**Green Pace Developer: Security Policy**



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 | All user input should be validated thoroughly prior to performing an operation with that input. This might include removing special characters or limiting number of characters. This will help prevent SQL injections and buffer overflows. |
| 1. Heed Compiler Warnings | Warnings can be used to catch bugs before testing. This can help lower cost and the number of vulnerabilities in the code. It is always important to eliminate all warnings by correcting the code to resolve the issue. |
| 1. Architect and Design for Security Policies | This outlines how each entity can access each other and manages what specific operations each can carry out. This will also determine the level of protection that is required for a system and the actions that should be performed with the security requirements are not met. |
| 1. Keep It Simple | Avoid exposing information that is not required by keeping code simple. Reducing surface area can help prevent adding additional vulnerabilities. It is also important to keep error messages simple and not provide any more information than is required to the user. |
| 1. Default Deny | Default deny means that if something is not specifically allowed then it should be denied. Essentially you are defining the specifics of what is allowed to be accessed and then setting everything else to be denied access unless explicit permission is granted. |
| 1. Adhere to the Principle of Least Privilege | Any process that runs a script or code should be ran on an account with the least privilege to help limit potential damage that can be done in the event the process is compromised. If a malicious user is able to inject code into a process the privileges that is granted to that process can determine the operations that user can perform in the system. Anything that requires raised privileges to be executed should be isolated in a separate process. |
| 1. Sanitize Data Sent to Other Systems | This may include the removal of any unwanted characters from the input by removing, replacing, encoding, or escaping the characters. This should occur after the input is entered by prior to the data being passed. |
| 1. Practice Defense in Depth | Practicing Defense in Depth means placing check points within each layer and subsystem in the application. These will act as gatekeepers that help ensure that only authenticated and authorized users can access the next layer. |
| 1. Use Effective Quality Assurance Techniques | This focuses on improving the software development process by making it efficient and effective while up keeping quality standards for the end user. This can be done by implementing automated and manual testing of code to look for bugs and vulnerabilities prior to merging new code to the main branch or releasing it. |
| 1. Adopt a Secure Coding Standard | Secure coding standards are practices that are implemented in order to prevent security vulnerabilities like bugs and broken logic. By following these standards a programmer can significantly reduce risks of these vulnerabilities prior to the code being deployed. |

## 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-CLG] | DCL31-C. Declare identifiers before using them. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example there is no type specifier for the data. |
| extern foo; |

| **Compliant Code** |
| --- |
| In this compliant code the solution assigns the variable to be an int data type. |
| extern int foo; |

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

|  |
| --- |
| **Principles(s):** Adopt a Secure Coding Standard – It should be a standard to declare identifiers before using the them. This will help ensure that the data type of the variable is correct. While this is a low severity issue, it should still be followed as a best practice. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **type-specifier**  **function-return-type**  **implicit-function-declaration**  **undeclared-parameter** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC-DCL31** | Fully implemented |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | -Wimplicit-int |  |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **MISRA C 2012 Rule 8.1** | Implemented |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL31** | Fully implemented |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect violations of this rule when the -Wimplicit and -Wreturn-type flags are used |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2018 | [CWARN.IMPLICITINT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [FUNCRET.IMPLICIT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [MISRA.DECL.NO\_TYPE](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [MISRA.FUNC.NOPROT.CALL](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [RETVOID.IMPLICIT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **24 D, 41 D, 20 S, 326 S, 496 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | **CERT\_C-DCL31-a** | All functions shall be declared before use |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **601, 718, 746, 808** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rule DCL31-C](https://www.mathworks.com/help/bugfinder/ref/certcruledcl31c.html) | Checks for:   * Types not explicitly specified * Implicit function declaration   Rule fully covered. |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **0434 (C)** **2050** **2051** **3335** | Fully implemented |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.07 | [V1031](https://www.viva64.com/en/w/v1031/) |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151949) | 3.11 | [S819](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-819)**,**[S820](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-820) | Partially implemented; implicit return type not covered. |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 20.10 | **type-specifier**  **function-return-type**  **implicit-function-declaration**  **undeclared-parameter** | Fully checked |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **type specifier missing** | Partially verified (exhaustively detects undefined behavior). |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | INT30-C. Ensure that unsigned integer operations do not wrap. |

