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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | This principle is for ensuring that the data entering a system is valid and meets the specific requirements and rules of the application/. Validating input will prevent injection attacks, malicious malware, etc.. |
| 1. Heed Compiler Warnings | This is where we as developers want to compile our code using the highest warning level possible. In this principle we will fix errors for the code and eliminate warnings because they might be vulnerable to attack. |
| 1. Architect and Design for Security Policies | The architect for our software needs to implement the security policies for all user levels. This is where we need to ensure that users do not have access to higher roles than they should be allowed to access. |
| 1. Keep It Simple | The software should be as simple as possible and avoid complexity which could result in difficult readability and maintainability. |
| 1. Default Deny | This principle is that by default access to the system is denied because it is based off permissions. |
| 1. Adhere to the Principle of Least Privilege | When processes are being executed, they should operate or function with the least set of privileges possible. |
| 1. Sanitize Data Sent to Other Systems | Data needs to be cleaned (sanitized) before being processed further to prevent the possibility of exposing vulnerabilities. |
| 1. Practice Defense in Depth | This principle is for implementing multiple layers of defense so that there is not a single point of failure in the security of your software. This will prevent hackers from being able to access sensitive data if they break through one layer because there will be more layers to exploit. |
| 1. Use Effective Quality Assurance Techniques | This is where the code/software needs to be tested and or audited periodically to expose vulnerabilities so that they can be fixed. Some techniques that can are very vital to software security is pen testing, auditing, user acceptance testing, etc.. |
| 1. Adopt a Secure Coding Standard | The developers team needs to have a set standard of secure coding techniques so that they can be practiced and followed through the entire development lifecycle. |

### 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** | **Never Qualify a Reference Type with a Const or a Volatile** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | C++ does not allow the change of reference types once it is initialized to an object, so doing so would result in redundancy due to implementing unnecessary code.  https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL52-CPP.+Never+qualify+a+reference+type+with+const+or+volatile |

| **Noncompliant Code** |
| --- |
| The const should be referenced as const-qualified char instead of the below code. |
| **void** f(**char** c) {  **char** &**const** p = c;    p = 'p';    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| The code below has removed the const qualifier |
| **void** f(**char** c) {  **char** &p = c;    p = 'p';    std::cout << c << std::endl;  } |

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

| **Principles(s):** **4 (Keep it Simple)** – We want the code to be as least complex as possible so that we maintain good quality for the code written.  **2 (Heed compiler Warnings)** – All warnings need to be addressed and fixed to ensure integrity of the code and data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT C++:DCL52-CPP | Checks for const qualified reference types and modification of const qualified reference types |
| Clang | 3.9 | N/A | Checks for violations of this rule and produces an error without the need to specify any special flags or options |
| SonarQube | 4.10 | S3708 | Provides reports for the quality of the code |
| Helix QAC | 7.2.0 | C++0014 | Prioritizes coding issues based on the severity of the risk |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do Not Access an Object Outside of Its Lifetime** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | We do not want to access the object outside of its lifetime because that can lead to the corruption of the data.  https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP54-CPP.+Do+not+access+an+object+outside+of+its+lifetime |

| **Noncompliant Code** |
| --- |
| Trying to call an object before the beginning of the pointers lifetime. |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s;    s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| The storing for the pointer has been obtained prior to calling the pointer. |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s = **new** S;    s->mem\_fn();  **delete** s;  } |

