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



# Green Pace Secure Development Policy

## Contents

[Green Pace Secure Development Policy 0](#_Toc87532843)

[Contents 1](#_Toc87532844)

[Overview 3](#_Toc87532845)

[Purpose 3](#_Toc87532846)

[Scope 3](#_Toc87532847)

[Module Three Milestone 3](#_Toc87532848)

[Ten Core Security Principles 3](#_Toc87532849)

[C/C++ Ten Coding Standards 4](#_Toc87532850)

[Coding Standard 1 5](#_Toc87532851)

[Coding Standard 2 7](#_Toc87532852)

[Coding Standard 3 9](#_Toc87532853)

[Coding Standard 4 11](#_Toc87532854)

[Coding Standard 5 14](#_Toc87532855)

[Coding Standard 6 16](#_Toc87532856)

[Coding Standard 7 18](#_Toc87532857)

[Coding Standard 8 20](#_Toc87532858)

[Coding Standard 9 22](#_Toc87532859)

[Coding Standard 10 24](#_Toc87532860)

[Defense-in-Depth Illustration 26](#_Toc87532861)

[Project One 26](#_Toc87532862)

[Revise the C/C++ Standards 26](#_Toc87532863)

[Risk Assessment 26](#_Toc87532864)

[Automated Detection 26](#_Toc87532865)

[Automation 26](#_Toc87532866)

[Summary of Risk Assessments 27](#_Toc87532867)

[Create Policies for Encryption and Triple A 27](#_Toc87532868)

[Map the Principles 28](#_Toc87532869)

[Audit Controls and Management 29](#_Toc87532870)

[Enforcement 29](#_Toc87532871)

[Exceptions Process 29](#_Toc87532872)

[Distribution 30](#_Toc87532873)

[Policy Change Control 30](#_Toc87532874)

[Policy Version History 30](#_Toc87532875)

[Appendix A Lookups 30](#_Toc87532876)

[Approved C/C++ Language Acronyms 30](#_Toc87532877)

## 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 | The first security principal is to validate input data. This is a way to prevent vulnerabilities such as bufferoverflow. The key to this principal is to validate all inputs, whether that be from untrusted data sources or from user input. |
| 1. Heed Compiler Warnings | The second security principal is heed compiler warnings. Whatever IDE you may be using, you want to enable the compiler warnings and set them to their highest available setting. Then, once the program is built, will display any number of warnings about potential vulnerabilities in the code. For example, in Visual Studio, these are the yellow warning signs followed by a brief description and sometimes solutions. This allows the programmer to correct these issues to provide a more secure code. |
| 1. Architect and Design for Security Policies | The third principal is architect and design for security policies. This policy has the programmer creating a software architecture and designing software to implement and enforce security policies. This security policy can take into account such as dividing the communication of the program into various subsets and then applying the principal of least privilege to allow unprivileged systems to still communicate with elevated privileges to ensure the overall security of the system. |
| 1. Keep It Simple | The fourth principal is keep it simple. This is the mantra of “keep it simple stupid” or KISS. This principal advises the developers to keep the design of the program as simplistic as possible. This can be anything from standard naming conventions for variables, commenting on codes, etc. If the program is too complex, the likelihood that vulnerabilities and errors will be included increases. |
| 1. Default Deny | The fifth principal is default deny. This means that the default setting is to deny permission and then utilize a need to know type scheme to determine when access should be authorized. This can be applied to any number of security concerns from firewall access, to changing filer permissions in the command line using commands such as CHMOD 777 to give read and write access when in all actuality it is not needed. |
| 1. Adhere to the Principle of Least Privilege | The sixth principal is adhere to the principal of least privilege. This is the principal that seeks to limit a user to the least amount of privilege necessary to complete a task. If elevated privileges are required, they are granted based on a standard set of guidelines and only granted while completing a specific task. This principal goes hand in hand with the default deny principal by starting from a more restrictive state. |
| 1. Sanitize Data Sent to Other Systems | The seventh principal is sanitize data sent to other systems. This principal seeks to sanitize any date being sent to another system or subsystem to remove unnecessary data or eliminate vulnerabilities that may be passed to that system. This principal will help prevent any type of injection attack and therefore will not compromise the other systems or subsystems |
| 1. Practice Defense in Depth | The eight principal is practice defense in depth. This principal will adhere to a set of redundant security systems that will minimize vectored attacks by ensuring there is not a single point of failure in the security of a system. |
| 1. Use Effective Quality Assurance Techniques | The nineth principal is use effective quality assurance techniques. This principal is implemented typically by a QA team that is dedicated to finding and eliminating vulnerabilities before a product is released. This can save time, money and reputation for the company and hopefully prevent any unintended consequences. |
| 1. Adopt a Secure Coding Standard | The tenth principal is adopt a secure coding standard. This principal is a guideline that every team member should adhere to and is a set standard for coding in the language the program is being written in. This will ensure conformity |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not define a C-style variadic function:  The declaration of a C-style variadic function that is never defined is permitted, as it is not harmful and can be useful in unevaluated contexts. This can potentially present vulnerabilities as the arguments are not checked |

