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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_gjdgxs)

[Purpose 2](#_30j0zll)

[Scope 2](#_1fob9te)

[Module Three Milestone 2](#_3znysh7)

[Ten Core Security Principles 2](#_2et92p0)

[C/C++ Ten Coding Standards 3](#_3dy6vkm)

[Coding Standard 1 4](#_1t3h5sf)

[Coding Standard 2 5](#_4d34og8)

[Coding Standard 3 6](#_2s8eyo1)

[Coding Standard 4 7](#_17dp8vu)

[Coding Standard 5 8](#_3rdcrjn)

[Coding Standard 6 9](#_26in1rg)

[Coding Standard 7 10](#_lnxbz9)

[Coding Standard 8 11](#_35nkun2)

[Coding Standard 9 13](#_1ksv4uv)

[Coding Standard 10 14](#_44sinio)

[Defense-in-Depth Illustration 15](#_2jxsxqh)

[Project One 15](#_z337ya)

[1.](#_3j2qqm3) Revise the C/C++ Standards 15

[2.](#_1y810tw) Risk Assessment 15

[3.](#_4i7ojhp) Automated Detection 15

[4.](#_2xcytpi) Automation 15

[5.](#_1ci93xb) Summary of Risk Assessments 16

[6.](#_3whwml4) Create Policies for Encryption and Triple A 16

[7.](#_2bn6wsx) Map the Principles 17

[Audit Controls and Management 18](#_qsh70q)

[Enforcement 18](#_3as4poj)

[Exceptions Process 18](#_1pxezwc)

[Distribution 19](#_49x2ik5)

[Policy Change Control 19](#_2p2csry)

[Policy Version History 19](#_147n2zr)

[Appendix A Lookups 19](#_3o7alnk)

[Approved C/C++ Language Acronyms 19](#_23ckvvd)

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | This principle emphasizes the importance of validating and sanitizing all input data to prevent malicious input from compromising the security of the software. By validating input data, the software can ensure that it meets the expected format and range of values, reducing the risk of vulnerabilities such as buffer overflows or SQL injection attacks. |
| 1. Heed Compiler Warnings | Pay close attention to and address any warnings or errors raised by the compiler, as they can indicate potential vulnerabilities that need to be resolved. |
| 1. Architect and Design for Security Policies | Incorporate security considerations into the software's architectural and design phases to mitigate risks and align with established security policies and best practices. |
| 1. Keep It Simple | The more complex a system is, the harder it is to ensure its security. By keeping systems and code simple and easy to understand, we reduce the potential for errors and vulnerabilities. Simple, well-structured code is also easier to test and maintain, further enhancing its security. |
| 1. Default Deny | The principle of default deny suggests that all access should be denied by default, and only authorized and necessary access should be explicitly granted. This approach helps to minimize the risk of unauthorized access or privilege escalation. By starting with a default deny stance and carefully controlling access permissions, the software can enforce the principle of least privilege and limit potential security breaches. |
| 1. Adhere to the Principle of Least Privilege | Each user, process, or system should have the minimum privileges necessary to perform its tasks. This reduces the potential for damage if a process is compromised, as a compromised process can only abuse the privileges it has been granted. |
| 1. Sanitize Data Sent to Other Systems | Data should be sanitized to ensure it does not contain any harmful elements, such as malicious commands or sensitive information. This applies both to data sent to other internal systems and to data sent to external systems or users. By sanitizing outgoing data, we prevent it from being used as a vector for attacks against other systems. |
| 1. Practice Defense in Depth | Defense in depth is a security strategy that involves implementing multiple layers of security controls and measures. By using a layered approach, even if one security control fails or is bypassed, there are additional layers of protection to prevent or detect attacks. This principle encourages the implementation of multiple security mechanisms, such as firewalls, intrusion detection systems, encryption, and access controls, to strengthen the overall security posture. |
| 1. Use Effective Quality Assurance Techniques | Incorporate rigorous quality assurance practices like code reviews, analysis, and testing to identify and rectify security vulnerabilities before software deployment. |
| 1. Adopt a Secure Coding Standard | Follow a recognized secure coding standard, such as the CERT C/C++ coding standards. Secure coding standards provide guidelines and best practices for writing secure code and avoiding common vulnerabilities. The standard should be continuously updated to address new security threats and should be used in conjunction with regular security training for developers. |

