**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 | Check all inputs to ensure it is safe, expected, and properly formatted. All inputs need to be considered untrusted until verified. It is the first line of defense, without its data is at risk of being compromised. |
| 1. Heed Compiler Warnings | Detecting warning tools for the compiler help eliminate any warning signs throughout the code. The compiler can find any errors and prevent the code from compiling. |
| 1. Architect and Design for Security Policies | Design software with security in mind from the beginning rather than an afterthought. Apply rules and policies that define how security should work. |
| 1. Keep It Simple | The simpler the code is the better to find any error or vulnerabilities. As code becomes more complex there is more room for errors to get missed. |
| 1. Default Deny | The assumption that any action could potentially be harmful to the system. This approach blocks all traffic by default unless it has been authorized by the firewall. |
| 1. Adhere to the Principle of Least Privilege | Giving too much privilege during the use of a system could negatively impact it by attackers taking advantage. Creating the least amount of privilege for most use could mitigate this. |
| 1. Sanitize Data Sent to Other Systems | Sanitation prevents malicious code being used during the transfer of data to another system. This could protect against attacks such as SQL injections. |
| 1. Practice Defense in Depth | Implement multiple defensive lines throughout the system to create layers of security. This could save any security flaws that might’ve been missed by creating a failure-safe system. |
| 1. Use Effective Quality Assurance Techniques | Having good QA techniques can help identify vulnerabilities early on and throughout the testing period. This ensures the integrity of the program eliminates overlooked vulnerabilities in the code. |
| 1. Adopt a Secure Coding Standard | Having standard help guide developers create software that is secure and reliable. Setting coding standards can eliminate missing steps in development that could lead to attacks later on. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Ensure that integer conversions do not result in lost or misinterpreted data |

| **Noncompliant Code** |
| --- |
| Loss of data can occur when converting unsigned int type to signed type. |
| void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This validates ranges when converting types. |
| void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  if (u\_a <= SCHAR\_MAX) {  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  } else {  /\* Handle error \*/  }  } |

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

| **Principles(s):** Policy 9: Use effective quality assurance technique, Mapping: something might get missed such as conversion of unsigned integers which is important to have a good quality assurance to pick up on small vulnerabilities like this. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | PROBABLE | MEDIUM | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | LANG.CAST.PC.AV  LANG.CAST.PC.CONST2PTR  LANG.CAST.PC.INT | Cast: arithmetic type/void pointer  Conversion: integer constant to pointer  Coercion alters value  Cast alters value  Truncation of allocation size  Tainted buffer access |
| Compass/ROSE | - | - | Can detect violation of this rule. However false warnings may be raised if limits.h is included |
| Astree | 24.04 | - | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| 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 |

Link: <https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data>

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not modify constant objects |

| **Noncompliant Code** |
| --- |
| The code below is attempting to change the value of a const object. |
| const int \*\*ipp;  int \*ip;  const int i = 42;  void func(void) {  ipp = &ip; /\* Constraint violation \*/  \*ipp = &i; /\* Valid \*/  \*ip = 0; /\* Modifies constant i (was 42) \*/  } |

| **Compliant Code** |
| --- |
| If the intent is to modify the value of I, then it should not be declared as const |
| int \*\*ipp;  int \*ip;  int i = 42;  void func(void) {  ipp = &ip; /\* Valid \*/  \*ipp = &i; /\* Valid \*/  \*ip = 0; /\* Valid \*/  } |

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

| **Principles(s):** Policy 4: Keep it simple, Mapping: Simple code helps find errors faster and less trouble such as using const |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | UNLIKELY | MEDIUM | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | assignment-to-non-modifiable-lvalue | Fully Checked |
| Coverity | 2017.07 | PW  MISRA C 2004 RULE 11.5 | Implemented |
| LDRA toll suite | 9.7.1 | 582 S | Fully implemented |
| Parasoft  C/C++ test | 2024.2 | CERT\_C-EXP40-A | A cast shall not remove any 'const' or 'volatile' qualification from the type of a pointer or reference |

#### Link: <https://wiki.sei.cmu.edu/confluence/display/c/EXP40-C.+Do+not+modify+constant+objects#:~:text=If%20an%20attempt%20is%20made,type%2C%20the%20behavior%20is%20undefined>.

