**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

# Amber Caldwell

## 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 | In essence, it’s best to assume all input data is malicious. Therefore, input should be validated from any source that isn’t trusted. Methods for validation can include a data range, expected type, and even expected length. |
| 1. Heed Compiler Warnings | Seacord and Schiela suggest, “Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws.” (Seacord, R., & Schiela, R., 2018) |
| 1. Architect and Design for Security Policies | Security policies help keep a system secure. Sometimes, a person or group may need more access in one application or area of a system than in another. Therefore, the more those areas can be segmented, and security can be divided into intercommunicating parts with their proper access groups, the more secure the system will be. |
| 1. Keep It Simple | The name really describes the principle. The simpler the security system in place the less likely for errors to occur. When dealing with overly complex systems there can be several failure points. |
| 1. Default Deny | Seacord and Schiela state, “Base access decisions on permission rather than exclusion.” (Seacord, R., & Schiela, R., 2018) A conditional access policy is a great example of this being used. One has the option base access on permission or exclusion. However, permission is usually the preferrable method. |
| 1. Adhere to the Principle of Least Privilege | This can best be explained simply by stating that no one should have more access than is required to perform their job. Additional access should not be granted just because it can be. This leaves a system more vulnerable. |
| 1. Sanitize Data Sent to Other Systems | Injection attacks are a way that hackers can make your program vulnerable. Eshkenazi explains, “To fix this, I can create a ValidateOutput class that will normalize the output and sanitize it using a whitelist…you can use a different whitelisting pattern depending on your output fields.” (Eshkenazi, 2020) |
| 1. Practice Defense in Depth | Use a multiple layer security defense strategy. A simple single layer security defense leaves your code or system vulnerable. However, when you add multiple layers that can cover the code or system from multiple angles and in multiple layers, then if one layer fails, you’re still protected. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance is maintaining a desired level of quality. Therefore, it’s important to always be testing, reviewing, and auditing to maintain the level you’re looking for. This could be through code reviews, penetration testing, or even having external reviewers audit for flaws. |
| 1. Adopt a Secure Coding Standard | The name of this principle really explains the principle itself. You want to adopt or create a standard that meets your needs and goals. This standard should be in the language that you’re using and on the platform that you use. |

*\*\* References on page 28*

### 

### 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** | DCL-52-CPP | C++ doesn’t allow you to change the value of a reference type, treating all references as being const qualified. You should not reference const for something that is already a qualifying. |

| **Noncompliant Code** |
| --- |
| This example declares p to be a reference to a const-qualified char. Then the modification of p makes the program incorrectly formed. |
| void f(char c) {  const char &p = c;  p = ‘p’; // Error: read-only variable is not assignable  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This removes the const qualifier. |
| void f(char c) {  char &p = c;  p = ‘p’;  std::cout << c << std::endl;  } |

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

| **Principles(s):** The principle of validate input data applies. When you validate the input data you are validate type. Therefore, this eliminates this issue and complies with this standard. One should also heed compiler warnings. However, if one did forget to validate the input data and did not pay attention to a compiler warning, then using effective quality assurance techniques would be sure to catch this issue. Overall, the goal should be to apply the principle of adopting a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |
| Polyspace Bug Finder | R2021b | CERT C++: DCL52-CPP | Checks for: const-qualified reference types, Modification of const-qualified reference types; Rule fully covered |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-CDL52 | N/A |
| Helix QAC | 2021.2 | C++0014 | N/A |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT-33-C | Make sure that division operations do not divide by zero. Basic math teaches us that dividing by zero you would receive an “undefined”. |

| **Noncompliant Code** |
| --- |
| Code prevents signed integer overflow, but fails to prevent a divide-by-zero error. |
| void func(signed long s\_a, signed long s\_b) {  signed long result;  if ((s\_a == LONG\_MIN) && (s\_b == -1)) {  /\* Handle error \*/  } else {  result = s\_a / s\_b;  }  /\* … \*/  } |

| **Compliant Code** |
| --- |
| This tests the division for no possibility of a divide-by-zero error. |
| void func(signed long s\_a, signed long s\_b) {  signed long result;  if ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {  /\* Handle error \*/  } else {  result = s\_a / s\_b;  }  /\* … \*/  } |