| **Noncompliant Code** |
| --- |
| This example can result in an unsigned integer wrap during the addition of the unsigned operands ui\_a and ui\_b. The resulting value might be used to allocate insufficient memory for an operation that can lead to an exploitable vulnerability. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {    unsigned int usum = ui\_a + ui\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In this compliant code the solution performs a precondition test to guarantee there is no unsigned wrap. |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {    unsigned int usum;    if (UINT\_MAX - ui\_a < ui\_b) {      /\* Handle error \*/    } else {      usum = ui\_a + ui\_b;    }    /\* ... \*/  } |

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

|  |
| --- |
| **Principles(s):** ValidateInput Data – In this example the compliant solution tests the operands to make sure there is no unsigned integer wrap. This is showing the importance of validating the input data to make sure it will not create any vulnerabilities. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **integer-overflow** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **ALLOC.SIZE.ADDOFLOW** **ALLOC.SIZE.IOFLOW** **ALLOC.SIZE.MULOFLOW** **ALLOC.SIZE.SUBUFLOW** **MISC.MEM.SIZE.ADDOFLOW** **MISC.MEM.SIZE.BAD** **MISC.MEM.SIZE.MULOFLOW** **MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule by ensuring that operations are checked for overflow before being performed (Be mindful of exception INT30-EX2 because it excuses many operations from requiring [validation](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validation), including all the operations that would validate a potentially dangerous operation. For instance, adding two unsigned ints together requires validation involving subtracting one of the numbers from UINT\_MAX, which itself requires no validation because it cannot wrap.) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **INTEGER\_OVERFLOW** | Implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2018 | [NUM.OVERFLOW](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [CWARN.NOEFFECT.OUTOFRANGE](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **493 S, 494 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | **CERT\_C-INT30-a** **CERT\_C-INT30-b** **CERT\_C-INT30-c** | Avoid integer overflows Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rule INT30-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint30c.html) | Checks for:   * Unsigned integer overflow * Unsigned integer constant overflow   Rule partially covered. |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **2910 [C], 2911 [D], 2912 [A],**  **2913 [S], 3383, 3384, 3385, 3386** | Partially implemented |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2910, 2911, 2912, 2913** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.07 | [V658,](https://www.viva64.com/en/w/v658/)[V1028](https://www.viva64.com/en/w/v1028/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **unsigned overflow** | Exhaustively verified. |

### 

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| The input in this code is unbounded and can lead to a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| In this example of compliant code it shows that the best solution for ensuring the data is not truncated and guarded against buffer overflows. |
| #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):** ValidateInput Data – This example is showing the importance of validating the input because if the buffer is too small to hold the data it can cause a buffer overflow. |

**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/cplusplus/CodeSonar) | 6.0p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO** **LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2018 | [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) | 2020.2 | **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) | R2020a | [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-JVA] | IDS00-J. Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| This example permits a SQL injection attack by incorporating unsanitized input into the SQL command allowing an attacker perform a SQL injection attack. |
| 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 + "'";  Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sqlString);    if (!rs.next()) {  throw new SecurityException(  "User name or password incorrect"  );  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  }  } |

| **Compliant Code** |
| --- |
| This solution uses a parametric query with a “?” character as a placeholder. The code also validates the length of the username argument preventing an attacker from submitting an excessively long user name. |
| 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):** ValidateInput Data, Sanitize Data Sent to Other Systems – This shows the importance of the principles Validate input data and sanitize data sent because by both sanitizing and validating the data it can prevent a SQL injection. Limiting and removing unwanted characters can help reduce 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) |
| [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) | 2020.2 | **BD-SECURITY-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] | MEM50-CPP. Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| In this example s is dereferenced after it has been deallocated. If this access results in a write-after-free, an exploitable vulnerability can be created that will run arbitrary code. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In the compliant solution the memory is dynamically allocated and is not deallocated until it is not required anymore. |
| #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):** Adopt a Secure Coding Standard – This shows that it is important to adopt a secure coding standard because it is important to make sure memory is not deallocated until it is no longer required in order to prevent vulnerabilities and the chance of running arbitrary code. |

