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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_gjdgxs)

[Purpose 2](#_30j0zll)

[Scope 2](#_1fob9te)

[Module Three Milestone 2](#_3znysh7)

[Ten Core Security Principles 2](#_2et92p0)

[C/C++ Ten Coding Standards 3](#_3dy6vkm)

[Coding Standard 1 4](#_1t3h5sf)

[Coding Standard 2 5](#_4d34og8)

[Coding Standard 3 6](#_2s8eyo1)

[Coding Standard 4 7](#_17dp8vu)

[Coding Standard 5 8](#_3rdcrjn)

[Coding Standard 6 9](#_26in1rg)

[Coding Standard 7 10](#_lnxbz9)

[Coding Standard 8 11](#_35nkun2)

[Coding Standard 9 13](#_1ksv4uv)

[Coding Standard 10 14](#_44sinio)

[Defense-in-Depth Illustration 15](#_2jxsxqh)

[Project One 15](#_z337ya)

[1.](#_3j2qqm3) Revise the C/C++ Standards 15

[2.](#_1y810tw) Risk Assessment 15

[3.](#_4i7ojhp) Automated Detection 15

[4.](#_2xcytpi) Automation 15

[5.](#_1ci93xb) Summary of Risk Assessments 16

[6.](#_3whwml4) Create Policies for Encryption and Triple A 16

[7.](#_2bn6wsx) Map the Principles 17

[Audit Controls and Management 18](#_qsh70q)

[Enforcement 18](#_3as4poj)

[Exceptions Process 18](#_1pxezwc)

[Distribution 19](#_49x2ik5)

[Policy Change Control 19](#_2p2csry)

[Policy Version History 19](#_147n2zr)

[Appendix A Lookups 19](#_3o7alnk)

[Approved C/C++ Language Acronyms 19](#_23ckvvd)

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating input data is very important because malicious users could potentially pass data through an input or even a user could accidentally include certain characters in their input that could inject SQL into our database that could get sensitive user data. |
| 1. Heed Compiler Warnings | There are a lot of great compiler warnings that have been put together over the years. We should pay close attention to them and work to reduce them as much as possible to keep our application as secure as possible. |
| 1. Architect and Design for Security Policies | Thinking about security should start as early as possible, even more than any coding begins. We should build our application architecture with the security policies and design in mind. |
| 1. Keep It Simple | Simple design and architecture is much easier to maintain and build upon. Adding additional complexities into a piece of software where it is not needed can also introduce unforeseen bugs and potential vulnerabilities that the programmers might not see. |
| 1. Default Deny | This means to by default deny access to all sensitive portions of a system and only allow explicit access to certain users rather than the other way around. This means there’s no chance of accidentally forgetting to lockdown a certain area of the application. |
| 1. Adhere to the Principle of Least Privilege | Users should only have access to the absolute least amount of privileges they need to do the actions they need within the system. Providing users with more access than what they need can lead to security holes in the application. |
| 1. Sanitize Data Sent to Other Systems | We should not only be responsible for the security of our system but also other third party systems as well. We should not allow users to send data to external systems without first checking and validating that input. |
| 1. Practice Defense in Depth | We should have multiple separate layers of security within our application. If a user somehow gets past one layer they should be greeted with another layer that they need to get through. This helps keep the application as secure as possible and deters possible hackers from getting through. |
| 1. Use Effective Quality Assurance Techniques | QA is a crucial role in building an application. Having QA Engineers thoroughly test the application. This means manual user testing and automated regression testing to ensure new changes made to the system are secure and correct. |
| 1. Adopt a Secure Coding Standard | There are a lot of great existing secure coding standards that have been compiled over decades and trial and error. Adopting one of these security policies rather than creating your own can avoid potential errors and speed up the time to production. |

### 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** | DCL40-C | Do not create incompatible declarations of the same function or object |

| **Noncompliant Code** |
| --- |
| Here we are assigning the extern i to an int data type, and then initialize a new i var with the short data type. |
| /\* In a.c \*/  **extern** **int** i; /\* UB 15 \*/    **int** f(**void**) {  **return** ++i; /\* UB 37 \*/  }    /\* In b.c \*/  **short** i; /\* UB 15 \*/ |

| **Compliant Code** |
| --- |
| Here both i variables are the int type. |
| /\* In a.c \*/  **extern** **int** i;    **int** f(**void**) {  **return** ++i;  }    /\* In b.c \*/  **int** i; |