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

| **Principles(s):** **2 (Heed compiler warnings)** – Warnings about the memory access and pointers must be addressed.  **3 (Architect and Design for Security Policies)** – The design needs to meet the specifications for the objects and memory so that objects and memory cannot be accessed outside of their intended usage.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Return-reference-local  Dangling\_pointer\_use | Partially Checked |
| Clang | 3.9 | -Wdangling-initializer-list | Catches some lifetime\_issues to incorrect use of std::initializer\_list<> |
| CodeSonar | 8.1p0 | IO.UAC  ALLOC.UAF | Use after close  Use after free |
| Helix QAC | 2024.3 | C++4003, C++4026  DF2812, DF2813, DF2814, DF2930, DF2931, DF2932, DF2933, DF2934 | Prioritizes coding issues based on the severity of the risk |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Use valid references, pointers, and iterators to reference elements of a basic\_string** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | If we try access elements of the string incorrectly it can lead to many errors and crashes.  https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR52-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+basic\_string |

| **Noncompliant Code** |
| --- |
| The calls to insert() is undefined. |
| **void f(const std::string &input) {**  **std::string email;**    **// Copy input into email converting ";" to " "**  **std::string::iterator loc = email.begin();**  **for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {**  **email.insert(loc, \*i != ';' ? \*i : ' ');**  **}**  **}** |

| **Compliant Code** |
| --- |
| The iterator that is incorrect/invalid is never accessed. |
| **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

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

| **Principles(s):** **2 (Heed complier warnings)** – All warnings should be fixed and addressed.  **3 (Architect and Design for security policies)**  - The usage and design of the memory must be correct so that referenced do not lead to out-of-bound access.  **6 (Adhere to Principle of least Privilege)** – We do not want users to be able to access data such or be able to manipulate strings in this instance and cause memory leaks or crashes.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle. |
| --- |

**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 | 2024.3 | DF4746, DF4747, DF4748, DF4749 | Prioritizes coding issues based on the severity of the risk |
| 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 | CERT C++:STR52-CPP | Checks for use of invalid string iterator(rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | We need to implement code to prevent injections from untrusted sources. If we do not implement code that has no vulnerabilities we put the company and its users at risk.  https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection |

| **Noncompliant Code** |
| --- |
| Unsanitized input argument for the username. |
| **class** Login {  **public** Connection getConnection() **throws** SQLException {      DriverManager.registerDriver(**new**              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  **return** DriverManager.getConnection(dbConnection);    }      String hashPassword(**char**[] password) {      // Create hash of password    }    **public** **void** doPrivilegedAction(String username, **char**[] password)  **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String pwd = hashPassword(password);          String sqlString = "SELECT \* FROM db\_user WHERE username = '"                           + username +                           "' AND password = '" + pwd + "'";        Statement stmt = connection.createStatement();        ResultSet rs = stmt.executeQuery(sqlString);    **if** (!rs.next()) {  **throw** **new** SecurityException(            "User name or password incorrect"          );        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| We are utilizing a placeholder for the argument. |
| **public** **void** doPrivilegedAction(    String username, **char**[] password  ) **throws** SQLException {    Connection connection = getConnection();  **if** (connection == **null**) {      // Handle error    }  **try** {      String pwd = hashPassword(password);        // Validate username length  **if** (username.length() > 8) {        // Handle error      }        String sqlString =        "select \* from db\_user where username=? and password=?";      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd);      ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");      }        // Authenticated; proceed    } **finally** {  **try** {        connection.close();      } **catch** (SQLException x) {        // Forward to handler      }    }  } |

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

| **Principles(s):** **1 (Validate Input Data)** – Input should be validated to ensure that unintended access is not gained by an attacker.  **2 (Heed compiler warnings)** – All warnings should be addressed and fixed  **3 (Architect and Design for security policies)** - The usage and design of the memory must be correct so that referenced do not lead to out-of-bound access.  **5 (Default Deny)** – Access to the system should be initially denied because it is based off permissions.  **6 (Adhere to Principle of least Privilege)** – We do not want users to be able to access data such or be able to manipulate strings in this instance and cause memory leaks or crashes.  **7 (Sanitize data sent to other systems)** - Data needs to be cleaned (sanitized) before being processed further to prevent the possibility of exposing vulnerabilities.  **8 (Practice Defense in depth)** - Implement multiple layers of defense so that there is not a single point of failure in the security of your software. This will prevent hackers from being able to access sensitive data if they break through one layer because there will be more layers to exploit.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The cheker framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Fortify | 1.0 | HTTP\_Response\_Splitting  SQL\_INJECTION\_PERSISTENCE | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do Not Access Freed Memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Accessing freed memory can result in exploits that would make the code vulnerable and a target for a malicious attack.  https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory |

