**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 | Input from users should be properly checked to make sure the input is valid. Check user input to determine if the input falls in line with what is expected in the input field. All input from a user should follow the same level of validation. This all works to prevent attacks like SQL injection that utilize input fields to gain access to data they are not supposed to have. |
| 1. Heed Compiler Warnings | Utilize the highest warning level for your compiler making sure to pay attention to the compiler warnings (Varne, 2021). When errors present themselves address the problem eliminating the problems in your code, ensuring the code both compiles and is free of potential issues (Varne, 2021). This will help ensure your code is error free, eliminating avenues of potential access (Varne, 2021). |
| 1. Architect and Design for Security Policies | Create system architecture in your design that enforces security policies (Varne, 2021). Create a design for your system that will enforce the necessary security in the correct areas of your system (Varne, 2021). If the system consists of multiple different sub systems, each of these systems may contain different security policies to support what the system performs (Varne, 2021). When designing a system, you need to keep in mind what security will be necessary for each of its components to make a coherent, secure design. |
| 1. Keep It Simple | You should aim to make your system as simple as you can according to its required functionality (Varne, 2021). A simple system will generally have less areas of compromise and chance of errors (Varne, 2021). More complex systems are more likely to have errors and will require a more robust system of security to cover each of the more complex areas of access in it (Varne, 2021). |
| 1. Default Deny | In your system, the default access for each user should be the denied state (Varne, 2021). Users should have to gain permission in your system rather than you excluding users which should not have permission (Varne, 2021). Creating a system that defaults to deny will work to enforce that users with access to the system should have access as they were able to prove their permissions (Varne, 2021). |
| 1. Adhere to the Principle of Least Privilege | Users should not be given any more access than what is absolutely necessary. Users should only be given access to information and functions that are necessary for them to have, but no more. This limited access helps to reduce the amount of access users have to areas of the system they should not be able to access. |
| 1. Sanitize Data Sent to Other Systems | All data sent outside of your system should be sanitized (Varne, 2021). Users may be able to access functions outside of what they are supposed to be able to be resulting in data being sent to your subsystems giving attackers access to unauthorized information (Varne, 2021). Sanitizing all of the data sent out of your system will help to eliminate this issue (Varne, 2021). |
| 1. Practice Defense in Depth | A system should be built on multiple layers of security for each aspect of the system. These layers should be additive meaning if one is bypassed the other will still be relevant. This creates a system with multiple layers of failsafe and caution. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance techniques help detect vulnerabilities present in your code (Varne, 2021). Implementing them into your development will help ensure vulnerabilities are detected (Varne, 2021). When implementing quality assurance techniques, it is important to implement multiple different techniques in order to benefit from all of them (Varne, 2021). |
| 1. Adopt a Secure Coding Standard | A secure coding standard is an effective way to ensure all of your code adheres to the same level of necessary security throughout your system. It is important to adopt a secure coding standard as guidance to make your code as secure as possible as it serves as a guide for your code to eliminate security vulnerabilities (Varne, 2021). It is important that this standard is implemented throughout all of your system and utilizing multiple can be beneficial (Varne, 2021). |

### C/C++ Ten Coding Standards

#### Coding Standard 1

| **Coding Standard** | **Label** | **INT35-CPP. Use correct integer precisions** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | The precision value of an integer is the number of bits that represent its value (Carnegie Mellon University, 2023). It is important to use the correct percussion value, so it is not too large which could cause unexpected behavior (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code the method ensures the argument will be less than the number of bits used to store the int, but this is not the proper way to determine precision (Carnegie Mellon University, 2023). Therefore, if this is run when int has more padding bits the precision value can still be too large (Carnegie Mellon University, 2023). |
| #include <limits.h>   unsigned int pow2(unsigned int exp) {  if (exp >= sizeof(unsigned int) \* CHAR\_BIT) {  /\* Handle error \*/  }  return 1 << exp; } |

| **Compliant Code** |
| --- |
| In this code the popcount() function is used to properly determine the precision value and define it, to prevent unexpected behavior (Carnegie Mellon University, 2023). |
| #include <stddef.h> #include <stdint.h>   /\* Returns the number of set bits \*/ size\_t popcount(uintmax\_t num) {  size\_t precision = 0;  while (num != 0) {  if (num % 2 == 1) {  precision++;  }  num >>= 1;  }  return precision; } #define PRECISION(umax\_value) popcount(umax\_value) |

