**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 ensure the system accept the expected format and type such as string, integer, float, or Boolean. |
| 1. Heed Compiler Warnings | To alert any potential issues and error for detecting any code’s inconsistency, so the final code can be improved and optimized. |
| 1. Architect and Design for Security Policies | To dictate any rules, practices and procedures in order to integrate proper security policies into the architecture and design phase before development stage. |
| 1. Keep It Simple | To reduced complexity in the design phase and improved maintainability towards the end of the development. |
| 1. Default Deny | To distinguish the privilege of authorized access and unauthorized access to minimized the risk of potential security breach |
| 1. Adhere to the Principle of Least Privilege | To restrict user’s access right due to not having enough privilege and credentials right. |
| 1. Sanitize Data Sent to Other Systems | Similar to quality assurance, this method involve whitelist sanitizing for allowing only valid character, blacklist sanitizing for cleaning unwelcomed characters, and escape sanitizing to reject invalid data request |
| 1. Practice Defense in Depth | To implement multiple layers of security for protecting core assets. |
| 1. Use Effective Quality Assurance Techniques | To detect system failure early throughout the development cycle and to ensure the software is bug free for performance consistency. |
| 1. Adopt a Secure Coding Standard | To avoid vulnerability and reduce the risk of exploitation |

### 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** | INT50-CPP | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two's complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in unspecified behavior |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator value. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Validate Input Data – unspecified values may result in buffer overflow, leading to the execution of arbitrary code by an attacker |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | certC++-INT50 |  |
| CodeSonar | 7.4p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2023.3 | C++3013 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT32-C | Ensure that operations on signed integers do not result in overflow |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b: |
| void func(signed int si\_a, signed int si\_b) {    signed int sum = si\_a + si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution ensures that the addition operation cannot overflow, regardless of representation: |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {    signed int sum;    if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||        ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {      /\* Handle error \*/    } else {      sum = si\_a + si\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** 10. Adopt a Secure Coding Standard – correct implementation avoid buffer overflow and the execution of arbitrary code by an attacker |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Integer-overflow | Fully checked |
| CodeSonar | 7.4p0 | 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 |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Helix QAC | 2023.3 | C2800, C2860  C++2800, C++2860  DF2801, DF2802, DF2803, DF2861, DF2862, DF2863 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR38-C | Do not confuse narrow and wide character strings and functions |

| **Noncompliant Code** |
| --- |
| incorrectly uses the strncpy() function in an attempt to copy up to 10 wide characters,  resulting in the truncation of the wide string.  incorrectly invokes the wcsncpy() function to copy up to 10 wide characters from narrow\_str1 to narrow\_str2. Because narrow\_str2 is a narrow string, it has insufficient memory to store the result of the copy and the copy will result in a buffer overflow. |
| #include <stddef.h>  #include <string.h>    void func(void) {    wchar\_t wide\_str1[]  = L"0123456789";    wchar\_t wide\_str2[] =  L"0000000000";      strncpy(wide\_str2, wide\_str1, 10);  }  #include <wchar.h>    void func(void) {    char narrow\_str1[] = "01234567890123456789";    char narrow\_str2[] = "0000000000";      wcsncpy(narrow\_str2, narrow\_str1, 10);  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the proper-width functions. Using wcsncpy() for wide character strings and strncpy() for narrow character strings ensures that data is not truncated and buffer overflow does not occur. |
| #include <string.h>  #include <wchar.h>    void func(void) {    wchar\_t wide\_str1[] = L"0123456789";    wchar\_t wide\_str2[] = L"0000000000";    /\* Use of proper-width function \*/    wcsncpy(wide\_str2, wide\_str1, 10);      char narrow\_str1[] = "0123456789";    char narrow\_str2[] = "0000000000";    /\* Use of proper-width function \*/    strncpy(narrow\_str2, narrow\_str1, 10);  } |

