**Green Pace Developer: Security Policy Guide**



# 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 is the process of making sure that the user’s input is clean, correct, and useful. There are many ways to make this happen some of those way include creating a list of all possible inputs, implementing specific rules to user inputs, implementing rules that reject bad input types, and checking that the data is both syntactically and semantically valid before processing it. It’s important to ensure this happens as early as possible in the data flow to catch issues. There are many ways to implement this but there are many data type validators available for usage to programmers to use. |
| 1. Heed Compiler Warnings | This is giving notice to watch for warnings in the code and not just errors as security vulnerabilities may present themselves in the warning. The best way to prevent issues from compiler warnings is to compile your code with high warnings that are more likely to catch issues. The next best step is to investigate all warning and eliminate them completely. |
| 1. Architect and Design for Security Policies | This is the idea to build your software around security policies and have them implemented through the whole process of the software. To make this happen it is important to fully understand these policies and where in the software’s life they will play a role and thus implement them at those stages. These policy’s play a crucial role in preventing known issues and should help to not only speed the development up but also issue as security vulnerabilities or prevented at all stages of the codes development. |
| 1. Keep It Simple | Keeping your work simple will help in many ways from developing code quicker to making it more secure. Simple code is easier to work with, easier to understand, and easier to fix if errors and warnings present themselves. Simple code also allows for quicker understanding of issues and then work is needed to be done to fix those issues. |
| 1. Default Deny   5. Default Deny (Cont.) | This is the process of denying all users initially and then verifying and allowing only specific set people who have been pre-approved to access that information. This makes it much easier for coders as they block all users and only grant access to specific users which helps to keep the code simple and development quick. This idea id very secure btu  due to it only allowing specific users it may be inconvenient and thus important to know when and when not to implement it depending on the security requirements and sections of the program. |
| 1. Adhere to the Principle of Least Privilege | This is the concept of only granting enough access to different users to allow them to perform their specific job. This prevents unnecessary users from gaining access to information they don’t need. It is also important in the step to do audits on a regular basis and add and remove those privileges when the tasks in those jobs change over time. In conclusion always meaning that users should only be given access to the information which is required by their job or account. |
| 1. Sanitize Data Sent to Other Systems | This is the process of removing specific information before the other information is sent to the new system. Some of the information that would be removed would be personal identifiable information, special unneeded characters, and information that is unnecessary to the system it is getting sent to. There are some tips to make this process easier such as normalizing data formats, validating and rejecting invalid data, avoid hardcoded values, and mask sensitive data. Following these steps should help to prevent unauthorized information from being sent to the wrong system. |
| 1. Practice Defense in Depth | This practice is the idea of having many layers of defense in place. This is important because of types of security may not be applicable for types of security vulnerabilities and threats while other may be made to prevent those. This also means if a security threat gets through one layer of security there are still multiple layers to catch the issues. It is also important to note that the more in-depth the defense is the more resources will be needed so it’s important to choose how much defense is needed based on what is being protected and the type of attacks that may happen on that information. |
| 1. Use Effective Quality Assurance Techniques | This technique ensures that even after the software has been developed that there are steps that ensure the code is being kept up to security standards which prevent current and new security vulnerabilities. Some examples of these include fuzz testing, penetration testing, and source code audits. This is a continuous action and should be done frequently to ensure the code is up to standards and catch issues before a security risk happens. |
| 1. Adopt a Secure Coding Standard | This is an important step to ensure standards are being met. It’s important to understand and then apply standards in secure coding so that the coding standards don’t slip, and issues start becoming unknown due to lack of standards. A lot of this comes with first initiating the standards, understanding those standards, and then being disciplined and having attention to details to follow those standards each time software is written and through the software whole life cycle. |

