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



# Green Pace Secure Development Policy

# By: Nicholas Wyrwas

# CS-405-15275-M01

## 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 | When it comes to validating input, especially from all sources it is essential for preventing a lot of different types of software vulnerabilities and security breaches. For example, it is critical to verify data that is from any untrusted or external sources. Which may include but is not limited to command-line arguments, user-uploaded files, network interfaces, and even environment variables. Making sure input is properly validated helps guard against different types of injection attacks and system compromises. |
| 1. Heed Compiler Warnings | Heed compiler warnings are essentially built-in safeguards that would flag any potentially risky code. It is beneficial to pay attention to these warnings, combining them with static and dynamic analysis tools and manual code reviews. Which would help with identifying and fixing any security issues early on during a sprint or development cycle. |
| 1. Architect and Design for Security Policies | Any software should be architected from the ground up while keeping security in mind. Which would include implementing any layered permissions, enforcing access controls, as well as designing each component with the correct level of protection., Structuring a system in accordance to authentication tiers would help ensure that resources are only accessible to properly authorized users. |
| 1. Keep It Simple | Most security would benefit from simplicity. When a system is less complex, the fewer places a vulnerability could hide. When developing a system it should be designed to remain as straightforward as you can make it to improve maintainability and reduce risks. |
| 1. Default Deny | When having a security model it should essentially operate on a “deny by default” basis. Meaning that users are only able to gain access once they are authorized. Depending on the access they still may be limited to certain aspects, if there are restrictions for different roles. Which would allow tighter control over the systems resources. |
| 1. Adhere to the Principle of Least Privilege | When it comes to every user and process, they should have only the minimum access needed to perform their roles or functions. By limiting permissions you are able to reduce any potential damage of accidental or detrimental actions caused by users and processes. |
| 1. Sanitize Data Sent to Other Systems | You want to ensure you sanitize your data before you transmit any data to any external systems. Which may include command shells, databases, or even third party components. If sanitation does not occur it could impose detrimental security risks, especially if the systems you transmit data to misuse or mishandle your data. Sanitation ensures that any harmful input would not compromise interconnected systems or expose vulnerabilities. |
| 1. Practice Defense in Depth | This is the strategy of deploying multiple layered security measures throughout your system. Where each layer would focus on different aspects of protection. For example, it would ensure that if one of your controls fail, the others would still provide coverage. This would allow a safeguards against a wide range of threats and compensates a weakness in any of the single layers. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance practices would help identify vulnerabilities before they can be exploited. Where you can have penetration testing, source code analysis, and even unit testing integrated throughout your development cycle. Along with, having unbiased assessments from external reviewers that had no prior exposure to the system or project would help with reducing the chance of overlooked flaws. |
| 1. Adopt a Secure Coding Standard | When adopting a secure coding standard you are ensuring that consistency and safety in software development is a primary focus. Making it easier to evaluate the code base for security compliance and maintaining a high level of quality across current and future projects. Allowing you or a company to promote reliable, maintainable, and secure software systems. |

### 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** | DCL51-CPP | Reserved identifiers are identifiers used by the compiler and the standard libraries. Any attempt to declare or define one of these results in name clashes, compile errors, or undefined behavior, making such code non-portable and unmaintainable. |

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

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

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

| **Principles(s):** [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** |
| --- | --- | --- | --- |
| Astree | 22.10 | Reserved-identifier | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL51 |  |
| Clang | 3.9 | -Wreserved-id-macro  -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names. |
| CodeSonar | 9.1p0 | LANG.ID.NU.MK  LANG.STRUCT.DECL.RESERVED | Macro name is C keyword  Declaration of reserved name. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT31-C | Incorrect conversions between integers can cause truncation, sign problems, or other data corruption. Compliance with this standard will eliminate logic errors and guarantee that numerical values are unchanged when being converted (cast) or used in arithmetic operations. |

