**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 | To validate inputs from untrusted data sources in order to eliminate software vulnerabilities. Any inputs from an untrusted source should be validated to prevent breaches. |
| 1. Heed Compiler Warnings | Eliminates compiler warnings and compiles code with the high warning level settings, as well as modifies the code when any warnings are received. This identifies and rids of security errors with the use of dynamic and static analysis tools. |
| 1. Architect and Design for Security Policies | Policies that are used to create software design and development that enforces security protection and policies. |
| 1. Keep It Simple | Keep the design and development simple to keep any errors to a minimum such as errors with implementation and/or configuration. |
| 1. Default Deny | A ruleset that denies incoming and outgoing traffic that is not permitted and that could be used to spread malware. |
| 1. Adhere to the Principle of Least Privilege | This is the lowest level of access to complete a mission set. This lowers any chances an attacker has to execute code with elevated privileges. This can limit access by anyone from the outside and any threats on the inside. Simply, this lowers any opportunities to cause damage. |
| 1. Sanitize Data Sent to Other Systems | Sanitizes data that is passed into subsystems such as shells and rational databases. |
| 1. Practice Defense in Depth | Create multi-level overlapping defensive strategies where if the first layer is breached or attacked, the other layers will counter the attack and prevent a security breach with any vulnerabilities. This simply manages risks with multiple layers of defensive. |
| 1. Use Effective Quality Assurance Techniques | Using and applying effective techniques that can be useful and efficient with identifying and destroying vulnerabilities such as code audits and penetration testing. |
| 1. Adopt a Secure Coding Standard | Applying a standard of coding that is secure to your platform and coding language. To create and adhere to a strict coding standard that is vital for developers to follow. |

### 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] | Do not cast to an enumeration value that is out of range. |

| **Noncompliant Code** |
| --- |
| This code example checks to see if the value is in range of the enum value. After the type is casted, it may not represent the given value of the integer. |
| enum EnumType {  First;  Second;  Third;  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  }  } |

| **Compliant Code** |
| --- |
| This example checks value of enum before conversion to make sure the output is not an error. |
| enum EnumType {  First;  Second;  Third;  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  }  EnumType enumVar = static\_cast<EnumType>(intVar); |

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

| **Principles(s):** Having values that are not specified can result in an overflow of the buffer. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | P4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS - Studio | 7.07 | V1016 | \_ |
| Helix QAC | 2021.1 | \_ | \_ |
| PRQA QA C++ | 2020.2 | CERT\_CPP – INT50-A | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration. |
| Axivion Bauhaus Suite | 6.9.0 | CertC++ - INT50 | \_ |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Define identifiers, pointers and references valid references properly to eliminate the chance of undefining identifiers. |

| **Noncompliant Code** |
| --- |
| In this code example, error due to the leading and ending underscores. |
| #ifndef \_ HEADER\_G\_  #define \_ HEADER\_G\_  // Contents of < header.g>  #endif // \_ HEADER\_G\_ |

| **Compliant Code** |
| --- |
| In this code example, the definitions are defined after removing the leading and ending underscore lines. |
| #ifndef HEADER\_G  #define HEADER\_G  // Contents of < header.g>  #endif // HEADER\_G |

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

| **Principles(s):**  Heed Compiler Warnings- Eliminates compiler warnings and compiles code with the high warning level settings.  Keep the code simple to aid in catching errors.  Follow Architect and Design for Security Policies to build code while preventing vulnerabilities  Use Effective Quality Assurance Techniques – Using and applying effective techniques that can be useful and efficient with identifying and destroying vulnerabilities such as code audits and penetration testing.  Adopt a secure coding standard – Applying a standard of coding that is secure to your platform and coding language. To create and adhere to a strict coding standard that is vital for developers to follow. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-INT50 | - |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | Reserved-identifier | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | Wreserved-id-macro  Wuser-defined-literals | -The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | If a reference type is qualified with const due to resulting in undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this code example, a char is created when there is a qualified const referenced. |
| #include <iostream>  void f(char m) {  char &const c = m;  c = 'c';  std::cout << m << std::endl;  } |

| **Compliant Code** |
| --- |
| In this code example , the qualifier cost was removed. |
| #include <iostream>  void f(char m) {  char &c = m;  c = 'c';  std::cout << m << std::endl; |