### 

### C/C++ Ten Coding Standards

### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | DCL-50-CPP | Name: Do not define a C-style Variadic Function  Rationalize: Trying to pass certain types of data can cause undefined behaviors which change what outputs are of the information. This shows it is important to check inputs of C-style variadic Functions |

| **Noncompliant Code** |
| --- |
| This example uses a C-style variadic function to add int together and keeps going until 0 is found. Due to how this code is set up with result in undefined behavior. Another issue here is passing a datatype other than int will cause undefined behaviors. |
| #include <cstdarg>    **int** add(**int** first, **int** second, ...) {  **int** r = first + second;  **va\_list** va;  **va\_start**(va, second);  **while** (**int** v = **va\_arg**(va, **int**)) {      r += v;    }  **va\_end**(va);  **return** r;  } |

| **Compliant Code** |
| --- |
| This example fixes the above examples issues due to the making sure that the inputs are integers as well it expands the function parameter pack into a list of values as part of a braced initializer list. |
| #include <type\_traits>    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg i, Arg j, Ts... all) {  **int** values[] = { j, all... };  **int** r = i;  **for** (auto v : values) {      r += v;    }  **return** r;  } |

| **Principles(s):** 1 (Validate Input Data) – Verifying the input data before trying to run it will ensure that there are not any unwanted actions to perform. |
| --- |

**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=222953724) | 22.10 | **Function-ellipsis** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++DCL50** | N/A |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | Cert-dc150-cpp | Checked by clang-tidy. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT. ELLIPSIS** | Ellipsis |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++2012, C++2625** | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.FUNC.VARARG** | N/A |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **41 S** | Fully Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL50-a** | Functions shall not be defined with a variable number of arguments |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: DCL50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl50cpp.html) | Checks for function definition with ellipsis notation (rule fully covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **function-ellipsis** | Fully checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**FunctionEllipsis**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-923) | N/A |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | EXP-63-CPP | Name: Do Not Rely on the value of a moved-from object  Rationalize: Many libraries’ support move semantics, but in some situations values will be inputted and moved as valid but will then put in a unspecified state after the objects value has been moved. |

| **Noncompliant Code** |
| --- |
| The example below will have issues due to the int between 0-9 having a standard output but due to the int being moved and reused there are chances for unexpected outputs to occur even when specifically activated. |
| #include <iostream>  #include <string>    **void** g(std::string v) {    std::cout << v << std::endl;  }    **void** f() {    std::string s;  **for** (unsigned i = 0; i < 10; ++i) {      s.append(1, **static\_cast**<**char**>('0' + i));      g(std::move(s));    }  } |

| **Compliant Code** |
| --- |
| This example of code works because it ensures the object is valid and specifies the state before the code accesses the information. |
| #include <iostream>  #include <string>    **void** g(std::string v) {    std::cout << v << std::endl;  }    **void** f() {  **for** (unsigned i = 0; i < 10; ++i) {      std::string s(1, **static\_cast**<**char**>('0' + i));      g(std::move(s));    }  } |

| **Principles(s):** 7 (Sanitize Data Sent to the other Systems) – This means that the information is checked before it is is being used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.MEM.NPD LANG.MEM.UVAR** | Null Pointer Dereference Uninitialized Variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **DF4701, DF4702, DF4703** | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-EXP63-a** | Do not rely on the value of a moved-from object |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: EXP63-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp63cpp.html) | Checks for read operations that reads the value of a moved-from object (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.29 | [**V1030**](https://pvs-studio.com/en/docs/warnings/v1030/) | N/A |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR-51-CPP | Name: Do not attempt to create a std::string from a null pointer  Rationalize: This is an issue because if a null pointer was passed to a function like this it would result in undefined behavior due to dereferencing the null pointer. |

| **Noncompliant Code** |
| --- |
| This example has issues if the environment variable does not exist, but the code still tries to return a null pointer resulting in undefined behavior in the program. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| This example fixes the above issue by the program checking for null before the std::string object is constructed |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {      // ...    }  } |

