**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

## Contents

[Green Pace Secure Development Policy 0](#_Toc179675451)

[Contents 1](#_Toc179675452)

[Overview 3](#_Toc179675453)

[Purpose 3](#_Toc179675454)

[Scope 3](#_Toc179675455)

[Module Three Milestone 3](#_Toc179675456)

[Ten Core Security Principles 3](#_Toc179675457)

[C/C++ Ten Coding Standards 4](#_Toc179675458)

[Coding Standard 1 5](#_Toc179675459)

[Coding Standard 2 8](#_Toc179675460)

[Coding Standard 3 10](#_Toc179675461)

[Coding Standard 4 12](#_Toc179675462)

[Coding Standard 5 15](#_Toc179675463)

[Coding Standard 6 18](#_Toc179675464)

[Coding Standard 7 20](#_Toc179675465)

[Coding Standard 8 22](#_Toc179675466)

[Coding Standard 9 25](#_Toc179675467)

[Coding Standard 10 27](#_Toc179675468)

[Defense-in-Depth Illustration 29](#_Toc179675469)

[Project One 29](#_Toc179675470)

[Revise the C/C++ Standards 29](#_Toc179675471)

[Risk Assessment 29](#_Toc179675472)

[Automated Detection 29](#_Toc179675473)

[Automation 29](#_Toc179675474)

[Summary of Risk Assessments 30](#_Toc179675475)

[Create Policies for Encryption and Triple A 30](#_Toc179675476)

[Map the Principles 31](#_Toc179675477)

[Audit Controls and Management 33](#_Toc179675478)

[Enforcement 33](#_Toc179675479)

[Exceptions Process 33](#_Toc179675480)

[Distribution 34](#_Toc179675481)

[Policy Change Control 34](#_Toc179675482)

[Policy Version History 34](#_Toc179675483)

[Appendix A Lookups 34](#_Toc179675484)

[Approved C/C++ Language Acronyms 34](#_Toc179675485)

## 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 | This principle is about ensuring the user’s input. Assume the user is trying to exploit code for unauthorized information. When you validate input you enforce the type of input, and the size of the input prior to processing it. |
| 1. Heed Compiler Warnings | When you are creating software, you should take any compiler warning seriously. If there is a vulnerability or warning about a specific variable you should make the necessary changes to correct the warning presented to you. |
| 1. Architect and Design for Security Policies | Security should not be an afterthought. Design and architecture decisions should consider security while being developed. You don’t want to set yourself up for multiple security issues that will need to be corrected later down the road. |
| 1. Keep It Simple | Do not overthink the design of an application for the sake of securing it. After a while development could become so muddled that it’s too difficult to do anything. |
| 1. Default Deny | It is easier to create a whitelist for access and make the deny a default option. This way, access is only for those specific cases, and all else is invalid and will be denied. |
| 1. Adhere to the Principle of Least Privilege | Be stingy with access and privileges. If something doesn’t need the privilege do not give it to them. Additional privilege can always be added later if necessary. |
| 1. Sanitize Data Sent to Other Systems | When data is sent to other systems make sure encryptions are right, and if specific data is not needed on the other system, remove it. |
| 1. Practice Defense in Depth | Be mindful of data both inside and outside of the software. Many attacks are successful because something is disclosed either physically or socially. |
| 1. Use Effective Quality Assurance Techniques | QA is in place to ensure a safe and successful product. Do not cut corners to deliver on time or get away with less work. Time needs to be taken to develop affective quality assurance. |
| 1. Adopt a Secure Coding Standard | There are many already known coding standards for specific frameworks and languages. It is best practice to adopt this into the SDLC because security is everyone’s responsibilities. |

