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

**Enrique Zarate**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_heading=h.gjdgxs)

[Purpose 2](#_heading=h.30j0zll)

[Scope 2](#_heading=h.1fob9te)

[Module Three Milestone 2](#_heading=h.3znysh7)

[Ten Core Security Principles 2](#_heading=h.4d34og8)

[C/C++ Ten Coding Standards 3](#_heading=h.2s8eyo1)

[Coding Standard 1 4](#_heading=h.3rdcrjn)

[Coding Standard 2 5](#_heading=h.26in1rg)

[Coding Standard 3 6](#_heading=h.lnxbz9)

[Coding Standard 4 7](#_heading=h.35nkun2)

[Coding Standard 5 8](#_heading=h.1ksv4uv)

[Coding Standard 6 9](#_heading=h.2jxsxqh)

[Coding Standard 7 10](#_heading=h.z337ya)

[Coding Standard 8 11](#_heading=h.28h4qwu)

[Coding Standard 9 13](#_heading=h.nmf14n)

[Coding Standard 10 14](#_heading=h.37m2jsg)

[Defense-in-Depth Illustration 15](#_heading=h.1mrcu09)

[Project One 15](#_heading=h.46r0co2)

[1.](#_heading=h.2lwamvv) Revise the C/C++ Standards 15

[2.](#_heading=h.111kx3o) Risk Assessment 15

[3.](#_heading=h.3l18frh) Automated Detection 15

[4.](#_heading=h.206ipza) Automation 15

[5.](#_heading=h.4k668n3) Summary of Risk Assessments 16

[6.](#_heading=h.2zbgiuw) Create Policies for Encryption and Triple A 16

[7.](#_heading=h.1egqt2p) Map the Principles 17

[Audit Controls and Management 18](#_heading=h.3ygebqi)

[Enforcement 18](#_heading=h.2dlolyb)

[Exceptions Process 18](#_heading=h.sqyw64)

[Distribution 19](#_heading=h.3cqmetx)

[Policy Change Control 19](#_heading=h.1rvwp1q)

[Policy Version History 19](#_heading=h.4bvk7pj)

[Appendix A Lookups 19](#_heading=h.2r0uhxc)

[Approved C/C++ Language Acronyms 19](#_heading=h.1664s55)

## 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 | All input being provided by an external party must be verified that it is output that matches that which is allowed. When input is not validated, it provides additional opportunities for those outside the system to gain unauthorized access. Improper input validation can lead to injection attacks, memory leakage, and compromised systems. Input validation can be done by limiting characters of certain types, allowing a string of less than or greater than a set size, or detection of fishy activity. |
| 1. Heed Compiler Warnings | When running the program in a compiler or debugger, warnings may pop up from the compiler. These warnings in many cases may be pointing to possible vulnerabilities. Items should be resolved in some way to mitigate or lessen the chance of a vulnerability. Compilers may not catch all vulnerabilities but any that are detected, it is our responsibility to ensure we address the warning. |
| 1. Architect and Design for Security Policies | Software created by our company needs to be designed with security in mind. Depending on the software, implementing specific user access based on privilege levels, implementing validations. The biggest thing is not building security to the software as an afterthought. |
| 1. Keep It Simple | Don’t over complicate the design of a program. Oftentimes with software, the more complex a program becomes, the likelihood of vulnerabilities may increase as the complexity increases. Complex programs may also be harder to identify where vulnerabilities exist when having to read and understand the program itself. Simplicity will help lower chances of vulnerability and make detection of those vulnerabilities simple. |
| 1. Default Deny | All access defaults shall be set to ‘deny’. Anyone that needs access to the system will need their own set of credentials. This helps build the security into the design. Other positives coming out of this may help such as making logs easier by being able to identify changes by users or monitoring access in general. |
| 1. Adhere to the Principle of Least Privilege | All processes should utilize the least privilege needed to complete that process. Time spent in the elevated permissions should also be reduced to eliminate the chances of attackers being able to use elevated permissions in other parts of the system. |
| 1. Sanitize Data Sent to Other Systems | When sending data to other systems that may change the data to other formats, sanitization of that data needs to happen to ensure that the data being sent can be used by attackers for attacks in the other system. Data sanitization needs to be done before sending data to another system. |
| 1. Practice Defense in Depth | Whenever we build security in the system, we want to build sufficient defenses so that if one fails, there may be another way to protect the data. Multiple securities in place are always helpful but it’s always important to think that even if a successful attack occurs and all defenses have been made, reducing the consequences as much as possible. For example, if they gained access to a specific portion of the program, ensure there is no way to jump from that portion of the program straight to another. |
| 1. Use Effective Quality Assurance Techniques | Adequate testing of the software is often required to ensure the amount of vulnerabilities have been reduced as much as possible. This can be done in multiple ways but testing should also be built into the system so it is constantly checked. As more iterations of a system come up, more bugs / vulnerabilities may be present requiremening more QA testing. |
| 1. Adopt a Secure Coding Standard | There are coding standards that exist for languages already or it may need to be developed if there are specifics wanted by a specific organization. Having these sets of rules will help all staff follow coding standards in hopes that all developers will utilize the standards and in result lower the number of vulnerabilities present. |