| **Principles(s):** 9 (Use Effective Quality Assurance Techniques) – This is to ensure the quality of the code is upheld and prevent common code quality issues. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **DF4770, DF4771, DF4772, DF4773, DF4774** | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.1 | **NPD.CHECK.CALL.MIGHT** **NPD.CHECK.CALL.MUST** **NPD.CHECK.MIGHT** **NPD.CHECK.MUST** **NPD.CONST.CALL** **NPD.CONST.DEREF** **NPD.FUNC.CALL.MIGHT** **NPD.FUNC.CALL.MUST** **NPD.FUNC.MIGHT** **NPD.FUNC.MUST** **NPD.GEN.CALL.MIGHT** **NPD.GEN.CALL.MUST** **NPD.GEN.MIGHT** **NPD.GEN.MUST** **RNPD.CALL** **RNPD.DEREF** | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023.1 | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | IDS-00-J | Name: Prevent SQL injection  Rationalize: This occurs when an attacker has the ability to enter more than one input and then the second input creates an authentic result and grants them access. |

| **Noncompliant Code** |
| --- |
| In this example the inputs are not sanitized and thus allows unsensitized input arguments for the username and then sends the input as a prepared statement. |
| **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;    **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;        PreparedStatement stmt = connection.prepareStatement(sqlString);          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        }      }    }  } |

| **Compliant Code** |
| --- |
| This example fixes the above issue by pre-sanitizing the information which in turn prevents multiple inputs being inputted. |
| **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      }    }  } |

| **Principles(s):** 10 (Adopt a Secure coding standard) – This is to ensure that SQL injection prevention practices are followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Fortify](https://wiki.sei.cmu.edu/confluence/display/java/Fortify) | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/java/Klocwork) | N/A | **SV.DATA.BOUND** **SV.DATA.DB** **SV.HTTP\_SPLIT** **SV.PATH** **SV.PATH.INJ** **SV.SQL** | Implemented |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2023.1 | **CERT.IDS00.TDSQL** | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 9.9 | [**S2077**](https://rules.sonarsource.com/java/RSPEC-2077)  [**S3649**](https://rules.sonarsource.com/java/RSPEC-3649) | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649) |
| [SpotBugs](https://wiki.sei.cmu.edu/confluence/display/java/SpotBugs) | 4.6.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** **SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM-34-C | Name: Only free Memory allocated dynamically  Rationalize: If free memory is not allocated dynamically it can result in heap corruption and other possible issues. This can occur if the developer calls free() on a pointer that is not returned by a standard memory allocation function. |

| **Noncompliant Code** |
| --- |
| The example below would cause issues due to the c\_str being set to reference either dynamically or statically which when called is erroneous. |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    **enum** { MAX\_ALLOCATION = 1000 };    **int** main(**int** argc, **const** **char** \*argv[]) {  **char** \*c\_str = NULL;  **size\_t** len;    **if** (argc == 2) {      len = **strlen**(argv[1]) + 1;  **if** (len > MAX\_ALLOCATION) {        /\* Handle error \*/      }      c\_str = (**char** \*)**malloc**(len);  **if** (c\_str == NULL) {        /\* Handle error \*/      }  **strcpy**(c\_str, argv[1]);    } **else** {      c\_str = "usage: $>a.exe [string]";  **printf**("%s\n", c\_str);    }  **free**(c\_str);  **return** 0;  } |

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

| **Principles(s):** 9 (Use Effective Quality Assurance Techniques) – This helps to ensure that the code is written well and there aren’t issues created by bad quality assurance. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **invalid-free** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM34** | Can detect memory deallocations for stack objects |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | **clang-analyzer-unix.Malloc** | Checked by clang-tidy; can detect some instances of this rule, but does not detect all |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | **ALLOC.TM** | Type Mismatch |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | Can detect some violations of this rule |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **BAD\_FREE** | Identifies calls to free() where the argument is a pointer to a function or an array. It also detects the cases where free() is used on an address-of expression, which can never be heap allocated. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **DF2721, DF2722, DF2723** | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **FNH.MIGHT** **FNH.MUST** | N/A |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **407 S, 483 S, 644 S, 645 S, 125 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MEM34-a** | Do not free resources using invalid pointers |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | N/A | N/A | Runtime analysis |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **424, 673** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | [CERT C: Rule MEM34-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem34c.html) | Checks for:   * Invalid free of pointer * Invalid reallocation of pointer   Rule fully covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.29 | [**V585**](https://pvs-studio.com/en/docs/warnings/v585/), [**V726**](https://pvs-studio.com/en/docs/warnings/v726/) | N/A |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 23.04 | **invalid-free** | Partially checked |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **unclassified ("free expects a free-able address")** | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/3168a3b1)). |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCL-03-C | Name: Use a static assertion to test the value of constant expression  Rationalize: This is a diagnostic tool that’s aids in finding and eliminating software defects that may result in vulnerabilities |