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| The char pointer str is initialized to the address of a string literal |
| char \*str = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| This creates a copy of the string literal in the space allocated to safely modify str |
| char str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** Policy 10: Adopt a Security Coding Standard, Mapping: Having standards help guide developers stay within boundaries of safety practices. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | LIKELY | MEDIUM | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | String-literal-modfication  Write-to-string-literal | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| Coverity | 2017.07 | PW | Deprecates conversion from a string literal to "char \*" |
| Polyspace Bug Finder | R2024b | Cert C: Rule STR30-C | Checks for writing to const qualified object (rule fully covered) |

#### Link: <https://wiki.sei.cmu.edu/confluence/display/c/STR30-C.+Do+not+attempt+to+modify+string+literals>

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| No validation was done before the query which can be used for injection. |
| String pwd = hashPassword(password);  String sqlString = "SELECT \* FROM db\_user WHERE username = '"  + username +  "' AND password = '" + pwd + "'";  Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sqlString); |

| **Compliant Code** |
| --- |
| Validated username before installed into the query. |
| String pwd = hashPassword(password);  // Validate username length  if (username.length() > 8) {  // Handle error  }  String sqlString =  "select \* from db\_user where username=? and password=?";  PreparedStatement stmt = connection.prepareStatement(sqlString);  stmt.setString(1, username);  stmt.setString(2, pwd);  ResultSet rs = stmt.executeQuery(); |

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

| **Principles(s):** Policy 1: Validate Input Data, Mapping: check all inputs that can potentially be harmful to the system like SQL injection. Policy 8: Practice Defense in Depth, Mapping: Implement multiple layers to protect from SQL injection |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | LIKELY | MEDIUM | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 9.0p0 | JAVA.IO.INJ.SQL | SQL injection |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Parasoft Jtest | 2024.2 | CERT.IDS00.TDSQL | Protect against SQL injection |

#### Link: <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection>

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| S is dereferenced after it has been deallocated. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory is not deallocated until it is no longer required |
| #include <new>  struct S {  void f();  };  void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** Policy 1: Validate Input Data, Mapping: Validating data can verify what’s coming into the input of the pointers. Policy 2: Heed Compiler Warnings, Mapping: having a tool warning system that can detect such vulnerability can mitigate exploitable code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | LIKELY | MEDIUM | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | ALLOC.UAF | Use after free |
| 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 |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | Partially implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |

#### Link: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory>

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| Multiple numeric values are converted from the standard input stream. However, if the text received from the standard input stream cannot be converted into a numeric value that can be represented by an int. |
| #include <iostream>  void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| Exceptions are enabled so that any conversion failure results in an exception being thrown. However, this approach cannot distinguish between which values are valid and which values are invalid and must assume that all values are invalid. |
| #include <iostream>  void f() {  int i, j;  std::cin.exceptions(std::istream::failbit | std::istream::badbit);  try {  std::cin >> i >> j;  // ...  } catch (std::istream::failure &E) {  // Handle error  }  } |

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

| **Principles(s):** Policy 2: Heed Compiler Warnings, Mapping: having a tool could help detect this type of error in the code. Policy 9: Use Effective Quality Assurance Techniques, Mapping: Utilizing good QA techniques will help find this vulnerabilities early on in the development process rather than later. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| MEDIUM | UNLIKELY | MEDIUM | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| CodeSonar | 9.0p0 | BADFUNC.ATOF  BADFUNC.ATOI  BADFUNC.ATOL  BADFUNC.ATOLL | Use of atof Use of atoi Use of atol Use of atoll |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-ERR62-a | The 'atof', 'atoi', 'atol' and 'atoll' functions from the 'stdlib.h' or 'cstdlib' library should not be used |
| Polyspace Bug Finder | R2024b | CERT C++:ERR62-CPP | Checks for unvalidated string-to-number conversion (rule fully covered) |