### 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** | **Ensure that integer conversions do not result in lost or misinterpreted data** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Integer conversions, both implicit and explicit (using a cast), must be guaranteed not to result in lost or misinterpreted data. The only integer type conversions that are guaranteed to be safe for all data values and all possible conforming implementations are conversions of an integral value to a wider type of the same signedness. The C Standard, subclause 6.3.1.3 [ISO/IEC 9899:2011], says  When a value with integer type is converted to another integer type other than \_Bool, if the value can be represented by the new type, it is unchanged.  Otherwise, if the new type is unsigned, the value is converted by repeatedly adding or subtracting one more than the maximum value that can be represented in the new type until the value is in the range of the new type.  Otherwise, the new type is signed and the value cannot be represented in it; either the result is implementation-defined or an implementation-defined signal is raised. |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type. This noncompliant code example results in a truncation error on most implementations. |
| #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** |
| --- |
| Validate ranges when converting from an unsigned type to a signed type. This compliant solution can be used to convert a value of unsigned long int type to a value of signed char type. |
| #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):** Heed Compiler Warnings, validate input data Keep It Simple, Adopt a Secure Coding Standard. The coding standard addresses a warning by applying a type cast, it simply uses a condition statement, and it creates a standard to adopt while converting integers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **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 |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity)\* | 2017.07 | **NEGATIVE\_RETURNS**  **REVERSE\_NEGATIVE**  **MISRA\_CAST** | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior  Can find instances where a negativity check occurs after the negative value has been used for something else  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |
| Cppcheck | 2.15 | **memsetValueOutOfRange** | Partially implemented  The second argument to memset() cannot be represented as unsigned char |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule INT31-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint31c.html) | Checks for:   * Integer conversion overflow * Call to memset with unintended value * Sign change integer conversion overflow * Tainted sign change conversion * Unsigned integer conversion overflow   Rule partially covered. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure size arguments for variable length arrays are in a valid range.** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Variable length arrays (VLAs), a conditionally supported language feature, are essentially the same as traditional C arrays except that they are declared with a size that is not a constant integer expression and can be declared only at block scope or function prototype scope and no linkage. If the size argument supplied to a variable length array is not a positive integer value, the behavior is undefined. The programmer must ensure that size arguments to variable length arrays, especially those derived from untrusted data, are in a valid range. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a variable length array of size *size* is declared. The size is declared as *size\_t* in compliance with INT01-C. Use *rsize\_t* or *size\_t* for all integer values representing the size of an object. However, the value of size may be zero or excessive, potentially giving rise to a security vulnerability. |
| #include <stddef.h>    **extern** **void** do\_work(**int** \*array, **size\_t** size);    **void** func(**size\_t** size) {  **int** vla[size];    do\_work(vla, size);  } |

| **Compliant Code** |
| --- |
| This compliant solution ensures the size argument used to allocate vla is in a valid range (between 1 and a programmer-defined maximum); otherwise, it uses an algorithm that relies on dynamic memory allocation. The solution also avoids unsigned integer wrapping that, given a sufficiently large value of size, would cause malloc to allocate insufficient storage for the array. |
| #include <stdint.h>  #include <stdlib.h>    **enum** { MAX\_ARRAY = 1024 };  **extern** **void** do\_work(**int** \*array, **size\_t** size);    **void** func(**size\_t** size) {  **if** (0 == size || SIZE\_MAX / **sizeof**(**int**) < size) {      /\* Handle error \*/  **return**;    }  **if** (size < MAX\_ARRAY) {  **int** vla[size];      do\_work(vla, size);    } **else** {  **int** \*array = (**int** \*)**malloc**(size \* **sizeof**(**int**));  **if** (array == NULL) {        /\* Handle error \*/      }      do\_work(array, size);  **free**(array);    }  } |

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

| **Principles(s):** Architect and Design for Security Policies, Use Effective Quality Assurance techniques. An effective quality assurance technique will spot when size arguments are out of range. Designing with security in mind will ensure you never accidently cause a buffer overflow within your software. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.SIZE.IOFLOW** **ALLOC.SIZE.MULOFLOW** **MISC.MEM.SIZE.BAD** | Integer Overflow of Allocation Size Multiplication Overflow of Allocation Size Unreasonable Size Argument |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **REVERSE\_NEGATIVE** | Fully implemented |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 2.15 | **negativeArraySize** | Context sensitive analysis Will warn only if given size is negative |
| [Cppcheck Premium](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck+Premium) | 24.9.0 | **negativeArraySize**  **premium-cert-arr32-c** | Context sensitive analysis Will warn only if given size is negative |