| **Noncompliant Code** |
| --- |
| The issue with the below code is assertions should be placed away from the structure in which it is being called and could cause issues with the main code due to where it was placed. |
| #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** |
| --- |
| The below code fixed the code above my intergrading it as a static assertion which is better for time and resources. |
| #include <assert.h>    **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    static\_assert(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**),                "Structure must not have any padding"); |

| **Principles(s):** 8 (Practice Defense in Depth) – This is a defense tactic to prevent defects and lower chances of vulnerabilities in the code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** | N/A |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR-51-CPP | Name: Handle all exceptions  Rationalize: The program will find an exception and default it to a handler if the system cannot fix the issue the exception will remain uncaught, and the program will likely fail to process. |

| **Noncompliant Code** |
| --- |
| The below code has issues due to there being a thrown exception but no catch handler to catch the issue and call the appropriate part of code. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| The below code fixes the issue by having a catch portion to catch the exception thrown and prevent issues. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // Handle error    }  } |

| **Principles(s):** 2 (Heed Compiler Warnings) – All warnings and issues that arise should be delt with . |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++4035, C++4036, C++4037** | N/A |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.CATCH.ALL** | N/A |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | CTR-57-CPP | Name: Provide a Valid ordering predicate  Rationalize: This helps to ensure a proper ordering requirement is met and weak ordering requirements are taken out to prevent issues in how the code is set up and run. |

| **Noncompliant Code** |
| --- |
| The below code will cause issues due to the object not adhering to the strict weak ordering requirements and due to this it will fail returning false for equivalent values |
| #include <functional>  #include <iostream>  #include <set>    **void** f() {    std::set<**int**, std::less\_equal<**int**>> s{5, 10, 20};  **for** (auto r = s.equal\_range(10); r.first != r.second; ++r.first) {      std::cout << \*r.first << std::endl;    }  } |

| **Compliant Code** |
| --- |
| This code fixes the above issue by following the strict weak ordering requirements and thus will provide valid information. |
| #include <iostream>  #include <set>    **void** f() {    std::set<**int**> s{5, 10, 20};  **for** (auto r = s.equal\_range(10); r.first != r.second; ++r.first) {      std::cout << \*r.first << std::endl;    }  } |

| **Principles(s):** 9 (Use Effective Quality Assurance Techniques) – Ensuring the quality of the code is upheld and written correctly with prevent the above issues. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3293** | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-CTR57-a** | For associative containers never use comparison function returning true for equal values |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: CTR57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr57cpp.html) | Checks for predicate lacking strict weak ordering (rule partially covered). |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input Output** | FIO-46-C | Name: Do not access a closed file  Rationalize: Make sure to check the status of a file because it is is closed and then it is called to there could be undefined behavior |

| **Noncompliant Code** |
| --- |
| The below code first closes a file and then calls to it but it cannot call to a close file so it cannot pull the needed information. |
| #include <stdio.h>    **int** close\_stdout(**void**) {  **if** (**fclose**(stdout) == EOF) {  **return** -1;    }    **printf**("stdout successfully closed.\n");  **return** 0;  } |

| **Compliant Code** |
| --- |
| The below code fixes the code above by first checking to see if the file is closed which if it is outputs a message telling the user the file is closed and if it is not, it continuous on the task it was originally tasked to do. |
| #include <stdio.h>    **int** close\_stdout(**void**) {  **if** (**fclose**(stdout) == EOF) {  **return** -1;    }    **fputs**("stdout successfully closed.", stderr);  **return** 0;  } |