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

| **Principles(s):**  Heed Compiler Warnings- Eliminates compiler warnings and compiles code with the high warning level settings.  Keep the code simple to aid in catching errors.  Follow Architect and Design for Security Policies to build code while preventing vulnerabilities  Use Effective Quality Assurance Techniques – Using and applying effective techniques that can be useful and efficient with identifying and destroying vulnerabilities such as code audits and penetration testing.  Adopt a secure coding standard – Applying a standard of coding that is secure to your platform and coding language. To create and adhere to a strict coding standard that is vital for developers to follow. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | CERT\_CPP-DCL52-a | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-DCL52-a | Never qualify a reference type with a ‘const’ or ‘volatile’ |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 | - |
| Polyspace Bug Finder | R2022b | CERT C++: DCL52-CPP | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| SonarQube C/C++ Plugin | 4.10 | S3708 | - |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | 0014 | - |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE | - |
| Helix QAC | 2022.3 | C++0014 | - |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CLG] | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| In this code example, the string literal is undefined behavior. |
| char \*str = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| In this code example, a string literal specifies the initial values of characters and the size of the array. The string stored in str can be modified safely without issues. |
| char str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):**  Heed Compiler Warnings- Eliminates compiler warnings and compiles code with the high warning level settings.  Keep the code simple to aid in catching errors.  Follow Architect and Design for Security Policies to build code while preventing vulnerabilities  Use Effective Quality Assurance Techniques – Using and applying effective techniques that can be useful and efficient with identifying and destroying vulnerabilities such as code audits and penetration testing.  Adopt a secure coding standard – Applying a standard of coding that is secure to your platform and coding language. To create and adhere to a strict coding standard that is vital for developers to follow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | String-literal-modifiaction  Write-to-string-literal | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| Compass/ROSE |  |  | Can detect simple violations of this rule |
| Coverity | 2017.07 | PW | Deprecates conversion from a string literal to “char\*” |
| Helix QAC | 2022.1 | C0556, C0752, C0753, C0754  C++ 3063, C++3064, C++3605, C++3606, C++3607 |  |
| Klockwork | 2022.1 | CERT.STR.ARG.CONST-TO-NONCONST  CERT.STR.ASSIGN.CONST-TO-NOCONST |  |
| LDRA tool suite | 9.7.1 | 157S | Partially implemented |
| Parasoft  C/C++ test | 2021.2 | CERT\_C-STR30-a  CERT\_C-STR30-b | A string literal shall not be modified. Do not modify string literals |
| PC-lint Plus | 1.4 | 489, 1776 | Partially supported |
| Polyspace Bug Finder | R2021a | CERT C: Rule STR30-C | Checks for writing to const qualified object (rule fully covered) |
| PRQA QA-C | 9.7 | 0556, 0752, 0753, 0754 | Partially implemented |
| PRQA QA-C++ | 4.4 | 3063, 3064, 3605, 3606, 3607, 3842 |  |
| PVS-Studio | 7.18 | V675 |  |
| RuleChecker | 20.10 | String-literal-modification | Partially checked |
| Splint | 3.1.1 |  |  |
| TrustInSoft Analyzer | 1.38 | Mem-access | Exhaustively verified (see one compliant and one non-compliant example). |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| This code example show the local variable is being passed to the new operator which caused the pointer to be passed to operator delete(). This caused undefined behavior when attempting to free memory that was not originally returned by operator new(). |
| #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** |
| --- |
| The call to ::operator delete() is removed. |
| #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; |