### 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** | **Data Types Complements Variable Name** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Data types shall be chosen to reflect the variable being used. Oftentimes with injection attacks, attackers will input whatever they can even if it may not match the data type so they can bypass permissions. To help us build security into the system, we should base some protections or detection mechanisms into place based on the data types. |

| **Noncompliant Code** |
| --- |
| When a data type is assigned to a variable that does not reflect the input that should be used. |
| boolean age; |

| **Compliant Code** |
| --- |
| When a data type is appropriately chosen that represents the variable name. |
| int age; |

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

| **Principles(s):**  **Validate Input Data** - Our company needs to ensure that we are properly validating input and the data type chosen will help us be more secure in our validation input practices.  **Architect and Design for Security Policies** - Making it easy for our programmers to understand what data type is intended and where it is essential for blocking attacks from outsiders who may try to use injection or overflow attacks.  **Keep It Simple** - It is common practice to choose data types and data names best suited for their intended purpose. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | type-specifier  function-return-type  implicit-function-declaration  undeclared-parameter | Fully checked |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | MISRA C 2012 Rule 8.1 | Implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 601, 718, 746, 808 | Fully supported |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 22.04 | type-specifier  function-return-type  implicit-function-declaration  undeclared-parameter | Fully checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Values consistent with Data Type** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Values entered into the data type needs to remain the same as their intended input. In many cases, numbers can be entered into strings but if the data type is of numbers only, then only numbers should be entered along with no characters of a-z. Even though characters can be entered into integer data types, these may be converted into a number which could lead to vulnerabilities. |

| **Noncompliant Code** |
| --- |
| When input that is used for an incompatible data types that may change from the intent of the user. |
| int age;  age = ‘h’; // outputs age as 104 |

| **Compliant Code** |
| --- |
| Correctly using a value that meets the intention of the programmer. |
| int age;  age = 31; |

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

| **Principles(s):**  **Validate Input Data** - Data for these types needs to be verified to ensure they are compatible for intended use.  **Keep It Simple** - The data value needs to be consistent with the variable and type. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 7.0p0 | BADFUNC.ATOF  BADFUNC.ATOI  BADFUNC.ATOL  BADFUNC.ATOLL  (customization) | Use of atof  Use of atoi  Use of atol  Use of atoll  Users can add custom checks for uses of other undesirable conversion functions. |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Fully implemented |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | 5030 | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | CERT\_C-ERR34-a | The library functions atof, atoi and atol from library stdlib.h shall not be used |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Use of alphanumeric characters only in strings.** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Many times, vulnerabilities can come up when characters other than alphanumeric characters are included in code such as % & ^ $ @ and others. For this reason, ensure that strings used for the program only contain alphanumeric characters to avoid vulnerabilities of this kind. |