| **Noncompliant Code** |
| --- |
| The freed memory is being accessed 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 deallocated memory is not being accessed. |
| #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):** **1 (Validate Input Data)** – Input should be validated to ensure that unintended access is not gained by an attacker.  **2 (Heed compiler warnings)** – All warnings should be addressed and fixed  **3 (Architect and Design for security policies)** - The usage and design of the memory must be correct so that referenced do not lead to out-of-bound access.  **5 (Default Deny)** – Access to the system should be initially denied because it is based off permissions.  **6 (Adhere to Principle of least Privilege)** – We do not want users to be able to access data such or be able to manipulate strings in this instance and cause memory leaks or crashes.  **7 (Sanitize data sent to other systems)** - Data needs to be cleaned (sanitized) before being processed further to prevent the possibility of exposing vulnerabilities.  **8 (Practice Defense in depth)** - Implement multiple layers of defense so that there is not a single point of failure in the security of your software. This will prevent hackers from being able to access sensitive data if they break through one layer because there will be more layers to exploit.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Dangling\_pointer\_use | Supported  Astree reports all accesses to freed allocated memory |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| CodeSonar | 8.1.p0 | ALLOC.UAF | Use after free |
| Cppcheck | 2.15 | USE\_AFTER\_FREE | Can detect the specific instance where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | This is only helpful to the developers when we are not performing runtime error checking.  https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression |

| **Noncompliant Code** |
| --- |
| The code below is using the assert() function to an object that is essential for the code to run correctly. |
| #include <assert.h>    **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    **int** func(**void**) {  **assert**(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**));  } |

| **Compliant Code** |
| --- |
| If the expression is a constantly only then the following conditional statement can be used. |
| **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    error "Structure must not have any padding"  #endif |

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

| **Principles(s):** **(Validate Input Data)** – Input should be validated to ensure that unintended access is not gained by an attacker.  **2 (Heed compiler warnings)** – All warnings should be addressed and fixed  **3 (Architect and Design for security policies)** - The usage and design of the memory must be correct so that referenced do not lead to out-of-bound access.  **5 (Default Deny)** – Access to the system should be initially denied because it is based off permissions.  **6 (Adhere to Principle of least Privilege)** – We do not want users to be able to access data such or be able to manipulate strings in this instance and cause memory leaks or crashes.  **7 (Sanitize data sent to other systems)** - Data needs to be cleaned (sanitized) before being processed further to prevent the possibility of exposing vulnerabilities.  **8 (Practice Defense in depth)** - Implement multiple layers of defense so that there is not a single point of failure in the security of your software. This will prevent hackers from being able to access sensitive data if they break through one layer because there will be more layers to exploit.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1p0 | Customization | User can implement a custom checker that reports uses of the asser() macro |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully Implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Do not return from a computational exception signal handler** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | If we return from a exception signal handler then we risk the possibility of corrupting memory, and terminating the program.  https://wiki.sei.cmu.edu/confluence/display/c/SIG35-C.+Do+not+return+from+a+computational+exception+signal+handler |