| **Noncompliant Code** |
| --- |
| This noncompliant code utilizes C-style variadic functions to add a series of integers together |
| #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** |
| --- |
| In this compliant solution, a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. |
| #include <type\_traits>    **template** <**typename** Arg, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Arg s) { **return** f + s; }    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Ts... rest) {  **return** f + add(rest...);  } |

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

| **Principles(s):**  1. Validate input - This principal ensures that proper inputs are entered  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**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) | 20.10 | Function-ellipsis | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-DCL50 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | Cert-dcl50-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.1p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++2012, C++2625 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2021.3 | [**MISRA.FUNC.VARARG**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [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) | 2021.2 | 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) | R2021b | CERT C++: DCL50-CPP | Checks for function definition with ellipsis notation (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | 2012, 2625 |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | Function-ellipsis | Fully Checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | FunctionEllipsis |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not declare or define a reserved identifier:  Declaring or defining an identifier in a context in which it is reserved results in undefined behavior. Do not declare or define a reserved identifier. |

| **Noncompliant Code** |
| --- |
| Utilizing reserved names that clash with header guards. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| Removing leading and trailing underscores alleviates noncompliant code. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

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

| **Principles(s):**  2. Heed compiler warnings - This principal ensures that compiler warnings are observed and corrected to eliminate vulnerabilities  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | reserved-identifier | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-DCL51 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro  -Wuser-defined-literals | The-Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverthing. This flag does not catch all instances of this rule, such as redefining reserved names |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.1p0 | LANG.ID.NU.MK  LANG.STRUCT.DECL.RESERVED | Macro name is C keyword  Declaration of reserved name |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++5003 |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 86 S, 218 S, 219 S, 580 S | Fully Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | CERT\_CPP-DCL51- a  CERT\_CPP-DCL51- b  CERT\_CPP-DCL51- c  CERT\_CPP-DCL51- d  CERT\_CPP-DCL51- e  CERT\_CPP-DCL51- f | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl51cpp.html) | Checks for redefinitions of reserved identifiers (rule partially covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | 5003 |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.15 | [**V1059**](https://pvs-studio.com/en/docs/warnings/v1059/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | Reserved-identifier | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**978**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-978) |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator:  This can potentially lead to a bufferoverflow if copying data to a buffer in which the buffer is not large enough. |

| **Noncompliant Code** |
| --- |
| This unbounded input can potentially lead to bufferoverflow |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Use std::string instead of bounded array to alleviate bufferoverflow |
| #include <iostream>  #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):**  2. Heed compiler warnings - This principal ensures that compiler warnings are observed and corrected to eliminate vulnerabilities  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.1p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No Space for null terminator  Buffer overrun  Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++2835, C++2836, C++2839, C++5216 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.3 | [**NNTS.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NNTS.TAINTED**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation   Rule partially covered. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3519**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3519) |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection:  Utilize sanitization and validation to prevent untrusted data from modifying SQL query |

| **Noncompliant Code** |
| --- |
| This noncompliant code utilizes prepared statement to permit SQL injection by incorporating the unsanitized input argument username into the 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** |
| --- |
| The compliant code utilizes a parametric query with a ? character as 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 - This principal ensures that proper inputs are entered  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | 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) | 6.1p0 | **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) |  | **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) | 2021.2 | **CERT.IDS00.TDSQL** | Protect against SQL injection | |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 6.7 | [**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) | |

#### 

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory:  Pointers to memory that has be deallocated are called dangling pointers and can result in exploitable vulnerabilities |

| **Noncompliant Code** |
| --- |
| Utilizing a write-after-free example to run arbitrary code with the permissions of the vulnerable process |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| Do not deallocate until it is no longer needed |
| #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 - This principal ensures that proper inputs are entered  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**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) | 20.10 | **dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.1p0 | **ALLOC.UAF** | Use after free |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **USE\_AFTER\_FREE** | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++4303, C++4304** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2021.3 | [**UFM.DEREF.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.DEREF.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.FFM.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.FFM.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.RETURN.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.RETURN.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.USE.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.USE.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | 4303, 4304 |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.15 | [**V586**](https://pvs-studio.com/en/docs/warnings/v586/), [**V774**](https://pvs-studio.com/en/docs/warnings/v774/) |  |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Splint) | 5.0 |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression:  Static assertions are useful in detecting software defects and eliminating them |

| **Noncompliant Code** |
| --- |
| Using the assert() macro to assert a property concerning a memory-mapped structure. |
| #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** |
| --- |
| Diagnosing incorrect assumptions at compile time |
| #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"); |