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

| **Principles(s):** Adopt Secure Coding Practice – correct implementation for narrow and wide string char prevent buffer overflow, data truncation, and other defects |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | P27 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | wide-narrow-string-cast wide-narrow-string-cast-implicit | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR38 | Fully implemented |
| Clang | 3.9 | -Wincompatible-pointer-types |  |
| CodeSonar | 7.4p0 | LANG.MEM.BO LANG.MEM.TBA | Buffer Overrun Tainted Buffer Access |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | FIO34-C | Distinguish between characters read from a file and EOF or WEOF |

| **Noncompliant Code** |
| --- |
| Although EOF is guaranteed to be negative and distinct from the value of any unsigned character, it is not guaranteed to be different from any such value when converted to an int. Consequently, when int has the same width as char, this loop may terminate prematurely. |
| #include <stdio.h>    void func(void) {    int c;      do {      c = getchar();    } while (c != EOF);  } |

| **Compliant Code** |
| --- |
| This compliant solution uses feof() and ferror() to test whether the EOF was an actual character or a real EOF because of end-of-file or errors: |
| #include <stdio.h>    void func(void) {    int c;      do {      c = getchar();    } while (c != EOF || (!feof(stdin) && !ferror(stdin)));  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems – improper use of EOF and WEOF will be resulted in significant vulnerabilities, potentially being targeted from injection attacks |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO34 |  |
| CodeSonar | 7.4p0 | LANG.CAST.COERCE | Coercion alters value |
| Helix QAC | 2017.07 | C2676, C2678  C++2676, C++2678, C++3001, C++3010, C++3051, C++3137, C++3717 |  |
| Klocwork | 1.2 | CWARN.CMPCHR.EOF |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | ERR57-CPP | Do not leak resources when handling exceptions |

| **Noncompliant Code** |
| --- |
| pst is not properly released when process\_item throws an exception, causing a resource 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** |
| --- |
| the exception handler frees pst by calling delete. |
| #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.      delete pst;      throw;    }    delete pst;  } |

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

| **Principles(s): 9.** Use Effective Quality Assurance Techniques – memory and resource leaks will cause the system to crash eventually. Proper implementation also prevent DDoS attack. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Code Sonar | 7.4p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2023.3 | DF4756, DF4757, DF4758 |  |
| Klocwork | 2023.3 | CL.MLK MLK.MIGHT MLK.MUST MLK.RET.MIGHT MLK.RET.MUST RH.LEAK |  |
| LDRA tool suite | 9.7.1 | 50 D | Partially implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | EXP34-C | Do not dereference null pointers |

| **Noncompliant Code** |
| --- |
| The sk pointer is initialized to tun->sk before checking if tun is a null pointer. Because null pointer dereferencing is [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior), the compiler (GCC in this case) can optimize away the if (!tun) check because it is performed after tun->sk is accessed, implying that tun is non-null. As a result, this noncompliant code example is vulnerable to a null pointer dereference exploit |
| static unsigned int tun\_chr\_poll(struct file \*file, poll\_table \*wait)  {    struct tun\_file \*tfile = file->private\_data;    struct tun\_struct \*tun = \_\_tun\_get(tfile);    struct sock \*sk = tun->sk;    unsigned int mask = 0;      if (!tun)      return POLLERR;      DBG(KERN\_INFO "%s: tun\_chr\_poll\n", tun->dev->name);      poll\_wait(file, &tun->socket.wait, wait);      if (!skb\_queue\_empty(&tun->readq))      mask |= POLLIN | POLLRDNORM;      if (sock\_writeable(sk) ||       (!test\_and\_set\_bit(SOCK\_ASYNC\_NOSPACE, &sk->sk\_socket->flags) &&       sock\_writeable(sk)))      mask |= POLLOUT | POLLWRNORM;      if (tun->dev->reg\_state != NETREG\_REGISTERED)      mask = POLLERR;      tun\_put(tun);    return mask;  } |

| **Compliant Code** |
| --- |
| This compliant solution eliminates the null pointer deference by initializing sk to tun->sk following the null pointer check. It also adds assertions to document that certain other pointers must not be null. |
| static unsigned int tun\_chr\_poll(struct file \*file, poll\_table \*wait)  {    assert(file);    struct tun\_file \*tfile = file->private\_data;    struct tun\_struct \*tun = \_\_tun\_get(tfile);    struct sock \*sk;    unsigned int mask = 0;      if (!tun)      return POLLERR;    assert(tun->dev);    sk = tun->sk;    assert(sk);    assert(sk->socket);    /\* The remaining code is omitted because it is unchanged... \*/  } |

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

| **Principles(s):** **9**.Use Effective Quality Assurance Techniques – dereferencing null pointer is undefined behavior may cause program termination or lead to execution of arbitrary code. Using proper assert function to pointers help eliminating incorrect null pointer dereference. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | [Insert text.] | [Insert text.] | Fully Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-EXP34 |  |
| CodeSonar | 7.4p0 | LANG.MEM.NPD  LANG.STRUCT.NTAD LANG.STRUCT.UPD | Null pointer dereference Null test after dereference Unchecked parameter dereference |
| Helix QAC | 2023.3 | DF2810, DF2811, DF2812, DF2813 | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-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):** 2. Heed Compiler Warnings – to catch uncaught exception for early security vulnerability which could be exploited by external threat |
| --- |