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

| **Principles(s):** The principle of validate input data applies. If you validate that one cannot divide by zero, then this issue can be mitigated. Therefore, this eliminates this issue and complies with this standard. The principle of heed compiler warnings and use effective quality assurance techniques would also apply. These would catch any missed validations. Overall, the goal should be to apply the principle of adopting a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Int-division-by-zero  Int-modulo-by-zero | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT33 | N/A |
| CodeSonar | 6.3p0 | LANG.ARITH.DIVZERO  LANG.ARITH.FDIVZERO | Division by zero  Float Divison By Zero |
| Coverity | 2017.07 | DIVIDE\_BY\_ZERO | Fully Implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR-50-CPP | Make sure that the null terminator is accounted for when copying data to a buffer. This will prevent buffer overflow. |

| **Noncompliant Code** |
| --- |
| The input is unbounded, which may lead to a buffer overflow. |
| void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The data is not truncated by using std::string. |
| void f() {  std::string input;  std:: string stringOne, stringTwo;  std:: cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** Make sure to validate any input data. Additionally, the principle of heed compiler warnings and the use effective quality assurance techniques would apply. These should catch any buffer overflow issues. Overall, the goal should be to apply the principle of adopting a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the ‘char’ buffer to store input from ‘std::cin’ |
| SonarQube C/C++ Plugin | 4.10 | S3519 | N/A |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | FIO-30-C | Do not call a formatted input / output function which may have untrusted data. This can lead to vulnerabilities and attacks. |

| **Noncompliant Code** |
| --- |
| This function is called for authentication and displays and error message if the data is incorrect. The function accepts a string. The function calculates the size of the message, allocates storage, constructs the message in the memory using the snprintf() function. Operations are not checked for overflow. The untrusted user message is passed as a string argument to fprintf(). |
| void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fprintf(stderr, msg);  free(msg);  } |

| **Compliant Code** |
| --- |
| This replaces fprintf() with fputs(), and outputs a direct message without analyzing its contents. |
| void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fputs(msg, stderr);  free(msg);  } |

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

| **Principles(s):** There are several principles which could apply. Heed any compiler warnings which may catch this issue. Also, sanitize data sent to other systems. This prevents attackers from making multiple areas of a system vulnerable. If the SQL injection were successful, it’s important to be using the principles of least privilege and default deny protecting as much data as possible. Next, use defense in depth. For instance, if one is using a multi-layered security policy if one area fails, then likely another area will protect other areas of your system. Use effective quality assurance techniques. By testing vulnerabilities such as this, these issues can be prevented. Overall, the goal should be to apply the principle of adopting a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-FIO30** | Partially implemented |
| CodeSonar | 6.2p0 | **IO.INJ.FMT** **MISC.FMT** | Format string injection Format string |
| Coverity | 2017.07 | **TAINTED\_STRING** | Implemented |
| LDRA tool suite | 9.7.1 | **86 D** | Partially Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM-31-C | Use free() to for the deallocation of memory. Without deallocating memory, space quickly becomes unavailable resulting in errors. |

| **Noncompliant Code** |
| --- |
| The object allocated by malloc() is not freed before the end of the last pointer text\_buffer. |
| enum { BUFFER\_SIZE = 32 };  int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }  return 0;  } |

| **Compliant Code** |
| --- |
| The pointer is deallocated with free(). |
| enum { BUFFER\_SIZE = 32 };  int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }  free(text\_buffer);  return 0;  } |

