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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating user inputs in C/C++ is integral to the safety and health of any system because by excepting invalid input data entire systems could crash, or buffer overflows can be exploited by malicious users. |
| 1. Heed Compiler Warnings | The compiler will notice and alert developers of logical or syntactic inconsistencies in their code. Ignoring or silencing these warnings could lead to extreme disarray in complex systems, potentially exposing the system to malicious exploitation or accidental release of sensitive data. |
| 1. Architect and Design for Security Policies | By beginning all development and system design with a security policy in place, security becomes a foundational part of the application rather than an after thought that is carefully woven into the application after its development, potentially introducing new bugs or errors. |
| 1. Keep It Simple | Simplicity is important to any set of rules or regulations as simple instructions are more easily followed and accepted amongst teams. |
| 1. Default Deny | By default, denying access to sensitive information or systems is the best practice. The term “0 trust” comes to mind as denying access to data and resources and only incrementally allowing them to trusted user’s is the most effective way of securing data or systems for medium to large user bases. |
| 1. Adhere to the Principle of Least Privilege | By allowing users only the most minimum amount of access that is needed, sensitive data and systems are protected. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data sent to other systems ensures that data is not leaked accidentally either through logical or malicious code exploits. Incorporating checks on outgoing or returned data ensures that the data being sent or returned form a function matches the parameters put in place by developers. |
| 1. Practice Defense in Depth | By practicing defense in depth, organizations can effectively secure their systems and resources. Because defense is an ever growing and evolving practicing, it requires teams to continually monitor and update their defenses. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance techniques is one method of adhering to security rules and guidelines as these standards are already in place to protect consumers or other users of an application. |
| 1. Adopt a Secure Coding Standard | By adopting a secure coding practice applications are more difficult to exploit, sensitive information is more easily protected, and end users can feel more comfortable utilizing an application from developers that utilize secure coding standards. |

### 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-DCL12-C | Implement abstract data types using opaque types |

| **Noncompliant Code** |
| --- |
| Developers are more likely to directly manipulate size and maxSize of string\_mx violating the principles of information hiding. |
| **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** |
| --- |
| In the compliant example two header files are created to hide the implementation of the data type from the users of the managed string library. |
| **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);  /\* ... \*/  **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  }; |

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

| **Principles(s):** Architect & Design for Security Policies |
| --- |

**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) | R2022b | [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) | 2022.2 | **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-INT31-C] | Ensure that integer conversions do not result in lost or misinterpreted data |

| **Noncompliant Code** |
| --- |
| Type range errors like loss of data and loss of sign can occur when converting from a value of unsigned integer to signed integer. |
| #include <limits.h>    **void** func(**void**) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;    sc = (**signed** **char**)u\_a; /\* Cast eliminates warning \*/    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| By validating ranges before converting from unsigned to signed int prevents loss of data and loss of sign errors. |
| #include <limits.h>    **void** func(**void**) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;  **if** (u\_a <= SCHAR\_MAX) {      sc = (**signed** **char**)u\_a;  /\* Cast eliminates warning \*/    } **else** {      /\* Handle error \*/    }  } |

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

| **Principles(s):** Architect & Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Tool | Version | Checker | Description |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **LANG.CAST.PC.AV** **LANG.CAST.PC.CONST2PTR** **LANG.CAST.PC.INT**  **LANG.CAST.COERCE** **LANG.CAST.VALUE**  **ALLOC.SIZE.TRUNC** **MISC.MEM.SIZE.TRUNC**  **LANG.MEM.TBA** | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value  Truncation of allocation size Truncation of size  Tainted buffer access |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-STR30-C] | Do not attempt to modify string literals. |

| **Noncompliant Code** |
| --- |
| Modifying the string literal of a char pointer results in undefined behavior. |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| By initializing the char as an array, the initial values of the string literal are specified as well as the size of the array allowing for modifications to the literal. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** Architect & Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **string-literal-modfication** **write-to-string-literal** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR30** | Fully implemented |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect simple violations of this rule |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-STR02-C] | Sanitize data passed to complex subsystems (inclusion of special characters could have undefined behaviors in complex subsystems like SQL databases) |