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

| **Principles(s):** Undefined behavior is caused with a passing a pointer value. This leads to deallocation function that was not obtained originally by the matching allocation function. This can cause potential issues with exploitable vulnerabilities.  Follow Architect and Design for Security Policies to build code while preventing vulnerabilities  Use Effective Quality Assurance Techniques – Using and applying effective techniques that can be useful and efficient with identifying and destroying vulnerabilities such as code audits and penetration testing. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Invalid\_dynamic\_memory\_alocation\_dangling\_pointer\_use | \_\_\_\_\_\_ |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++MEM51 | \_\_\_\_\_\_ |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDelete.Leaks-Wminsmiatched-new-deleteclang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations |
| CodeSonar | 6.0p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM | Double free  Free non-heap variable  Type Mistmatch |
| Helix QAC | 2021.1 | \_\_\_\_\_\_ | \_\_\_\_\_\_ |
| Klocwork | 2021.1 | CL.FFM.ASSIGNFM  CL.FFM.COPY  CL.FMM  FMM.MIGHT  FMM.MUST  FNH.MIGHT  FNH.MUST  FUM.GEN.MIGHT  FUM.GEN.MUST  UNINIT.CTOR.MIGHT  UNINIT.CTOR.MUST  UNINIT.HEAP.MIGHT  UNINIT.HEAP.MUST | \_\_\_\_\_\_ |
| 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 |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Using the same form in corresponding calls to new and delete. Always provide empty brackets for when deallocating arrays. |
| Polyspace Bug Finder | R2020a | CERT C++: MEM51-CPP | Checks for invalid deletion and free of pointer, deallocation of previously deallocated pointer. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use static assertions to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| In this example, the assert() is being displayed and is used to assert which is not supposed to used. |
| #include <assert.h>  Struct time {  Unsigned char MODE;  Unsigned int DATA;  Unsigned int COUNT;  }  Int func(void) {  Assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| In this example, when doing a static assertion, the preprocessor conditional statement will be used for assertions that only contain constant expression. |
| #include <assert.h>  Struct time {  Unsigned char MODE;  Unsigned int DATA;  Unsigned int COUNT;  }  #if(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  #error “Structure must not have any padding  #endif |

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

| **Principles(s):** Static assertion lets users locate and find errors in code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | \_\_\_\_\_\_ | Users can create a customer check that uses the assert() function. |
| ÉCLAIR | 1.2 | CC2.DCL03 | \_\_\_\_\_\_\_ |
| Clang | 3.9 | Misc-static.assert | Checked by clang-tidy |
| Axivion Bauhaus Suite | 6.9.0 | CERTC-DCL03 | \_\_\_\_\_\_ |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handles all exceptions thrown before main () begins executing. |

| **Noncompliant Code** |
| --- |
| With this example, the constructor throws exception that is not caught when the new object is constructed. |
| struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| This makes sTest into a local variable with storage. This lets exceptions that are thrown to be seen and caught before execution. |
| struct S {  S() noexcept(false);  };    S &globalS() {  try {  static S s;  return s;  } catch (...) {  // Handle error, fix needed.  }  // Unreachable.  } |

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

| **Principles(s):** When an exception is thrown that is not caught, this results in abnormal program termination. This also leads to being vulnerable of attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Potentially-throwing-static-initialization | Patially Checked |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++-ERR58 | \_\_\_\_\_\_ |
| Clang | 3.9 | CERT-EER58-CPP | Checked by clang-tidy |
| PRQA QA-C++ | 4.4 | 4634, 4636, 4637, 4639 | \_\_\_\_\_\_ |
| Rule Checker | 20.10 | Potentially-throwing-static-initialization | Partially Checked |

#### Coding Standard 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| | **Coding Standard** | **Label** | **Name of Standard** | | --- | --- | --- | | Output Input | [STD-008-CPP] | Have a positioning call when alternating output and input from a file stream |   **Noncompliant Code** |
| This example does not include the proper positioning call between the input and output calls and because of this, an error is created. |
| #include <fstream>  #include <string>  void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output data";  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| In this example, the undefined behavior is eliminated because of the seekg() fuction that is called between input and output. |
| #include <fstream>  #include <string>  void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  file << "Output data";  std::string str;  file.seekg(0, std::ios::beg);  file >> str;  } |

