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



# 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 | As data is entered, ensure that it is sufficient enough not to be greater than what is acceptable. Data entered should screened for proper formatting such as, if it string data or int. |
| 1. Heed Compiler Warnings | Warnings can be seen as a hinderance, but they can be a gateway for a potential attack down the road. It is important to make note of why a warning is there in the first place, because unlike error messages, a warning will still allow the code to run. If this can be removed by refactoring the code to be compliant and still safe that would be the best case. |
| 1. Architect and Design for Security Policies | The code should be implemented with security policies in mind while also maintaining the function of its intended purpose. Permissions or privileges should be sectioned into separate areas based on these different levels. |
| 1. Keep It Simple | Simplicity keeps everything a little bit easier to maintain and also easier to debug. If there is an elaborate amount to go through, this can make debugging more cumbersome. |
| 1. Default Deny | The default setting is that access is denied. Access is gained by ensuring proper credentials are utilized. |
| 1. Adhere to the Principle of Least Privilege | Using the minimal number of permissions to accomplish a task. As these permissions increase due to the complexity of a task, it is important to only have access to these for as short of a time as possible. This reduces the amount of time an attacker could have with advanced access. |
| 1. Sanitize Data Sent to Other Systems | Data can be corrupted intentionally such as a SQL injection. Sanitizing allows for data to be scrubbed before passing onto another system causing issues. |
| 1. Practice Defense in Depth | Multiple layers of protection will allow a greater chance of keeping privacy and security in mind. The more layers, the increased likelihood of success against attack. |
| 1. Use Effective Quality Assurance Techniques | QA is imperative for any program to be successful. Maintaining a strong adherence to screening such as penetration testing can identify and potentially eliminate threats. |
| 1. Adopt a Secure Coding Standard | Applying best practices during an entire project’s development will help increase security. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not write syntactically ambiguous declarations. |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL53-CPP.+Do+not+write+syntactically+ambiguous+declarations**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL53-CPP.+Do+not+write+syntactically+ambiguous+declarations)

| **Noncompliant Code** |
| --- |
| The syntax is ambiguous because it can be interpreted incorrectly. This example the latter is defined while the former is not, and that also means that the mutex object is never locked. |
| #include <mutex>  static std::mutex m;  static int shared\_resource;  void increment\_by\_42() {  std::unique\_lock<std::mutex>(m);  shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| The lock object is given an identifier other than m and converting is executed properly. |
| #include <mutex>    static std::mutex m;  static int shared\_resource;  void increment\_by\_42() {  std::unique\_lock<std::mutex> lock(m);  shared\_resource += 42;  } |

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

| **Principles(s):** Heed Compiler Warnings (2) – Warnings are a sign that something is potentially an issue and could be interpreted as ambiguous by the IDE. There are steps to resolve the issue in the compiler.  Keep it Simple (4) – Keeping code simple will help reduce any chances of complexities being made, reducing any ambiguity. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | LANG.STRUCT.DECL.FNEST | Nested Function Declaration |
| Helix QAC | 2025.1 | C++1109, C++ 2510 |  |
| Klocwork | 2025.1 | CERT.DCL.AMBIGUOUS |  |
| LDRA tool suite | 9.7.1 | 296 S | Partially implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b  CERT\_CPP-DCL53-c | Parameter names in function declarations should not be enclosed in parentheses Local variable names in variable declarations should not be enclosed in parentheses Avoid function declarations that are syntactically ambiguous |
| Polyspace Bug Finder | R2024b | CERT C++: DCL53CPP | Checks for declarations that can be confused between:   * Function and object declaration * Unnamed object or function parameter declaration   Rule fully covered. |
| Clang | 3.9 | -Wvexing-parse |  |
| SonarQube C/C++ Plugin | 4.10 | S3468 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized memory. |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory)

| **Noncompliant Code** |
| --- |
| An uninitialized local variable results in undefined behavior. |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The variable is initialized which will print the value. |
| #include <iostream>    void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):** Heed Compiler Warnings (2) – Warnings are a sign that something is potentially an issue and could be interpreted as ambiguous by the IDE. There are steps to resolve the issue in the compiler. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | uninitialized-read | Partially checked |
| Clang | 3.9 | -Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| CodeSonar | 9.0p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local Uninitialized variable |
| Helix QAC | 2025.1 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 |  |
| Klocwork | 2025.1 | UNINIT.CTOR.MIGHT  UNINIT.CTOR.MUST  UNINIT.HEAP.MIGHT  UNINIT.HEAP.MUST  UNINIT.STACK.ARRAY.MIGHT  UNINIT.STACK.ARRAY.MUST  UNINIT.STACK.ARRAY.PARTIAL.MUST  UNINIT.STACK.MIGHT  UNINIT.STACK.MUST |  |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2024b | CERT C++: EXP53-CPP | Checks for:  Non-initialized variable  Non-initialized pointer  Rule partially covered. |
| PVS-Studio | 7.37 | V546, V573, V614, V670, V679, V730, V788, V1007, V1050 |  |
| RuleChecker | 22.10 | uninitialized-read | Partially checked |

