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



Green Pace Secure Development Policy

<https://resources.sei.cmu.edu/asset_files/BookChapter/2005_009_001_52692.pdf>

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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Project One 24](#_Toc52464070)

[1. Revise the C/C++ Standards 24](#_Toc52464071)

[2. Risk Assessment 24](#_Toc52464072)

[3. Automated Detection 24](#_Toc52464073)

[4. Automation 24](#_Toc52464074)

[5. Summary of Risk Assessments 25](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 25](#_Toc52464076)

[7. Map the Principles 26](#_Toc52464077)

[Audit Controls and Management 28](#_Toc52464078)

[Enforcement 28](#_Toc52464079)

[Exceptions Process 28](#_Toc52464080)

[Distribution 29](#_Toc52464081)

[Policy Change Control 29](#_Toc52464082)

[Policy Version History 29](#_Toc52464083)

[Appendix A Lookups 29](#_Toc52464084)

[Approved C/C++ Language Acronyms 29](#_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 daa 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 | Validate Input Data is a programming technique that ensures only properly formatted data may enter a software system component. An application should check that data is both syntactically and semantically valid (in that order) before using it in anyway (including displaying it back to the user).  In other words, the user can only put in data as designed by the programmer and the system can only display it back to the user as designed by the programmer. Input from untrusted sources must be validate. |
| 1. Heed Compiler Warnings | A compiler has warnings for a reason. So, a good programmer will go ahead and fix any warnings when they appear. It will help to eliminate all warnings as this helps to catch bugs before testing eliminating cost to fix problems. These warnings catch bugs that may be hard to find in testing. So, maintaining warning free code will help to pinpoint which code may be the problem if all of sudden warnings begin to appear. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set.  This will pay off in the future as it will reduce technical debt and risk mitigation. |
| 1. Keep It Simple | As complex designs can increase the likelihood that errors will be made in the implementation, configuration and use of the code, the design should be as simple and as small as possible. As security mechanisms become more complex, the effort required to achieve an appropriate level of assurance will increase dramatically. Keeping code simple and clean is an important step in the process. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted [Saltzer 74, Saltzer 75].  In other words, limit privileges and restrict access to secure data to only users who need it. Deny access to any user that cannot demonstrate authorization. Ensure that requests for sensitive are checked to verify that the user is authorized to access it. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges [Saltzer 74, Saltzer 75]. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems [[C STR02-A](https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems)] such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a [security flaw](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-securityflaw) from becoming an exploitable vulnerability and/or limit the consequences of a successful [exploit](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-exploit). For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment [Seacord 05].  When this is applied to software quality, DiD involves setting up a series of inspections designed to prevent or detect quality issues before code gets into production. At each step, there is a chance that the defect will be detected or prevented so that it will drastically reduce the likelihood that the defect will make it into production. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions [Seacord 05]. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

## 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 Value** | [STD-001-CPP] | Do not declare of define a reserved identifier |

| **Noncompliant Code** |
| --- |
| A common practice is to use a macro in a preprocessor conditional that guards against multiple inclusions of a header file. While this is a recommended practice, many programs use reserved names as the header guards. Such a name may clash with reserved names defined by the implementation of the C++ standard template library in its headers or with reserved names implicitly predefined by the compiler even when no C++ standard library header is included. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| This compliant solution avoids using leading or trailing underscores in the name of the header guard. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

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

|  |
| --- |
| **Principles(s): DCL51-CPP ~ There is a standard for reserved names that should be followed. It is IS0/IEC 14882-2014.** |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Reserver-identifier | Partially checked |
| Axivion Bauhaus Suiite | 7.2.0 | CertC ++-DCL51 |  |
| 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. |
| CodeSonar | 6.0p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not cast to an out of range enumeration value |

| **Noncompliant Code** |
| --- |
| Bounds Checking  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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-unspecifiedbehavior). |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| Bounds Checking  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):** **Int50-CPP** ~ It is possible for unspecified values to result in a buffer overflow, leading to the execution of arbitrary code by an attacker. However, because enumerators are rarely used for indexing into arrays or other forms of pointer arithmetic, it is more likely that this scenario will result in data integrity violations rather than arbitrary code execution. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-Int50 |  |
| Helix QAC | 2021.1 | **C++3013** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-INT50-a** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.13 | [**V1016**](https://pvs-studio.com/en/docs/warnings/v1016/) |  |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | [Guarantee that storage for strings has sufficient space for character data and the null terminator](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) |