| **Noncompliant Code** |
| --- |
| The operation is undefined. |
| #include <errno.h>  #include <limits.h>  #include <signal.h>  #include <stdlib.h>    **volatile** **sig\_atomic\_t** denom;    **void** sighandle(**int** s) {    /\* Fix the offending volatile \*/  **if** (denom == 0) {      denom = 1;    }  }    **int** main(**int** argc, **char** \*argv[]) {  **if** (argc < 2) {  **return** 0;    }    **char** \*end = NULL;  **long** temp = **strtol**(argv[1], &end, 10);    **if** (end == argv[1] || 0 != \*end ||        ((LONG\_MIN == temp || LONG\_MAX == temp) && **errno** == ERANGE)) {      /\* Handle error \*/    }      denom = (**sig\_atomic\_t**)temp;  **signal**(SIGFPE, sighandle);    **long** result = 100 / (**long**)denom;  **return** 0;  } |

| **Compliant Code** |
| --- |
| No user-defined handler is required. |
| #include <errno.h>  #include <limits.h>  #include <signal.h>  #include <stdlib.h>    **int** main(**int** argc, **char** \*argv[]) {  **if** (argc < 2) {  **return** 0;    }    **char** \*end = NULL;  **long** denom = **strtol**(argv[1], &end, 10);    **if** (end == argv[1] || 0 != \*end ||        ((LONG\_MIN == denom || LONG\_MAX == denom) && **errno** == ERANGE)) {      /\* Handle error \*/    }    **long** result = 100 / denom;  **return** 0;  } |

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

| **Principles(s):** **(Validate Input Data)** – Input should be validated to ensure that unintended access is not gained by an attacker.  **2 (Heed compiler warnings)** – All warnings should be addressed and fixed  **3 (Architect and Design for security policies)** - The usage and design of the memory must be correct so that referenced do not lead to out-of-bound access.  **5 (Default Deny)** – Access to the system should be initially denied because it is based off permissions.  **6 (Adhere to Principle of least Privilege)** – We do not want users to be able to access data such or be able to manipulate strings in this instance and cause memory leaks or crashes.  **7 (Sanitize data sent to other systems)** - Data needs to be cleaned (sanitized) before being processed further to prevent the possibility of exposing vulnerabilities.  **8 (Practice Defense in depth)** - Implement multiple layers of defense so that there is not a single point of failure in the security of your software. This will prevent hackers from being able to access sensitive data if they break through one layer because there will be more layers to exploit.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.STRUCT.RFCESH | Return from computational exception signal handler |
| CPPcheck Premium | 24.9.0 | Premium-cert-sig35-c | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced Enforcement |
| PC-lint Plus | 1.4 | 2671, 2764 | Fully supported |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| Integers | [STD-008-CPP] | We do not want to exceed the maximum value allowed for the unsigned integers. This works together with the assignment from week 2 and buffer overflow. This can cause a security vulnerability and be exploited by hackers.  https://wiki.sei.cmu.edu/confluence/display/c/INT30-C.+Ensure+that+unsigned+integer+operations+do+not+wrap |

| **Noncompliant Code** |
| --- |
| The value from the code could result in that specific value trying to be allocated to insufficient memory. |
| **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum = ui\_a + ui\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This could uses a pre-test for the operands so that a wrap is not possible. |
| #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):** **(Validate Input Data)** – Input should be validated to ensure that unintended access is not gained by an attacker.  **2 (Heed compiler warnings)** – All warnings should be addressed and fixed  **3 (Architect and Design for security policies)** - The usage and design of the memory must be correct so that referenced do not lead to out-of-bound access.  **5 (Default Deny)** – Access to the system should be initially denied because it is based off permissions.  **6 (Adhere to Principle of least Privilege)** – We do not want users to be able to access data such or be able to manipulate strings in this instance and cause memory leaks or crashes.  **7 (Sanitize data sent to other systems)** - Data needs to be cleaned (sanitized) before being processed further to prevent the possibility of exposing vulnerabilities.  **8 (Practice Defense in depth)** - Implement multiple layers of defense so that there is not a single point of failure in the security of your software. This will prevent hackers from being able to access sensitive data if they break through one layer because there will be more layers to exploit.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle. |
| --- |