**Principles: 10 and 2**

| **This standard adheres to the principle of adopting a secure coding standard as utilizing the correct precision value is a standard that should be followed throughout your code. This standard also helps to adhere to the principle of heed compiler warning as incorrect precision values could cause complier warnings that this standard would address.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | Supported by Astree | “Supported: Astrée reports overflows due to insufficient precision.” (Carnegie Mellon University, 2023). |
| CodeSonar | 9.0p0 | LANG.ARITH.BIGSHIFT | “Shift Amount Exceeds Bit Width” (Carnegie Mellon University, 2023). |
| Parasoft C/C++test | 2024.2 | CERT\_C-INT35-a | “Use correct integer precisions when checking the right hand operand of the shift operator” (Carnegie Mellon University, 2023). |
| Polyspace Bug Finder | R2024b | CERT C: Rule INT35-C | “Checks for situations when integer precisions are exceeded (rule fully covered)” (Carnegie Mellon University, 2023). |

#### Coding Standard 2

| **Coding Standard** | **Label** | **INT30-CPP. Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Unsigned int variables can wrap resulting in the data value being incorrect for that variable (Carnegie Mellon University, 2023). It is important to establish ways to prevent this from happening using data types max and minimum to keep data values correct (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code no check is present therefore when performing the addition a wrap occurs in the int variable (Carnegie Mellon University, 2023). |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/ } |

| **Compliant Code** |
| --- |
| In this code a check occurs checking if the addition will result in value over the int max value (Carnegie Mellon University, 2023). This prevents the int variable from wrapping as it stops the addition from happening if it would (Carnegie Mellon University, 2023). |
| #include <limits.h>   void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum;  if (UINT\_MAX - ui\_a < ui\_b) {  /\* Handle error \*/  } else {  usum = ui\_a + ui\_b;  }  /\* ... \*/ } |

**Principles: 2 and 10**

| **This standard adheres to the principle of Heed Compiler Warnings. Utilizing unsigned int variable will result in a compiler warning, and following this standard will ensure you are heeding its warning. This standard would also relate to the principle of Adopt a Secure Coding Standard as utilizing assigned int variable is a secure measure that this standard ensures you follow.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | integer-overflow | “Fully checked” (Carnegie Mellon University, 2023). |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT30 | “Implemented” (Carnegie Mellon University, 2023). |
| CodeSonar | 9.0p0 | 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” (Carnegie Mellon University, 2023). |
| Coverity | 2017.07 | INTEGER\_OVERFLOW | “Implemented”(Carnegie Mellon University, 2023). |

#### Coding Standard 3

| **Coding Standard** | **Label** | **STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | When copying data to a buffer that is not large enough for that data a buffer overflow will occur (Carnegie Mellon University, 2023). This can be prevented by ensuring the location is large enough to hold the data (Carnegie Mellon University, 2023). When working with strings there will need to be enough space for the string and the null character (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code user input is not limited therefore users could enter an input larger than the max size of char buf, which would result in a buffer overflow (Carnegie Mellon University, 2023). |
| #include <iostream>   void f() {  char buf[12];  std::cin >> buf; } |

| **Compliant Code** |
| --- |
| In this code the location variable for user input is a string variable instead of a bounded array preventing buffer overflow (Carnegie Mellon University, 2023). |
| #include <iostream> #include <string>   void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo; } |