**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) | 6.9.0 | **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) | 6.0p0 | **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 |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2018 | [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) | 2020.2 | **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) | R2020a | [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) | 7.07 | [**V586**](https://www.viva64.com/en/w/v586/), [V774](https://www.viva64.com/en/w/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-CLG] | MSC11-C. Incorporate diagnostic tests using assertions |

| **Noncompliant Code** |
| --- |
| This code uses assert() macro to verify the memory allocation succeeded. Since memory availability depends on the overall state of the system, using this macro might lead to abrupt termination of the process and a potential denial-of-service attack. |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;    len = strlen(c\_str);  dup = (char \*)malloc(len + 1);  assert(NULL != dup);    memcpy(dup, c\_str, len + 1);  return dup;  } |

| **Compliant Code** |
| --- |
| This code shows how to properly detect and handle possible memory exhaustion using assertions. |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;    len = strlen(c\_str);  dup = (char\*)malloc(len + 1);  /\* Detect and handle memory allocation error \*/  if (NULL == dup) {  return NULL;  }    memcpy(dup, c\_str, len + 1);  return dup;  } |

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

|  |
| --- |
| **Principles(s):** Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard – Using tests and assertions is a great way to ensure that the code is functioning properly, this can help verify that the correct outcome of the code should be doing is actually happening. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **LANG.FUNCS.ASSERTS** | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECT** | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | **CERT\_C-MSC11-a** | Assert liberally to document internal assumptions and invariants |

### 

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | ERR51-CPP. Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this example, neither f() and main() catch the exceptions being thrown, this then calls std::terminate(). |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this solution the main entry point handles the exceptions which ensures the stack is unwound up to the main function. |
| 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):** Adopt a Secure Coding Standard, Heed Compiler Warnings – This example touches on these two principles because the original code was not set up to throw an exception. It is best practices to implement exceptions and error messages to explain to the user why the code was forced to quit, or why something is invalid This also goes hand in hand with heed compiler warnings because when building code if you ignore the warnings you are leaving room for attackers to possibly exploit the code. Allowing an application to terminate abnormally can tie up resources and is often a vector for denial of service attacks. |

**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) | 6.9.0 | **CertC++-ERR51** |  |  |
| [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) | 2020.2 | **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) | R2020a | [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** |
| --- | --- | --- |
| **Memory** | [STD-008-CPP] | MEM57-CPP. Avoid using default operator new for over-aligned types |

| **Noncompliant Code** |
| --- |
| The new expression is used to invoke the default operator new to obtain storage then constructs an object of the vector type. |
| struct alignas(32) Vector {  char elems[32];  };    Vector \*f() {  Vector \*pv = new Vector;  return pv;  } |

| **Compliant Code** |
| --- |
| An overloaded operator new function is defined to obtain appropriately aligned storage by calling the C11 function aligned\_alloc(). Programs that use the array form of the new expression must define the corresponding member array operator new[] and operator delete[]. |
| #include <cstdlib>  #include <new>    struct alignas(32) Vector {  char elems[32];  static void \*operator new(size\_t nbytes) {  if (void \*p = std::aligned\_alloc(alignof(Vector), nbytes)) {  return p;  }  throw std::bad\_alloc();  }  static void operator delete(void \*p) {  free(p);  }  };    Vector \*f() {  Vector \*pv = new Vector;  return pv;  } |

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

|  |
| --- |
| **Principles(s):** ValidateInput Data – This example shows the importance of avoiding relying on the default operator new when obtaining storage for objects of over aligned types. This can cause the code to have unexpected behavior. Validating the data will allow you to ensure there is enough storage. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P6 | L2 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| Tool | Version | **Checker** | Description Tool |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-MEM57-a** | Avoid using the default operator 'new' for over-aligned types |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | [CERT C++: MEM57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem57cpp.html) | Checks for situations where operator new is not overloaded for possibly overaligned types (rule fully covered) |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-009-CPP] | INT32-C. Ensure that operations on signed integers do not result in overflow |