| **Noncompliant Code** |
| --- |
| Special characters are allowed resulting in undefined behavior or malicious code injection to complex subsystems. |
| **sprintf**(buffer, "/bin/mail %s < /tmp/email", addr);  **system**(buffer); |

| **Compliant Code** |
| --- |
| Creating an array of valid characters allows the application to easily filter out special characters that may have undefined results on complex subsystems like SQL databases. |
| **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):** Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **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 |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-MEM11-C] | Do not assume infinite heap space. |

| **Noncompliant Code** |
| --- |
| Because no bound is placed on memory, an undetected memory leak could exhaust the entire heap leading to computer failure, crashes, or other undefined behavior. |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>    **enum** {MAX\_LENGTH=100};    **typedef** **struct** namelist\_s {  **char** name[MAX\_LENGTH];  **struct** namelist\_s\* next;  } \*namelist\_t;    **int** main() {    namelist\_t names = NULL;  **char** new\_name[MAX\_LENGTH];    **do** {      /\*       \* Adding unknown number of records to a list;       \* the user can enter as much data as he wants       \* and exhaust the heap.       \*/  **puts**("To quit, enter \"quit\"");  **puts**("Enter record:");  **fgets**(new\_name, MAX\_LENGTH, stdin);  **if** (**strcmp**(new\_name, "quit") != 0) {        /\*         \* Names continue to be added without bothering         \* about the size on the heap.         \*/        unsigned **int** i = **strlen**(new\_name) - 1;  **if** (new\_name[i] == '\n') new\_name[i] = '\0';        namelist\_t new\_entry = (namelist\_t) **malloc**(**sizeof**( **struct** namelist\_s));  **if** (new\_entry == NULL) {          /\* Handle error \*/        }  **strcpy**( new\_entry->name, new\_name);        new\_entry->next = names;        names = new\_entry;      }  **puts**(new\_name);    } **while** (**strcmp**( new\_name, "quit") != 0);    **return** 0;  } |

| **Compliant Code** |
| --- |
| If heap exhaustion is a potential after code refactoring, storing large amounts of data to the disk may be a more manageable solution for handling the large amounts of data. |
| Writing data to a simple database file could potentially resolves issues with memory exhaustion. |

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

| **Principles(s):** Architect & Design for Security Policies |
| --- |

**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/c/CodeSonar) | 7.2p0 | **ALLOC.LEAK** **IO.TAINT.SIZE** **MISC.MEM.SIZE.BAD** **(general)** | Leak Tainted allocation size Unreasonable size argument Library models account for allocator failure cases |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **26 S, 140 S, 6 D, 28 D, 5 C, 1 U** | Partially implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **413, 613** | Assistance provided: reports use of null pointers including those which could be returned when a call to an allocation function fails |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C: Rec. MEM11-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.mem11c.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-DCL03-C] | Use a static assertion to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| Because the assertion is not placed in a function, the assertion occurs only at runtime if its specific parameters are present. |
| #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** |
| --- |
| Preprocessor conditional statements allow for assertions during compilation and development, giving clear warnings before runtime errors. |
| **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):** Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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) | 7.2p0 | **(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 |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-ERR55-CPP] | Honor exception specifications |

| **Noncompliant Code** |
| --- |
| A function is declared as nonthrowing, but can still indeed through a buffer overthrow or other runtime error. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| Allows for exception to be thrown because programmers are already anticipating said exception. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++Test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR55-a** | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: ERR55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr55cpp.html) | Checks for noexcept functions exiting with exception (rule fully covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4035, 4036, 4632** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **unhandled-throw-noexcept** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-CRT53-CPP] | Use valid integer ranges. |

| **Noncompliant Code** |
| --- |
| Invalid logic in this code results in the foreach loop iterating past the range of the integer. |
| #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** |
| --- |
| By passing the iterators in the correct order overflow is avoided. |
| #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):** Validate User Input & Validate Input Being Sent To External Systems |
| --- |

**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) | 22.10 | **overflow\_upon\_dereference** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **LANG.MEM.BO** | Buffer Overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.4 | **C++3802** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **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 |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory | [STD-MEM50-CPP] | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| Because s is referenced after is memory has been deallocated, this can be exploited to access to the new code or data now being referenced at this pointer address. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| Dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s;  } |