| **Noncompliant Code** |
| --- |
| When characters are used that are outside of the alphanumeric range. |
| String welcomeMessage;  welcomeMessage = “Hello there @FistName @Lastname, your room number is <room#$>.” |

| **Compliant Code** |
| --- |
| Remove any character that are anything other than the alphabet, numbers, and punctuation. During validations, if anything else shows up, reroute code to restart. |
| String welcomeMessage;  welcomeMessage = “Hello there, your room will be shown below.” |

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

| **Principles(s):**  **Validate Input Data** - All input should be validated to ensure we are not allowing characters that are unnecessary.  **Architect and Design for Security Policies** - This is a company choice in how we design the input necessary for the software.  **Practice Defense in Depth** - We want to prevent any mishaps from occurring.  **Adopt a Secure Coding Standard** - Only allowing alphanumeric entries for string type of data is more of a secure choice. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-FIO30 | Partially implemented |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | TAINTED\_STRING | Implemented |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 | N/A | Can detect violations of this rule when the -Wformat-security flag is used |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022a | CERT C: Rule FIO30-C | Checks for tainted string format (rule partially covered) |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Avoid Injections by ignoring ‘=’** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Injections often take the form of expressions to bypass unauthorized access checkpoints. Our software needs to be able to detect these by searching for characters often used in injection attacks. |

| **Noncompliant Code** |
| --- |
| Allowing the use of ‘=’ character for inputs. |
| String passcode;  passcode = “1=1”; |

| **Compliant Code** |
| --- |
| Detect the use of these characters and create an error. |
| String passcode;  passcode = “1=1”;  if (passcode uses ‘=’) {  error}  else {  pass} |

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

| **Principles(s):**  **Practice Defense in Depth** - Esure SQL Injections are prevented.  **Sanitize Data Sent to Other Systems** - Ensure items being brought in are being sanitized so that way SQL injection is less likely.  **Architect and Design for Security Policies -** Design the system so it is least likely to fall prey to SQL injection attacks.  **Validate Input Data** - User entered data is rarely to be trusted. Putting things into place to lower the risks. |
| --- |

**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) | 7.0p0 | IO.INJ.COMMAND  IO.INJ.FMT  IO.INJ.LDAP  IO.INJ.LIB  IO.INJ.SQL  IO.UT.LIB  IO.UT.PROC | Command injection  Format string injection  LDAP injection  Library injection  SQL injection  Untrusted Library Load  Untrusted Process Creation |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022a | CERT C: Rec. STR02-C | Checks for:  Execution of externally controlled command  Command executed from externally controlled path  Library loaded from externally controlled path  Rec. partially covered. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | CERT\_C-STR02-a  CERT\_C-STR02-b  CERT\_C-STR02-c | Protect against command injection  Protect against file name injection  Protect against SQL injection |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2022.2 | NNTS.TAINTED  SV.TAINTED.INJECTION |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Evaluating a pointer that has been deallocated from memory can lead to issues. Since deallocated pointers are dangling pointers, these pointers can be used as a point of entry for vulnerabilities. (From CEI CERT C++ Coding) |

| **Noncompliant Code** |
| --- |
| s is dereferenced after it has been deallocated. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