### 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** | **Ensure that division and remainder operations do not result in divide-by-zero errors** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | The result of the / operator is the quotient from the division of the first arithmetic operand by the second arithmetic operand. Division operations are susceptible to divide-by-zero errors. Overflow can also occur during two's complement signed integer division when the dividend is equal to the minimum (most negative) value for the signed integer type and the divisor is equal to −1. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example prevents signed integer overflow in compliance with INT32-C. Ensure that operations on signed integers do not result in overflow but fails to prevent a divide-by-zero error during the division of the signed operands s\_a and s\_b: |
| #include <limits.h>    void func(signed long s\_a, signed long s\_b) {  signed long result;  if ((s\_a == LONG\_MIN) && (s\_b == -1)) {  /\* Handle error \*/  } else {  result = s\_a / s\_b;  }  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution tests the division operation to guarantee there is no possibility of divide-by-zero errors or signed overflow: |
| #include <limits.h>    void func(signed long s\_a, signed long s\_b) {  signed long result;  if ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {  /\* Handle error \*/  } else {  result = s\_a / s\_b;  }  /\* ... \*/  } |

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

| **Principles(s):** ValidateInput Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | int-division-by-zero  int-modulo-by-zero | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | CertC-INT33 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | LANG.ARITH.DIVZERO  LANG.ARITH.FDIVZERO | Division by zero  Float Division By Zero |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule (In particular, it ensures that all operations involving division or modulo are preceded by a check ensuring that the second operand is nonzero.) |
| Coverity | 2017.07 | DIVIDE\_BY\_ZERO | Fully implemented |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | zerodiv  zerodivcond | Context sensitive analysis of division by zero  Not detected for division by struct member / array element / pointer data that is 0  Detected when there is unsafe division by variable before/after test if variable is zero |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | C2830  C++2830  DF2831, DF2832, DF2833 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | DBZ.CONST  DBZ.CONST.CALL  DBZ.GENERAL  DBZ.ITERATOR  DBZ.ITERATOR.CALL |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 43 D, 127 D, 248 S, 629 S, 80 X | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-INT33-a | Avoid division by zero |
| [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) | R2023a | [CERT C: Rule INT33-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint33c.html) | Checks for:   * Integer division by zero * Tainted division operand * Tainted modulo operand   Rule fully covered. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151949) | 3.11 | [**S3518**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-3518) |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.25 | [**V609**](https://pvs-studio.com/en/docs/warnings/v609/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | division\_by\_zero | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/c37797b7) ). |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Declare objects with appropriate storage durations** |
| --- | --- | --- |
| **Data Value** | [STD-002-CLG] | Every object has a storage duration that determines its lifetime: static, thread, automatic, or allocated.  According to the C Standard, 6.2.4, paragraph 2 [ISO/IEC 9899:2011],  The lifetime of an object is the portion of program execution during which storage is guaranteed to be reserved for it. An object exists, has a constant address, and retains its last-stored value throughout its lifetime. If an object is referred to outside of its lifetime, the behavior is undefined. The value of a pointer becomes indeterminate when the object it points to reaches the end of its lifetime.  Do not attempt to access an object outside of its lifetime. Attempting to do so is undefined behavior and can lead to an exploitable vulnerability. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the address of the variable c\_str with automatic storage duration is assigned to the variable p, which has static storage duration. The assignment itself is valid, but it is invalid for c\_str to go out of scope while p holds its address, as happens at the end of dont\_do\_this(). |
| #include <stdio.h>    const char \*p;  void dont\_do\_this(void) {  const char c\_str[] = "This will change";  p = c\_str; /\* Dangerous \*/  }    void innocuous(void) {  printf("%s\n", p);  }    int main(void) {  dont\_do\_this();  innocuous();  return 0;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, p is declared with the same storage duration as c\_str, preventing p from taking on an indeterminate value outside of this\_is\_OK(): |
| void this\_is\_OK(void) {  const char c\_str[] = "Everything OK";  const char \*p = c\_str;  /\* ... \*/  }  /\* p is inaccessible outside the scope of string c\_str \*/ |