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

| **Principles(s):** Architect & Design for Security Policies |
| --- |

**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 | **dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **ALLOC.UAF** | Use after free |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-EXP51-CPP] | Do not delete an array through a pointer of the incorrect type. |

| **Noncompliant Code** |
| --- |
| Deleting an array through a pointer of the incorrect type results in CRT56-CPP. Do not use pointer arithmetic on polymorphic objects. |
| **struct** Base {  **virtual** ~Base() = **default**;  };    **struct** Derived final : Base {};    **void** f() {     Base \*b = **new** Derived[10];     // ...  **delete** [] b;  } |

| **Compliant Code** |
| --- |
| The Derived array is now deleted from its own type Derived, rather than its inherited type Base from the previous example. |
| **struct** Base {  **virtual** ~Base() = **default**;  };    **struct** Derived final : Base {};    **void** f() {     Derived \*b = **new** Derived[10];     // ...  **delete** [] 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** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -analyzer-checker=cplusplus | Checked with clang -cc1 or (preferably) scan-build |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **ALLOC.TM** | Type Mismatch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.4 | **C++3166** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.4 | **CERT.EXPR.DELETE\_ARR.BASE\_PTR** |  |

### 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.

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-DCL12 | Low | Unlikely | High | P1 | L3 |
| STD-INT31-C | High | Probable | High | P6 | L2 |
| STD-STR30-C | Low | Likely | Low | P9 | L2 |
| STD-STR02-C | High | Likely | Medium | P18 | L1 |
| STD-MEM11-C | Low | Probable | High | P2 | L3 |
| STD-DCL03-C | Low | Unlikely | High | P1 | L3 |
| STD-ERR55-CPP | Low | Likely | Low | P9 | L2 |
| STD-CTR53-CPP | High | Probable | High | P6 | L2 |
| STD-MEM50-CPP | High | Likely | Medium | P18 | L1 |
| STD-EXP51-CPP | Low | Unlikely | Medium | P2 | L3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Data at rest should be encrypted if it contains any identifying or confidential information. Cipher algorithms should be robust. Specifically AES256 ciphers are optimal to protect data that is most confidential with AES128 being the minimum standard deemed acceptable. |
| Encryption at flight | Usage of plain text protocols like ftp, telnet, and http do not encrypt traffic during transit, for this reason and because more robust modern protocols are readily available, i.e https, SMB3.X, etc. older file transfer protocols should not be utilized. Threat actors capturing network traffic locally on company networks could gain access to confidential information by not encrypting traffic during transit. Additionally, recent WiFi standards should be taken into consideration, WiFi 6 has protections to protect packets that are broadcast to all local devices. Network structures should also be segregated over multiple VLANs to create a separation between resources. |
| Encryption in use | Remote locations and home users should only be accessing company data via a minimum of AES128 encrypted VPN tunnels. Newer VPN standards like WireGuard can provide robust ciphers with minimal performance impacts to connection speed and SMB over VPN compared to IPSEC. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the backbone of our company’s network and data security. Users should be authenticating with both robust passwords and some form of 2 factor authentication with an authentication TOTP app being the most optimal and secure as opposed to SMS or email. |
| Authorization | Authorization is an important standard to decide what network resources and data a specific user has access to. Our principle of least privilege is important to take into consideration when making justification for user authentication levels. Users should only be authenticated for the very specific resources that they need and nothing more. Any changes to company data should be authorized and after a brief timeout users will need to reauthenticate to continue accessing company resources. |
| Accounting | By monitoring active sessions strange behaviors and accesses can be identified and investigated before a malicious actor can gain access to valuable resources. Honeypots or thinkst canaries can be especially useful to gain information about attackers and to track the actions of already authenticated users. |

**\***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
* Static Code Analysis Tools
* Linters To Adhere To Coding Best Practices
* Remote Access Logs
* Intrusion Detection Software
* Database Authorization 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 | 01/29/2023 | Revisions to policies and examples | Hunter Ashner | Hunter Ashner |
| 1.2 | 2/12/2023 | Added summary ok risks. | Hunter Ashner | Hunter Ashner |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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