**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 |
| Axivion Bauhaus | 7.2.0 | CertC-INT30 | Implemented |
| Coverity | 2017.07 | Integer\_overflow | Implemented |
| Cppcheck Premium | 24.9.0 | Premium-cert-int30-c | Partially implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not subtract or compare two pointers that do not refer to the same array** |
| --- | --- | --- |
| Arrays | [STD-009-CPP] | If the two pointers are not referring to the same array it can cause logical errors. The pointers must be part of the same array to prevent these errors.  https://wiki.sei.cmu.edu/confluence/display/c/ARR36-C.+Do+not+subtract+or+compare+two+pointers+that+do+not+refer+to+the+same+array |

| **Noncompliant Code** |
| --- |
| The example below is trying to see how many of the elements in the array “nums” are free. |
| #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** |
| --- |
| This is utilizing the subtraction of the pointers from the same array. |
| #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):** **(Validate Input Data)** – Input should be validated to ensure that unintended access is not gained by an attacker.  **2 (Heed compiler warnings)** – All warnings should be addressed and fixed  **3 (Architect and Design for security policies)** - The usage and design of the memory must be correct so that referenced do not lead to out-of-bound access.  **5 (Default Deny)** – Access to the system should be initially denied because it is based off permissions.  **6 (Adhere to Principle of least Privilege)** – We do not want users to be able to access data such or be able to manipulate strings in this instance and cause memory leaks or crashes.  **7 (Sanitize data sent to other systems)** - Data needs to be cleaned (sanitized) before being processed further to prevent the possibility of exposing vulnerabilities.  **8 (Practice Defense in depth)** - Implement multiple layers of defense so that there is not a single point of failure in the security of your software. This will prevent hackers from being able to access sensitive data if they break through one layer because there will be more layers to exploit.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Pointer-subtraction | Partially Checked |
| Axivion | 7.2.0 | CertC-ARR36 | Can detect operations on pointers that are unrelated |
| CodeSonar | 8.1p0 | LANG.STRUCT.CUP  LANG.STRUCT.SUP | Comparison of unrelated pointers  Subtraction of unrelated pointers |
| Cppcheck | 2.15 | compartPointers | Fully implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Exclude user input from format strings** |
| --- | --- | --- |
| Input Output (FIO) | [STD-010-CPP] | Attackers can exploit user input from strings that are formatted. If this happens the attacker can view contents of data, write to memory locations, etc..  https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings |

| **Noncompliant Code** |
| --- |
| The code below shows that a function will accept untrusted data from the string “user”. |
| #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** |
| --- |
| The fprintf() was replaced with fputs() which will output the message to stderr without evaluating the contents. |
| #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): 1** **(Validate Input Data)** – Input should be validated to ensure that unintended access is not gained by an attacker.  **2 (Heed compiler warnings)** – All warnings should be addressed and fixed  **3 (Architect and Design for security policies)** - The usage and design of the memory must be correct so that referenced do not lead to out-of-bound access.  **5 (Default Deny)** – Access to the system should be initially denied because it is based off permissions.  **6 (Adhere to Principle of least Privilege)** – We do not want users to be able to access data such or be able to manipulate strings in this instance and cause memory leaks or crashes.  **7 (Sanitize data sent to other systems)** - Data needs to be cleaned (sanitized) before being processed further to prevent the possibility of exposing vulnerabilities.  **8 (Practice Defense in depth)** - Implement multiple layers of defense so that there is not a single point of failure in the security of your software. This will prevent hackers from being able to access sensitive data if they break through one layer because there will be more layers to exploit.  **9 (Use Effective Quality Assurance Techniques)** – Testing of the code will decrease the likelihood of a vulnerability.  **10 (Adopt a secure coding standard)**- This will ensure that all developers are practicing a secure coding technique through the entire lifecycle.  **4 (Keep it simple) -** The software should be as simple as possible and avoid complexity which could result in difficult readability and maintainability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion | 7.2.0 | CertC-FIO30 | Partially Implemented |
| CodeSonar | 8.1p0 | IO.INJ.FMT | Format String Injection  Format String |
| Coverity | 2017.07 | TAINTED\_STRING | Implemented |
| Cppcheck Premium | 24.9.0 | Premium-cert-fio30-c | Partially Implemented |