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

| **Principles(s):** Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | pointered-deallocation  return-reference-local | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL30 | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | LANG.STRUCT.RPL | Returns pointer to local |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule. It automatically detects returning pointers to local variables. Detecting more general cases, such as examples where static pointers are set to local variables which then go out of scope, would be difficult |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | RETURN\_LOCAL | Finds many instances where a function will return a pointer to a local stack variable. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | C3217, C3225, C3230, C4140  C++2515, C++2516, C++2527, C++2528, C++4026, C++4624, C++4629 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | LOCRET.ARG  LOCRET.GLOB  LOCRET.RET |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 42 D, 77 D, 71 S, 565 S | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-DCL30-a  CERT\_C-DCL30-b | The address of an object with automatic storage shall not be returned from a function  The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 604, 674, 733, 789 | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule DCL30-C](https://www.mathworks.com/help/bugfinder/ref/certcruledcl30c.html) | Checks for pointer or reference to stack variable leaving scope (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.25 | [**V506**](https://pvs-studio.com/en/docs/warnings/v506/), [**V507**](https://pvs-studio.com/en/docs/warnings/v507/), [**V558**](https://pvs-studio.com/en/docs/warnings/v558/), [**V623**](https://pvs-studio.com/en/docs/warnings/v623/), [**V723**](https://pvs-studio.com/en/docs/warnings/v723/), [**V738**](https://pvs-studio.com/en/docs/warnings/v738/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 23.04 | return-reference-local | Partially checked |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 |  |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | dangling\_pointer | Exhaustively detects undefined behavior (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/d86f1a03) ). |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | 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):** Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | C++5216  DF2835, DF2836, DF2839 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN |  |
| [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) | 2023.1 | 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) | R2023a | CERT C++: STR50-CPP | 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  Insufficient destination buffer size  Rule partially covered. |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | stream-input-char-array | Partially checked |
| [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** | **Prevent SQL injection** |
| --- | --- | --- |
| **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. |

| **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 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 {  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 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):** Keep It Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| [Fortify](https://wiki.sei.cmu.edu/confluence/display/java/Fortify) | 1.0 | HTTP\_Response\_Splitting  SQL\_Injection\_\_Persistence  SQL\_Injection | Implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/java/Klocwork) |  | SV.DATA.BOUND  SV.DATA.DB  SV.HTTP\_SPLIT  SV.PATH  SV.PATH.INJ  SV.SQL | Implemented |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 9.9 | [**S2077**](https://rules.sonarsource.com/java/RSPEC-2077)  [**S3649**](https://rules.sonarsource.com/java/RSPEC-3649) | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649) |
| [SpotBugs](https://wiki.sei.cmu.edu/confluence/display/java/SpotBugs) | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Detect and handle memory allocation errors** |
| --- | --- | --- |
| **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):** Default Deny |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 45 D | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| [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) | R2023a | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | V522, V668 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the termination behavior of assert() and abort()** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Because assert() calls abort(), cleanup functions registered with atexit() are not called. If the intention of the programmer is to properly clean up in the case of a failed assertion, then runtime assertions should be replaced with static assertions where possible. (See DCL03-C. Use a static assertion to test the value of a constant expression.) When the assertion is based on runtime data, the assert should be replaced with a runtime check that implements the adopted error strategy (see ERR00-C. Adopt and implement a consistent and comprehensive error-handling policy). |

| **Noncompliant Code** |
| --- |
| This noncompliant code example defines a function that is called before the program exits to clean up: |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }    /\* ... \*/    assert(/\* Something bad didn't happen \*/);    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the call to assert() is replaced with an if statement that calls exit() to ensure that the proper termination routines are run: |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }    /\* ... \*/    if (/\* Something bad happened \*/) {  exit(EXIT\_FAILURE);  }    /\* ... \*/  } |

