

# 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 | It’s important to validate all input from untrusted data sources to decrease the chances of software vulnerabilities. These untrusted data sources include command lines, network interfaces, environmental variables, and used-controlled files. |
| 1. Heed Compiler Warnings | Code should be compiled using the highest warning level available for the compiler. Eliminate the warning by modifying the code as well as use static and dynamic analysis tools to find security flaws. |
| 1. Architect and Design for Security Policies | Create software architecture to implement and enforce security policies. Set user privileges if a system requires different privileges at certain times. |
| 1. Keep It Simple | Keep the design simple to avoid errors made during implementation, configuration and during use. Effort to achieve assurance increases as security mechanisms become complex. |
| 1. Default Deny | Access decisions should be based on permission rather than exclusion. Access should be denied unless the protection identifies conditions that would grant the user access. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least number of privileges needed to do the job in order to prevent unwanted intrusion. This gives an attack less opportunities to execute code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Data that is passed through command shells, databases and commercial off-the-shelf components should be sanitized. This is to prevent attackers from attacking systems through injection. |
| 1. Practice Defense in Depth | There should be multiple layers of security to prevent security flaws. Combine secure programming techniques with secure environments to reduce vulnerabilities in the operation environment. |
| 1. Use Effective Quality Assurance Techniques | Use different testing practices to identify and eliminate vulnerabilities through a quality assurance program. |
| 1. Adopt a Secure Coding Standard | It’s important to develop secure coding standards for the development language and platform used to build the system.001 |

### 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] | Abstract data types are not restricted to object-oriented languages such as C++ and Java. They should be created and used in C language programs as well. Abstract data types are most effective when used with private (opaque) data types and information hiding. |

| **Noncompliant Code** |
| --- |
| The implementation of the string\_mx type is fully visible to the user of the data type after including the string\_m.h file. Programmers are consequently more likely to directly manipulate the fields within the structure, violating the software engineering principles of information hiding and data encapsulation and increasing the probability of developing incorrect or nonportable code. |
| **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  };    **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| This compliant solution reimplements the string\_mx type as a private type, hiding the implementation of the data type from the user of the managed string library. Modules that implement the abstract data type include both the external and internal definitions, whereas users of the data abstraction include only the external string\_m.h file. This allows the implementation of the string\_mx data type to remain private. |
| **struct** string\_mx;  **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/  **002struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  }; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion | Bauhaus | CertC-DCL12 | NA |
| LDRA Tool Suite | 9.7.1 | 104 D | Partially Implemented |
| Polyspace Bug Finder | R2022b | CERTC: Rec. DCL12-C | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced (rule partially covered) |
| Parasoft C/C++ Test | 2022.1 | CERT\_C-DCL12-a | If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | A value-returning function must return a value from all code paths; otherwise, it will result in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the programmer forgot to return the input value for positive input, so not all code paths return a value. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, all code paths now return a value. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  **return** a;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | return-implicit | Fully Checked |
| Axivion | 7.2.0 | CertC++-<SC52 | NA |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| CodeSonar | 7.2p0 | LANG.STRUCT.MRS | Missing return statement |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CLG] | Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings [Seacord 2013b]. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is of sufficient size to hold the character data to be copied and the null-termination character. (See STR03-C. Do not inadvertently truncate a string.) |