#### Coding Standard 3

| **Coding Standard** | **Label** | [**Guarantee that storage for strings has sufficient space for character data and the null terminator**](https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings [Seacord 2013b]. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is sufficient size to hold the character data to be copied and the null-termination character. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example demonstrates an *off-by-one* error [Dowd 2006]. The loop copies data from src to dest. However, because the loop does not account for the null-termination character, it may be incorrectly written 1 byte past the end of dest. |
| #include <stddef.h>    **void** copy(**size\_t** n, **char** src[n], **char** dest[n]) {  **size\_t** i;    **for** (i = 0; src[i] && (i < n); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the loop termination condition is modified to account for the null-termination character that is appended to dest. |
| #include <stddef.h>    **void** copy(**size\_t** n, **char** src[n], **char** dest[n]) {  **size\_t** i;    **for** (i = 0; src[i] && (i < n - 1); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |
|  |

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

| **Principles(s):** Sanitize Data Sent to Other Systems, Use Effective Quality Assurance Techniques. Another example where effective QA can ensure sizes do not exceed the side of an array. Additionally, there are many use cases where a sting can be sent or copied to another system. Programmers will want to ensure string data does not get truncated or cause a buffer overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.2 | **C2840,  C5009, C5038**  **C++0145, C++5009, C++5038**  **DF2840, DF2841, DF2842, DF2843, DF2845, DF2846, DF2847, DF2848, DF2930, DF2931, DF2932, DF2933, DF2935, DF2936, DF2937, DF2938** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.2 | **SV.FMT\_STR.BAD\_SCAN\_FORMAT** **SV.UNBOUND\_STRING\_INPUT.FUNC** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152309) | 9.7.1 | **489 S, 109 D, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR31-a** **CERT\_C-STR31-b** **CERT\_C-STR31-c** **CERT\_C-STR31-d** **CERT\_C-STR31-e** | Avoid accessing arrays out of bounds Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Avoid using unsafe string functions which may cause buffer overflows |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Ensure that privilege relinquishment is successful** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | The meaning of "appropriate privileges" varies from platform to platform. For example, on Solaris, appropriate privileges for setuid() means that the PRIV\_PROC\_SETID privilege is in the effective privilege set of the process. On BSD, it means that the effective user ID (EUID) is zero (that is, the process is running as root) or that uid=geteuid(). On Linux, it means that the process has CAP\_SETUID capability and that setuid(geteuid()) will fail if the EUID is not equal to 0, the real user ID (RUID), or the saved set-user ID (SSUID). |

| **Noncompliant Code** |
| --- |
| This noncompliant code example compiles cleanly on most POSIX systems, but no explicit checks are made to ensure that privilege relinquishment has succeeded. This may be dangerous depending on the sequence of the preceding privilege changes. |
| /\* Code intended to run with elevated privileges \*/    /\* Temporarily drop privileges \*/  **if** (seteuid(getuid()) != 0) {    /\* Handle error \*/  }    /\* Code intended to run with lower privileges \*/    **if** (need\_more\_privileges) {    /\* Restore privileges \*/  **if** (seteuid(0) != 0) {      /\* Handle error \*/    }      /\* Code intended to run with elevated privileges \*/  }    /\* ... \*/    /\* Permanently drop privileges \*/  **if** (setuid(getuid()) != 0) {    /\* Handle error \*/  }    /\*   \* Code intended to run with lower privileges,   \* but if privilege relinquishment failed,   \* attacker can regain elevated privileges!   \*/ |

| **Compliant Code** |
| --- |
| This compliant solution was implemented in sendmail, a popular mail transfer agent, to determine if superuser privileges were successfully dropped [Wheeler 2003]. If the setuid() call succeeds after (supposedly) dropping privileges permanently, then the privileges were not dropped as intended. |
| /\* Store the privileged ID for later verification \*/  uid\_t privid = geteuid();    /\* Code intended to run with elevated privileges   \*/    /\* Temporarily drop privileges \*/  **if** (seteuid(getuid()) != 0) {    /\* Handle error \*/  }    /\* Code intended to run with lower privileges  \*/    **if** (need\_more\_privileges) {    /\* Restore Privileges \*/  **if** (seteuid(privid) != 0) {      /\* Handle error \*/    }      /\* Code intended to run with elevated privileges   \*/  }    /\* ... \*/    /\* Restore privileges if needed \*/  **if** (geteuid() != privid) {  **if** (seteuid(privid) != 0) {      /\* Handle error \*/    }  }    /\* Permanently drop privileges \*/  **if** (setuid(getuid()) != 0) {    /\* Handle error \*/  }    **if** (setuid(0) != -1) {    /\* Privileges can be restored, handle error \*/  }    /\*   \* Code intended to run with lower privileges;   \* attacker cannot regain elevated privileges   \*/ |