**Principle: 10**

| **This standard adheres to the principle of Use Effective Quality Assurance Techniques, ensuring string size is a technique you can use in your code to prevent unexpected errors. Utilizing this technique when working with strings will help prevent errors and align with the principle of utilizing quality techniques in your code.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stream-input-char-array | “Partially checked + soundly supported”(Carnegie Mellon University, 2023). |
| CodeSonar | 9.0p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | “No space for null terminator  Buffer overrun  Type overrun”(Carnegie Mellon University, 2023). |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | “Partially implemented”(Carnegie Mellon University, 2023). |
| Polyspace Bug Finder | R2024b | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | “Checks for:    Use of dangerous standard function  Missing null in string array  Buffer overflow from incorrect string format specifier  Destination buffer overflow in string manipulation  Insufficient destination buffer size  Rule partially covered.”(Carnegie Mellon University, 2023). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **FIO30-C. Exclude user input from format strings** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Input data from a user can be tainted so it is important to not use it with a formated i/o function (Carnegie Mellon University, 2023). This can result in users gaining access to information they are not supposed to have access to (Carnegie Mellon University, 2023). An example of this could be SQL injection tainted data. |

| **Noncompliant Code** |
| --- |
| In this code the input data is called to fprintsf() which allows that data to attack the system (Carnegie Mellon University, 2023). |
| #include <stdio.h> #include <stdlib.h> #include <string.h>   void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {   /\* Handle error \*/   } else if (ret >= len) {   /\* Handle truncated output \*/   }  fprintf(stderr, msg);  free(msg); } |

| **Compliant Code** |
| --- |
| In this code the fprintf() is replaced with fputs() so the input cannot attack the system as its contents are not passed (Carnegie Mellon University, 2023). |
| #include <stdio.h> #include <stdlib.h> #include <string.h>   void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {   /\* Handle error \*/   } else if (ret >= len) {   /\* Handle truncated output \*/   }  fputs(msg, stderr);  free(msg); } |

**Principle: 1**

| **This standard adheres to the principle of Validate user input data, as this standard works to handle user input data properly. This standard excludes input data where it should not be used allowing you to properly validate the data and adhere to the principle.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | " Partially implemented”(Carnegie Mellon University, 2023). |
| CodeSonar | 9.0p0 | IO.INJ.FMT  MISC.FMT | " Format string injection  Format string”(Carnegie Mellon University, 2023). |
| Coverity | 2017.07 | TAINTED\_STRING | " Implemented”(Carnegie Mellon University, 2023). |
| LDRA tool suite | 9.7.1 | 86 D | " Partially Implemented”(Carnegie Mellon University, 2023). |

#### Coding Standard 5

| **Coding Standard** | **Label** | **MEM50-CPP. Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | A Freed memory location can still appear to be valid while written to or read from leading to unexpected results as the location should be inaccessible (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code s gets dereferenced after it has already been deallocated, this is accessing freed memory (Carnegie Mellon University, 2023). Therefore, the vulnerability with be present in this code and the unexcepted behavior can occur (Carnegie Mellon University, 2023). |
| #include <new>   struct S {  void f(); };   void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f(); } |

| **Compliant Code** |
| --- |
| In this code the vulnerability is not present as the memory is not deallocated until after all of it use functionality has been done (Carnegie Mellon University, 2023). No freed memory is accessed in this code. |
| #include <new>  struct S {  void f(); };  void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s; } |

**Principles: 10 and 2**

| **This standard adheres to the principle of Adopt a Secure Coding Standard. Making sure to avoid accessing any freed memory throughout your code is a standard that should be adopted and used to prevent unexpected behavior. Utilizing this standard throughout your code will help to adhere to this principle. This standard also applies to the principle of heed compiler warnings as accessing freed memory may cause a warning that should not be ignored. Utilizing this standard will help ensure that warning is addressed.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | “Checked by clang-tidy, but does not catch all violations of this rule.” (Carnegie Mellon University, 2023). |
| CodeSonar | 9.0p0 | ALLOC.UAF | “Use after free” (Carnegie Mellon University, 2023). |
| Coverity | v7.5.0 | 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” (Carnegie Mellon University, 2023). |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MEM50-a | “Do not use resources that have been freed” (Carnegie Mellon University, 2023). |