#### Coding Standard 3

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

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator**](https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator)

| **Noncompliant Code** |
| --- |
| This code reads one character at a time instead of reading the entire line and then from stdin instead of reading the entire line at once. The stdin stream is read until end-of-file is encountered or a newline character is read. This could result in an overflow. |
| #include <stdio.h>    enum { BUFFERSIZE = 32 };    void func(void) {  char buf[BUFFERSIZE];  char \*p;  int ch;  p = buf;  while ((ch = getchar()) != '\n' && ch != EOF) {  \*p++ = (char)ch;  }  \*p++ = 0;  if (ch == EOF) {  /\* Handle EOF or error \*/  }  } |

| **Compliant Code** |
| --- |
| This allows for the loop to read the entire line and will terminate the string at the appropriate time. |
| #include <stdio.h>    enum { BUFFERSIZE = 32 };  void func(void) {  char buf[BUFFERSIZE];  int ch;  size\_t index = 0;  bool truncated = false;  while ((ch = getchar()) != '\n' && ch != EOF) {  if (index < sizeof(buf) - 1) {  buf[index++] = (char)ch;  } else {  truncated = true;  }  }  buf[index] = '\0'; /\* Terminate string \*/  if (ch == EOF) {  /\* Handle EOF or error \*/  }  if (truncated) {  /\* Handle truncation \*/  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard (10) – Following secure coding best practices can reduce the amount of code that is improperly initialized. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | No | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 |  | 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 | 9.0p0 | 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 |  |  | 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 |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| Fortify SCA | 5.0 |  |  |
| Helix QAC | 2024.4 | C2840, C5009, C5038  C++0145, C++5009, C++5038  DF2840, DF2841, DF2842, DF2843, DF2845, DF2846, DF2847, DF2848, DF2930, DF2931, DF2932, DF2933, DF2935, DF2936, DF2937, DF2938 |  |
| Klocwork | 2024.4 | SV.FMT\_STR.BAD\_SCAN\_FORMAT SV.UNBOUND\_STRING\_INPUT.FUNC |  |
| LDRA tool suite | 9.7.1 | 489 S, 109 D, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++test | 2024.2 | CERT\_C-STR31-a  CERT\_C-STR31-b  CERT\_C-STR31-c  CERT\_C-STR31-d  CERT\_C-STR31-e | Avoid accessing arrays out of bounds  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Avoid using unsafe string functions which may cause buffer overflows |
| PC-lint Plus | 1.4 | 421, 498 | Partially supported |
| Polyspace Bug Finder | R2024b | CERT C: Rule STR31-C | 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. |
| PVS-Studio | 7.37 | V518, V645, V727, V755 |  |
| Splint | 3.1.1 |  |  |
| TrustInSoft Analyzer | 1.38 | mem\_access | Exhaustively verified (see one compliant and one non-compliant example). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV] | Prevent SQL injection. |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection**](https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection)

| **Noncompliant Code** |
| --- |
| This code still allows for a username to be bypassed through a SQL injection. |
| 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 corrected code checks for the length of a username entered which will prevent an overflow from a large number of characters being entered. |
| 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):** Validate Input Data (1) – Data must be validated to prevent SQL injection, overflow, or other unexpected behavior.  Keep it Simple (4) – Keeping code simple will help reduce any chances of complexities being made, reducing any ambiguity. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| CodeSonar | 9.0p0 | JAVA.IO.INJ.SQL | SQL Injection |
| 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 |
| Fortify | 1.0 | HTTP\_Response\_Splitting  SQL\_Injection\_\_Persistence  SQL\_Injection | Implemented |
| Klocwork | 2024.4 | SV.DATA.DB  SV.SQL  SV.SQL.DBSOURCE | Implemented |
| Parasoft Jtest | 2024.2 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SonarQube | 9.9 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |
| SpotBugs | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Detect and handle memory allocation errors. |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors)