| **Principles(s):** 3 (Architect and Design for security Policies) – Ensuring the design of the code is written properly will ensure that the security of the project is upheld. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **48 D** | Partially implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **2471** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | [CERT C: Rule FIO46-C](https://www.mathworks.com/help/bugfinder/ref/certcrulefio46c.html) | Checks for use of previously closed resource (rule partially covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-FIO46-a** | Do not use resources that have been freed |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **DF2696, DF2697, DF2698** | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **IO.UAC** | Use after close |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151949) | 3.11 | [**S3588**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-3588) | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **SV.INCORRECT\_RESOURCE\_HANDLING.URH** | N/A |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **USE\_AFTER\_FREE** | Implemented |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | N/A | Supported |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | N/A |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Integers** | INT-35-C | Name: Use correct integer precisions  Rationalize: This has to do with the precision of an integer based on how how many bits if information it has and if the correct size isn’t used could return incorrect information. |

| **Noncompliant Code** |
| --- |
| The code below would most likely have a issue with integer precision as there is nothing preventing it from preventing precision issues. |
| #include <limits.h>    unsigned **int** pow2(unsigned **int** **exp**) {  **if** (**exp** >= **sizeof**(unsigned **int**) \* CHAR\_BIT) {      /\* Handle error \*/    }  **return** 1 << **exp**;  } |

| **Compliant Code** |
| --- |
| This fixes the above code by implementing the popCount() function which counts the number of bits and then the precision of the integer can be set by the program or manually by the user. |
| #include <stddef.h>  #include <stdint.h>    /\* Returns the number of set bits \*/  **size\_t** popcount(uintmax\_t num) {  **size\_t** precision = 0;  **while** (num != 0) {  **if** (num % 2 == 1) {        precision++;      }      num >>= 1;    }  **return** precision;  }  #define PRECISION(umax\_value) popcount(umax\_value) |

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

| **Principles(s):** 1 (Validate Input Data) – Ensuring the input data is correct with prevent issues from compiling integer information. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | N/A | Supported: Astrée reports overflows due to insufficient precision. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.ARITH.BIGSHIFT** | Shift Amount Exceeds Bit Width |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **C0582**  **C++3115** | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-INT35-a** | Use correct integer precisions when checking the right hand operand of the shift operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | [CERT C: Rule INT35-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint35c.html) | Checks for situations when integer precisions are exceeded (rule fully covered) |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### 

### Automation



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

There are a few areas where the DevOps process can be modified to automate enforcement of the standards. The most important automation technique would be testing and can be done in numerous ways. Using the DevSecOps toolchain we can see automation testing should be completed after the build phase as the system works the testing phase will happen though the life of the software. Some examples of these tests include dynamic testing which tests the application during runtime with techniques such as penetration testing and fuzz testing which simulate attacks which help to reveal vulnerabilities. Then there is specific automated testing which includes Static application security test and dynamic application security testing which ensures both consistent and efficient security validation. Another automation tactic is using continuous monitoring with tools such as Prometheus, Grafana, and Elasticsearch which helps to track metrics, detect anomalies, and have response procedures in place through an automation sequence. These are just a few ways to implement automation into the life cycle of the software to uphold the DevSecOps toolchain.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| IDS-00-J | High | Likely | Medium | 18 – High | 1 |
| DCC-03-C | Low | Unlikely | High | 1 – Low | 3 |
| MEM-34-C | High | Likely | Medium | 18 – High | 1 |
| INT-35-C | Low | Unlikely | Medium | 2 – Low | 3 |
| FIO-46-CPP | Medium | Unlikely | Medium | 4 – Low | 3 |
| DCL-50-CPP | High | Probable | Medium | 12 – High | 1 |
| ERR-51-CPP | Low | Probable | Medium | 4 – Low | 3 |
| STR-51-CPP | High | Likely | Medium | 18 – High | 1 |
| CTR-57-CPP | Low | Probable | High | 2 – Low | 3 |
| EXP-63-CPP | Medium | Probable | Medium | 8 – Medium | 2 |