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

| **Principles(s):** Adhere to the Principle of Least Privilege, Default Deny, Adopt a Secure Coding Standard.  Remove privilege to deny access for any data or memory not needed to know by the current user. Defaulting to deny access to data memory or other databases is a good practice to maintain a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Low | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-POS37-a** | Ensure that privilege relinquishment is successful |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule POS37-C](https://www.mathworks.com/help/bugfinder/ref/certcrulepos37c.html) | Checks for priviledge drop not verified (rule fully covered) |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **user\_defined** | Soundly supported |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-POS37** |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Only free memory allocated dynamically** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Freeing memory that is not allocated dynamically can result in heap corruption and other serious errors. Do not call free() on a pointer other than one returned by a standard memory allocation function, such as malloc(), calloc(), realloc(), or aligned\_alloc(). |

| **Noncompliant Code** |
| --- |
| This noncompliant code example sets c\_str to reference either dynamically allocated memory or a statically allocated string literal depending on the value of argc. In either case, c\_str is passed as an argument to free(). If anything other than dynamically allocated memory is referenced by c\_str, the call to free(c\_str) is erroneous. |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    **enum** { MAX\_ALLOCATION = 1000 };    **int** main(**int** argc, **const** **char** \*argv[]) {  **char** \*c\_str = NULL;  **size\_t** len;    **if** (argc == 2) {      len = **strlen**(argv[1]) + 1;  **if** (len > MAX\_ALLOCATION) {        /\* Handle error \*/      }      c\_str = (**char** \*)**malloc**(len);  **if** (c\_str == NULL) {        /\* Handle error \*/      }  **strcpy**(c\_str, argv[1]);    } **else** {      c\_str = "usage: $>a.exe [string]";  **printf**("%s\n", c\_str);    }  **free**(c\_str);  **return** 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution eliminates the possibility of c\_str referencing memory that is not allocated dynamically when passed to free(). |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    **enum** { MAX\_ALLOCATION = 1000 };    **int** main(**int** argc, **const** **char** \*argv[]) {  **char** \*c\_str = NULL;  **size\_t** len;    **if** (argc == 2) {      len = **strlen**(argv[1]) + 1;  **if** (len > MAX\_ALLOCATION) {        /\* Handle error \*/      }      c\_str = (**char** \*)**malloc**(len);  **if** (c\_str == NULL) {        /\* Handle error \*/      }  **strcpy**(c\_str, argv[1]);    } **else** {  **printf**("%s\n", "usage: $>a.exe [string]");  **return** EXIT\_FAILURE;    }  **free**(c\_str);  **return** 0;  } |

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

| **Principles(s):** Keep It Simple, Adopt a Secure Coding Standard.  This creates a standard step before passing memory reference to the free() function. |
| --- |