#### Link: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR62-CPP.+Detect+errors+when+converting+a+string+to+a+number>

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| Neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| The main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):** Policy 4: Keep it simple, Mapping: having simple handlers for each exception mitigate the error issue. Policy 10: Adopt a Secure Coding Standard, Mapping: a guide for developer can maintain a proper standard for how to handle all exceptions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | PROBABLE | MEDIUM | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| CodeSonar | 7.2.0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Polyspace Bug Finder | R2024b | CERT C++: ERR51-CPP | Checks for unhandled exception (rule partially covered) |
| RuleChecker | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |

#### Link: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-008-CPP] | Guarantee that container indices and iterators are within the valid range |

| **Noncompliant Code** |
| --- |
| insert\_in\_table(), that has two int parameters, pos and value, both of which can be influenced by data originating from untrusted sources. The function performs a range check to ensure that pos does not exceed the upper bound of the array, specified by tableSize, but fails to check the lower bound. Because pos is declared as a (signed) int, this parameter can assume a negative value, resulting in a write outside the bounds of the memory referenced by table. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

| **Compliant Code** |
| --- |
| The parameter pos is declared as size\_t, which prevents the passing of negative arguments. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

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

| **Principles(s):** Policy 3: Architect and Design for Secuirty, Mapping: the design of the code is highly important to keep array references within the bounds of the array. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | LIKELY | HIGH | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Overflow\_upon\_deference | - |
| LDRA tool suite | 9.7.1 | 45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X | Partially implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2024b | CERT C++: CTR50-CPP | Checks for:  Array access out of bounds  Array access with tainted index  Pointer dereference with tainted offset  Rule partially covered. |

LINK: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR50-CPP.+Guarantee+that+container+indices+and+iterators+are+within+the+valid+range>

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| File Management | [STD-009-CPP] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| The file opened by the call to fopen() is not closed before function func() returns: |
| #include <stdio.h>    int func(const char \*filename) {  FILE \*f = fopen(filename, "r");  if (NULL == f) {  return -1;  }  /\* ... \*/  return 0;  } |

| **Compliant Code** |
| --- |
| The file pointed to by f is closed before returning to the caller. |
| #include <stdio.h>    int func(const char \*filename) {  FILE \*f = fopen(filename, "r");  if (NULL == f) {  return -1;  }  /\* ... \*/  if (fclose(f) == EOF) {  return -1;  }  return 0;  } |

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

| **Principles(s):** Policy 2: Heed Compiler Warning, Mapping: have a tool to detect when a file didn’t close could help mitigate this issue. Policy 8: Practice Defense in Depth, Mapping: multiple security layers can protect of any unclosed files. Policy 10: Adopt a Secure Coding Standard, Mapping: Setting up a standard can help guide developers to look for closing of file if there was a file opened. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| MEDIUM | UNLIKELY | MEDIUM | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2025.1 | DF4786, DF4787, DF4788 | - |
| Klocwork | 2025.1 | RH.LEAK | - |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP\_FI1051-a | Ensure resources are freed |

Link: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming (OOP) | [STD-010-CPP] | Honor replacement handler requirements |

| **Noncompliant Code** |
| --- |
| A replacement new\_handler is written to attempt to release salvageable resources when the dynamic memory manager runs out of memory. |
| #include <new>    void custom\_new\_handler() {  // Returns number of bytes freed.  extern std::size\_t reclaim\_resources();  reclaim\_resources();  }    int main() {  std::set\_new\_handler(custom\_new\_handler);    // ...  } |

| **Compliant Code** |
| --- |
| ustom\_new\_handler() uses the return value from reclaim\_resources(). If it returns 0, then there will be insufficient memory for operator new to succeed. |
| #include <new>  void custom\_new\_handler() noexcept(false) {  // Returns number of bytes freed.  extern std::size\_t reclaim\_resources();  if (0 == reclaim\_resources()) {  throw std::bad\_alloc();  }  }    int main() {  std::set\_new\_handler(custom\_new\_handler);    // ...  } |

