**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 for 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 | Validated Input data is considered already an extraordinarily strong defense for any system. Things like buffer overflows need a good defense. Since this course is about C++ the best practice for buffer overflow is validated input data. CERT C++ will maintain and show validating input data with examples. |
| 1. Heed Compiler Warnings | At high warning levels in compliance with “MSC00-C” causes a warning to be displayed. When converting from an unassigned character to a constant character when char is called. CASTS are required to eliminate these warnings, but excessive casts can make code difficult to read and hide legitimate warning messages. When doing buffer overflow, pay attention to all warnings displayed. |
| 1. Architect and Design for Security Policies | Security policies can be both implicit and explicit to the system. Security policies that are considered documented and well-known while still being visibly enforced will help the expected user behavior. Security is particularly useful in the S.D.L.C. to contain a valid yet wise strategy. Capturing the approach on architect and design implement splitting the system into distinct intercommunicating subsystems. They would need each to contain a certain appropriate set of privileges. |
| 1. Keep It Simple | To keep it simplified and less over the place start by needing to make sure every class is given a one responsibility while still removing all redundant behaviors that have not been used. Even remove the redundant instances, methods to help the processes. Lastly you can start by adding clarity to your code which anyone should do. Giving variables descriptive names and using comments in your system will help you and the next person save time on your code. |
| 1. Default Deny | Default Deny is denying permissions. Of course, by default. Whenever the protection deems permission is not authorized it will not grant access. The process is provided by blocking and protecting the firewall policy. There are even ties into SQL server allowing the unaccustomed users added to execute permissions. |
| 1. Adhere to the Principle of Least Privilege | Least privilege is when every program and every user of the system should be operating using the least amount of set privileges necessary to complete what needs to be done. This is smart because the strategy for mitigating vulnerabilities have withstood the test of time. Here is an example, if the process that contains vulnerabilities is not more privilege than the attacker then there is little to be gained by exploiting the vulnerabilities. Consider your system is running with elevated privileges and still accessing files within shaved or user directories which end up being a chance it might be exploited to perform when user does not have authorized privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing the data is before invoking a function in a different logical unit which cannot understand the context. The Best practices are data sanitization in your code. Analyzing data before sending it off helps prevent things like probing, snooping, and spoofing any attacks. You can do this with examining and validating data that is exchanged. |
| 1. Practice Defense in Depth | Defense in depth otherwise labeled as DiD is to manage risk with multiple defensive strategies given that one layer of defense turns out to be inadequate. As it is that the other layer of defense can prevent a security flaw from happening. The best alternative is to rely on one single strategy. For example, in best practices, IT maintenance would implement firewalls, intrusion detection, antivirus/antimalware, VPN or even virtual machine. When you complete input validation it could hypothetically take away the need for other defenses in the system. Yet every now and then rather than completely relying on the given mitigation strategy it is often wise and smart to help you follow defense in depth tactic. |
| 1. Use Effective Quality Assurance Techniques | To use quality assurance techniques you apply any available tools, the process, and techniques. This is while it still is in the discovery and prevention of the integer vulnerabilities. Static analysis is useful for finding error in your code. We can even use CppCheck on compiled code that is checking for undefined behaviors (Example dead pointers and out of bounds checking). Even seeing the system follows user requirements by approaches like SCRUM even AGILE |
| 1. Adopt a Secure Coding Standard | The best secure coding standard you can take is following CERT C++. This is a useful standard that provides explanation to guidelines that will help it be incorporated into the integration process. To adopt for things like input and output, error handling, and much more you can secure code by applying a better and yet secure code. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | (INT32) Obey the one def. rule |

| **Noncompliant Code** |
| --- |
| Defining the class of the same location with different definitions. |
| Struct type\_name{  Member\_type1  };  class student{  public:  member\_name1;  }; |

| **Compliant Code** |
| --- |
| Inducted into both translation units |
| Struct type\_name  {  Member\_type1 member\_name1;  } object\_names;  Struct student  {  Int roll;  String name;  }; |