| **Noncompliant Code** |
| --- |
| This noncompliant code example demonstrates an off-by-one error [Dowd 2006]. The loop copies data from src to dest. However, because the loop does not account for the null-termination character, it may be incorrectly written 1 byte past the end of dest. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the loop termination condition is modified to account for the null-termination character that is appended to dest: |
| #include <stddef.h>    **void** copy(**size\_t** n, **char** src[n], **char** dest[n]) {  **size\_t** i;    **for** (i = 0; src[i] && (i < n - 1); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | NA | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR31 | Detects calls to unsafe string function that may cause buffer overflow  Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| CodeSonar | 7.2p0 | **LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\*** | Buffer overrun  Type overrun  No space for null terminator  A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Compass/ROSE | NA | NA | Can detect violations of the rule. However, it is unable to handle cases involving strcpy\_s() or manual string copies such as the one in the first example |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV] | SQL injection vulnerabilities arise in applications where elements of a SQL query originate from an untrusted source. Without precautions, the untrusted data may maliciously alter the query, resulting in information leaks or data modification. The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures.  Suppose a system authenticates users by issuing the following query to a SQL database. If the query returns any results, authentication succeeds; otherwise, authentication fails. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows JDBC code to authenticate a user to a system. The password is passed as a char array, the database connection is created, and then the passwords are hashed.  Unfortunately, this code example permits a SQL injection attack by incorporating the unsanitized input argument username into the SQL command, allowing an attacker to inject validuser' OR '1'='1. The password argument cannot be used to attack this program because it is passed to the hashPassword() function, which also [sanitizes](https://wiki.sei.cmu.edu/confluence/display/java/Rule+BB.+Glossary#RuleBB.Glossary-sanitize) the input. |
| **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 {  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;    class Login {  public Connection getConnection() throws SQLException {  DriverManager.registerDriver(new  com.microsoft.sqlserver.jdbc.SQLServerDriver());  String dbConnection =  PropertyManager.getProperty("db.connection");  // Can hold some value like  // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  return DriverManager.getConnection(dbConnection);  }    String hashPassword(char[] password) {  // Create hash of password  }    public void doPrivilegedAction(  String username, char[] password  ) throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);  String sqlString = "select \* from db\_user where username=" +  username + " and password =" + pwd;  PreparedStatement stmt = connection.prepareStatement(sqlString);    ResultSet rs = stmt.executeQuery();  if (!rs.next()) {  throw new SecurityException("User name or password incorrect");  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a parametric query with a ? character as a placeholder for the argument. This code also validates the length of the username argument, preventing an attacker from submitting an arbitrarily 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| CodeSonar | 7.2p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | The default memory allocation operator, ::operator new(std::size\_t), throws a std::bad\_alloc exception if the allocation fails. Therefore, you need not check whether calling ::operator new(std::size\_t) results in nullptr. The nonthrowing form, ::operator new(std::size\_t, const std::nothrow\_t &), does not throw an exception if the allocation fails but instead returns nullptr. The same behaviors apply for the operator new[] versions of both allocation functions. Additionally, the default allocator object (std::allocator) uses ::operator new(std::size\_t) to perform allocations and should be treated similarly. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program. |
| #include <cstring>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new int[size];  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new (std::nothrow) int[size];  if (!copy) {  // Handle error  return;  }  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled. |
| Hewlix QAC | 2022.4 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 | NA |
| Klockwork | 2022.4 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF | NA |
| LDRA tool suite | 9.7.1 | 45 D | Partially Implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-JAV] | Diagnostic tests can be incorporated into programs by using the assert statement. Assertions are primarily intended for use during debugging and are often turned off before code is deployed by using the -disableassertions (or -da) Java runtime switch. Consequently, assertions should be used to protect against incorrect programmer assumptions and not for runtime error checking.  Assertions should never be used to verify the absence of runtime (as opposed to logic) errors, such as  Invalid user input (including command-line arguments and environment variables)  File errors (for example, errors opening, reading, or writing files)  Network errors (including network protocol errors)  Out-of-memory conditions (when the Java Virtual Machine cannot allocate space for a new object and the garbage collector cannot make sufficient space available)  System resource exhaustion (for example, out-of-file descriptors, processes, threads)  System call errors (for example, errors executing files, locking or unlocking mutexes)  Invalid permissions (for example, file, memory, user)  Code that protects against an I/O error, for example, cannot be implemented as an assertion because it must be present in the deployed executable.  Assertions are generally unsuitable for server programs or embedded systems in deployment. A failed assertion can lead to a denial-of-service (DoS) attack if triggered by a malicious user. In such situations, a soft failure mode, such as writing to a log file and rejecting the request, is more appropriate. |

| **Noncompliant Code** |
| --- |
| Because input availability depends on the user and can be exhausted at any point during program execution, a robust program must be prepared to gracefully handle and recover from the unavailability of input. However, using the assert statement to verify that some significant input was available is inappropriate because it might lead to an abrupt termination of the process, resulting in a denial of service. |
| BufferedReader br;    // Set up the BufferedReader br    String line;    // ...    line = br.readLine();    assert line != null; |