| **Noncompliant Code** |
| --- |
| The results can lead to abnormal termination of the program due to how the array is created. The caller assumes no exemptions are to be thrown due to the noexcept function. |
| #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** |
| --- |
| The std::bad\_alloc exepction is used in this example if sufficient memory cannot be allocated. |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy;  try {  copy = new int[size];  } catch(std::bad\_alloc) {  // Handle error  return;  }  // At this point, copy has been initialized to allocated memory  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):** Architect and Design for Security Policies (3) – deleting of old data will help alleviate resources that would be spent maintaining that information. This leftover data could be leveraged if not properly disposed of.  Practice Defense in Depth (8) – multiple layers are in place to ensure that if a layer is compromised, there are still many more that are all sectioned off, compartmentalizing the threat. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  |  |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| Helix QAC | 2025.1 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 |  |
| Klocwork | 2025.1 | 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 | 9.7.1 | 45 D | Partially implemented |
| Parasoft C/C++test | 2024.2 | 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++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2024b | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| PVS-Studio | 7.37 | V522, V668 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression. |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression**](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression)

| **Noncompliant Code** |
| --- |
| The assert() macro is used to help with memory mapping and is not a proper solution. This assert() macro is vital for function. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| A conditional statement is added which evaluates assertions at compile time. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** [Architect and Design for Security Policies (3) – Proper implementation of assert() will ensure proper evaluation occurs appropriately. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 9.0p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not leak resources when handling exceptions. |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR57-CPP.+Do+not+leak+resources+when+handling+exceptions**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR57-CPP.+Do+not+leak+resources+when+handling+exceptions)

| **Noncompliant Code** |
| --- |
| Due to improper handling, an exception is thrown causing a resource leak. |
| #include <new>    struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };    void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }    try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  throw;  }  delete pst;  } |

| **Compliant Code** |
| --- |
| Resources are no longer leaked, due to a delete call that is made when an exception is thrown. This will need to be implemented for each distinct clean up. |
| #include <new>  struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };  void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }  try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  delete pst;  throw;  }  delete pst;  } |

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

| **Principles(s):** Architect and Design for Security Policies (3) – Proper handling of exceptions should take place. Exceptions should be thrown when expected and no other information should able to be accessed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2025.1 | DF4756, DF4757, DF4758 |  |
| Klocwork | 2025.1 | CL.MLK  MLK.MIGHT  MLK.MUST  MLK.RET.MIGHT  MLK.RET.MUST  RH.LEAK |  |
| LDRA tool suite | 9.7.1 | 50 D | Partially implemented |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-ERR57-a | Ensure resources are freed |
| Polyspace Bug Finder | R2024b | CERT C++: ERR57-CPP | Checks for:  Resource leak caused by exception  Object left in partially initialized state  Bad allocation in constructor |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice  Memory Protection] | [STD-008-CPP] | Properly deallocate dynamically allocated resources |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources)

| **Noncompliant Code** |
| --- |
| A local variable is passed incorrectly, causing undefined behavior. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  // ...  delete s1;  } |

| **Compliant Code** |
| --- |
| A destructor is added to eliminate the undefined behavior from the above example. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...  s1->~S();  } |

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

| **Principles(s):** Architect and Design for Security Policies (3) – Proper handling of resources should occur especially when deallocating resources. Guidelines are in place to ensure resources are not mishandled. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | invalid\_dynamic\_memory\_allocation  dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM51 |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 9.0p0 | ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Double free  Type mismatch  Leak |
| Helix QAC | 2025.1 | C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339, C++4262, C++4263, C++4264 |  |
| Klocwork | 2025.1 | CL.FFM.ASSIGN  CL.FFM.COPY  CL.FMM  CL.SHALLOW.ASSIGN  CL.SHALLOW.COPY  FMM.MIGHT  FMM.MUST  FNH.MIGHT  FNH.MUST  FUM.GEN.MIGHT  FUM.GEN.MUST  UNINIT.CTOR.MIGHT  UNINIT.CTOR.MUST  UNINIT.HEAP.MIGHT  UNINIT.HEAP.MUST |  |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially implemented |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Use the same form in corresponding calls to new/malloc and delete/free  Always provide empty brackets ([]) for delete when deallocating arrays  Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor  Properly deallocate dynamically allocated resources |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2024b | CERT C++: MEM51-CPP | Checks for:  Invalid deletion of pointer  Invalid free of pointer  Deallocation of previously deallocated pointer  Rule partially covered. |
| PVS-Studio | 7.37 | V515, V554, V611, V701, V748, V773, V1066 |  |
| SonarQube C/C++ Plugin | 4.10 | S1232 |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice  String Correctness] | [STD-009-CPP] | Do not confuse narrow and wide character strings and functions |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/c/STR38-C.+Do+not+confuse+narrow+and+wide+character+strings+and+functions**](https://wiki.sei.cmu.edu/confluence/display/c/STR38-C.+Do+not+confuse+narrow+and+wide+character+strings+and+functions)