### 

### Create Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest is the practice of protecting information stored on physical devices such as hard drives, phones, laptops, etc. through encoding by using encryption algorithms. These encryptions on the devices can then be decrypted with the appropriate key and this ensures that on physical devices the information is kept secure if the device gets lost or stolen. Its also important to know that rest encryptions rely on symmetric cryptography where the same key is responsible for encryption and decryption. There are usually four different levels of at rest encryption which include application-level encryption which covers encryptions for workstations or sever hosts. There are also Database encryptions which provide encryptions for some or all parts of a physical database. Next, there is file system encryption which covers system file systems which allows suers to boot into a system but needs a password to unlock certain part of the file system. Lastly, there is full disk encryption which converts data on an entire hard drive to a nonsensical form. This encryption type is essential in covering physical devices and data from unauthorized access and should be applied to any physical item needing extra security. |
| Encryption in flight | Encryption in flight is the practice of protecting data which travels between two points. This means that the data is encrypted before transfer and then once it arrives it is decrypted. The best ways to apply this is to avoid unencrypted protocols, use TLS for https, always consider using VNP’s, and use certificates signed by a certificate authority for self-signed certificates. This should be implemented due to the chance of attacks and vulnerabilities that arise from information that is between the sender and receiver. In technical terms the information is encrypted with a 128-bit AES key before travel and then once it meets its target location the session key is extracted using the decryption with a secure hash of the password which then allows the information to be accessed. This provides a layer of security to the information that can’t be copied, captured, and intercepted when traveling between the two parties. |
| Encryption in use | Encryption in use is the practice of encrypting the data while it is actively being processed or accessed. Its important to know that in most circumstances data in use is the most vulnerable data as the data is stored in clear text in the devices memory during the length of its usage and attackers will look for the opportunity to attack this one over the other two options here are usually two types of encryption to use during this stage which include partially homomorphic encryption which allows a device to only perform one type of mathematical operation on encrypted data and cannot support more than one operation at a time. The other one being fully homomorphic encryption which allows encryption for arbitrary sequences which is more versatile than PHE but also has higher computational requirements so it’s important for the user to understand the pros and cons. There are challenges to encryption in use which include computational overhead, limited operations, data size increase, limited protection against authorized entities, compatibility challenges, and key management complexity. This policy must apply to ensure information is safeguarded while it is in use and at its most vulnerable stage. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the first step in the Triple-A framework and has the task of identifying the user. During this step the system checks to see if the user has the proper credentials to enter and access the systems. This can be done by matching the credentials to the account database information which unlocks their account information if it matches correctly. In most databases there are three types of authentications which include secret known information such as a password, a physical security item such as a credential card or usb, and a personal physical identification item such as a eye or finger printer scan. For many an active directory is used as the database to store and analyze these credential types. This policy applies because it allows the initial access to users and checks to ensure the proper user is accessing the right user’s information. |
| Authorization | Authorization is the next step after access which determines the level of access to the system the specific user has and helps o enforce policy. This information is also stored in the database and the level of access is specified by an admin account there. This policy must be applied for many reasons one is it restricts access to sensitive items that not all member needs access to, it also ensures that only specific account such as admin account can change access to other members, and it also ensures that users aren’t trying to access information that may slow down the system. During this stage it’s important to implement restrict all access which means that every user has no access until changed specifically by an admin account, this makes its easier to control access, and lower chances of accidental granted access. It’s important to know that all steps but this step specifically audits should occur routinely as users will need and loss access depending on there role and make updates according to that information. |
| Accounting | Accounting is the last process which includes a few important steps such as measuring the usage of system resources by users. In this step things such as login time, data sent, data received, logout times, IP addresses, and URI are tracked. This information is sent once the user logs out and is stored to help admins with authorization control, billing, and resource utilization. This policy applies because it helps organizations see statistics on user usages, ensure proper access is bring done, see if there are anomalies to user operations, and many more user specific items that come from the information collected during this stage. |

### 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 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 | N/A |
| 1.1 | 03/24/2024 | Inputted Module Three Milestone Information | Andre Burton | N/A |
| 1.2 | 04/13/2024 | Inputted Project One Information | Andre Burton | N/A |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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