| **Compliant Code** |
| --- |
| This compliant solution demonstrates the recommended way to detect and handle unavailability of input: |
| BufferedReader br;    // Set up the BufferedReader br    String line;    // ...    line = br.readLine();    if (line == null) {  // Handle error  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| NA | NA | NA | NA | NA |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Jtest | 2022.2 | CERT.MSC60.ASSERT | Do not use assertions in production code |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | If a function throws an exception other than one allowed by its exception-specification, it can lead to an implementation-defined termination of the program ([except.spec], paragraph 9). |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a function is declared as nonthrowing, but it is possible for std::vector::resize() to throw an exception when the requested memory cannot 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 solution, the function's noexcept-specification is removed, signifying that the function allows all exceptions.Compliant description] |
| #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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | unhandled-throw-noexcept | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR55 | NA |
| CodeSonar | 7.2p0 | LANG.STRUCT.EXCP.THROW | Use of Throw |
| Helix QAC | 2022.4 | C++4035, C++4036, C++4632 | NA |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| String Correctness | [STD-008-LLL] | Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings [Seacord 2013]. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is of sufficient size to hold the data to be copied. C-style strings require a null character to indicate the end of the string, while the C++ std::basic\_string template requires no such character. |

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 7.2p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Helix QAC | 2022.4 | C++5216  DF2835, DF2836, DF2839, | NA |
| Klockwork | 2022.4 | NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN | NA |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| SQL Injection | [STD-009 -JAV] | Code injection can occur when untrusted input is injected into dynamically constructed code. One obvious source of potential vulnerabilities is the use of JavaScript from Java code. The javax.script package consists of interfaces and classes that define Java scripting engines and a framework for the use of those interfaces and classes in Java code. Misuse of the javax.script API permits an attacker to execute arbitrary code on the target system.  This guideline is a specific instance of IDS00-J. Prevent SQL injection. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example incorporates untrusted user input into a JavaScript statement that is responsible for printing the input: |
| private static void evalScript(String firstName) throws ScriptException {  ScriptEngineManager manager = new ScriptEngineManager();  ScriptEngine engine = manager.getEngineByName("javascript");  engine.eval("print('"+ firstName + "')");  } |

| **Compliant Code** |
| --- |
| The best defense against code injection vulnerabilities is to prevent the inclusion of executable user input in code. User input used in dynamic code must be sanitized, for example, to ensure that it contains only valid, whitelisted characters. Sanitization is best performed immediately after the data has been input, using methods from the data abstraction used to store and process the data. Refer to IDS00-J. Sanitize untrusted data passed across a trust boundary for more details. If special characters must be permitted in the name, they must be normalized before comparison with their equivalent forms for the purpose of input validation. This compliant solution uses whitelisting to prevent unsanitized input from being interpreted by the scripting engine. |
| private static void evalScript(String firstName) throws ScriptException {  // Allow only alphanumeric and underscore chars in firstName  // (modify if firstName may also include special characters)  if (!firstName.matches("[\\w]\*")) {  // String does not match whitelisted characters  throw new IllegalArgumentException();  }    ScriptEngineManager manager = new ScriptEngineManager();  ScriptEngine engine = manager.getEngineByName("javascript");  engine.eval("print('"+ firstName + "')");  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| NA | NA | NA | NA | NA |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| Parasoft Jtest | 2022.2 | CERT.IDS52.TDCODE | Validate potentially tainted data before it is used in methods that generate code |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | [STD-010-CPP] | When an exception is thrown, the exception object operand of the throw expression is copied into a temporary object that is used to initialize the handler. If the copy constructor for the exception object type throws during the copy initialization, std::terminate() is called, which can result in possibly unexpected implementation-defined behavior. For more information on implicitly calling std::terminate(), see ERR50-CPP. Do not abruptly terminate the program.  The copy constructor for an object thrown as an exception must be declared noexcept, including any implicitly-defined copy constructors. Any function declared noexcept that terminates by throwing an exception violates ERR55-CPP. Honor exception specifications.  The C++ Standard allows the copy constructor to be elided when initializing the exception object to perform the initialization if a temporary is thrown. Many modern compiler implementations make use of both optimization techniques. However, the copy constructor for an exception object still must not throw an exception because compilers are not required to elide the copy constructor call in all situations, and common implementations of std::exception\_ptr will call a copy constructor even if it can be elided from a throw expression. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an exception of type S is thrown in f(). However, because S has a std::string data member, and the copy constructor for std::string is not declared noexcept, the implicitly-defined copy constructor for S is also not declared to be noexcept. In low-memory situations, the copy constructor for std::string may be unable to allocate sufficient memory to complete the copy operation, resulting in a std::bad\_alloc exception being thrown. |
| #include <exception>  #include <string>    class S : public std::exception {  std::string m;  public:  S(const char \*msg) : m(msg) {}    const char \*what() const noexcept override {  return m.c\_str();  }  };    void g() {  // If some condition doesn't hold...  throw S("Condition did not hold");  }    void f() {  try {  g();  } catch (S &s) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This compliant solution assumes that the type of the exception object can inherit from std::runtime\_error, or that type can be used directly. Unlike std::string, a std::runtime\_error object is required to correctly handle an arbitrary-length error message that is exception safe and guarantees the copy constructor will not throw [ ISO/IEC 14882-2014 ]. |
| #include <stdexcept>  #include <type\_traits>    struct S : std::runtime\_error {  S(const char \*msg) : std::runtime\_error(msg) {}  };    static\_assert(std::is\_nothrow\_copy\_constructible<S>::value,  "S must be nothrow copy constructible");    void g() {  // If some condition doesn't hold...  throw S("Condition did not hold");  }    void f() {  try {  g();  } catch (S &s) {  // Handle error  }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | cert-err60-cpp | Checked by clang-tidy |
| Helix QAC | 2022.4 | C++3508 | NA |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-ERR60-a  CERT\_CPP-ERR60-b | Exception objects must be nothrow copy constructible  An explicitly declared copy constructor for a class that inherits from 'std::exception' should have a non-throwing exception specification |
| PRQA QA-C++ | 4.4 | 3508 |  |

