flight. No data being sent to another system to be altered should be sent in plaintext but encrypted as ciphertext and returned as ciphertext. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is verification of a person/system’s identity. We cannot allow just anyone to access our system or certain parts of it. User logins with emails/passwords, one-time passwords sent via text message, or biometric scans can help us verify if a legitimate actor is accessing our system. Unless it is a public webpage, no one should be able to access a system without being authenticated first. |
| Authorization | Not all actors should have the same level of authority. Authorization determines a user’s level of access to a system and is used to enforce the principle of least privilege. Not everyone needs the same level of access to perform their actions, and they should not be granted excess permissions. A lack of authorization can be disastrous when authentication is bypassed.  A malicious actor would easily gain excessive privileges and make damaging changes to a database or leak sensitive information. Authorization can mitigate if not prevent disasters from occurring. |
| Accounting | Accounting is the process of tracking system activity. Accounting keeps track of which user makes changes to the database. Accounting logs whether new users are added and who added them. It also tracks changes in user level of access, which files a user accesses, and more. Accounting enables a system to detect intrusions or unusual activity by tracking all activity conducted by users. Accounting helps system administrators ensure users are only able to perform the actions that they are allowed to have and can take corrective measures if there are any violations in policy or escalations of privilege. |

**\***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 |  |
| 1.1 | 05/23/2025 | Initial coding standards and principle definitions added. | Keegan Sevener |  |
| 1.2 | 06/13/2025 | Risk assessment and mapping of principles to coding standards. Automation addressed and policies for encryption and AAA defense added. | Keegan Sevener |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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