#### Coding Standard 6

| **Coding Standard** | **Label** | **ERR50-CPP. Do not abruptly terminate the program** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Abruptly terminating a program can result in many issues with your code including by not limited to losing temporary files, corrupting data, leaving system elements in an open state (Carnegie Mellon University, 2023). This is something that you want to avoid doing in your code prioritizing properly terminating a program (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code calling f can result in std::at\_exit which is an abrupt end to the program (Carnegie Mellon University, 2023). |
| #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 code exceptions are caught and handled so throwing func will not rethrow and cause an abrupt end to the program (Carnegie Mellon University, 2023). |
| #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  }  // ... } |

**Principle 9**

| **This standard adheres to the principle of using effective quality assurance techniques. Utilizing this standard employs a secure technique that ensures the program is not ended improperly. This helps the program end correctly and applying this technique falls in line with this principle's guidance.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stdlib-use | “Partially checked” (Carnegie Mellon University, 2023). |
| CodeSonar | 9.0p0 | BADFUNC.ABORT  BADFUNC.EXIT | “Use of abort  Use of exit” (Carnegie Mellon University, 2023). |
| LDRA tool suite | 9.7.1 | 122 S | “Enhanced Enforcement” (Carnegie Mellon University, 2023). |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-ERR50-a  CERT\_CPP-ERR50-b  CERT\_CPP-ERR50-c  CERT\_CPP-ERR50-d  CERT\_CPP-ERR50-e  CERT\_CPP-ERR50-f  CERT\_CPP-ERR50-g  CERT\_CPP-ERR50-h  CERT\_CPP-ERR50-i  CERT\_CPP-ERR50-j  CERT\_CPP-ERR50-k  CERT\_CPP-ERR50-l  CERT\_CPP-ERR50-m  CERT\_CPP-ERR50-n | “The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception Never allow an exception to be thrown from a destructor, deallocation, and swap Do not throw from within destructor There should be at least one exception handler to catch all otherwise unhandled exceptions An empty throw shall only be used in the compound-statement of a catch handler Exceptions shall be raised only after start-up and before termination of the program Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) Function called in global or namespace scope shall not throw unhandled exceptions Always catch exceptions Properly define exit handlers The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid throwing exceptions from functions that are declared not to throw The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used” (Carnegie Mellon University, 2023). |

#### Coding Standard 7

| **Coding Standard** | **Label** | **ERR51-CPP. Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | When an exception is thrown and not handled in the code, the default handling of exceptions will occur (Carnegie Mellon University, 2023). This means the std::terminate() will be called which ends the exception abrupt and unsafely (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code there is no catch for the exception thrown by throwing\_func() (Carnegie Mellon University, 2023). In this code the default exception handling will occur cause an abrupt and unsafe end to it (Carnegie Mellon University, 2023). |
| void throwing\_func() noexcept(false);   void f() {  throwing\_func(); }   int main() {  f(); } |

| **Compliant Code** |
| --- |
| In this code there is a catch present in the int main() for the exception thrown from throwing \_func, where the exception is handled after catch (Carnegie Mellon University, 2023). |
| void throwing\_func() noexcept(false);   void f() {  throwing\_func(); }   int main() {  try {  f();  } catch (...) {  // Handle error  } } |

**Principle: 2**

| **This standard adheres to the principle of heed compiler warnings. Unhandled exceptions will create a warning in the compiler that should not be ignored. Following this standard will help ensure that this warning is addressed and the principle is respected.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | main-function-catch-all  early-catch-all | “Partially checked” (Carnegie Mellon University, 2023). |
| CodeSonar | 9.0p0 | LANG.STRUCT.UCTCH | “Unreachable Catch” (Carnegie Mellon University, 2023). |
| LDRA tool suite | 9.7.1 | 527 S | “Partially implemented” (Carnegie Mellon University, 2023). |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | “Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point” (Carnegie Mellon University, 2023). |