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    void f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #include <iostream>  #include <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):** **STR50-CPP ~** Copying string data to a buffer that is too small to hold that data results in a buffer overflow. Attackers can [exploit](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-exploit) this condition to execute arbitrary code with the permissions of the vulnerable process. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | **C++2835, C++2836, C++2839, C++5216** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.1 | [**NNTS.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NNTS.TAINTED**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | [Do not use deprecated or obsolescent functions](https://wiki.sei.cmu.edu/confluence/display/c/MSC24-C.+Do+not+use+deprecated+or+obsolescent+functions) |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the obsolescent functions strcat() and strcpy() are used: |
| #include <string.h>  #include <stdio.h>    enum { BUFSIZE = 32 };  void complain(const char \*msg) {      static const char prefix[] = "Error: ";    static const char suffix[] = "\n";    char buf[BUFSIZE];      strcpy(buf, prefix);    strcat(buf, msg);    strcat(buf, suffix);    fputs(buf, stderr);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, strcat() and strcpy() are replaced by strcat\_s() and strcpy\_s(): |
| #define \_\_STDC\_WANT\_LIB\_EXT1\_\_  #include <string.h>  #include <stdio.h>    enum { BUFFERSIZE = 256 };    void complain(const **char** \*msg) {    static const **char** prefix[] = "Error: ";    static const **char** suffix[] = "\n";  **char** buf[BUFFERSIZE];      strcpy\_s(buf, BUFFERSIZE, prefix);    strcat\_s(buf, BUFFERSIZE, msg);    strcat\_s(buf, BUFFERSIZE, suffix);  **fputs**(buf, stderr);  } |

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

|  |
| --- |
| **Principles(s):** **MSC24-C**~ To remediate invocations of unchecked obsolescent functions, an application might use inline coding that, in all respects, conforms to this guideline, or an alternative library that, in all respects, conforms to this guideline, or alternative nonobsolescent functions from C11, |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **stdlib-use-ato**  **stdlib-macro-ato**  **stdlib-use-atoll**  **stdlib-macro-atoll** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MSC24** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **BADFUNC.\*  (customization)** | A number of CodeSonar's "Use of \*" checks are for deprecated/obsolescent functions CodeSonar also provides a mechanism for users to create custom checks for uses of specified functions |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.MSC34** | Fully implemented |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | [Free dynamically allocated memory when no longer needed](https://wiki.sei.cmu.edu/confluence/display/c/MEM31-C.+Free+dynamically+allocated+memory+when+no+longer+needed) |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the object allocated by the call to malloc() is not freed before the end of the lifetime of the last pointer text\_buffer referring to the object: |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    **int** f(void) {  **char** \*text\_buffer = (**char** \*)**malloc**(BUFFER\_SIZE);    if (text\_buffer == NULL) {      return -1;    }    return 0;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the pointer is deallocated with a call to free(): |
| #include <stdlib.h>    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):MEM31-C** Before the lifetime of the last pointer that stores the return value of a call to a standard memory allocation function has ended, it must be matched by a call to free() with that pointer value. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 |  | Supported, but no explicit checker |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM31** | Can detect dynamically allocated resources that are not freed |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **ALLOC.LEAK** | Leak |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |

### 

### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly: |
| #include <assert.h>    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** |
| --- |
| [Compliant description] For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution: |
| struct timer {  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):** **DCL03-C** ~ Static assertion is a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities at compile time. The absence of static assertions, however, does not mean that code is incorrect. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | [Do not leak resources when handling exceptions](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR57-CPP.+Do+not+leak+resources+when+handling+exceptions) |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, 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** |
| --- |
| [Compliant description] n this compliant solution, 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):** ERR57-CPP Memory and other resource leaks will eventually cause a program to crash. If an attacker can provoke repeated resource leaks by forcing an exception to be thrown through the submission of suitably crafted data, then the attacker can mount a [denial-of-service attack](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions). |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | **C++4756, C++4757, C++4758** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **50 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | **CERT\_CPP-ERR57-a** | Ensure resources are freed |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Ouput | [STD-008-CPP] | [CPP. Close files when they are no longer needed](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed) |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. 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** |
| --- |
| In this compliant solution, 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):** FIO51 ~ Failing to properly close files may allow an attacker to exhaust system resources and can increase the risk that data written into in-memory file buffers will not be flushed in the event of [abnormal program termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination). |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | **C++4786, C++4787, C++4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.1 | [**RH.LEAK**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-009-CPP] | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the two iterators that delimit the range point into the same container, but the first iterator does not precede the second. On each iteration of its internal loop, std::for\_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in undefined behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.end(), c.begin(), [](int i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the iterator values passed to std::for\_each() are passed in the proper order. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.begin(), c.end(), [](int i) { std::cout << i; });  } |