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

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

| **Principles(s):**  **Heed Compiler Warnings** - Ensure all warnings are payed attention to as this one should be indicated by the compiler.  **Adopt a Secure Coding Standard** - General good practices such as not leaving dangling pointers is essential in secure coding. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.0p0 | ALLOC.DF  ALLOC.UAF | Double free  Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022a | CERT C: Rec. MEM01-C | Checks for missing reset of a freed pointer (rec. fully covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | CERT\_C-MEM01-a  CERT\_C-MEM01-b  CERT\_C-MEM01-c  CERT\_C-MEM01-d | Do not use resources that have been freed  Always assign a new value to an expression that points to deallocated memory  Always assign a new value to global or member variable that points to deallocated memory  Always assign a new value to parameter or local variable that points to deallocated memory |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Document with Assertions** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions can be used to help with documentation. These are written into the code to verify that something is what it needs to be at the time that this command comes up and if not, can help find deficiencies in the code. |

| **Noncompliant Code** |
| --- |
| Using comments to document what variables may need to be at a point in time in the code. |
| int x = 2;  int y = 3;  int z;  z = x + y; // z should be equal to 5. |

| **Compliant Code** |
| --- |
| Rather than using comments, use assert commands to help with verifying variables at specific points in time. |
| int x = 2;  int y = 3;  int z;  z = x + y;  ASSERT(z == 5); |

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

| **Principles(s):**  **Use Effective Quality Assurance Techniques** - Assertions help to ensure our code is functioning correctly at specific checkpoints.  **Heed Compiler Warnings** - Assertions can be used as a more direct way of verifying computations being done by the compiler rather than relying on comments. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | CERT\_C-EXP20-a | Avoid comparing values with TRUE macro/enum constant using equality operators ("==", "!=") |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | No Information | Supported indirectly via MISRA C:2004 Rule 13.2. |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 114 S | Partially implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 697 | Partially supported: reports comparisons of Boolean values to constants other than 0 |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all errors and avoid aborting programs** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Errors and exceptions should be handled within the program. Aborting the program in the middle may be bad for data handling and can possibly corrupt some files for the program. |

| **Noncompliant Code** |
| --- |
| Noncompliance with this code would be not indicating the error or aborting if an error is found. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  if (error) {  abort()  }  }    **int** main() {  f();  if(error) {  abort(); }  } |

| **Compliant Code** |
| --- |
| Handling errors within the code. |
| 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):**  **Practice Defense in Depth** - Handle all errors in the program and prevent the system from crashing.  **Use Effective Quality Assurance Techniques** - All error handling needs to be checked to ensure no crashing occurs.  **Architect and Design for Security Policies & Adopt a Secure Coding Standard** - The design for error handling should always be considered in every project. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.0p0 | LANG.FUNCS.IRV | Ignored return value |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | error-information-unused  error-information-unused-computed | Partially checked |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | pointer arithmetic | Exhaustively verified. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | CERT\_C-ERR33-a  CERT\_C-ERR33-b  CERT\_C-ERR33-c  CERT\_C-ERR33-d | The value returned by a function having non-void return type shall be used  The value returned by a function having non-void return type shall be used  Avoid null pointer dereferencing  Always check the returned value of non-void function |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Ensure numeric data types do not wrap** |
| --- | --- | --- |
| Integer Wrapping | STD-008-CPP | Wrapping in numbers can lead to insufficient memory being allocated and can lead to vulnerabilities. Not only that, but it can lead to incorrect values that may mess with the rest of the program. |

| **Noncompliant Code** |
| --- |
| This code takes to integers and adds them without checking for wrapping. |
| void func(unsigned **int** ui\_a, unsigned **int** ui\_b) {  unsigned **int** usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Checks must be put into place to detect when wrapping may occur by ensuing the data type max is not within range of the addition of the two numbers. If so, an error should be listed. |
| #include <limits.h>    void func(unsigned **int** ui\_a, unsigned **int** ui\_b) {  unsigned **int** usum;  if (UINT\_MAX - ui\_a < ui\_b) {  /\* Handle error \*/  } else {  usum = ui\_a + ui\_b;  }  /\* ... \*/  } |