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

| **Principles(s):** Policy 10: Adopt a Secure Coding Standard, Mapping: Having a handler standard can help with guidance for replacement handler. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | PROBABLE | HIGH | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2025.1 | DF4776, DF4777, DF4778, DF4779 | - |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-OOP56-a  CERT\_CPP-OOP56-b  CERT\_CPP-OOP56-c | Properly define terminate handlers Properly define unexpected handlers Properly define new handlers |
| Polyspace Bug Finder | R2024b | CERT C++: OOP56-CPP | Checks for replacement handler function that does not meet requirements (rule fully covered) |

Link: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP56-CPP.+Honor+replacement+handler+requirements>

### 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.

Creating guidance of how to integrate automate enforcement into the current policy we can take a look at the above diagram that has been converted into DevSecOps. The pre-production starts with an assess and plan phase that can create a landscape of threats. This phase can help design and build the system with security being a priority from the start using standard security guidance. As pre-production is coming to an end it is being tested and verified for any vulnerabilities.

Once pre-production is completed the production phases start. The production starts with configuring a security setting that can help alert any event and intrusion detection. Networks will be monitored and block any attacks to then stabilize and maintain security baselines. Once an attack is dealt with the system is brought back to pre-production to assess and plan new features to deal with such attacks.

### 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-005-CPP | HIGH | LIKELY | MEDIUM | P18 (HIGH) | 5 |
| STD-004-CPP | HIGH | LIKELY | MEDIUM | P18 (HIGH) | 5 |
| STD-008-CPP | HIGH | LIKELY | HIGH | P9 (HIGH) | 4 |
| STD-001-CPP | HIGH | PROBABLE | MEDIUM | P12 (HIGH) | 4 |
| STD-003-CPP | LOW | LIKELY | MEDIUM | P4 (MEDIUM) | 3 |
| STD-009-CPP | MEDIUM | UNLIKELY | MEDIUM | P4 (MEDIUM) | 3 |
| STD-006-CPP | MEDIUM | UNLIKELY | MEDIUM | P4 (LOW) | 3 |
| STD-007-CPP | LOW | PROBABLE | MEDIUM | P4 (LOW) | 2 |
| STD-010-CPP | LOW | PROBABLE | HIGH | P2 (LOW) | 2 |
| STD-002-CPP | LOW | UNLIKELY | MEDIUM | P2 (LOW) | 1 |

### 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 | Protects data stored in databases or backups by encrypting where it rests. It should be applied with key management systems for secure access control. The reason for its use it helps prevent unauthorize access if physical media is stolen, which also is compliance with the GDPR. |
| Encryption in flight | Secures data while it’s transiting across networks such as a server and a user. It should be applied with a secure network like HTTPS or secure APIs. The reason for use is that it helps ensure data confidentiality during transmission. |
| Encryption in use | Protects data while it is being actively processed in memory. It should be applied by using memory encryptions for system handling private data. The reason for use is that it helps protect data from insider threats like malware, and enables processing data without exposing it in plain text. |

Link: <https://nsysgroup.com/blog/encryption-for-the-three-states-of-data-best-practices/?utm_source=chatgpt.com>

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process of verifying the identity of a user or system before granting access. The authentication process works by systems using a username and password for accounts. The reason this policy applies is to prevent unauthorize users from accessing the system. |
| Authorization | This determines what actions and permission a user or system has after they have been authenticated. It applies by using the least privilege principle as users only get access to what they need. The reason it applies is because it helps prevent abuse of access and enforces segregation of duties like who approves what. |
| Accounting | These tracks and logs actions performed by a user or system. The application is used by network monitoring tools or application logs. The reason it applies is because it enables an incident response and helps detect insider threats. |

Link: <https://www.techtarget.com/searchsecurity/definition/authentication-authorization-and-accounting?utm_source=chatgpt.com>

**\***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 | 05/25/2025 | 3-2 Milestone | Anthony Miranda Aponte | [Insert text.] |
| 3.0 | 06/12/2025 | Project One | Anthony Miranda Aponte | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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