| **Noncompliant Code** |
| --- |
| Incorrect use of the strncpy() function may cause earlier termination due to null bytes being contained inside wide characters. |
| #include <stddef.h>  #include <string.h>    void func(void) {  wchar\_t wide\_str1[] = L"0123456789";  wchar\_t wide\_str2[] = L"0000000000";  strncpy(wide\_str2, wide\_str1, 10);  } |

| **Compliant Code** |
| --- |
| Both the strncpy () and wscncpy() functions are used properly for narrow and wide characters respectively. |
| #include <string.h>  #include <wchar.h>    void func(void) {  wchar\_t wide\_str1[] = L"0123456789";  wchar\_t wide\_str2[] = L"0000000000";  /\* Use of proper-width function \*/  wcsncpy(wide\_str2, wide\_str1, 10);  char narrow\_str1[] = "0123456789";  char narrow\_str2[] = "0000000000";  /\* Use of proper-width function \*/  strncpy(narrow\_str2, narrow\_str1, 10);  } |

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

| **Principles(s):** Heed Compiler Warnings (2) – If narrow strings are used when wide strings are required an overflow can occur. It is critical to know the differences between the string types.  Architect and Design for Security Policies (3) – Proper handling of string types should occur to prevent overflow from occurring. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | wide-narrow-string-cast  wide-narrow-string-cast-implicit | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR38 | Fully implemented |
| Clang | 3.9 | -Wincompatible-pointer-types |  |
| CodeSonar | 9.0p0 | LANG.MEM.BO  LANG.MEM.TBA | Buffer Overrun  Tainted Buffer Access |
| Coverity | 2017.07 | PW | Implemented |
| Cppcheck Premium | 24.11.0 | premium-cert-str38-c |  |
| Helix QAC | 2024.4 | C0432  C++0403 |  |
| Klocwork | 2024.4 | CXX.DIFF.WIDTH.STR\_AND\_FUNC |  |
| Parasoft C/C++test | 2024.2 | CERT\_C-STR38-a | Do not confuse narrow and wide character strings and functions |
| PC-lint Plus | 1.4 | 2454, 2480, 2481 | Partially supported: reports illegal conversions involving pointers to char or wchar\_t as well as byte/wide-oriented stream inconsistencies |
| Polyspace Bug Finder | R2024b | CERT C: Rule STR38-C | Checks for misuse of narrow or wide character string (rule fully covered) |
| RuleChecker | 24.04 | wide-narrow-string-cast  wide-narrow-string-cast-implicit | Partially checked |
| TrustInSoft Analyzer | 1.38 | pointer arithmetic | Partially verified. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice  Memory Protection] | [STD-010-CPP] | Do not dereference null pointers |

**Source:** [**https://wiki.sei.cmu.edu/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers**](https://wiki.sei.cmu.edu/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers)

| **Noncompliant Code** |
| --- |
| Due to a dereference of a null pointer, there is overwriting memory of user-defined data. |
| #include <png.h> /\* From libpng \*/  #include <string.h>    void func(png\_structp png\_ptr, int length, const void \*user\_data) {  png\_charp chunkdata;  chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);  /\* ... \*/  memcpy(chunkdata, user\_data, length);  /\* ... \*/   } |

| **Compliant Code** |
| --- |
| This corrected code ensures that user data is not null. |
| #include <png.h> /\* From libpng \*/  #include <string.h>   void func(png\_structp png\_ptr, size\_t length, const void \*user\_data) {  png\_charp chunkdata;  if (length == SIZE\_MAX) {  /\* Handle error \*/  }  if (NULL == user\_data) {  /\* Handle error \*/  }   chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);  if (NULL == chunkdata) {  /\* Handle error \*/  }  /\* ... \*/  memcpy(chunkdata, user\_data, length);  /\* ... \*/   } |