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

| **Principles(s):**  **Architect and Design for Security Policies** - Our company’s programmers need to be aware of the possibility of wrapping issues and should be building the program with this in mind.  **Use Effective Quality Assurance Techniques** - Testing should be done with these variables to ensure wrapping does not exists in areas where it may cause a vulnerability.  **Adopt a Secure Coding Standard** - Wrapping is a general rule and a general standard in most programming. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | integer-overflow | Fully checked |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | INTEGER\_OVERFLOW | Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | CERT\_C-INT30-a  CERT\_C-INT30-b  CERT\_C-INT30-c | Avoid integer overflows  Integer overflow or underflow in constant expression in '+', '-', '\*' operator  Integer overflow or underflow in constant expression in '<<' operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022a | CERT C: Rule INT30-C | Checks for:  Unsigned integer overflow  Unsigned integer constant overflow  Rule partially covered. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not read uninitialized memory** |
| --- | --- | --- |
| Uninitialized Memory | STD-009-CPP | This is a bit more of a complex issue but reading variables that have not been initialized was never intended to be done. In some cases, there are some default values also known as indeterminate values but often these are not meant to be read in the unitialized state which can lead to vulnerabilities depending on functions being used. It is best to avoid accessing variables that have not been initialized. |

| **Noncompliant Code** |
| --- |
| Reading an uninitialized variable. |
| #include <iostream>  void f() {  int i;  cout << i << endl;  } |

| **Compliant Code** |
| --- |
| Reading a variable after it has been properly initialized. |
| #include <iostream>  void f() {  int i = 5;  cout << i << endl;  } |

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

| **Principles(s):**  **Heed Compiler Warnings** - General warnings usually pop-up for unused variables, this is when they should be removed.  **Architect and Design for Security Policies** - Keeping Code clean and free from “litter” and unused items is important for security built-in design.  **Sanitize Data Sent to Other Systems** - Much like data being sanitized before being sent to other systems, the cleanliness of the code itself should be kept up to standard.  **Adopt a Secure Coding Standard** - This is one of the general rules in SEI Cert C++ Coding. |
| --- |

**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) | 22.04 | uninitialized-local-read  uninitialized-variable-use | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.0p0 | LANG.MEM.UVAR | Uninitialized variable |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | uninitvar  uninitdata  uninitstring  uninitMemberVar  uninitStructMember | Detects uninitialized variables, uninitialized pointers, uninitialized struct members, and uninitialized array elements (However, if one element is initialized, then cppcheck assumes the array is initialized.)  There are FN compared to some other tools because Cppcheck tries to avoid FP in impossible paths. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | CERT\_C-EXP33-a | Avoid use before initialization |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Avoid setting limits for strings** |
| --- | --- | --- |
| String Storage | STD-010-CPP | Setting boundaries or limits on strings creates the possibility of buffer overflow vulnerabilities. Not only that, but having these added and the need to have range checks or buffer overflow detection methods will increase the complexity of the code that may possibly add more vulnerabilities. |

| **Noncompliant Code** |
| --- |
| Setting a limit on a string array. |
| #include <iostream>  void f() {  char welcome[12];  cin >> welcome;  } |

| **Compliant Code** |
| --- |
| Properly using string to auto allocate size based on items placed into string. |
| #include <iostream>  #include <string>  void f() {  string input1;  string input2;  cin >> input1;  cin >> input2;  } |

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

| **Principles(s):**  **Adopt a Secure Coding Standard** - This is one of the general rules for SEI C++ Cert Secure Coding.  **Architect and Design for Security Policies** - Ensuring the variable capacity is long enough to prevent buffer overflows is critical for design with security in mind. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | mem\_access | Exhaustively verified (see one compliant and one non-compliant example). |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | No information available | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.0p0 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | 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 |

### 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.

With our DevOps process in place, we can easily identify some places where we would like to add some automatic checks into the process to help with continuing to integrate security checks into our system and move to a DevSecOps process. All of the programmers should be familiar with current standards such as SEI Cert C Coding Standards. If they are not, it’s important to have staff certified or re-trained on these standards with the basics. As they build the programs, during the verification process, we can implement automation checks with the automation checks that are best suited for our needs. Our standards show which tools are available for each of the standards and based on those lists, we can determine which tools will be the ones we need to implement that catch most of the vulnerabilities we are likely to have. Those reports will bring up the vulnerabilities that have to be addressed and can be done so before production.