**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 | **invalid-free** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM34** | Can detect memory deallocations for stack objects |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | **clang-analyzer-unix.Malloc** | Checked by clang-tidy; can detect some instances of this rule, but does not detect all |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.TM** | Type Mismatch |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the termination behavior of assert() and abort()** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | The assert macro puts diagnostic tests into programs; it expands to a void expression. When it is executed, if expression (which shall have a scalar type) is false (that is, compares equal to 0), the assert macro writes information about the particular call that failed (including the text of the argument, the name of the source file, the source line number, and the name of the enclosing function—the latter are respectively the values of the pre-processing macros \_\_*FILE\_*\_ and \_\_*LINE\_*\_ and of the identifier \_\_*func\_*\_) on the standard error stream in an implementation-defined format. It then calls the abort function. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example defines a function that is called before the program exits to clean up. However, the code also has an assert, and if the assertion fails, the cleanup() function is *not* called. |
| **void** cleanup(**void**) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(**void**) {  **if** (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/    **assert**(/\* Something bad didn't happen \*/);      /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the call to assert() is replaced with an if statement that calls exit() to ensure that the proper termination routines are run. |
| **void** cleanup(**void**) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(**void**) {  **if** (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/    **if** (/\* Something bad happened \*/) {  **exit**(EXIT\_FAILURE);    }      /\* ... \*/  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques, Architect and Design for Security Policies.  Any bug is a vulnerability. If clean-up is never executed after an assertion it puts the software at risk of exploitation. Security should be in mind while developing, and a strong QA foundation should be able to catch scalarly errors such as this. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-ERR06-a | Do not use assertions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Adopt and implement a consistent and comprehensive error-handling policy** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | A secure system is invariably subject to stresses, such as those caused by attack, erroneous or malicious inputs, hardware or software faults, unanticipated user behavior, and unexpected environmental changes that are outside the bounds of "normal operation." Yet the system must continue to deliver essential services in a timely manner, safely and securely. To accomplish this, the system must exhibit qualities such as robustness, reliability, error tolerance, fault tolerance, performance, and security. All these system-quality attributes depend on consistent and comprehensive error handling that supports the goals of the overall system. |

| **Noncompliant Code** |
| --- |
| Many implementations of the stdio package adjust their behavior slightly if stdout is a terminal. To make the determination, these implementations perform some operation that fails (with ENOTTY) if stdout is not a terminal. Although the output operation goes on to complete successfully, errno still contains ENOTTY. This behavior can be mildly confusing, but it is not strictly incorrect because it is meaningful for a program to inspect the contents of errno only after an error has been reported. More precisely, errno is meaningful only after a library function that sets errno on error has returned an error code. |
| **errno** = 0;  **printf**("This\n");  **printf**("is\n");  **printf**("a\n");  **printf**("test.\n");  **if** (**errno** != 0) {  **fprintf**(stderr, "printf failed: %s\n", **strerror**(**errno**));  } |

| **Compliant Code** |
| --- |
| This compliant solution uses ferror() to detect an error. In addition, if an early call to printf() fails, later calls may modify errno, whether they fail or not, so the program cannot rely on being able to detect the root cause of the original failure if it waits until after a sequence of library calls to check. |
| **printf**("This\n");  **printf**("is\n");  **printf**("a\n");  **printf**("test.\n");  **if** (**ferror**(stdout)) {  **fprintf**(stderr, "printf failed\n");  } |

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

| **Principles(s):** Adopt a Secure Coding Standard.  Having a sound policy in place for error handling can ensure the product continues to perform and manage potential attacks while remaining transparent to users. If implemented correctly, could mean maintaining the system’s operability while under stress of an attack. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rec. ERR00-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.err00c.html) | Checks for situations where error information is not checked (rec. partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not call system()** |
| --- | --- | --- |
| System | [STD-008-CPP] | Use of the system() function can result in exploitable vulnerabilities, in the worst case allowing execution of arbitrary system commands. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the system() function is used to execute any\_cmd in the host environment. If this code is compiled and run with elevated privileges on a Linux system, for example, an attacker can create an account by entering the following string: *happy'; useradd 'attacker*  The shell would interpret this string as two separate commands:  **$** any\_cmd 'happy'  **$** useradd 'attacker'  and create a new user account that the attacker can use to access the compromised system. |
| #include <string.h>  #include <stdlib.h>  #include <stdio.h>    **enum** { BUFFERSIZE = 512 };    **void** func(**const** **char** \*input) {  **char** cmdbuf[BUFFERSIZE];  **int** len\_wanted = snprintf(cmdbuf, BUFFERSIZE,                              "any\_cmd '%s'", input);  **if** (len\_wanted >= BUFFERSIZE) {      /\* Handle error \*/    } **else** **if** (len\_wanted < 0) {      /\* Handle error \*/    } **else** **if** (**system**(cmdbuf) == -1) {      /\* Handle error \*/    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the call to system() is replaced with a call to execve(). The exec family of functions does not use a full shell interpreter, so it is not vulnerable to command-injection attacks, such as the one illustrated in the noncompliant code example. |
| #include <sys/types.h>  #include <sys/wait.h>  #include <unistd.h>  #include <errno.h>  #include <stdlib.h>    **void** func(**char** \*input) {    pid\_t pid;  **int** status;    pid\_t ret;  **char** \***const** args[3] = {"any\_exe", input, NULL};  **char** \*\*env;  **extern** **char** \*\*environ;      /\* ... Sanitize arguments ... \*/      pid = fork();  **if** (pid == -1) {      /\* Handle error \*/    } **else** **if** (pid != 0) {  **while** ((ret = waitpid(pid, &status, 0)) == -1) {  **if** (**errno** != EINTR) {          /\* Handle error \*/  **break**;        }      }  **if** ((ret == 0) ||          !(WIFEXITED(status) && !WEXITSTATUS(status))) {        /\* Report unexpected child status \*/      }    } **else** {      /\* ... Initialize env as a sanitized copy of environ ... \*/  **if** (execve("/usr/bin/any\_cmd", args, env) == -1) {        /\* Handle error \*/        \_Exit(127);      }    }  } |