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

| **Principles(s):** A undefined behavior will be created if input and output without the proper positioning call.  Follow Architect and Design for Security Policies to build code while preventing vulnerabilities  Use Effective Quality Assurance Techniques – Using and applying effective techniques that can be useful and efficient with identifying and destroying vulnerabilities such as code audits and penetration testing. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2020a | ECRT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call  Rule fully covered |
| Helix QAC | 2021.1 | - | - |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP\_FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Namespace modifications | [STD-009-CPP] | Do not modify standard namespaces. New declarations in the namespace can cause undefined behavior. |

| **Noncompliant Code** |
| --- |
| This example caused an undefined behavior. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| After changing to non-standard, this caused a placement without reserved name. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):**  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | CertC++: DCL58-CPP | Checks for modification of standard namespaces  Rule fully covered |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | CertC++-DCL58 | - |
| [Parasoft C/C++ test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | CERT\_CPP-DCL58-a | Do not modify the standard namespaces 'std' and 'posix |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | 4032, 4035, 4631 | - |

#### Coding Standard 10

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

| **Noncompliant Code** |
| --- |
| In this example, process item throws exception because pst isn’t released correctly. This results in a leak. |
| #include <new>    struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };    void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }    try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  throw;  }  delete pst;  } |

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

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

| **Principles(s):**  Sanitize Data Sent to Other Systems- Sanitizes data that is passed into subsystems such as shells and rational databases.  Adhere to the Principle of Least Privilege-This is the lowest level of access to complete a mission set. This lowers any chances an attacker has to execute code with elevated privileges. This can limit access by anyone from the outside and any threats on the inside. Simply, this lowers any opportunities to cause damage.  Architect and Design for Security Policies-Policies that are used to create software design and development that enforces security protection and policies. |
| --- |

**Threat Level**

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

**Automation**

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

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

This automation structure will automate our DecOps process. It will incorporate unit testing throughout the software and will be integrated within stage of the process. It will start from auditing the authorization levels of employees to determine what adjustments will be needed. It will also utlizie the mitigation tools to aid in creating secure code to be delivered to customers. This process will continue with automating modifications after the release of products which will allow for new versions to be pushed.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CLG | High | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CLG | Low | Unlikely | Low | P3 | L3 |
| STD-004-CLG | Low | Likely | Low | P9 | L2 |
| STD-005-CLG | High | Likely | Medium | P18 | L1 |
| STD-006-CLG | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | High | Likely | High | P9 | L2 |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-010-CPP | Low | Probable | High | P2 | L3 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | This means that data will be protected while it is being stored. An encryption key protects data when it is being written into storage and protects it from access that is not authorized. |
| Encryption at flight | This encrypts and protects data when it is being moved. For example, if data is put on a hard drive to be stored, becomes encrypted while being transmitted for protection. This simply adds an additional layer of protection to data. |
| Encryption in use | This means that data will be protected while is being used, accessed or created. Encryption in use is important to protect memory from being hacked or maliciously attacked. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication provides a method for identifying users with their own unique login credentials in order to access a network. |
| Authorization | Authorization gives users permission to access data or make changes. This is similar to the principle of least privilege which gives authorized users enough access to data to complete a task. This simply makes data more secure and protected. |
| Accounting | Accounting monitors data usage in order see trends that are occurring within the data. This helps with generating statistics. Accounting is used to determine the level of access a user needs and the authorization levels. Accounting also helps with finding possible mischief of data. |

**\***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**

* 4- to keep the design and development simple to help eliminate errors
* 5- denies access that is not permitted.
* 6- the lowest level of access necessary to complete the mission set
* 8- Create multi-level overlapping defensive strategies
* 10- develop and apply a secure coding standard. Adhere to the secure coding standard

**Firewall logs**

* 4- to keep the design and development simple to help eliminate errors
* 5- denies access that is not permitted.
* 6- the lowest level of access necessary to complete the mission set
* 7- sanitize data that is passed into subsystems
* 8- Create multi-level overlapping defensive strategies
* 10- develop and apply a secure coding standard. Adhere to the secure coding standard

**Anti-malware logs**

* 4- to keep the design and development simple to help eliminate errors
* 5- denies access that is not permitted.
* 6- the lowest level of access necessary to complete the mission set
* 7- sanitize data that is passed into subsystems
* 8- Create multi-level overlapping defensive strategies
* 10- develop and apply a secure coding standard. Adhere to the secure coding standard

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 |  |
| [Insert text.] | 11/20/2022 | Project One | Sarah Brady | [Insert text.] |
| [Insert text.] | 12/10/2022 | Project One Final Revision | Sarah Brady | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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