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

| **Principles(s):** Heed compiler warnings and use effective quality assurance techniques. This combination should find any instances of memory leaks. Again, the goal should be to apply the principle of adopting a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-MEM31** | Can detect dynamically allocated resources that are not freed |
| CodeSonar | 6.2p0 | **ALLOC.LEAK** | Leak |
| Coverity | 2017.07 | **RESOURCE\_LEAK**  **ALLOC\_FREE\_MISMATCH** | Finds resource leaks from variables that go out of scope while owning a resource |
| Cppcheck | 1.66 | leakReturnValNotUsed | Doesn't use return value of memory allocation function |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCL-03-C | Use a static assertion for testing the value of a constant expression. Static remains constant by definition. |

| **Noncompliant Code** |
| --- |
| The use of the runtime assertion is a better assertion than no assertion at all, but it should be placed into a function. |
| struct timer {  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** |
| --- |
| Static assertions allow incorrect diagnoses at compile time instead of resulting in a runtime error. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  }  static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int), “Structure must not have any padding”); |

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

| **Principles(s):** Heed compiler warnings to catch instances of this issue. Also, use effective quality assurance techniques which may include cross referencing with a static analysis tool. The principle of adopting a secure coding standard for this issue should be the goal. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.2p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully Implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR-55-CPP | To stop abnormal termination of a program exception-specifications should be upheld. |

| **Noncompliant Code** |
| --- |
| It’s possible for std::vector::resize() to throw an exception if the memory cannot be allocated. |
| void f(std::vector<int> &v, size\_t s) noexcept(true) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| All exceptions are allowed. |
| void f(std::vector<int> &v, size\_t s) {  v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** While this isn’t a large issue overall heeding compiler warnings would help address the issue. By cross referencing an automation tool like Cppcheck, one is further guaranteed to catch the issue. By using effective quality assurance techniques this issue should be fixed altogether. Aiming to achieve a secure coding standard is the optimal outcome. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Unhandled-throw-noexcept | Partially checked |
| LDRA tool suite | 9.7.1 | 56 D | Partially checked |
| Polyspace Bug Finder | R2021b | CERT C++: ERR55-CPP | Checks for noexcept functions exiting with exception (rule fully covered) |
| RuleChecker | 20.10 | Unhandled-throw-noexcept | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Integer Conversion** | INT-36-C | Do not convert an integer to a pointer type and vice versa if the correct steps to convert are not in place. |

| **Noncompliant Code** |
| --- |
| The pointer size can be great than the integer. Therefore, the conversion cannot be represented. |
| void f(void) {  char \*ptr;  /\* ... \*/  unsigned int number = (unsigned int)ptr;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The C standard guarantees that a pointer to void may be converted to or from a pointer to any object type and back again and that the result must compare equal to the original pointer. The use of intptr\_t or uintptr\_t allows conversion without change in value. |
| void f(void) {  char \*ptr;  /\* ... \*/  uintptr\_t number = (uintptr\_t)ptr;  /\* ... \*/  } |

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

| **Principles(s):** Input Validation would apply to this issue. Furthermore, one should heed compiler warnings. Lastly, effective quality assurance techniques would eliminate the issue. While this is a probable issue, it should be easily fixed. Overall, the goal should be to apply the principle of adopting a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **pointer-integral-cast**  **pointer-integral-cast-implicit**  **function-pointer-integer-cast**  **function-pointer-integer-cast-implicit** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-INT36** | Fully implemented |
| Clang | 3.9 | -Wint-to-pointer-cast, -Wint-conversion | Can detect some instances of this rule, but does not detect all |
| Helix QAC | 2021.3 | **C0303, C0305, C0306, C0309, C0324, C0326, C0360, C0361, C0362**  **C++3040, C++3041, C++3042, C++3043, C++3044, C++3045, C++3046, C++3047, C++3048** | N/A |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Function Declaration** | MSC-53-CPP | If a function is declared as noreturn, do not return a value. This results in undefined behaviors and / or errors. |

| **Noncompliant Code** |
| --- |
| If zero is passed, then control will flow off of the end of the function. This results in undefined behavior. |
| [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  else if (i < 0)  std::exit(0);  } |

| **Compliant Code** |
| --- |
| If zero is passed, the function does not return on any code path. |
| [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  std::exit(0);  } |