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

| **Principles(s):** Practice Defense in Depth.  Applications and code are not the only thing vulnerable to attack. The system upon which the application runs on is also a potential asset to be used by attackers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **stdlib-use-system** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-ENV33** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | cert-env33-c | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **BADFUNC.PATH.SYSTEM** **IO.INJ.COMMAND** | Use of system Command injection |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Never hard code sensitive information** |
| --- | --- | --- |
| Sensitive Data | [STD-009-CPP] | Hard coding sensitive information, such as passwords or encryption keys can expose the information to attackers. Anyone who has access to the executable or dynamic library files can examine them for strings or other critical data, revealing the sensitive information. Leaking data protected by International Traffic in Arms Regulations (ITAR) or the Health Insurance Portability and Accountability Act (HIPAA) can also have legal consequences. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example must authenticate to a remote service with a code, using the authenticate() function declared below. It passes the authentication code to this function as a string literal. The authentication code exists in the program's binary executable and can be easily discovered. |
| /\* Returns nonzero if authenticated \*/  **int** authenticate(**const** **char**\* code);    **int** main() {  **if** (!authenticate("correct code")) {  **printf**("Authentication error\n");  **return** -1;    }    **printf**("Authentication successful\n");    // ...Work with system...  **return** 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution requires the user to supply the authentication code, and securely erases it when done, using memset\_s(), an optional function provided by C11's Annex K. |
| /\* Returns nonzero if authenticated \*/  **int** authenticate(**const** **char**\* code);    **int** main() {  #define CODE\_LEN 50  **char** code[CODE\_LEN];  **printf**("Please enter your authentication code:\n");  **fgets**(code, **sizeof**(code), stdin);  **int** flag = authenticate(code);    memset\_s(code, **sizeof**(code), 0, **sizeof**(code));  **if** (!flag) {  **printf**("Access denied\n");  **return** -1;    }  **printf**("Access granted\n");    // ...Work with system...  **return** 0;  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems, Practice Defense in Depth, Architect and Design for Security Policies.  Hardcoding Password will mean the password is coded into the programs binary executable. This also means patches would have to be sent to other systems that contain the software. Linux has a utility called strings that can print strings found within a binary. Practicing defense in depth with other system capabilities in mind is good security practice when designing for security policies. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **HARDCODED.AUTH** **HARDCODED.DNS** **HARDCODED.KEY** **HARDCODED.SALT** **HARDCODED.SEED** | Hardcoded Authentication Hardcoded DNS Name Hardcoded Crypto Key Hardcoded Crypto Salt Hardcoded Seed in PRNG |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.2 | **C3122**  **C++3842** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.2 | **HCC** **HCC.PWD** **HCC.USER** **CXX.SV.PWD.PLAIN** **CXX.SV.PWD.PLAIN.LENGTH** **CXX.SV.PWD.PLAIN.ZERO** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MSC41-a** | Do not hard code string literals |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Properly pair allocation and deallocation** |
| --- | --- | --- |
| Windows OS | [STD-010-CPP] | Windows provides several APIs for allocating memory.  While some of these functions have converged over time, it is still important to always properly pair allocations and deallocations. |