**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** |
| --- | --- | --- | --- |
| Astree | 24.04 | **integer-overflow** | Fully checked |
| CodeSonar | 8.1p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| Coverity | 2017.07 | **TAINTED\_SCALAR**  **BAD\_SHIFT** | Implemented |
| Helix QAC | 2024.2 | **C2800, C2860**  **C++2800, C++2860**  **DF2801, DF2802, DF2803, DF2861, DF2862, DF2863** |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-001-CPP] | (INT30) Reserving the identifier of the data incorrectly can cause issues in the system. |

| **Noncompliant Code** |
| --- |
| Needs not being met. Causing an undefined behavior for the system. Hence when a value is too big to be the result. |
| Int main(){  Int dataValue =95;  If dataValue  Std::cout << “Data Value Accepted” << std::endl;  } |

| **Compliant Code** |
| --- |
| making it initialized |
| Int main(){  Int dataValue =55;  Int maxValue = std::numeric\_limits,int.::max();  If (dataValue <= maxValue){  Std::cout << “Data Value Accepted” << std::endl;  } else {  Std::cout << “Data value goes past the limit” << std::endl; |

**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** |
| --- | --- | --- | --- |
| codeSonar | 8.1p0 | **JAVA.MATH.APPROX.E** **JAVA.MATH.APPROX.PI** **JAVA.CAST.FTRUNC** **JAVA.ARITH.FPEQUAL** | Approximate e Constant (Java) Approximate pi Constant (Java) Cast: Integer to Floating Point (Java) Floating Point Equality (Java) |
| Parasoft jtest | 2024.1 | **CERT.NUM12.CLP** | Do not cast primitive data types to lower precision |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | STR30)Four common errors are unbounded string copies, off-by-one errors, mention errors, and string truncation. Things like different string sizes are then required as well as negative buffer boundaries which will alert of buffer overflow to the system. |

| **Noncompliant Code** |
| --- |
| There is no check for alphabetic character. |
| bool isAlphaString(const sting& str) {    }  Int main() {  string input;  cout << “Enter a sting: “;  getline(cin, input); |

| **Compliant Code** |
| --- |
| There is no overflow in this example. |
| bool isAlphaString(const sting& str) {  For (char ch : str) {  If (!isAlpha(ch)){  Return false;  }  }  Return true;  }  Int main() {  string input;  cout << “Enter a sting: “;  getline(cin, input); |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| high | probable | high | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| codeSonar | 8.1p0 | **ALLOC.UAF** | Use after free |
| Helix QAC | 2024.2 | **DF4746, DF4747, DF4748, DF4749** |  |
| Parasoft C/C++ test | 2023.1 | **CERT\_CPP-STR52-a** | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| Polyspace Bug finder | R2024a | **CERTC++:STR52-CPP** | Checks for use of invalid string iterator (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | (STR31)SQL injection vulnerabilities arise in applications where elements of a SQL query originate from an untrusted source. The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures. |

| **Noncompliant Code** |
| --- |
| information can be accessed directly |
| Sql::resultSet \*res = pstmt->executeQuery();  While (res->next()) {  Std::cout << “user: “ << res->getstring(“username”) << std::endl;  } |

| **Compliant Code** |  |
| --- | --- |
| the list is acceptable char and can be sanitized |  |
| Pstmt = con->prepareStatment(“Select \* From users where username = ? and password = ?”);  Pstmt->setString(1, “user\_input\_username”);  Pstmt->setString(2, “user\_input\_password”); |  |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| high | likely | medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| astree | 24.04 |  | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| Axivion Bauhaus suite | 7.2.0 | **CertC-STR31** | Detects calls to unsafe string function that may cause buffer overflow Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| CodeSonar | 8.1p0 | **LANG.MEM.BO** | Buffer overrun Type overrun No space for null terminator A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Compass/ROSE |  |  | Can detect violations of the rule. However, it is unable to handle cases involving strcpy\_s() or manual string copies such as the one in the first example |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | (MEM08) Memory protection uses new and then delete rather than raw memory allocation and deallocation to help truly avoid memory errors that will be popping up. |

| **Noncompliant Code** |
| --- |
| Added more than needed the solution isn’t waiting for memory to require deallocate. |
| #inlcude <new>  struct i {  void f();  }; |

| **Compliant Code** |
| --- |
| We don’t need to do anything just wait until the memory isn't given requirements to deallocate. |
| #include <new> |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| high | likely | medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| 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 |
| Astree | 22.10 | **invalid\_dynamic\_memory\_allocation dangling\_pointer\_use** | [Insert text.] |
| CodeSonar | 8.1p0 | **ALLOC.FNH ALLOC.DF ALLOC.TM ALLOC.LEAK** | Free non-heap variable Double free Type mismatch Leak |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-MEM51** | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | (MSC11) The goal is to prevent information leaks when passing a class object across a trusted boundary. If it isn’t verified data it could cause unneeded issues. |

| **Noncompliant Code** |
| --- |
| Code can change nothing is asserted. |
| Int main () {  Int x = 7;  X = 9; |

| **Compliant Code** |
| --- |
| Asserting X into it still being 7 |
| Int main() {  Int x = 7;  X = 9  Assert(x == 7); |