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

| **Principles(s):** Adhere to the Principle of Least Privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-ERR06-a | Do not use assertions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 586 | Fully supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. The C++ Standard, [except.handle], paragraph 9 [ISO/IEC 14882-2014], states the following:  If no matching handler is found, the function std::terminate() is called; whether or not the stack is unwound before this call to std::terminate() is implementation-defined.  The default terminate handler called by std::terminate() calls std::abort(), which abnormally terminates the process. When std::abort() is called, or if the implementation does not unwind the stack prior to calling std::terminate(), destructors for objects may not be called and external resources can be left in an indeterminate state. Abnormal process termination is the typical vector for denial-of-service attacks. For more information on implicitly calling std::terminate(), see ERR50-CPP. Do not abruptly terminate the program.  All exceptions thrown by an application must be caught by a matching exception handler. Even if the exception cannot be gracefully recovered from, using the matching exception handler ensures that the stack will be properly unwound and provides an opportunity to gracefully manage external resources before terminating the process. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| 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):** Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.1 | main-function-catch-all  early-catch-all | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-ERR51 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | C++4035, C++4036, C++4037 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | MISRA.CATCH.ALL |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 527 S | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.1 | main-function-catch-all  early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not attempt to create a std::string from a null pointer** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | The std::basic\_string type uses the traits design pattern to handle implementation details of the various string types, resulting in a series of string-like classes with a common, underlying implementation. Specifically, the std::basic\_string class is paired with std::char\_traits to create the std::string, std::wstring, std::u16string, and std::u32string classes. The std::char\_traits class is explicitly specialized to provide policy-based implementation details to the std::basic\_string type. One such implementation detail is the std::char\_traits::length() function, which is frequently used to determine the number of characters in a null-terminated string. According to the C++ Standard, [char.traits.require], Table 62 [ISO/IEC 14882-2014], passing a null pointer to this function is undefined behavior because it would result in dereferencing a null pointer. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.1 | assert\_failure |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | LANG.MEM.NPD | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | DF4770, DF4771, DF4772, DF4773, DF4774 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Value-returning functions must return a value from all exit paths** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | The C++ Standard, [stmt.return], paragraph 2 [ISO/IEC 14882-2014], states the following:  Flowing off the end of a function is equivalent to a return with no value; this results in undefined behavior in a value-returning function.  A value-returning function must return a value from all code paths; otherwise, it will result in undefined behavior. This includes returning through less-common code paths, such as from a function-try-block, as explained in the C++ Standard, [except.handle], paragraph 15:  Flowing off the end of a function-try-block is equivalent to a return with no value; this results in undefined behavior in a value-returning function (6.6.3). |

| **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):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.1 | return-implicit | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MSC52 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | LANG.STRUCT.MRS | Missing return statement |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | DF2888 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | FUNCRET.GEN  FUNCRET.IMPLICIT |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 2 D, 36 S | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-MSC52-a | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.1 | [S935](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-935) |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | [V591](https://pvs-studio.com/en/docs/warnings/v591/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.1 | return-implicit | Fully checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities.  It is at the memory manager's discretion when to reallocate or recycle the freed memory. When memory is freed, all pointers into it become invalid, and its contents might either be returned to the operating system, making the freed space inaccessible, or remain intact and accessible. As a result, the data at the freed location can appear to be valid but change unexpectedly. Consequently, memory must not be written to or read from once it is freed. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #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 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.1 | dangling\_pointer\_use |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MEM50 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | ALLOC.UAF | Use after free |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | C++4303, C++4304 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | UFM.DEREF.MIGHT  UFM.DEREF.MUST  UFM.FFM.MIGHT  UFM.FFM.MUST  UFM.RETURN.MIGHT  UFM.RETURN.MUST  UFM.USE.MIGHT  UFM.USE.MUST |  |
| [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) | 2023.1 | 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) | R2023a | [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. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | V586, V774 |  |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Splint) | 5.0 |  |  |