| **Noncompliant Code** |
| --- |
| Signed to Unsigned - Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of a signed type to a value of an unsigned type. This noncompliant code example results in a negative number being misinterpreted as a large positive number. |
| #include <limits.h>    **void** func(**signed** **int** si) {    /\* Cast eliminates warning \*/    unsigned **int** ui = (unsigned **int**)si;      /\* ... \*/  }    /\* ... \*/    func(INT\_MIN); |

| **Compliant Code** |
| --- |
| Validate ranges when converting from a signed type to an unsigned type. This compliant solution converts a value of a signed int type to a value of an unsigned int type: |
| #include <limits.h>    **void** func(**signed** **int** si) {    unsigned **int** ui;  **if** (si < 0) {      /\* Handle error \*/    } **else** {      ui = (unsigned **int**)si;  /\* Cast eliminates warning \*/    }    /\* ... \*/  }  /\* ... \*/    func(INT\_MIN + 1); |

**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 | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4 10.6 and 10.7 |
| CodeSonar | 9.1p0 | 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 |  |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |
| Cppcheck Premium | 24.11.0 | memsetValueOutOfRange  premium-cert-int31-c |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR32-C | Many functions in the C standard library operate on null-terminated strings and expect the terminating null to mark the end of the string. Passing an un-null-terminated sequence as a string causes undefined behavior, including reading past the end of the intended buffer, possibly resulting in an access violation or security bug (buffer overread). |

| **Noncompliant Code** |
| --- |
| This code example is noncompliant because the wide character sequence cur\_msg will not be null-terminated when passed to wcslen(). This will occur if lessen\_memory\_usage() is invoked while cur\_msg\_size still has its initial value of 1024. |
| #include <stdlib.h>  #include <wchar.h>    **wchar\_t** \*cur\_msg = NULL;  **size\_t** cur\_msg\_size = 1024;  **size\_t** cur\_msg\_len = 0;    **void** lessen\_memory\_usage(**void**) {  **wchar\_t** \*temp;  **size\_t** temp\_size;      /\* ... \*/    **if** (cur\_msg != NULL) {      temp\_size = cur\_msg\_size / 2 + 1;      temp = **realloc**(cur\_msg, temp\_size \* **sizeof**(**wchar\_t**));      /\* temp &and cur\_msg may no longer be null-terminated \*/  **if** (temp == NULL) {        /\* Handle error \*/      }        cur\_msg = temp;      cur\_msg\_size = temp\_size;      cur\_msg\_len = wcslen(cur\_msg);    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, cur\_msg will always be null-terminated when passed to wcslen(): |
| #include <stdlib.h>  #include <wchar.h>    **wchar\_t** \*cur\_msg = NULL;  **size\_t** cur\_msg\_size = 1024;  **size\_t** cur\_msg\_len = 0;    **void** lessen\_memory\_usage(**void**) {  **wchar\_t** \*temp;  **size\_t** temp\_size;      /\* ... \*/    **if** (cur\_msg != NULL) {      temp\_size = cur\_msg\_size / 2 + 1;      temp = **realloc**(cur\_msg, temp\_size \* **sizeof**(**wchar\_t**));      /\* temp and cur\_msg may no longer be null-terminated \*/  **if** (temp == NULL) {        /\* Handle error \*/      }        cur\_msg = temp;      /\* Properly null-terminate cur\_msg \*/      cur\_msg[temp\_size - 1] = L'\0';      cur\_msg\_size = temp\_size;      cur\_msg\_len = wcslen(cur\_msg);    }  } |