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

| **Principles(s):**  1. Validate input - This principal ensures that proper inputs are entered  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**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** |  |
| [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) | 6.1p0 | **(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) |  |  | 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** | [STD-007-CPP] | Handle all exceptions:  Catch all exceptions thrown by an application |

| **Noncompliant Code** |
| --- |
| Failing to catch f () or main () exception thrown by throwing\_func () |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Utilizing the main entry point to handle all exceptions |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):**  1. Validate input - This principal ensures that proper inputs are entered  2. Heed compiler warnings - This principal ensures that compiler warnings are observed and corrected to eliminate vulnerabilities  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**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) | 20.10 | **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** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++4035, C++4036, C++4037** |  |
| [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) | 2021.2 | **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) | R2021b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4035, 4036, 4037** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | [STD-008-CPP] | Do not destroy a mutex while it is locked:  This vulnerability is exploited by destroying a mutex object while a thread is blocked waiting for the lock, which can expose critical sections and shared data |

| **Noncompliant Code** |
| --- |
| This noncompliant code example creates several threads that each invoke the do\_work() function, passing a unique number as an ID.  Unfortunately, this code contains a race condition, allowing the mutex to be destroyed while it is still owned, because start\_threads() may invoke the mutex's destructor before all of the threads have exited. |
| #include <mutex>  #include <thread>    const size\_t maxThreads = 10;    void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);    // Access data protected by the lock.  }    void start\_threads() {  std::thread threads[maxThreads];  std::mutex m;    for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  }  } |

| **Compliant Code** |
| --- |
| This compliant solution eliminates the race condition by extending the lifetime of the mutex. |
| #include <mutex>  #include <thread>    **const** **size\_t** maxThreads = 10;    **void** do\_work(**size\_t** i, std::mutex \*pm) {    std::lock\_guard<std::mutex> lk(\*pm);      // Access data protected by the lock.  }    std::mutex m;    **void** start\_threads() {    std::**thread** threads[maxThreads];    **for** (**size\_t** i = 0; i < maxThreads; ++i) {      threads[i] = std::**thread**(do\_work, i, &m);    }  } |

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

| **Principles(s):**  1. Validate input - This principal ensures that proper inputs are entered  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++4961, C++4962** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.3 | [**CERT.CONC.MUTEX.DESTROY\_WHILE\_LOCKED**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-CON50-a** | Do not destroy another thread's mutex |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: CON50-CPP](https://www.mathworks.com/help/bugfinder/ref/certccon50cpp.html) | Checks for destruction of locked mutex (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4961, 4962** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-009-CPP] | Use valid iterator ranges:  Failing to use a range of two iterators that are valid an refer into the same container can cause undefined behavior |

| **Noncompliant Code** |
| --- |
| n this noncompliant example, the two iterators that delimit the range point into the same container, but the first iterator does not precede the second. On each iteration of its internal loop, std::for\_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <algorithm>  #include <iostream>  #include <vector>    **void** f(**const** std::vector<**int**> &c) {    std::for\_each(c.end(), c.begin(), [](**int** i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the iterator values passed to std::for\_each() are passed in the proper order. |
| #include <algorithm>  #include <iostream>  #include <vector>    **void** f(**const** std::vector<**int**> &c) {    std::for\_each(c.begin(), c.end(), [](**int** i) { std::cout << i; });  } |

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

| **Principles(s):**  1. Validate input - This principal ensures that proper inputs are entered  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **overflow\_upon\_dereference** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++3802** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-CTR53-a** **CERT\_CPP-CTR53-b** | Do not use an iterator range that isn't really a range Do not compare iterators from different containers |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **3802** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.15 | [**V539**](https://pvs-studio.com/en/docs/warnings/v539/), [**V662**](https://pvs-studio.com/en/docs/warnings/v662/), [**V789**](https://pvs-studio.com/en/docs/warnings/v789/) |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-010-CPP] | Do not cast to an out-of-range enumeration value:  Ensure that the arithmetic value being cast must be within the range of values the enumeration can represent to avoid unspecified values |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {    EnumType enumVar = **static\_cast**<EnumType>(intVar);    **if** (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator value. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {  **if** (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = **static\_cast**<EnumType>(intVar);  } |

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

| **Principles(s):**  1. Validate input - This principal ensures that proper inputs are entered  3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind  4. Keep it simple - This principal ensures that the code written is as simplistic as possible  9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code  10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-INT50** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++3013** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-INT50-a** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **3013** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.15 | [**V1016**](https://pvs-studio.com/en/docs/warnings/v1016/) |  |

### 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.

[Insert your written explanations here.]