### 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.

By implementing automation for the enforcement and compliance of security standards at all stages of the DevOps pipeline we can ensure that the DevSecOps approach will increase security standards and efficiency. Throughout all of the stages, automation will provide assurance that security standards are being followed throughout the entire DevSecOps pipeline. These automation tools give us the ability and streamline the production of the applications without putting security at risk. From the pre-production phase to the production phase, the compliance and standards for security will be followed. For example, during the “Verify and Test” stage, we can utilize Google Unit testing as we did in our previous module to automate this testing and have confidence that all warnings, and vulnerabilities have been assessed.

### 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 | Low (3) | 3 |
| STD-002-CPP | High | Probable | High | High (6) | 2 |
| STD-003-CPP | High | Probable | High | High (6) | 2 |
| STD-004-CPP | High | Likely | Medium | High (18) | 1 |
| STD-005-CPP | Low | Unlikely | High | Low (1) | 3 |
| STD-006-CPP | Low | Unlikely | High | Low (1) | 3 |
| STD-007-CPP | Low | Unlikely | High | Low (1) | 3 |
| STD-008-CPP | High | Likely | High | High (9) | 2 |
| STD-009-CPP | Medium | Probable | Medium | Medium (8) | 2 |
| STD-010-CPP | High | Likely | Medium | High (18) | 1 |

### 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 | This specific type of encryption is used to protect data that is stored on drives such as HDD, SSD, the cloud, any virtual storage, etc. Encryption at rest should be used because it helps maintain the protection of data in the presence of an attack. If the attacker was to gain access to the data, they would be unable to translate/read it due to the encryption. This type of encryption should be applied whenever there is data that is sensitive to an organization/company being stored on one of the types of storage options such as HDD, SSD, Cloud, virtual storage, etc. |
| Encryption in flight | This type of encryption is used when data is being transmitted over a network. Encryption in flight is especially useful when transferring data over the open internet, like public clouds. The reason this type of encryption is used and is very important is because it maintains the integrity of the data as it moves between systems and keeps it from being accessible by attackers and third parties. |
| Encryption in use | This type of encryption is used when data is being actively used or input into the system. When talking about data being actively used, that can be files being uploaded/downloaded, reading/writing processes, etc. Encryption in use is extremely important because it helps prevent data leaks/breaches while data is being processed in the memory. This specific type of encryption should be applied whenever sensitive data is actively being processed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Ensures that the user, system, or outside source trying to access the network is allowed. This is applied with usernames/password, 2FA, biometrics, etc. It is applied because it is a layer of security that ensure unauthorized sources cannot access the network. The policy applies to our specific scenario because it will prevent unauthorized access to accounts and maintain the integrity of the user’s data, and the addition of new users. |
| Authorization | This is what determines the type of access a user/account has on the network or system. The way this portion of the framework works is verifying the user’s permission once they have logged in. Each user is stored into a database with specific privileges, and these are accessed once a user logs in. For example, an admin has different permissions than a normal user. This applies to our company and applications because the rules of least privilege should be applied so that a user cannot have access to sensitive data that they should not have access to. |
| Accounting | Accounting is used for keeping a record of user’s activities so that they can be used in audits and for reporting purposes. The way that is works is it logs activities that are performed by users such as login, changes to the database, files accessed by users, settings changes, etc. reason that this applies to our scenario is we want to ensure that users are not preforming activities that could be harmful to applications and user data. |

**\***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 | 11/17/2024 | Module 3 Milestone | Jacob Griffin |  |
| 1.2 | 12/8/2024 | Project One | Jacob Griffin |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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