**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 | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Supported  Astree supports the implementation of library stubs to fully verify this guideline. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR32 | Partially implemented: can detect some violation of the rule |
| CodeSonar | 9.1p0 | MISC.MEM.NTERM.CSTRING | Unterminated C String |
| Coverity | 2017.07 | STRING\_NULL | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | MSC32-C | Seeding pseudorandom number generators is important to the overall unpredictability of the resulting pseudorandom values. Predictable or insecure seeding can lead to vulnerabilities such as SQL injection or token predictability and can otherwise breach integrity and confidentiality. |

| **Noncompliant Code POSIX** |
| --- |
| This noncompliant code example generates a sequence of 10 pseudorandom numbers using the random() function. When random() is not seeded, it behaves like rand(), producing the same sequence of random numbers each time any program that uses it is run. |
| #include <stdio.h>  #include <stdlib.h>    **void** func(**void**) {  **for** (unsigned **int** i = 0; i < 10; ++i) {      /\* Always generates the same sequence \*/  **printf**("%ld, ", random());    }  } |

| **Compliant Code** |
| --- |
| Call srandom() before invoking random() to seed the random sequence generated by random(). This compliant solution produces different random number sequences each time the function is called, depending on the resolution of the system clock: |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>    **void** func(**void**) {  **struct** timespec ts;  **if** (timespec\_get(&ts, TIME\_UTC) == 0) {      /\* Handle error \*/    } **else** {      srandom(ts.tv\_nsec ^ ts.tv\_sec);  **for** (unsigned **int** i = 0; i < 10; ++i) {        /\* Generates different sequences at different runs \*/  **printf**("%ld, ", random());      }    }  } |

**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 | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Supported, but no explicit checker |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MSC32 |  |
| CodeSonar | 9.1p0 | HARDCODED.SEED  MISC.CRYPTO.TIMESEED | Hardcoded Seed in PRNG  Predictable Seed in PRNG |
| Cppcheck Premium | 24.11.0 | Premium-cert-msc32-c |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM54-CPP | The purpose of this standard is to disallow the use of uninitialized memory to prevent undefined behavior. Access to an object outside of its lifetime may result in program crashes, memory corruptions, or security problems like use-after-free vulnerabilities. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example ensures that the long is constructed into a buffer of sufficient size. However, it does not ensure that the alignment requirements are met for the pointer passed into placement new. To make this example clearer, an additional local variable c has also been declared. |
| #include <new>    **void** f() {  **char** c; // Used elsewhere in the function    unsigned **char** buffer[**sizeof**(**long**)];  **long** \*lp = ::**new** (buffer) **long**;      // ...  } |

| **Compliant Code Alignas** |
| --- |
| In this compliant solution, the alignas declaration specifier is used to ensure the buffer is appropriately aligned for a long. |
| #include <new>    **void** f() {  **char** c; // Used elsewhere in the function    alignas(**long**) unsigned **char** buffer[**sizeof**(**long**)];  **long** \*lp = ::**new** (buffer) **long**;      // ...  } |

**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** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM54 |  |
| CodeSonar | 9.1p0 | LANG.MEM.BO | Buffer Overrun |
| Helix QAC | 2025.2 | C++3119, C++3128, DF3520, DF2521, DF3522, DF3523 |  |
| Klockwork | 2025.2 | CERT.MEM.PLACEMENTNEW.MISALIGNED  CERT.MEM.PLACEMENTNEW.TOOSMALL |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | EXP05-C | Assertions are intended to check that conditions that should never fail if the program is correct. Inserting input values (particularly untrusted input) in assertions negates the purpose of assertions because they can be manipulated to cause assertion failures, which can result in unexpected termination or denial of service. Defensive checks (not assertions) should be used to guard against invalid input. |

| **Noncompliant Code** |
| --- |
| The remove\_spaces() function in this noncompliant code example accepts a pointer to a string str and a string length slen and removes the space character from the string by shifting the remaining characters toward the front of the string. The function remove\_spaces() is passed a const char pointer as an argument. The const qualification is cast away, and then the contents of the string are modified. |
| **void** remove\_spaces(**const** **char** \*str, **size\_t** slen) {  **char** \*p = (**char** \*)str;  **size\_t** i;  **for** (i = 0; i < slen && str[i]; i++) {  **if** (str[i] != ' ') \*p++ = str[i];    }    \*p = '\0';  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function remove\_spaces() is passed a non-const char pointer. The calling function must ensure that the null-terminated byte string passed to the function is not const by making a copy of the string or by other means. |
| **void** remove\_spaces(**char** \*str, **size\_t** slen) {  **char** \*p = str;  **size\_t** i;  **for** (i = 0; i < slen && str[i]; i++) {  **if** (str[i] != ' ') \*p++ = str[i];    }    \*p = '\0';  } |