### 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.



In the transition from DevOps to DevSecOps, automation plays a crucial role in ensuring the enforcement of security standards throughout the software development process. By integrating security practices at every stage, Green Pace can proactively identify and mitigate vulnerabilities, thereby reducing the risk of potential breaches and data leaks. To automate the enforcement of security standards defined in the policy, Green Pace should incorporate various tools and practices within its existing DevOps process.

One key area where automation can be applied is code scanning and analysis. By integrating automated code scanning tools, such as Static Application Security Testing and Dynamic Application Security Testing, into the CI/CD pipeline, developers can receive immediate feedback on potential security vulnerabilities, coding errors, and misconfigurations. This enables them to take corrective actions early in the development cycle, ensuring that security is considered from the start.

Another aspect of automation is vulnerability management. By implementing automated vulnerability scanning tools that continuously monitor the software components and dependencies used in the application, Green Pace can detect and alert developers about any known vulnerabilities or outdated libraries. This allows them to promptly address such issues by applying patches or updating dependencies, minimizing the risk of exploitation.

Lastly, security testing can also be automated. By implementing automated security testing frameworks, such as penetration testing tools, Green Pace can simulate real-world attacks and identify vulnerabilities that may have been missed during development. Integrating these tools into the CI/CD pipeline allows for regular security testing to be performed automatically before deploying new code or infrastructure changes.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Likely | Medium | P6 | L2 |
| STD-002-CLG | High | Probable | High | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-JAV | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CLG | Medium | Unlikely | Medium | P4 | L3 |
| STD-007-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-008-CPP | High | Likely | Medium | P18 | L1 |
| STD-009-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-010-CPP | High | Likely | Medium | P18 | L1 |

### 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 | All sensitive data stored on databases, file systems, or persistent storage must be encrypted at rest. Encryption at rest ensures that sensitive data remains encrypted when it is stored on disks or other storage devices such as user databases, client records, etc. This protects the data from unauthorized access in case of unauthorized access to the physical storage devices. |
| Encryption at flight | Encryption in flight safeguards data while it is being transmitted across systems or networks. The policy for transmitting data over public networks or insecure channels requires encryption using secure protocols such as TLS/SSL. This applies to data transmitted over internet connections or networks. |
| Encryption in use | Encryption in use refers to the protection of sensitive data while being used within applications or systems. Encryption must be applied to all sensitive data when being processed or when in use in any application or system. This will help mitigate data breaches or unauthorized access to sensitive data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to verifying who someone is before granting them access to sensitive systems or resources. All users who access Green Pace systems or software must authenticate themselves using unique and secure credentials before being allowed to access anything in conjunction with having 2-factor authentication active or a 2-factor authentication hardware device. |
| Authorization | Authorization is the level of permissions granted to a user once given access after authenticating themselves. Users should be granted permissions or privileges starting with no privileges or permissions and adding the minimum of what is needed for them to complete their roles and requirements inside the company. |
| Accounting | Accounting is also known as auditing or logging and requires that there are recordings and monitoring of all user activities within any system or application. All significant user activities such as additions of new users, modifications to privileges, or access or changes to databases, must be logged and auditable. |