| **Noncompliant Code** |
| --- |
| In this example, the FormatMessage() function allocates a buffer and stores it in the buf parameter.  From the documentation of FORMAT\_MESSAGE\_ALLOCATE\_BUFFER [MSDN]. Instead of freeing the memory using LocalFree(), this code example uses GlobalFree() erroneously. |
| **LPTSTR** buf;  **DWORD** n = FormatMessage(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER |                          FORMAT\_MESSAGE\_FROM\_SYSTEM |                          FORMAT\_MESSAGE\_IGNORE\_INSERTS, 0, GetLastError(),                          LANG\_USER\_DEFAULT, (**LPTSTR**)&buf, 1024, 0);  **if** (n != 0) {    /\* Format and display the error to the user \*/      GlobalFree(buf);  } |

| **Compliant Code** |
| --- |
| The compliant solution uses the proper deallocation function as described by the documentation. |
| **LPTSTR** buf;  **DWORD** n = FormatMessage(FORMAT\_MESSAGE\_ALLOCATE\_BUFFER |                          FORMAT\_MESSAGE\_FROM\_SYSTEM |                          FORMAT\_MESSAGE\_IGNORE\_INSERTS, 0, GetLastError(),                          LANG\_USER\_DEFAULT, (**LPTSTR**)&buf, 1024, 0);  **if** (n != 0) {    /\* Format and display the error to the user \*/      LocalFree(buf);  } |

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

| **Principles(s):** Keep It Simple, Use Effective Quality Assurance Techniques.  There is a table of pairing for Windows API memory allocation calls. Use what is already appropriate and documented. Quality QA will be aware of these pairings and prevent incorrect pairing from occurring. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported: Can be checked with appropriate analysis stubs. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.TM** | Type mismatch |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ALLOC\_FREE\_MISMATCH (needs improvement)** | Partially implemented; needs improvement |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule WIN30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulewin30c.html) | Checks for mismatched alloc/dealloc functions on Windows (rule fully covered) |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation can be used to enforce the standards found in this policy. Training should be put into place prior to implementation. Part of planning should be familiarizing with security tools and threat models. After assessing and planning, IDE security plug-ins can be put in place for the development team so that code is flagged when it deviates from these coding standards and policies.

After release there should be integrity checks and defense in depth measures in place. Network monitoring and penetration testing will be used to continuously assess the product. Security personnel should be ready and prepared to respond after delivery in case of any incidents.

### 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 | Probable | High | Medium | 2 |
| STD-002-CPP | High | Probable | High | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Probable | Low | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-007-CPP | Medium | Probable | High | Low | 3 |
| STD-008-CPP | High | Probable | Medium | High | 1 |
| STD-009-CPP | High | Probable | Medium | High | 1 |
| STD-010-CPP | Low | Probable | Low | Medium | 2 |

### 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 when data is encrypted when stored on a disk, a database or any other form of persistent storage. This should be used whenever sensitive information is stored but not actively being sent or used. |
| Encryption in flight | Encryption in flight is when data is encrypted while being sent over a network from one device or system to another. This type of encryption prevents eavesdropping and protects against interception. |
| Encryption in use | Encryption in use is when data is actively being used such as in memory or during any data processing or calculations. This type of encryption prevents unauthorized access to sensitive information while it is being utilized. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is when you verify the identity of the user to ensure they are who they say they are. This policy applies because one of our principles is defense in depth. Whenever a user is gaining access to a system we must confirm they are the actual individual. |
| Authorization | Once the identity of the user is confirmed we can establish what type of privileges they are granted. This policy applies because we adhere to the principle of least privilege. We only what to grant permission for what is necessary for the user. |
| Accounting | This is a way to track activity of a user. This policy applies because we adopt security coding standards. We must be able to track activity and monitor for abnormal behavior of a user in case it is a threat actor. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 10/07/2024 | Added Principles, and Coding Standards. | Joshua Battaglia |  |
| 1.2 | 10/13/2024 | Improved Coding Standards. Added Risk Assessment, Automation, Detection, Risk Summary, Encryption policies, Mapping principles to standards. | Joshua Battaglia | TBD |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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