Moving into the maintenance phase of the software, items will continue to be improved upon and bugs will be detected from main users and normal use. Depending on the amount of changes being made to the program after in the maintenance phase, we need to determine how often these need to be checked. I would recommend that any changes being made to the code after improvements should be ran through the automations. It is also important to have automation checks run when there haven’t been changes because updated to dependencies or the frameworks we may be using may have had some changes to them that can also affect the software. The last part of the maintenance phase, we also need to ensure that our company is utilizing error logs and monitoring for changes made by users to track any abnormal behavior. These logs can be essential in observing any unfamiliar behavior that our programmers may not be aware of or unfamiliarities that can not be detected by our chosen tools as not all tools will be 100%. It’s important to use a few of these tools so we can cover as many bases as possible. These changes should bring our current process into a more DevSecOps approach with continuing to keep security in mind.

### 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 | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Probable | Medium | P12 | L1 |
| STD-005-CPP | High | Unlikely | Low | P9 | L2 |
| STD-006-CPP | Medium | Probable | Low | P12 | L1 |
| STD-007-CPP | High | Likely | Medium | P18 | L1 |
| STD-008-CPP | High | Likely | High | P9 | L2 |
| STD-009-CPP | High | Probable | Medium | P12 | L1 |
| STD-010-CPP | High | Likely | Medium | P18 | L1 |

\*\*\* For priority, the higher the P#, the higher the priority.

### 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 | To prevent data leaks or data theft, we must encrypt our databases so the data in them is safe and protected. Our users will have access to the keys that will allow them to decrypt and view files for productive work flows. Data that would be considered high priority items such as protected health information may need extra protection with an additional layer of encryption or encryption with keys that are only given to specific users that need that information. |
| Encryption at flight | All files being sent between employees / vendors / users through electronic use, communication channels need to be encrypted. This is to prevent man in the middle attacks where someone may be trying to retrieve files in transit. Use of secure sockets layers or transport layers security will be able to help this. Another way we can keep items safe is by ensuring only staff and users have the required keys and certificates to view the files between our company and it’s users. |
| Encryption in use | Encryption will be put into place where the database will continue to be encrypted at all times. When items in the database need to be used, they will be pulled from the database and once opened by the user, the file being used will be decrypted in order to be updated. The key here is that the database itself will never be decrypted. This helps prevent an attack from accessing the whole database in the case that a user is in the middle of a project, attackers may have access to the current file in use but not the entire database. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the policy that covers general user access to our system. All of our permitted users need to have some form of sign-in to look into the application. It also provides a way for security to monitor those coming and going into the software. All users will be given some form of login and password and we need to consider some form of multifactor sign-ins for added protections such as code on phones of those customers and such. This policy is to ensure that only users with proper access are allowed in and lower the chances of ungranted accesses. |
| Authorization | A set of predetermined accesses will be created to ensure that employees are only given access to portions of the software they need. Not all employees will need access to the entire system at any one time with the exception of some of the managers. We need to follow the standard of deny by default. This will lessen the chances of attackers using specific users access to access the entire system. |
| Accounting | Policies will be put into place that ensure that there are multiple ways to monitor what is being done in the software. For example, the database needs to ensure changes are monitored by users that are logged in by including items such as time, data, user number, and items that were changed. This will help our security team and users track abnormal behavior in the case that any attackers are able to access the system and make unpermitted changes. |

### 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 | 07/17/2022 | Define Top 10 Standards | Enrique Zarate |  |
| 1.2 | 08/07/2022 | Complete Risk Assessment and Set Priorities | Enrique Zarate |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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