**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 | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Pointer-qualifier-cast-const  Pointer-qualifier-cast-const-implicit | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-EXP05 | Fully implemented |
| CodeSonar | 9.1p0 | LANG.CAST.PC.CRCQ | Cast removes const qualifier |
| ÉCLAIR | 1.2 | CC2.EXP05 | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Allowing exceptions to propagate without handling them can cause the program to terminate, lead to resource leaks, and result in inconsistent system states. Handling all exceptions ensures that the program remains stable, performs any necessary recovery, and executes any critical cleanup operations. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // Handle error    }  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| CondeSonar | 9.1p0 | LANG.STRUCT.UCTCH  PARSE.MBDH | Masked by handler  Masked by default handler |
| Klockwork | 2025.2 | MISRA.CATCH.ALL |  |
| Parasoft C/C++ Test | 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 |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Resource Management | FIO42-C | Failure to close files can cause resource leaks. These may lead to exhaustion of file descriptors, loss of performance, or loss of data. Closing files as soon as they are no longer needed explicitly relinquishes resources back to the system and flushes buffered output. |

| **Noncompliant Code** |
| --- |
| This code example is noncompliant because the file opened by the call to fopen() is not closed before function func() returns: |
| #include <stdio.h>    **int** func(**const** **char** \*filename) {  **FILE** \*f = **fopen**(filename, "r");  **if** (NULL == f) {  **return** -1;    }    /\* ... \*/  **return** 0;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the file pointed to by f is closed before returning to the caller: |
| #include <stdio.h>    **int** func(**const** **char** \*filename) {  **FILE** \*f = **fopen**(filename, "r");  **if** (NULL == f) {  **return** -1;    }    /\* ... \*/  **if** (**fclose**(f) == EOF) {  **return** -1;    }  **return** 0;  } |

**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 | Low | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Supported, but no explicit checker |
| CodeSonar | 9.1p0 | ALLOC.LEAK | Leak |
| Compass/ROSE |  |  |  |
| Coverity | 2017.07 | RESOURCE\_LEAK (partial) | Partially Implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | CON37-C | Usage of signal() is also considered unsafe in a multithreaded environment, since the behavior is implementation-defined and signal handling can be adversely affected by thread switching. Safer signal() function alternatives such as sigaction() exist. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example invokes the signal() function from a multithreaded program: |
| #include <signal.h>  #include <stddef.h>  #include <threads.h>    **volatile** **sig\_atomic\_t** flag = 0;    **void** handler(**int** signum) {    flag = 1;  }    /\* Runs until user sends SIGUSR1 \*/  **int** func(**void** \*data) {  **while** (!flag) {      /\* ... \*/    }  **return** 0;  }    **int** main(**void**) {  **signal**(SIGUSR1, handler); /\* Undefined behavior \*/    thrd\_t tid;    **if** (thrd\_success != thrd\_create(&tid, func, NULL)) {      /\* Handle error \*/    }    /\* ... \*/  **return** 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution uses an object of type atomic\_bool to indicate when the child thread should terminate its loop: |
| #include <stdatomic.h>  #include <stdbool.h>  #include <stddef.h>  #include <threads.h>    atomic\_bool flag = ATOMIC\_VAR\_INIT(**false**);    **int** func(**void** \*data) {  **while** (!flag) {      /\* ... \*/    }  **return** 0;  }    **int** main(**void**) {    thrd\_t tid;    **if** (thrd\_success != thrd\_create(&tid, func, NULL)) {      /\* Handle error \*/    }    /\* ... \*/    /\* Set flag when done \*/    flag = **true**;    **return** 0;  } |