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

| **Principles(s):** The principle of heed compiler warnings and use effective quality assurance techniques would apply. Using not only a standard compiler by an automation tool would likely catch an error like this. Additionally, by using effective testing techniques, this should not be an issue. Overall, the goal should be to apply the principle of adopting a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Invalid-noreturn | Fully checked |
| CodeSonar | 6.2p0 | LANG.STRUCT.RFNR | Return from noreturn |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MSC53-a | Never return from functions that should not return |
| RuleChecker | 20.10 | Invalid-noreturn | Fully checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **File Management** | FIO-51-CPP | File management is important. If a file is opened, then it must be closed before the lifetime of the last pointer or the termination of the program. |

| **Noncompliant Code** |
| --- |
| The file is not properly closed. There is a call to open the file, and no matching close call. The program terminates before the file can be closed properly, too. |
| void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

| **Compliant Code** |
| --- |
| The close call is called before the termination call. This ensures the file is closed correctly. |
| 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):** The principle of heed compiler warnings and use effective quality assurance techniques would apply. Using not only a standard compiler by an automation tool would likely catch an error like this. Additionally, by using effective testing techniques, this should not be an issue. Overall, the goal should be to apply the principle of adopting a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ | N/A | N/A | Runtime Detection |
| Polyspace Bug Finder | R2021b | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

### 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.

Lightweight directory access protocol (LDAP) queries information regarding users in a directory service like Microsoft Active Directory (AD). AD can be synced to the cloud in services such as Azure Active Directory (AAD). These queries are vulnerable to SQL injection attacks. It’s important to use a defense in depth strategy for security. This can include firewalls, anti-virus, encryption, and a triple-a framework approach. Data should be sanitized that is sent to other systems due to the vulnerability of injection attacks. Furthermore, administrators should be adhering to the principles of least privilege and default deny for further data protection. Additionally, any applications that are published and available to authenticated users should be thoroughly checked for areas where injection attacks may occur.

Current static application security testing (SAST) and dynamic application security testing (DAST) techniques cannot keep up with the pace of modern development. The goal of automation is to integrate a static analysis tool or mechanism into the development environment to alert us of issues as the code is being written. Additionally, any changes can be analyzed quickly so changes can be made quickly. It is recommended that the static analysis tool be put into pre-production in the verify and test phase. Once verified and tested, the newly secure application can be pushed into production for authenticated users.

### 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 |
| --- | --- | --- | --- | --- | --- |
| DCL-03-C | Low | Unlikely | High | P1 | L3 |
| DCL-52-CPP | Low | Unlikely | Low | P3 | L3 |
| ERR-55-CPP | Low | Likely | Low | P9 | L2 |
| FIO-30-C | High | Likely | Medium | P18 | L1 |
| FIO-51-CPP | Medium | Unlikely | Medium | P4 | L3 |
| INT-33-C | Low | Likely | Medium | P6 | L2 |
| INT-36-C | Low | Probable | High | P2 | L3 |
| MEM-31-C | Medium | Probable | Medium | P8 | L2 |
| MSC-53-CPP | Medium | Unlikely | Low | P2 | L3 |
| STR-50-CPP | High | Likely | Medium | P18 | L1 |