**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 | high | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Findbugs | 2.0.3 | **MSF\_MUTABLE\_SERVLET\_FIELD MTIA\_SUSPECT\_STRUTS\_INSTANCE\_FIELD MTIA\_SUSPECT\_SERVLET\_INSTANCE\_FIELD** | implemented |
| Fotify | 6.10.0120 | **Singleton\_Member\_Field** | implemented |
| sonarQube | 9.9 | S2226 |  |
|  |  |  |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | (ERR56) Exceptions can be used to check for errors that are in the input validation on any of the parameters of public functions displayed. if a function has happened to be reannounced during its initialization phase. Any interruptions will most likely cause undefined behavior. |

| **Noncompliant Code** |
| --- |
| Can’t catch exceptions being |
| #include <thread>    void throwing\_func() noexcept(false);    void thread\_start() {    throwing\_func();  }    void f() {    std::thread t(thread\_start);    t.join();  } |

| **Compliant Code** |
| --- |
| This would handle all exceptions in thrown |
| #include <thread>    void throwing\_func() noexcept(false);    void thread\_start(void) {    try {      throwing\_func();    } catch (...) {      }  }    void f() {    std::thread t(thread\_start);    t.join();  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| low | probable | medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-ERR51** |  |
| CodeSonar | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| Helix QAC | 2024.2 | **C++4035, C++4036, C++4037** |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **API** | [STD-008-CPP] | Each imported API has its reserved spot in the IAT where the address of the imported function is written by the Windows loader. |

| **Noncompliant Code** |
| --- |
| vulnerabilities being assessed |
| ++len;  size = line + len > line\_end  memcpy(line : len);  line+= len; |

| **Compliant Code** |
| --- |
| Show vulnerabilities |
| ++len;  size = line + len > line\_end  memcpy(line : len);  line+= len;  FreePic\_unload(dib);  FreePic\_closememory(mem);  FreePic\_DeInitialise();  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 | likely | high | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Precision of an integer | [STD-009-CPP] | The precision of an integer type is the number of bits it uses to represent values, excluding any sign and padding bits. for example, x86-32 where no padding bits are used, the precision of signed types is w(type)-1, while for unsigned types, the precision equals w(type). |

| **Noncompliant Code** |
| --- |
| unsigned integer has padding bits |
| #include <limits.h>    unsigned int pow2(unsigned int exp) {    if (exp >= sizeof(unsigned int) \* CHAR\_BIT) {      /\* Handle error \*/    }    return 1 << exp;  } |

| **Compliant Code** |
| --- |
| Counts the number of bits provided in the unassigned integer |
| #include <stddef.h>  #include <stdint.h>    /\* Returns the number of set bits \*/  size\_t popcount(uintmax\_t num) {    size\_t precision = 0;    while (num != 0) {      if (num % 2 == 1) {        precision++;      }      num >>= 1;    }    return precision;  } |

**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 | 24.04 |  | Supported: Astrée reports overflows due to insufficient precision. |
| CodeSonar | 8.1p0 | **LANG.ARITH.BIGSHIFT** | Shift Amount Exceeds Bit Width |
| Helix QAC | 2024.2 | **C0582**  **C++3115** |  |
| Parasoft C/C++ test | 2023.1 | **CERT\_C-INT35-a** | Use correct integer precisions when checking the right hand operand of the shift operator |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Buffer Overflow** | [STD-010-CPP] | Buffer overflows need to not be oblivious. They can’t have any error where they can be exploited to furthermore cause buffer overflow. They are usually caused by inadequately bounded loops. These loops are most commonly loops limited by an upper bound, loops limited by a lower bound, loops limited by the address of the last element of the array, the address of the first element of the array, and loops limited by the null terminator. |

| **Noncompliant Code** |
| --- |
| Std::copy() does nothing to expand the dest vector and the program will overflow the buffer on copying the very first element. |
| #include <algorithm>  #include <vector>    void f(const std::vector<int> &src) {    std::vector<int> dest;    std::copy(src.begin(), src.end(), dest.begin());    } |

| **Compliant Code** |
| --- |
| I ensured std:copy() went to the destination container that can hold all the elements being copied to it. |
| #include <algorithm>  #include <vector>  void f(const std::vector<int> &src) {      std::vector<int> dest(src.size());    std::copy(src.begin(), src.end(), dest.begin());    } |