**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 | Low | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Stdlib-use-signal | Fully Checked |
| CodeSonar | 9.1p0 | BADFUNC.SIGNAL | Use of signal |
| Coverity | 2017.07 | MISRA C 2012 RULE 21.5 | Over-constraining |
| CppCheck Premium | 24.11.0 | Premium-cert-con37-c |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Initialization | MSC50-CPP | std: :rand() is a low-quality random number generator, and since it is a global function, applications running it on different threads may have undefined behavior due to non-determinism of that global state. Newer random number generators are found in the <random> header and include better random generation, a more cryptographically secure API, deterministic behavior for debuggers, and thread safety (e.g. std::mt19937). |

| **Noncompliant Code** |
| --- |
| The following noncompliant code generates an ID with a numeric part produced by calling the rand() function. The IDs produced are predictable and have limited randomness. Further, depending on the value of RAND\_MAX, the resulting value can have modulo bias. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    id += std::to\_string(std::**rand**() % 10000);    // ...  } |

| **Compliant Code** |
| --- |
| The C++ standard library provides mechanisms for fine-grained control over pseudorandom number generation. It breaks random number generation into two parts: one is the algorithm responsible for providing random values (the engine), and the other is responsible for distribution of the random values via a density function (the distribution). The distribution object is not strictly required, but it works to ensure that values are properly distributed within a given range instead of improperly distributed due to bias issues. This compliant solution uses the [Mersenne Twister](http://dl.acm.org/citation.cfm?doid=272991.272995) algorithm as the engine for generating random values and a uniform distribution to negate the modulo bias from the noncompliant code example. |
| #include <random>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    std::uniform\_int\_distribution<**int**> distribution(0, 10000);    std::random\_device rd;    std::mt19937 engine(rd());    id += std::to\_string(distribution(engine));    // ...  } |

**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 | Low | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Bad-function(AUTOSAR.26.5.1A) | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MSC50 |  |
| Clang | 4.0(prerelease) | Cert-msc50-cpp | Checked by clang-tidy |
| CodeSonar | 9.1p0 | BADFUNC.RANDOM.RAND | Use of rand |

### 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 utilized 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.

It is always immensely helpful to use automation throughout the development of software. From automated tests, to builds we can leverage automation to increase productivity and accuracy. However, it is very important that automation is used responsibly, or it could have the opposite effect. When use in the correct way we can use automation throughout the entire software development cycle.