| **Noncompliant Code** |
| --- |
| This code can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In this example the addition operation cannot overflow. |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Adopt a Secure Coding Standard – It is important to validate the input in this code to prevent buffer flow and prevent attackers from executing arbitrary code. This should be standard to validate any user input and declare all variables to prevent these vulnerabilities. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **integer-overflow** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **ALLOC.SIZE.ADDOFLOW** **ALLOC.SIZE.IOFLOW** **ALLOC.SIZE.MULOFLOW** **ALLOC.SIZE.SUBUFLOW** **MISC.MEM.SIZE.ADDOFLOW** **MISC.MEM.SIZE.BAD** **MISC.MEM.SIZE.MULOFLOW** **MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_SCALAR**  **BAD\_SHIFT** | Implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **493 S, 494 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | **CERT\_C-INT32-a** **CERT\_C-INT32-b** **CERT\_C-INT32-c** | Avoid integer overflows Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) |  |  | Runtime analysis |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rule INT32-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint32c.html) | Checks for:   * Integer overflow * Tainted division operand * Tainted modulo operand   Rule partially covered. |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **2800, 2801, 2802, 2803,**  **2860, 2861, 2862, 2863** | Fully implemented |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2800, 2801, 2802, 2803,**  **2860, 2861, 2862, 2863** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.07 | [V1026](https://www.viva64.com/en/w/v1026/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **signed\_overflow** | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/06486475)). |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-010-CPP] | ERR55-CPP. Honor exception specifications |

| **Noncompliant Code** |
| --- |
| In this example a function is declared as nonthrowing but still possible for std::vector::resize() to throw an exception if the requested memory is unable to be allocated. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) noexcept(true) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| In this compliant code the function's noexcept-specification is removed, signifying that the function allows all exceptions. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) {  v.resize(s); // May throw, but that is okay  } |

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

|  |
| --- |
| **Principles(s):** Keep It Simple, Heed Compiler Warnings – It is important to be cautious of any warnings provided by the compiler, and also include exceptions to be thrown when there is an issue in the application. These should be expected exceptions as unexpected exceptions can disrupt the control flow and cause premature termination and denial of service. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **unhandled-throw-noexcept** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++-ERR55** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **56 D** | Partially implemented |
| [Parasoft C/C++Test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-ERR55-a** | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4035, 4036, 4632** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **unhandled-throw-noexcept** | Partially checked |

## 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.

When looking at the attached image for the DevSecOps process I think that the best choices for automation would be in the processes of verify and test, monitor and detect, and respond. I feel that during the verify and test stage it is a great time to create and run automated tests to ensure that there are no bugs or vulnerabilities prior to the code being released. There are many tools that can help with this, one that I currently use is called Virtuoso. This will allow you to build end to end tests or tests on a specific feature that will be automated to help catch any issues.

When entering the monitor and detect phase adding automation to help automate the collection of logs, analytics and alerts. This will make it easier to ensure that the correct data is collected and there are many solutions that can help collect and aggregate the data to make searching the log files easier since all of your logs from different servers, applications, or together network assets can be accessed in one place automatically.

In the Respond phase it would be ideal to have automation that can be used to quarantine any files that have been detected until someone can review and identify if the file or data has been infected by a virus or been targeted by a malicious attacker. Using a tool like this can help increase security as it can isolate these files to prevent further attacks at deeper levels of the network.

## 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-CLG | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-JVA | High | Probable | Medium | P12 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CLG | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Medium | Unlikely | Low | P6 | L2 |
| STD-009-CPP | High | Likely | High | P9 | L2 |
| STD-010-CPP | Low | Likely | Low | P9 | L2 |