**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 |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 | [Insert text.] |
| CodeSonar | 7.4p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2023.3 | C++4035, C++4036, C++4037 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Orientation Programming | OOP53-CPP | Write constructor member initializers in the canonical order |

| **Noncompliant Code** |
| --- |
| declaration order of the member variables does not match the member initializer order, attempting to read the value of someVal results in an unspecified value being stored into dependsOnSomeVal. |
| class C {    int dependsOnSomeVal;    int someVal;    public:    C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| changes the declaration order of the class member variables so that the dependency can be ordered properly in the constructor's member initializer list. |
| class C {    int someVal;    int dependsOnSomeVal;    public:    C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):** Keep it simple – correct implementation such as naming convention, member initialize order reduce code complexity. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Initializer-list-order | Fully Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ -OOP53 |  |
| Clang | 3.9 | -Wreorder |  |
| CodeSonar | 7.4p0 | LANG.STRUCT.INIT.OOMI | Out of Order Member Initializers |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declaration | DCL60-CPP | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| two different translation units define a class of the same name with differing definitions violates the ODR and results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| // a.cpp  struct S {    int a;  };    // b.cpp  class S {  public:    int a;  }; |

| **Compliant Code** |
| --- |
| the solution is to use a header file to introduce the object into both translation units, as shown in this compliant solution. |
| // S.h  struct S {    int a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):** Architect and Design for Security Policies – implement one definition rule to prevent violation of undefined behavior as well as potentially DDoS attack if hacker can access the system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading | Partially Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 |  |
| CodeSonar | 7.4p0 | LANG.STRUCT.DEF.FDH LANG.STRUCT.DEF.ODH | Function defined in header file Object defined in header file |
| Helix QAC | 2023.3 | C++1067, C++1509, C++1510 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| File Input/Ouput | FIO51-CPP | Close file when they are no longer needed |

| **Noncompliant Code** |
| --- |
| The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):** Architect and Design for Security Policies – properly close file after reduce the risk of abnormal program termination and prevent attacker leech system resources. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2023.3 | DF4786, DF4787, DF4788 |  |
| Klocwork | 2023.3 | RH.LEAK |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |

### 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.

In the pre-production phase, we start to assess and plan security test driven design at an early stage of the development cycle to ensure the final build is more securable. By following OWASP best practices, we can integrate proper security tools in our development pipeline to check for any issues or vulnerability at the testing stage; in order to have a smooth transition at the deployment stage.

Since the security test has already produced at the design stage, we can monitor and detect any abnormal behavior towards the end of deployment, so we can respond to the attacks accordingly. Bt using correct coding standards implementation, we can reduce risk of security breach as well as data breach if the system is compromised.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| INT50-CPP | High | Unlikely | Medium | P4 | L3 |
| INT32-C | High | Likely | High | P9 | L2 |
| STR38-C | High | Likely | Low | P27 | L1 |
| FIO34-C | High | Probable | Medium | P12 | L1 |
| ERR57-CPP | Low | Probable | High | P2 | L3 |
| EXP34-C | High | Likely | Medium | P18 | L1 |
| ERR51-CPP | Low | Probable | Medium | P4 | L3 |
| OOP53-CPP | Medium | Unlikely | Medium | P4 | L3 |
| DCL60-CPP | High | Unlikely | High | P3 | L3 |
| FIO51-CPP | Medium | Unlikely | Medium | P4 | L3 |

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 | Data stored in the database behind firewall. Using AES(Advanced Encryption Standard) to encrypt data at rest to verify the key between sender and receiver so data can be readable |
| Encryption at flight | Data transmitted over the network. Using TLS (Transport Layer Security) or SSL (Secure Sockets Layers) to encrypt the communication between client and server, and prevent man in the middle attacks |
| Encryption in use | Data used in application. Using symmetric encryption such as AES, DES, or Triple DES or asymmetric encryption such as RSA to protect sensitive data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | To confirm the user’s identity for system access such as user login |
| Authorization | To ensure user have right and privilege to access specific resources of the system such as file access permissions. |
| Accounting | To track user’s activities accordingly in order to prevent any odd behavior in the future |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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