#### Coding Standard 8

| **Coding Standard** | **Label** | **MEM51-CPP. Properly deallocate dynamically allocated resources** |
| --- | --- | --- |
| **Memory Protection** | STD-008-CPP | Improperly deallocating a dynamically allocated pointer or deallocating a pointer value that no longer exists is undefined behavior and will result in unexpected results (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code a local variable is passed as the expression in the placement new operator, it is then passed to operator delete (Carnegie Mellon University, 2023). This is undefined behavior and will result in unexpected results (Carnegie Mellon University, 2023). |
| #include <iostream>   struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; } };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;   // ...   delete s1; } |

| **Compliant Code** |
| --- |
| In this code the operator delete is not present and instead s1 destructor is called explicitly, this is not undefined behavior (Carnegie Mellon University, 2023). |
| #include <iostream>   struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; } };   void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...   s1->~S(); } |

**Principle: 9**

| **This standard adheres to the principle of Use Effective Quality Assurance Techniques. Utilizing this standard will employ a quality technique that properly deallocate dynamically allocated resources. This will help to prevent unexpected behavior and also fall in line with the guiding principle of using quality techniques throughout your code.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-unix.MismatchedDeallocator | “Checked by clang-tidy, but does not catch all violations of this rule” (Carnegie Mellon University, 2023). |
| CodeSonar | 9.0p0 | ALLOC.DF  ALLOC.TM  ALLOC.LEAK | “Double free  Type mismatch Leak” (Carnegie Mellon University, 2023). |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | “Partially implemented” (Carnegie Mellon University, 2023). |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | “Use the same form in corresponding calls to new/malloc and delete/free  “Always provide empty brackets ([]) for delete when deallocating arrays  Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor  Properly deallocate dynamically allocated resources” (Carnegie Mellon University, 2023). |

#### Coding Standard 9

| **Coding Standard** | **Label** | **MEM52-CPP. Detect and handle memory allocation errors** |
| --- | --- | --- |
| **Memory Protection** | STD-009-CPP | The memory allocation operator will throw an exception if allocation fails by default (Carnegie Mellon University, 2023). It is imporant to check for if allocation fails so it can be handled properly (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code no check is present for failed allocation, and this can result in an abrupt ending of the program as std::size\_t can thow an exception when the allocation fails (Carnegie Mellon University, 2023). |
| #include <cstring>   void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new int[size];  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy; } |

| **Compliant Code** |
| --- |
| In this code allocation fail is properly checked for (Carnegie Mellon University, 2023). std::nothrow will return a nullptr if allocation fails (Carnegie Mellon University, 2023). This program properly checks for the nullptr value and handles it appropriately (Carnegie Mellon University, 2023). |
| #include <cstring> #include <new>   void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new (std::nothrow) int[size];  if (!copy) {  // Handle error  return;  }  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy; } |

**Principles: 9 and 2**

| **This standard adheres to the principle of Use Effective Quality Assurance Techniques. Employing this technique will implement a quality technique into your code that will properly handle memory allocation. This will adhere to the principle of utilizing these techniques throughout your code. This standard also adheres to the technique of heed compiler warnings. This is because improperly handled memory allocation may cause compiler warnings that this standard would address.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | “Finds inconsistencies in how function call return values are handled” (Carnegie Mellon University, 2023). |
| LDRA tool suite | 9.7.1 | 45 D | “Partially implemented” (Carnegie Mellon University, 2023) |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | “Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined” (Carnegie Mellon University, 2023) |
| Polyspace Bug Finder | R2024b | CERT C++: MEM52-CPP | “Checks for unprotected dynamic memory allocation (rule partially covered)” (Carnegie Mellon University, 2023) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **MEM56-CPP. Do not store an already-owned pointer value in an unrelated smart pointer** |
| --- | --- | --- |
| **Memory Protection** | STD-010-CPP | Creating an unrelated smart pointer object using a pointer value already owned can result in resetting the already owned pointer value causing unexpected behavior (Carnegie Mellon University, 2023). |