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

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

**Threat Level**

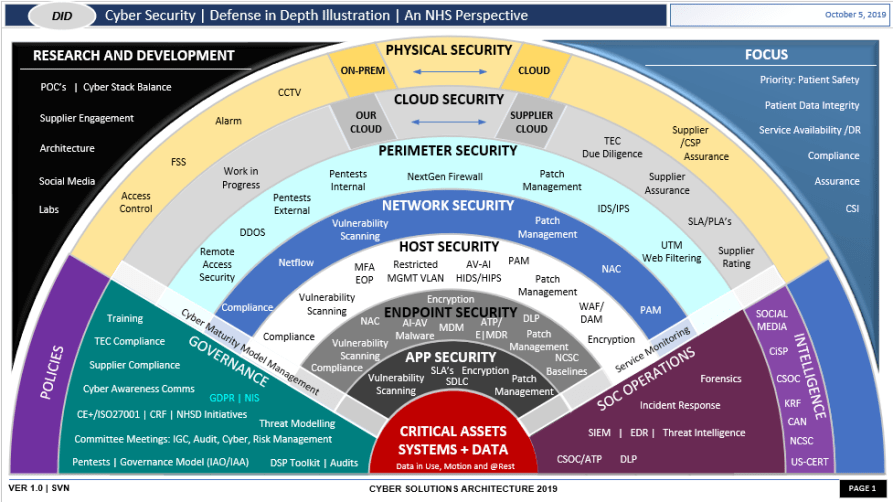
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| high | likely | medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **invalid\_pointer\_dereference** |  |
| CodeSonar | 8.1p0 | **BADFUNC.BO.\* LANG.MEM.BO LANG.MEM.TBA** | A collection of warning classes that report uses of library functions prone to internal buffer overflows. Buffer Overrun Tainted Buffer Access |
| Helix QAC | 2024.2 | **DF3526, DF3527, DF3528, DF3529, DF3530, DF3531, DF3532, DF3533, DF3534** |  |
| Parasoft C/C++ test | 2023.1 | **CERT\_CPP-CTR52-a** | Do not pass empty container iterators to std algorithms as destinations |

### 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

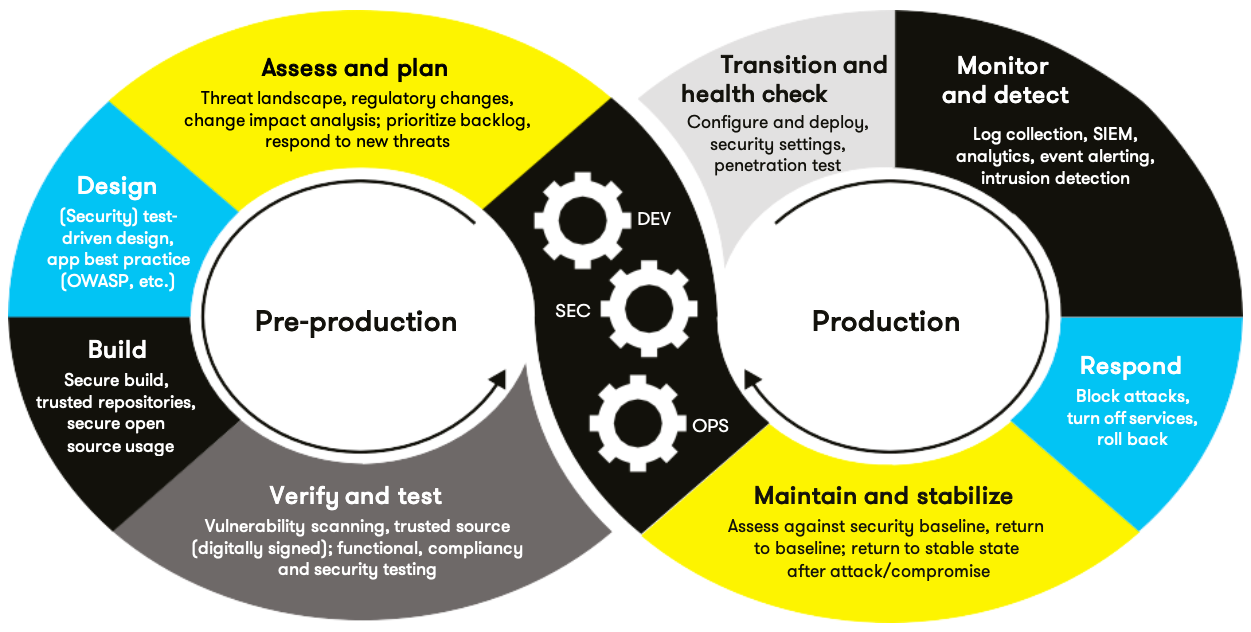
Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority,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