### 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 |
| --- | --- | --- | --- | --- | --- |
| CON37-C | Low | Probable | Low | P4 | L3 |
| DCL51-CPP | Low | Unlikely | Medium | P2 | L3 |
| ERR51-CPP | Low | Probable | Low | P6 | L2 |
| EXP-5-C | Medium | Probable | Medium | P4 | L3 |
| FIO42-C | Medium | Unlikely | Low | P2 | L3 |
| INT31-C | High | Probable | High | P12 | L1 |
| MEM54-C | High | Likely | High | P9 | L2 |
| MSC32-C | Medium | Likely | Medium | P18 | L1 |
| MSC50-CPP | Medium | Unlikely | Low | P4 | L3 |
| STR32-C | High | Probable | High | P12 | L1 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest is a security measure that secures data that is stored on physical storage media (servers, databases, backups) against unauthorized access or theft. This policy ensures that if the media is lost or stolen, the data is still unreadable and cannot be used by malicious actors without the proper encryption key. Encryption at rest typically involves using cryptographic algorithms such as AES-256 to transform data into an unreadable form that is difficult to decipher without a decryption key. The policy can be applied to data at rest whenever it is being stored. This includes data on physical servers, virtual machines, cloud storage, databases, and backups. It is important to manage encryption keys securely and only allow access to authorized individuals or systems. |
| Encryption in flight | Encryption in flight protects data that is being sent over a network by encrypting it as it travels from one point to another. This usually involves using a secure protocol like TLS or IPSec. The goal is to prevent any unauthorized person from intercepting or modifying the data. The policy applies to all network communications that contain sensitive or regulated data. This can include web traffic, API calls, or remote access sessions. |
| Encryption in use | In-use encryption (sometimes described as encryption-in-use) refers to encryption of data when it is loaded into the memory of an application and is in use, generally using homomorphic encryption or a secure enclave. This policy is appropriate for protecting large sensitive datasets, in applications such as private analytics or machine learning services running in the cloud or other shared environments, where the data is being actively processed or read. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication ensures that users or systems are who they claim to be before being allowed access. This can be achieved through various methods such as passwords, biometrics, multi-factor authentication (MFA), or certificates. The benefit of this policy is that it restricts actions within the system to authenticated and legitimate users or systems only. It helps mitigate unauthorized access. This policy needs to be applied at all entry points to the system. |
| Authorization | Authorization defines the actions an authenticated user is permitted to perform. Authorization policies are frequently based on role-based access control (RBAC) or attribute-based access control (ABAC). Authorization is a policy that is implemented post-authentication. If a user is authenticated, the system determines what resources and applications that the user is permitted to access according to the authorization policy. Authorization is an important part of an organization's security defenses and limits employees to access only those resources that are required to do their jobs, which greatly reduces the risk of insider attacks and privilege escalation. |
| Accounting | Accounting (or auditing) involves the tracking of user activities, resource consumption, and system modifications via logs and monitoring systems. This policy aims to facilitate compliance, incident investigation, and performance analysis. It applies to all critical systems and must ensure that logs are tamper-proof and securely stored. |

**\***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

|  |  |  |
| --- | --- | --- |
| Policy | Principle(s) | Justification |
| Encryption at Rest | 3.Architect and Design for Security Policies  5. Default Deny  8. Practice Defense in Depth | Encrypting data at rest is consistent with the principle of designing systems to enforce access controls and "layered" protections. The "default deny" strategy means that without the decryption key, no one has access. It also provides another layer of protection should physical storage be breached. |
| Encryption in Flight | 1.Validate Input Data  3.Architect and Design for Security Policies  8. Practice Defense in Depth | Encryption of data in transit provides data integrity and confidentiality during its travel between systems. Encryption also enables the implementation of secure architecture guidelines and acts as a defense in depth mechanism. In the event of a defense breach (such as a network filtering device), encryption can help limit data exposure. |
| Encryption in Use | 6. Adhere to the Principle of Least Privilege  8. Practice Defense in Depth | Encrypting the data while it is being processed would limit exposure to privileged users. Which would ensure that access is only granted to where it would be essential.  While also strengthening layered defenses by preventing any leaks from memory. |
| Authentication | 1.Validate Input Data  6. Adhere to the Principle of Least Privilege | Verifying the users identity would ensure that only trusted and valid inputs enter the system. While also ensuring access rights would be limited to their respective privileges |
| Authorization | 5. Default Deny  6. Adhere to the Principle of Least Privilege | Users would be restricted to necessary functions, while all other access would be denied by default. Mitigating the risk of unauthorized data access or misuse. |
| Accounting | 8. Practice Defense in Depth  9. Use Effective Quality Assurance Techniques | Tracking activities would allow an early detection of unusual or unauthorized behavior.  With the addition of logging and auditing they would add depth to overall security measures |

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 | 07/18/2025 | Initial Template / Covering core security principles and coding standards. | Nicholas Wyrwas |  |
| 1.1 | 08/08/2025 | Completion of:  C/C++ Standards  Risk Assessment  Automated Detection  Automation  Summary of Risk Assessments  Policies  Mapping of Principles | Nicholas Wyrwas |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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