| **Noncompliant Code** |
| --- |
| In this code two smart pointers are created with the same pointer value causing them to both result in an empty value (Carnegie Mellon University, 2023). |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i); } |

| **Compliant Code** |
| --- |
| In this code shared\_ptr is used to avoid an empty value in the ptrs when the first ptr value is destroyed (Carnegie Mellon University, 2023). |
| #include <memory>  void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1); } |

**Principles: 10 and 9**

| **This standard adheres to the principle of adopting a secure coding standard. This standard establishes behavior that should not be performed throughout all of the code. This standard also helps to follow the principle of Use Effective Quality Assurance Techniques as when following this standard, you are employing effective techniques to avoid storing already owned pointer values in an unrelated smart pointer.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | dangling\_pointer\_use | “Fully Checked” (Carnegie Mellon University, 2023). |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM56 | “Implemented” (Carnegie Mellon University, 2023). |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MEM56-a | “Do not store an already-owned pointer value in an unrelated smart pointer” (Carnegie Mellon University, 2023) |
| Polyspace Bug Finder | R2024b | CERT C++: MEM56-CPP | “Checks for use of already-owned pointers (rule fully covered)” (Carnegie Mellon University, 2023) |

### Defense-in-Depth Illustration

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



### Automation



The existing DevSecOps diagram does a good job of creating, testing, and maintaining a system. The infinite loop of it is a good representation of the constant flow and development that a system would have, and it matches very well with what actual production, testing, and maintenance would look like. When adding automation into this existing process, it is important to look at where it would fit best. Automation would be most fitting inside the verify and test phase of the process. This allows for automated testing to be most beneficial as directly after the testing phase is the health check which will apply changes found in the testing phase. So, implementing automation into the verify and test phase would allow developers to benefit from what is found from the automated tests as they can implement any changes needed as necessary in the already existing health check phase. This would also ensure that before anything is released that the system is adhering to the standards the automated tests would find as they would take place in the pre-production section.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Medium | P1 | 3 |
| STD-002-CPP | High | Likely | Medium | P9 | 2 |
| STD-003-CPP | High | Likely | Medium | P18 | 1 |
| STD-004-CPP | High | Likely | Medium | P18 | 1 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CPP | low | Probable | Medium | P4 | 3 |
| STD-007-CPP | low | Probable | Medium | P4 | 3 |
| STD-008-CPP | High | Likely | Medium | P18 | 1 |
| STD-009-CPP | High | Likely | Medium | P18 | 1 |
| STD-010-CPP | High | Likely | Medium | P18 | 1 |

### Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest refers to properly encrypting data that is stored on disks or another form of backup media (Google, n.d). To apply encryption at rest, all data being stored should be encrypted using an encryption standard (Google, n.d). This means that any data being stored in a system will be encrypted and will have to go through the encryption process to be accessed and utilized. This helps improve the security of the data as even if an attacker were to gain access to the data, they would only have access to the encrypted unusable data (Google, n.d). This creates another layer of security surrounding the data where an attacker would then also have to find a way past the encryption to get the data to a usable state (Google, n.d). |
| Encryption in flight | Encryption in flight refers to properly encrypting data that is moving from one location to another, including moving internally to another area of the system or over the internet (Google, 2025). To apply encryption in flight you would ensure that proper encryption practices are applied to data when it is in transport (Google, 2025). This includes the start and end point of the data, ensuring the data is properly handled at both points maintaining proper encryption (Google, 2025). Encryption in flight is important to implement as if the data were to be compromised during the transportation process the attackers would only have access to the encrypted data (Google, 2025). This adds another layer of security to the data that is in transport, making the risk of attackers intercepting the data less detrimental as they would also have to infiltrate the encryption (Google, 2025). |
| Encryption in use | Encryption in use refers to properly encrypting data while it is being utilized (Velimirovic, 2023). In use data refers to data that is currently being used by the user, for example data that is being inputted, updated, accessed, and any other uses the user may have with the data (Velimirovic, 2023). To apply encryption to this data real time encryption must be applied to the data encrypting any data that is being input and decrypting data for the user as it is in use (Velimirovic, 2023). Encryption in use is important to implement as data within a system is always encrypted as long as the other encryption policies are also applied (Velimirovic, 2023). This helps to make any attacks that compromise data to be less impactful as the data will always be encrypted, meaning attackers will have to bypass the encryption as well (Velimirovic, 2023). This adds another layer of security to the data that users are actively using (Velimirovic, 2023). |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a way to verify who a user is, through the use of data only relevant to them (Gillis, 2024). This could include implementation of a username and password system, biometric scanner, or any other system that works to establish the identity of an individual (Gillis, 2024). This is important to implement into a system as properly identifying who the user is the first step and will allow you to implement authorization and accounting (Gillis, 2024). Proper authentication in a system helps build up the security that the user in the system is accounted for their identity has been verified, providing records and systems based on authentication (Gillis, 2024). |
| Authorization | Authorization is a way to properly provide access to aspects of a system to the users based on their identity (Gillis, 2024). Authorization must follow authentication as it relies directly on knowing the identity of the users (Gillis, 2024). In authorization after a user attempt to perform something in the system, the system will determine whether a user should have access to that function and authorize it accordingly (Gillis, 2024). This is important to implement in a system as it is integral to following the principle of least privilege. This can limit the access of what each user has access to, to only what is needed for them. This also works to make the system more secure as if one user is compromised (Gillis, 2024). The only aspects that are likely to be affected would be the ones that users have access to (Gillis, 2024). |
| Accounting | Accounting is a way to properly keep track of information related to how each user is using the system (Gillis, 2024). This can include information like time spent in the system, number of requests sent, data sent, and much more (Gillis, 2024). This is important to implement into a system as this information can be very useful in keeping a system secure (Gillis, 2024). This information can be used for analysis to determine if anything out of the ordinary is happening within the system (Gillis, 2024). An example being if a user is suddenly sending different types of data throughout the system this could indicate that they have been compromised. This data can also be used to access the performance of a system looking for problematic areas to address (Gillis, 2024). A well performing system will be more secure as it is less likely to perform unexpected behavior. |

## 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 | 05/26/2025 | First Draft | Trenton Mendiola |  |
| 3.0 | 06/15/2025 | Final Draft | Trenton Mendiola |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

References

Carnegie Mellon University. (n.d.). *SEI CERT C++ coding standard - SEI CERT C++ coding standard - Confluence*. Confluence. Retrieved May 25, 2025, from <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

Gillis, A. (2024, November 16). *What is authentication, authorization, and accounting (AAA)? | Definition from TechTarget*. Security. Retrieved June 15, 2025, from <https://www.techtarget.com/searchsecurity/definition/authentication-authorization-and-accounting>

Google. (2025, April). *Encryption in transit*. Google Cloud. Retrieved June 15, 2025, from <https://cloud.google.com/docs/security/encryption-in-transit>

Google. (n.d.). *Default encryption at rest*. Google Cloud. Retrieved June 15, 2025, from <https://cloud.google.com/docs/security/encryption/default-encryption>

Grammarly.Retrieved May 25, 2025, from <https://app.grammarly.com/>

Varne, A. (2021, June 9). *Secure coding guidelines*. Medium. Retrieved May 25, 2025, from <https://varneanand-88.medium.com/secure-coding-guidelines-126890bca67a>

Velimirovic, A. (2023, November 16). *Data encryption in use explained*. phoenixNAP Blog. Retrieved June 15, 2025, from <https://phoenixnap.com/blog/encryption-in-use>