### Defense-in-Depth Illustration

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





Automation plays a critical role in the development of software and can be implemented in numerous parts of the DevOps cycle. Starting from the beginning, Automation can help us save time and resources if implemented properly. When developing code, we need to be able to automate the code as well as running and automating test cases to ensure the security of the code as well as the functionality. Packaging and deploying code can be automated as well once it is ready to be released after the testing phase. Automation can be used in configuring the servers, networks, firewalls and monitor the application as well. Finally, automation can find vulnerabilities within the live product as well as help the product stay current when the requirements and standards need to be updated.

### 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 | High | P1 | L3 |
| STD-002-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-003-CLG | High | Likely | Medium | P18 | L1 |
| STD-004-JAV | High | Probably | Medium | P12 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-JAV | NA | NA | NA | NA | NA |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-LLL | High | Likely | Medium | P18 | L1 |
| STD-009-JAV | NA | NA | NA | NA | NA |
| STD-010-CPP | Low | Probable | 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 | Encryption at rest is protection for stored data. This is designed to prevent attackers from accessing any unencrypted data by ensuring it’s encrypted. It’s used to prevent attacks where a hacker would obtain the storage, but wouldn’t have the keys to bypass the encryption. It is a high priority for companies and highly recommended. |
| Encryption at flight | Encryption at flight is when data is encrypted when it’s being transmitted over a network. It should be applied by using secure transfer such as SFTP. This is to protect data that is transferred between servers online and can contain important data. Data is vulnerable during flight if a secure method is not used since it is unencrypted and insecure. VPN’s are also used wisely to ensure encryption at flight is used. |
| Encryption in use | Encryption in use is when data is accessed or used by a an application or end user. This is the most vulnerable form and needs to be limited to users who need it. It is also important and should be used to prevent hackers from seeing data and deciphering it. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process when the user wants to access the network and needs credentials such as a username and password. The policy applies to limit access to certain networks and databases. |
| Authorization | Enforces policies on networks for the user and gives them access to only the resources that they need. |
| Accounting | A way of monitoring and capturing logs of what is occurring by each user when accessing the network. This is to hold users accountable for their actions and to keep track of who is accessing what. |

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

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

### Map the Principles

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

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

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

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.2 | [Insert text.] | Updated Security and Coding Standards | Christopher Gaceta | John Smith |
| 2.0 | [Insert text.] | Automation/Encryption/Triple A | Christopher Gaceta | John Smith |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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