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

| **Principles(s):** This falls in the Adopt a secure coding standard principle. This also fallins into the heed compiler warnings since it is fully checked in our automation tools. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | type-compatibility  type-compatibility-link  distinct-extern | Fully checked |
| Axivion Bauhaus Suite | [Insert text.] | CertC-DCL40 | Fully implemented |
| CodeSonar | [Insert text.] | LANG.STRUCT.DECL.IF  LANG.STRUCT.DECL.IO | Inconsistent function declarations  Inconsistent object declarations |
| Coverity | [Insert text.] | MISRA C 2012 Rule 8.4 | Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | EXP33-C | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| Here there is no control flow for if the user sets the sign to 0. |
| **void** set\_flag(**int** number, **int** \*sign\_flag) {  **if** (NULL == sign\_flag) {  **return**;  }    **if** (number > 0) {  \*sign\_flag = 1;  } **else** **if** (number < 0) {  \*sign\_flag = -1;  }  }    **int** is\_negative(**int** number) {  **int** sign;  set\_flag(number, &sign);  **return** sign < 0;  } |

| **Compliant Code** |
| --- |
| Here we are accounting for all numbers, including 0. |
| **void** set\_flag(**int** number, **int** \*sign\_flag) {  **if** (NULL == sign\_flag) {  **return**;  }    /\* Account for number being 0 \*/  **if** (number >= 0) {  \*sign\_flag = 1;  } **else** {  \*sign\_flag = -1;  }  }    **int** is\_negative(**int** number) {  **int** sign = 0; /\* Initialize for defense-in-depth \*/  set\_flag(number, &sign);  **return** sign < 0;  } |

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

| **Principles(s):**This falls in the adopt a secure coding policy principle. Also in heed compiler warnings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | uninitialized-local-read  uninitialized-variable-use | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-EXP33 |  |
| CodeSonar | 7.4p0 | LANG.MEM.UVAR | Uninitialized variable |
| Compass/ROSE |  |  | Automatically detects simple violations of this rule, although it may return some false positives. It may not catch more complex violations, such as initialization within functions taking uninitialized variables as arguments. It does catch the second noncompliant code example, and can be extended to catch the first as well |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR51-CPP | Do not attempt to create a std::string from a null pointer. |

| **Noncompliant Code** |
| --- |
| Here we are not checking if the std::getenv value is valid, so we’re setting the string directly. |
| #include <cstdlib>  #include <string>    **void** f() {  std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| Here we’re checking if the value is valid. If it is not valid for any reason we’re setting the string value to an empty string. |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** This would fall into the adopt a secure coding policy standard. We should not allow any code to enter our production env without having a standard for checking null strings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | assert\_failure |  |
| CodeSonar | 7.4p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Helix QAC | 2023.3 | DF4770, DF4771, DF4772, DF4773, DF4774 |  |
| Klocwork | 2023.3 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | FIO30-C | Exclude user input from format strings |

| **Noncompliant Code** |
| --- |
| Here we’re just printing a message directly from the user input. This can expand to not ever insert a SQL statement directly from user input either. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    **void** incorrect\_password(**const** **char** \*user) {  **int** ret;  /\* User names are restricted to 256 or fewer characters \*/  **static** **const** **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + **sizeof**(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);  **if** (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  **if** (ret < 0) {  /\* Handle error \*/  } **else** **if** (ret >= len) {  /\* Handle truncated output \*/  }  **fprintf**(stderr, msg);  **free**(msg);  } |

| **Compliant Code** |
| --- |
| Here we’re just checking if the user input is valid and a valid length, this can expand to SQL by sanitizing and removing any illegal chars before running the SQL statement. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    **void** incorrect\_password(**const** **char** \*user) {  **int** ret;  /\* User names are restricted to 256 or fewer characters \*/  **static** **const** **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + **sizeof**(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);  **if** (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  **if** (ret < 0) {  /\* Handle error \*/  } **else** **if** (ret >= len) {  /\* Handle truncated output \*/  }  **fputs**(msg, stderr);  **free**(msg);  } |