## 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 | Encryption in rest was designed with the intention to prevent attackers from accessing unencrypted data that could be stored on disk. This ensures the data is encrypted while on disk so if an attacker obtains the drive with the encrypted data and they do not have access to the encryption keys they will have to defeat the encryption prior to being able to access any data. It is a good idea to use strong encryption methods on this data like AES or RSA. It is also best practice to keep this data encrypted if the incorrect username or password is entered when attempting to access this data. |
| Encryption at flight | Encryption in flight is the process of encrypting data while the data is being transmitted or moving from one location to another. Since data is generally less secure while in motion this is a critical implementation to ensure that data is not intercepted by attackers prior to reaching the destination. This protection can be achieved by being sure to use robust network security controls like firewalls and network access control. |
| Encryption in use | Encryption in use references active data that is generally stored in RAM or CPU caches/registers. It is a good idea to include things like full memory encryption, CPU based key storage or enclaves as ways to ensure that this data is encrypted while it is in use. This will help prevent malicious attacks on this data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a method of identifying each user that is accessing the network by requiring them to login with a username and password. This should be implemented in every system and should block access to users who cannot properly authenticate. |
| Authorization | Authorization is the next step that should be executed prior to gaining access to the network. This will check that the Authenticated user has the correct level of access. Each user should be able to have different levels of accessed assigned that will prevent them from accessing or changing any information in the database without the proper level of authorization. This might include restricting the ability to add a new user to those with a specific level of access. |
| Accounting | Accounting is the part of the AAA framework that monitors the specific resources that a user consumes while accessing the network. This is done by logging session statistics and other information pertaining to usage. This can also be used to see which files were accessed by each user in order to track changes made to the database or any other files. This is key to be able to quickly find the user responsible for making changes that could have led to a vulnerability or malicious attack. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

## Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

| **Standard** | **Principle(s)** | **Explanation** |
| --- | --- | --- |
| STD-001-CLG | 10 | It should be a standard to declare identifiers before using the them. This will help ensure that the data type of the variable is correct. While this is a low severity issue, it should still be followed as a best practice. |
| STD-002-CPP | 1 | In this example the compliant solution tests the operands to make sure there is no unsigned integer wrap. This is showing the importance of validating the input data to make sure it will not create any vulnerabilities. |
| STD-003-CPP | 1 | This example is showing the importance of validating the input because if the buffer is too small to hold the data it can cause a buffer overflow. |
| STD-004-JVA | 1, 7 | This shows the importance of the principles Validate input data and sanitize data sent because by both sanitizing and validating the data it can prevent a SQL injection. Limiting and removing unwanted characters can help reduce vulnerabilities. |
| STD-005-CPP | 10 | This shows that it is important to adopt a secure coding standard because it is important to make sure memory is not deallocated until it is no longer required in order to prevent vulnerabilities and the chance of running arbitrary code. |
| STD-006-CLG | 9, 10 | Using tests and assertions is a great way to ensure that the code is functioning properly, this can help verify that the correct outcome of the code should be doing is actually happening. |
| STD-007-CPP | 2, 10 | This example touches on these two principles because the original code was not set up to throw an exception. It is best practices to implement exceptions and error messages to explain to the user why the code was forced to quit, or why something is invalid This also goes hand in hand with heed compiler warnings because when building code if you ignore the warnings you are leaving room for attackers to possibly exploit the code. Allowing an application to terminate abnormally can tie up resources and is often a vector for denial of service attacks. |
| STD-008-CPP | 1 | This example shows the importance of avoiding relying on the default operator new when obtaining storage for objects of over aligned types. This can cause the code to have unexpected behavior. Validating the data will allow you to ensure there is enough storage. |
| STD-009-CPP | 1, 10 | It is important to validate the input in this code to prevent buffer flow and prevent attackers from executing arbitrary code. This should be standard to validate any user input and declare all variables to prevent these vulnerabilities. |
| STD-010-CPP | 2, 4 | It is important to be cautious of any warnings provided by the compiler, and also include exceptions to be thrown when there is an issue in the application. These should be expected exceptions as unexpected exceptions can disrupt the control flow and cause premature termination and denial of service. |

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 |  |
| 2.0 | 04/10/2020 | Updated document to include more information on Principles and risk analysis. | Daniel DeGouveia |  |

# Appendix A Lookups

[How this Coding Standard Is Organized - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/How+this+Coding+Standard+Is+Organized#HowthisCodingStandardIsOrganized-NoncompliantCodeExamplesandCompliantSolutions)

## Approved C/C++ Language Acronyms

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