### 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 usually refers to stored data on backups, archives, or even cloud storage which is inactive. This data is going to be translated from readable form into encrypted form. Therefore, only authorized users with the correct key will have access. The policy applies because only users who can decrypt this data can see it, and this prevents unwanted breaches. |
| Encryption at flight | This is the process of encrypting data while it’s being transmitted. The best way to ensure data being transmitted on the web is safe is by using a layer of security during transport such as TLS or transport layer security. This will change the http to https. This can be achieved using either asymmetric or symmetric keys. This would apply to the policy because it would ensure that only authorized receivers can gain access to the sender’s sensitive information. |
| Encryption in use | Encryption in use can refer to data which is active. This data may be under constant change. However, this data is likely stored on a CPU, within files, or possibly even a user’s RAM. According to Das, “All sensitive data is encrypted, including all data fields in all applications, adhering to the AES-256 standards.” (Das) This applies to the policy as it does not allow unauthorized access. It also strengthens endpoints, such as workstations. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication can refer to user logins. This is a way of user identification and making sure that a user is who they are claiming to be. A user will likely have a username or email and a password. Authenticated user’s credentials will match a set of user’s credentials that are stored within the user database. Plain-text passwords should not be stored. However, OWASP clears the storing of salted cryptographic hashes of passwords. Additionally, password entry will be denied after a certain number of attempts. The policy should deny access to a user who cannot authenticate. |
| Authorization | New users will be issued an access level. This should adhere to the policy of least privilege and default deny will be in place. This means that the level of access the user is authorized to have is sufficient for their job to be done and no extra access. Access will be denied to those who are not authorized. This authorization would apply to services as well as files accessed by the users. This policy ensures that the data is protected. |
| Accounting | Accounting is a way of logging information. This would allow administrators to view logins, database changes, the addition of new users as well as the level of access, files being accessed, and any session information. For instance, an administrator may wish to see what application was accessed from which location at what time and from what IP address. If this is inconsistent from a user’s normal login behavior, then the account may have been compromised. This type of auditing is important to the policy, because any unwanted behavior or actions can be tracked and mitigated quickly. |

**\***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: 3, 4, 5, 6, 8, 10
* Firewall logs: 3, 4, 8, 10
* Anti-malware logs: 8, 10
* Data Type: 1, 2, 9, 10
* Data Value: 1, 2, 9, 10
* String Correctness: 1, 2, 9, 10
* SQL Injection: 2, 5, 6, 7, 8, 9, 10
* Memory Protection: 2, 9, 10
* Assertions: 2, 9, 10
* Exceptions: 2, 9, 10
* Integer Conversion: 1, 2, 9, 10
* Function Declaration: 2, 9, 10
* File Management: 2, 9, 10

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 | 02/13/2022 | Policy Upgrade | Amber Caldwell |  |
| [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 |

References:

Das, P. (2021, December 29). *Protect sensitive data: Data in use encryption: Tech brief*. Sotero. Retrieved February 13, 2022, from https://www.soterosoft.com/blog/data-in-use-encryption-data-in-motion-encryption/

Eshkeazi, S. (2020, July 30). *Secure coding done right*. SecureCoding. Retrieved January 24, 2022, from https://www.securecoding.com/blog/secure-coding-done-right/#:~:text=Sanitize%20Data%20Sent%20to%20Other%20Systems&text=To%20prevent%20attackers%20from%20taking,output%20in%20a%20given%20context.

Lo Faso, V. (2015, April 28). *How to protect data in flight*. Worldwide IT Training. Retrieved February 13, 2022, from https://www.globalknowledge.com/us-en/resources/resource-library/articles/how-to-protect-data-in-flight/#gref

Mylonas, L. (2018, November 27). *What is AAA Security? an introduction to authentication, Authorisation and Accounting*. Codebots. Retrieved February 6, 2022, from https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting

Seacord, R., & Schiela, R. (2018, May 2). *Confluence*. Top 10 Secure Coding Practices - CERT Secure Coding - Confluence. Retrieved January 24, 2022, from <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices?focusedCommentId=88044413>

Vijayan, J. (2021, June 19). *Why automating your security testing is mission-critical*. TechBeacon. Retrieved February 13, 2022, from https://techbeacon.com/security/why-automating-your-security-testing-mission-critical