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

| **Principles(s):** Heed Compiler Warnings (2) – Dereferencing null pointers will cause unwanted behavior leaving the code more prone to exposure. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | null-dereferencing | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-EXP34 |  |
| CodeSonar | 9.0p0 | LANG.MEM.NPD  LANG.STRUCT.NTAD  LANG.STRUCT.UPD | Null pointer dereference  Null test after dereference  Unchecked parameter dereference |
| Compass/ROSE |  |  | Can detect violations of this rule. In particular, ROSE ensures that any pointer returned by malloc(), calloc(), or realloc() is first checked for NULL before being used (otherwise, it is free()-ed). ROSE does not handle cases where an allocation is assigned to an lvalue that is not a variable (such as a struct member or C++ function call returning a reference) |
| Coverity | 2017.07 | CHECKED\_RETURN  NULL\_RETURNS  REVERSE\_INULL  FORWARD\_NULL | Finds instances where a pointer is checked against NULL and then later dereferenced  Identifies functions that can return a null pointer but are not checked  Identifies code that dereferences a pointer and then checks the pointer against NULL  Can find the instances where NULL is explicitly dereferenced or a pointer is checked against NULL but then dereferenced anyway. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary |
| Cppcheck | 2.15 | nullPointer, nullPointerDefaultArg, nullPointerRedundantCheck |  |
| Cppcheck Premium | 24.11.0 | nullPointer, nullPointerDefaultArg, nullPointerRedundantCheck |  |
| Helix QAC | 2024.4 | DF2810, DF2811, DF2812, DF2813 | Fully implemented |
| Klocwork | 2024.4 | 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 | Fully implemented |
| LDRA tool suite | 9.7.1 | 45 D, 123 D, 128 D, 129 D, 130 D, 131 D, 652 S | Fully implemented |
| Parasoft C/C++test | 2024.2 | CERT\_C-EXP34-a | Avoid null pointer dereferencing |
| Parasoft Insure++ |  |  | Runtime analysis |
| PC-lint Plus | 1.4 | 413, 418, 444, 613, 668 | Partially supported |
| Polyspace Bug Finder | R2024b | CERT C: Rule EXP34-C | Checks for use of null pointers (rule partially covered) |
| PVS-Studio | 7.37 | V522, V595, V664, V713, V1004 |  |
| SonarQube C/C++ Plugin | 3.11 | S2259 |  |
| Splint | 3.1.1 |  |  |
| TrustInSoft Analyzer | 1.38 | mem\_access | Exhaustively verified (see one compliant and one non-compliant example). |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

The security principles can be incorporated within the already existing DevSecOps. The pre-production phase plays a vital role in laying the foundation for what the future project will look like. The Assess and Plan phase can incorporate the overall architecture and design for the project while revisiting the plan as new threats arise. Principles such as validating input data, heed compiler warnings, and keeping it simple, using effective Quality Assurance techniques, and adopting a secure coding standard are going to be focused on. Once in the production phase aspects like sanitizing data sent to other systems, default deny and practicing defense in depth are now to be focused because there is no indication when an attacker will strike. Being proactive rather than reactive is going to save time and money.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Probable | Medium | P12 | L1 |
| STD-003-CPP | High | Likely | No | P9 | L2 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely |  | P3 | L3 |
| STD-007-CPP | Low | Probable | High | P2 | L3 |
| STD-008-CPP | High | Likely | Medium | P18 | L1 |
| STD-009-CPP | High | Likely |  | P18 | L1 |
| STD-010-CPP | High | Likely |  | 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 at rest | This refers to data being encrypted will not in use. This data is useless unless there is access to the decryption key. Encryption can help during an attack as long as the key remains uncompromised because without the key, the data is still safe within and cannot be accessed. |
| Encryption in flight | This refers to the transportation of data from one location to another. To safely do this, data must first be encrypted before the transit process can begin. |
| Encryption in use | This refers to accessing data after using the decryption key. The data is exposed and must find a way to encrypt it during use to prevent loss of precious data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is verifying that the user is the correct user by providing login information and possible additional forms of verifications such as Two-Step authentication. Biometric data can also be used to authenticate a user if the resources are allowed to use it. |
| Authorization | Once a user is verified, only certain permissions will be accessible to them. The principle of least privilege is an excellent use of this. If a user had higher levels of authorization for their account, a hacker could try and exploit this and have much higher levels of access they should not be able to. |
| Accounting | This is overseeing and monitoring what is occurring with all users. This is another blockade that is in place to ensure security is maintained. If during monitoring it is revealed that someone has given themselves “super” permissions this could flag the system because for what reason was someone given much higher levels of authorization. |

**\***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 |  |
| 1.1 | 5/25/2025 | Milestone 3 Requirements | Robert Lulashi |  |
| 1.2 | 6/14/2025 | Project 1 Requirements | Robert Lulashi |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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