**\***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

| **Principles** | **Principles That Map to Each Standard** | **Justification for Connection** |
| --- | --- | --- |
| ValidateInput Data | Ensure that division and remainder operations do not result in divide-by-zero errors, Guarantee that storage for strings has sufficient space for character data and the null terminator, Prevent SQL injection | Any of these standards when not adhered to can result in unexpected results leaving a system or software vulnerable to various vulnerabilities such as buffer overflows or SQL injection attacks. By validating and preventing invalid input of data this can be prevented. |
| Heed Compiler Warnings | Detect and handle memory allocation errors, Understand the termination behavior of assert() and abort(), and Handle all exceptions | Do not ignore warnings or errors in software just because it compiles there may be unforeseen vulnerabilities that can be resolved. It is also important to go beyond the more common errors caught by IDE’s by using automated software to check for errors such as CppCheck to look for issues the IDE misses. |
| Architect and Design for Security Policies | Declare objects with appropriate storage durations, Guarantee that storage for strings has sufficient space for character data and the null terminator, Prevent SQL injection, Do not attempt to create a std::string from a null pointer, Do not access freed memory | The following standard are best practices that should be standard in all security policies. If you come across a security policy that does not include these you should suggest an immediate change to the policy as they will mitigate risk to software and systems. |
| Keep It Simple | Ensure that division and remainder operations do not result in divide-by-zero errors, Prevent SQL injection, Handle all exception, Value-returning functions must return a value from all exit paths, Do not access freed memory | Following these simple standards will keep the system from becoming overcomplicated and hard to understand, thus making it easier to manage and maintain and test system security. |
| Default Deny | Prevent SQL injection, Understand the termination behavior of assert() and abort(), Do not access freed memory | Securing the system against vulnerabilities will protect against unauthorized access to data and systems thus aiding in upholding the principle of default deny . |
| Adhere to the Principle of Least Privilege | Ensure that division and remainder operations do not result in divide-by-zero errors,  Guarantee that storage for strings has sufficient space for character data and the null terminator  Prevent SQL injection, Understand the termination behavior of assert() and abort(), Do not access freed memory | Even with the mitigation of access to a system or database a vulnerable system can present unexpected results that can be exploited by bad actors. Therefore ensure the system does not perform unexpectedly to protect against SQL injections or buffer overflow which could lead to the circumvention of the principle of least privilege by granting bad actors access to unauthorized systems or databases. |
| Sanitize Data Sent to Other Systems | Declare objects with appropriate storage durations, Prevent SQL injection, Do not attempt to create a std::string from a null pointer | These coding standards ensure that data coming in and out of the system are first sanitized. |
| Practice Defense in Depth | Prevent SQL injection, Do not attempt to create a std::string from a null pointer,  Do not access freed memory | SQL injections and buffer overflows are the most common vulnerabilities that bad actors attempt to take advantage of when trying to exploit a system. |
| Use Effective Quality Assurance Techniques | Understand the termination behavior of assert() and abort(),  Handle all exceptions | It is important to test code often and thoroughly through code review, analysis, and testing to identify any vulnerabilities that can occur through out development even if it was once secure changes to the code in other places could open the door to vulnerabilities in previously secure areas of the code. |
| Adopt a Secure Coding Standard | Ensure that division and remainder operations do not result in divide-by-zero errors,  Declare objects with appropriate storage durations,  Guarantee that storage for strings has sufficient space for character data and the null terminator, Prevent SQL injection,  Detect and handle memory allocation errors,  Understand the termination behavior of assert() and abort(),  Handle all exceptions, Do not attempt to create a std::string from a null pointer,  Value-returning functions must return a value from all exit paths,  Do not access freed memory | All of the standards derive from the CERT C/C++ or Java coding standards thus all of them apply. |

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 |  |
| 1.5 | 05/28/2023 | Revised transition to DevSecOps | John Austin | Professor Prasad |
| 2.0 | 06/17/2023 | Polished transition to DevSecOps | John Austin | Pending Approval |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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