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

| **Principles(s):** This falls in the validate user input principle, because we should always sanitize and validate user input before using them in the system to prevent any malicious code injection. Also heed compiler warnings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 |  | Supported via stubbing/taint analysis |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | Partially implemented |
| CodeSonar | 7.4p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| Compass/ROSE |  |  |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM51-CPP | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| Here we’re calling the delete method. We are attempting to free memory from an undefined value. |
| #include <iostream>    **struct** S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {  alignas(**struct** S) **char** space[**sizeof**(**struct** S)];  S \*s1 = **new** (&space) S;    // ...    **delete** s1;  } |

| **Compliant Code** |
| --- |
| Here we are calling S’s destructor which will properly deallocate the memory. |
| #include <iostream>    **struct** S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {  alignas(**struct** S) **char** space[**sizeof**(**struct** S)];  S \*s1 = **new** (&space) S;    // ...    s1->~S();  } |

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

| **Principles(s):** This would probably fall into the Ensure Effective QA standard. We should have proper automated testing that tests for these kinds of errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | invalid\_dynamic\_memory\_allocation  dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM51 |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 7.4p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Free non-heap variable  Double free  Type mismatch  Leak |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | EXP45-C | Do not perform assignments in selection statements |

| **Noncompliant Code** |
| --- |
| Here we are accidentally assigning a to the value of b in an if block |
| **if** (a = b) {  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Here we are doing a comparison correctly. |
| **if** (a == b) {  /\* ... \*/  } |

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

| **Principles(s):** This falls in heed compiler warnings principle. This should never get past an initial error and should never make it into a production environment. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | assignment-conditional | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-EXP45 |  |
| Clang | 3.9 | -Wparentheses | Can detect some instances of this rule, but does not detect all |
| CodeSonar | 7.4p0 | LANG.STRUCT.CONDASSIG  LANG.STRUCT.SE.COND  LANG.STRUCT.USEASSIGN | Assignment in conditional  Condition contains side effects  Assignment result in expression |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| We are not handling any exceptions. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {  throwing\_func();  }    **int** main() {  f();  } |

| **Compliant Code** |
| --- |
| We are properly specifying a try catch to handle any exceptions. |
| **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):** This falls in the adopt a secure coding policy standard. We should have a standard in our PR review that we should not allow code to enter our production env without making sure all exceptions are caught. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | main-function-catch-all  early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| CodeSonar | 7.4p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2023.3 | C++4035, C++4036, C++4037 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Floating Point | FLP30-C | Do not use floating-point variables as loop counters |

| **Noncompliant Code** |
| --- |
| This gives us an inconsistent output because it could iterate either 9 or 10 times. |
| **void** func(**void**) {  **for** (**float** x = 0.1f; x <= 1.0f; x += 0.1f) {  /\* Loop may iterate 9 or 10 times \*/  }  } |

| **Compliant Code** |
| --- |
| This will always output the same exact value. |
| #include <stddef.h>    **void** func(**void**) {  **for** (**size\_t** count = 1; count <= 10; ++count) {  **float** x = count / 10.0f;  /\* Loop iterates exactly 10 times \*/  }  } |

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

| **Principles(s):** This would fall into the heed compiler warnings principle. As we can see below it is fully checked by all of our automation tools. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | for-loop-float | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FLP30 | Fully implemented |
| Clang | 3.9 | cert-flp30-c | Checked by clang-tidy |
| CodeSonar | 7.4p0 | LANG.STRUCT.LOOP.FPC | Float-typed loop counter |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Environment | ENV33-C | Do not call system() |

| **Noncompliant Code** |
| --- |
| This example uses the system() method. |
| #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** |
| --- |
| Replacing the system() call with execve() |
| #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):** This would fall into the default deny principle. We should by default never allow code to call any system functions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | stdlib-use-system | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ENV33 |  |
| Clang | 3.9 | cert-env33-c | Checked by clang-tidy |
| CodeSonar | 7.4p0 | BADFUNC.PATH.SYSTEM  IO.INJ.COMMAND | Use of system  Command injection |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Arrays | ARR36-C | Do not subtract or compare two pointers that do not refer to the same array |