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

|  |
| --- |
| **Principles(s):** CTR53-53CPP ~ Using an invalid iterator range is similar to allowing a buffer overflow, which can lead to an attacker running arbitrary code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **overflow\_upon\_dereference** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | **C++3802** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | **CERT\_CPP-CTR53-a** **CERT\_CPP-CTR53-b** | Do not use an iterator range that isn't really a range Do not compare iterators from different containers |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **3802** |  |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-010-CPP] | [Ensure that operations on signed integers do not result in overflow](https://wiki.sei.cmu.edu/confluence/display/c/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):** INT32-C ~ Integer overflow can lead to buffer overflows and the execution of arbitrary code by an attacker. Invalidate input tha may contribute/cause overflow. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **integer-overflow** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_SCALAR**  **BAD\_SHIFT** | Implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.1 | **C2800, C2801, C2802, C2803, C2860, C2861, C2862, C2863**  **C++2800, C++2801, C++2802, C++2803, C++2860, C++2861, C++2862, C++2863** |  |

## 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.

Within the preproduction there are several phases with each phase addressing such things such as threat modeling, security tool training IDE security, static application testing, automated security scans, unit, integration and other tests.

Once things have moved into production, automated testing is being don and prevention continues by integrity checks and defense in depth measures. Some threat detection methods that can be used are as follows:

* Network monitoring
* Penetration testing
* Performance logs

When development of software, QA should be done often, so should testing in the security sense as well. Test often and test early!

## 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 | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Probable | Medium | P12 | L1 |
| STD-005-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | High | P2 | L3 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-009-CPP | High | Probable | High | P6 | L2 |
| STD-010-CPP | High | Likely | High | P9 | L2 |
|  |  |  |  |  |  |

## 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 | Protection of data where it is stored (on phone/computer, hard drive, cloud, database, etc.). There are many tools to help prevent attackers from getting to data and to prevent the data at rest from being breached as in being physically stolen, logically stolen or breached. Tools such as VeryaCrypt could be used. |
| Encryption at flight | Protects data as it is being moved from one location to the then next such as sending email or browsing the internet. There are several email encryption tools available that a company could use to encrypt email. While on the web, web traffic could be sent via SSL or TLS or the company could obtain certificates from certificate authorities. By using these tools, companies/organizations can keep the data safer and keep their customers safer as well as the data. |
| Encryption in use | Data encryption while being created, edited, processed or viewed. When the data is at rest or in flight, for instance, in the middle of accessing a website on the server (this is when the data is most vulenerable). Encryption of this data is a must and this is when the encryption key needs to be kept secret. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Validates the user’s identity as authentic before granting the login for the user. Verification of a user’s credentials. The policy can be achieved in several way:   * SSO * Username/password verification * Biometrics * Card Readers/Scanners   All users within the network, whether new or existing, shall be verified. Without being authenticated, identify theft and/or unauthorized system access can occur. |
| Authorization | Grants the user the privilege to access networks and commands. Authorization must be set for new and existing users to access files, directories and or/applications for each individual user. Each user or group shall be permitted to have read, write and/or execution permission, depending on the user’s role within an organization. For example, a data entry clerk may have read / write access to view or edit a particular file while a manager may warrant administrative permissions. |
| Accounting | Collects data to track usage patterns by individual user, service, host time of day, day of week, and so on. |

**\***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 |  |
| 1.1 | 06/21/2021 | First Revision | Karen Bevis | Karen Bevis |
| 1.2 | 06/27/2021 | Final Revision | Karen Bevis | Karen Bevis |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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