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Probable | Medium | P12 | L1 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Probable | Medium | P12 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Medium | Probable | High | P4 | L3 |
| STD-009-CPP | High | Probable | High | P6 | L2 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | This policy has to do with encrypting data as it written and stored when on a hard drive. The data must then be decrypted using a key in order to read the data. This is important because it will ensure that data is secure should the hard drive be stolen as the perpetrators will not be able to read the data without a key |
| Encryption at flight | This policy has to do with encrypting data as its being transferred. In some cases, data may be in an unencrypted form when it is at rest and therefore by encrypting the data once it is in flight, will serve as a viable security measure to ensure the data remains safe |
| Encryption in use | This policy has to do with encrypting data while being used. This is done by encrypting the data in the memory so that only valid users can access and decrypt the data. This would be similar to password protecting certain files on a hardrive |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of validating a user against a user name and password in a database. The user would login to a system and has to have verified credentials in order to pass into the system. User name and password are a basic form of authentication and more robust 2-factor authentication techniques should be used |
| Authorization | Authorization is the process of authorizing a user certain portions of a system. Each user should be given the least amount of privlges possible and therefore each user only has access to a limited portion of a system. This is important because of the simple fact that need to know applies here and there should be very few users who have complete access |
| Accounting | Accounting is the process of logging each user’s actions on the system. This is important because it will allow the system administrators to track what each user is doing once fully authenticated. This can also help troubleshoot certain things and identify security breaches or unauthorized access. |

**\***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
  + - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 5. Default Deny - Set the default to deny any traffic that has not been already approved
    - 6. Adhere to the principal of least privilege - Users are given the least number of privileges as possible to complete their task
    - 7. Sanitize data sent to other systems - Encrypting data being transferred to other systems
    - 8. Practice defense in depth - A redundant set of defensive protocols
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Firewall logs
  + - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 5. Default Deny - Set the default to deny any traffic that has not been already approved
    - 6. Adhere to the principal of least privilege - Users are given the least number of privileges as possible to complete their task
    - 7. Sanitize data sent to other systems - Encrypting data being transferred to other systems
    - 8. Practice defense in depth - A redundant set of defensive protocols
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Anti-malware logs
  + - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 5. Default Deny - Set the default to deny any traffic that has not been already approved
    - 6. Adhere to the principal of least privilege - Users are given the least number of privileges as possible to complete their task
    - 7. Sanitize data sent to other systems - Encrypting data being transferred to other systems
    - 8. Practice defense in depth - A redundant set of defensive protocols
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Data Type
  + - 1. Validate input - This principal ensures that proper inputs are entered
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Data Value
  + - 2. Heed compiler warnings - This principal ensures that compiler warnings are observed and corrected to eliminate vulnerabilities
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* String Correctness
  + - 2. Heed compiler warnings - This principal ensures that compiler warnings are observed and corrected to eliminate vulnerabilities
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* SQL Injection
  + - 1. Validate input - This principal ensures that proper inputs are entered
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Memory Protection
  + - 1. Validate input - This principal ensures that proper inputs are entered
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Assertions
  + - 1. Validate input - This principal ensures that proper inputs are entered
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Exceptions
  + - 1. Validate input - This principal ensures that proper inputs are entered
    - 2. Heed compiler warnings - This principal ensures that compiler warnings are observed and corrected to eliminate vulnerabilities
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Concurrency
  + - 1. Validate input - This principal ensures that proper inputs are entered
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Containers
  + - 1. Validate input - This principal ensures that proper inputs are entered
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities
* Integers
  + - 1. Validate input - This principal ensures that proper inputs are entered
    - 3. Architect and design for security policies - This principal ensures that the code that is created is done with security in mind
    - 4. Keep it simple - This principal ensures that the code written is as simplistic as possible
    - 9. Use effective quality assurance techniques - This principal ensures a certain level of QA and utilizes valid tests to test the code
    - 10. Adopt a secure coding standard - This principal ensures a standardized coding standard to help eliminate vulnerabilities

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 | 11/08/2021 | 3-2 Milestone: Coding Standards | Daryl Miller | Trevor Hodde |
| 1.2 | 12/1/2021 | 6-2 Project One: Security Policy | Daryl Miller | Trevor Hodde |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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