| **Noncompliant Code** |
| --- |
| We are subtracting the free elements in the num array from a pointer. |
| #include <stddef.h>    **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** end;  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;    /\* Increment next\_num\_ptr as array fills \*/    free\_elements = &end - next\_num\_ptr;  } |

| **Compliant Code** |
| --- |
| Here we use the same array to determine how many free elements are left. |
| #include <stddef.h>  **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;    /\* Increment next\_num\_ptr as array fills \*/    free\_elements = &(nums[SIZE]) - next\_num\_ptr;  } |

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

| **Principles(s):** This would fall into the heed compiler warnings principle. It should be automatically checked and throw a warning in our automation tools. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | pointer-subtraction | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ARR36 | Can detect operations on pointers that are unrelated |
| CodeSonar | 7.4p0 | LANG.STRUCT.CUP  LANG.STRUCT.SUP | Comparison of unrelated pointers  Subtraction of unrelated pointers |
| Helix QAC | 2023.3 | C0487, C0513  DF2668, DF2669, DF2761, DF2762, DF2763, DF2766, DF2767, DF2768, DF2771, DF2772, DF2773 |  |

### 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.

There are certain standards that should never make it past the local dev build of the developer working on a new feature. For example, if the standard is caught by a compiler warning, it should be fixed immediately before entering a deployed environment. This could also be checked at the “Verify and Test” phase before pre-production enters into production, but in reality should be caught in the “Build” phase of pre-production. Other standards like “Ensure Effective QA” should be caught in the “verify and test” phase in pre-production. Everything that enters into production should be thoroughly tested. In an ideal world all standards should be caught somewhere in the pre-production phase. We should do our best to never allow anything in production that could cause harm or not be secure. If something were to make it through it should be caught in the “transition and health check” phase, as a last resort we have a monitor and detect phase if something were to make it all the way into a production environment.

### 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 |
| --- | --- | --- | --- | --- | --- |
| ARR36-C | Medium | Probable | Medium | P8 | L2 |
| DCL40-C | Low | Unlikely | Medium | P2 | L3 |
| ENV33-C | High | Probable | Medium | P12 | L1 |
| ERR51-CPP | Low | Probable | Medium | P4 | L3 |
| EXP33-C | High | Probable | Medium | P12 | L1 |
| EXP45-C | Low | Likely | Medium | P6 | L2 |
| FIO30-C | High | Likely | Medium | P18 | L1 |
| FLP30-C | Low | Probable | Low | P6 | L2 |
| MEM51-CPP | High | Likely | Medium | P18 | L1 |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STR51-CPP | High | Likely | Medium | P18 | 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 in rest | This policy states that any data that is stored at rest on a physical media device should be encrypted. |
| Encryption at flight | This policy states that data that is being transmitted over a network should always be encrypted. |
| Encryption in use | This policy states that we should protect any data that is currently in use. This means things like files that are open, database connections or anything a user could be using. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This states that we should consider extra security layers like MFA, and also ensure things like user passwords and login information are properly encrypted. |
| Authorization | This states that we should have appropriate roles and responsibilities for multiple user types, and only provide the least amount of access possible for the user to interact with the program in the ways they need. |
| Accounting | This states that we should keep appropriate records of access in the system for auditing purposes. Keeping track of access in the system can protect us if a breach were to occur, we could more easily track it down and stop it. |

**\***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

1. Standard #’s: 4
2. Standard #’s: 1, 2, 4, 6, 8, 10
3. Standard #’s:
4. Standard #’s:
5. Standard #’s: 9
6. Standard #’s:
7. Standard #’s:
8. Standard #’s:
9. Standard #’s: 5
10. Standard #’s: 2, 3, 7

Each standard above is mapped to the corresponding principle. I have filled out the principle the standard belongs to and why in the principle section of each standard. Please refer to that section for more information on the mappings.

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 | 12/02/2023 | Initial Security Policy Completion | Chris Blair | [Insert text.] |
| [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 |