Defining the guidance on where and how to modify DevOps process in which to automate enforcement of the standards in the policy by making sure the measures for the security are constantly missing. Making sure security measures are in every process of DevOps toolchain. Which turns into the DevSecOps that contains create, verify, plan, preprod, adapt, release, prevent, detect, predict, and respond.

**Pre-production**

* {Verify and test} – Static application testing, Automated security, Unit test, and Integration.
* {Asses and plan} – threat modeling.
* {Build} - Automation created.
* {Design} – OWASP

**Production**

* {Maintain and stabilize} – assess after attack/compromised.
* {Transition and health check} – Configure and deploy security tests.
* {Monitor and Detection} – SEIM analysis.
* {Respond} – Turn off services.

Finally, when in production all testing for automated goes on to preventing by integrity checks and DiD. You will need to use unlimited threat protection. Ways to do that are network monitoring, performance logs, and penetration testing where these methods can protect any potential threats. Testing consistently and ever certain amount of code makes sure there is less errors that need to be attacked later. Giving the system less vulnerabilities throughout itself. Allowing no hackers or compromised issues with the system. DevSecOps and DevOps do happen to run at the same frequency. Hence that even with that DevOps additions help with SecOps with examples of unit testing. This follows into the developmental side of it. Yet when referring to the operations side DevOps will still need to do integrity checks with DiD for necessary prevention.

### 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 | likely | high | P9 | L2 |
| STD-002-CPP | Low | unlikely | medium | P2 | L3 |
| STD-003-CPP | high | probable | high | P6 | L2 |
| STD-004-CPP | high | likely | medium | P18 | L1 |
| STD-005-CPP | high | likely | medium | P18 | L1 |
| STD-006-CPP | medium | likely | high | P6 | L2 |
| STD-007-CPP | low | probable | medium | P4 | L3 |
| STD-008-CPP | medium | likely | high | P12 | L1 |
| STD-009-CPP | low | unlikely | medium | P2 | L3 |
| STD-010-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 at rest | Is the practice to protecting data. This is stored on a device like a hard drive. Does this by encoding it using encryption algorithms. The policy applies by helping to ensure important and sensitive information to keep it strictly confidential. Encryption at rest is used because it prevents all unauthorized access to the stored data. |
| Encryption in flight | I I s the encryption of data that moves over a network. When doing things on an open browser when moving data. This can be considered the public cloud implementations. The policy is done by two steps encryption and decryption. Why this is done is to prevent hackers and unauthorized users getting the information off the cloud or any way through unsecured browser. |
| Encryption in use | Is the practice of encrypting data while still being accessed or read. Why it is done is because it ensures all sensitive data is secured. No matter the lifecycle stage or location. It done by data being scrambled into a unreadable format using encryption algorithms. Also, only will authorized users be able to with the right decryption key can unscramble it. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Process of identifying any user and granting access to the network. How it is done is by username and password credentials. This is done to stop everyone from accessing unauthentic information for them self. |
| Authorization | Process enforces the network polices and user privileges. It is done because it allows to assign the tasks and activities that users are granted access too. How it is done is in the system you set permissions to the users to create authorization for appropriate access. |
| Accounting | This is the final process in the framework. Its process is all about measuring the networks. How it will collect and log data on user sessions like their time, which session chosen, and of course resource usage used. Why it is done network admin will look at users access privileges to specific resources to any changes. |

**\***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 | 09/20/2024 | SNHU M3 | Win Inlaw |  |
| 1.1.1 | 10/9/2024 | SNHU M6 | Win Inlaw |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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