**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 | All input data should be check and validated to prevent any vulnerabilities. |
| 1. Heed Compiler Warnings | Compiler warnings should be looked at and considered because they could show issues and vulnerabilities that may not be noticed during development. |
| 1. Architect and Design for Security Policies | Consider the architecture and design of a system when creating and implementing security policies. |
| 1. Keep It Simple | Keep programs simple and on task so that there are not any unnecessary systems in place that could lead to issues and vulnerabilities. |
| 1. Default Deny | By default, deny all access that has not been permitted by the policy being used. |
| 1. Adhere to the Principle of Least Privilege | Only grant users permission for files that are necessary to complete their job. |
| 1. Sanitize Data Sent to Other Systems | Check data that is sent between systems for things that could cause issues such as unused functions and calls. |
| 1. Practice Defense in Depth | Have multiple layers of defense in a system to keep vulnerabilities to a minimum. |
| 1. Use Effective Quality Assurance Techniques | Use techniques that makes sure that a system meets all quality standards and safety standards. |
| 1. Adopt a Secure Coding Standard | Have a standard for how secure all code should be to make sure everything is consistent between projects and employees. |

### 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 use floating point variable as loop counters |

https://wiki.sei.cmu.edu/confluence/display/java/NUM09-J.+Do+not+use+floating-point+variables+as+loop+counters

| **Noncompliant Code** |
| --- |
| The non-compliant code is an issue because it uses a float for the loop which when iterated can cause the loop to iterate the wrong amount of times based on how it rounds which can lead to errors down the line. |
| float i;  for (i = 0.1, i < 10, i++) {  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The compliant code works because it uses an integer for the loop which means that it should iterate the correct number of times because there wont be any issues from the rounding. |
| int i;  for (i = 1, i < 10, i++) {  std::cout << i;  } |

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

| **Principles(s):** Validate Input Data – making sure that data for loops makes sense can help prevent errors.  Keep It Simple – using non integer values for loops over complicates the loop and can lead to unforeseen errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Low | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | For-loop-float | Fully Checked |
| Clang | 3.9 | Cert-flp30-c | Checked by clang-tidy |
| CodeSonar | 8.1p0 | LANG.STRUCT.LOOP.FPC | Float-typed loop counter |
| LDRA tool suite | 9.7.1 | 39 S | Fully Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure that integer conversion do not result in lost or misinterpreted data |

https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data

| **Noncompliant Code** |
| --- |
| The non-compliant code does not work because it does not check that the integer conversion does not effect the data. |
| void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The compliant code works because while it still does the integer conversion it does also check using an if-else statement that the conversion does not effect the data. |
| void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  if (u\_a <= SCHAR\_MAX) {  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  } else {  /\* Handle error \*/  }  } |

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

| **Principles(s):** Validate Input Data – The integers being converted, and the accuracy and effects need to be verified. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Klocwork | 2024.1 | PORTING.CAST.SIZE |  |
| TrustInSoft Analyzer | 1.38 | Signed\_downcast | Exhaustively verified. |
| Astrée | 23.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Exclude user input from format strings |

https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings

| **Noncompliant Code** |
| --- |
| The non-compliant code does not work because with the msg includes user inputs that are passed as a format string argument. |
| void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fprintf(stderr, msg);  free(msg);  } |

| **Compliant Code** |
| --- |
| The compliant code works because it keeps user inputs out of the formatted string by using fputs() instead of fprint(). |
| void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fputs(msg, stderr);  free(msg);  } |

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

| **Principles(s):** Validate Input Data – If user inputs are able to be effect format strings it can cause them to gain access where they shouldn’t so any strings entered by users should be validated to make sure that they are not trying to affect these format strings. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIgh | Likely | Medium | 18 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | Partially implemented. |
| CodeSonar | 8.1p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| Coverity | 2017.07 | TAINTED\_STRING | Implemented |
| Helix QAC | 2024.1 | DF4916, DF4917, DF4918 |  |

#### Coding Standard 4

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

https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection

| **Noncompliant Code** |
| --- |
| The noncompliant code does not meet the standard because it still allows injection through unsanitized inputs. |
| 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** |
| --- |
| The compliant code works because it validates user inputs to make sure that injection cannot occur. |
| 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 - SQL injection can happen because of user input so these inputs need to be validated that they do not attempt this.  Use Effective Quality Assurance Techniques – Since SQL injection is something that should always be prevented there should be techniques that are used to ensure it does not happen.  Sanitize Data Sent to Other Systems – since SQL injection is caused by sending a malicious SQL query any data inputs should be checked and cleared of any possible queries that could cause injection. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 18 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL injection (java) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Ensure that operations and signed integers do not result in overflow. |

https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow

| **Noncompliant Code** |
| --- |
| The non-compliant code allows overflow to occur during the addition of si\_a and si\_b. |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The compliant code works because it uses parameters to ensure that the addition cannot cause an overflow to occur. |
| void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

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

| **Principles(s):** Verify Input Data – Because user inputs can also cause overflow their input should be verified that it will not cause it.  Adopt A Secure Coding Standard – This should be a standard used by all programmers because overflow is important to protect against.  Sanitize Data Sent to Other Systems – Because these operations and integers can be used by other systems, they should be made sure to not cause issues for those systems as well. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | 9 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | Integer-overflow | Fully Checked |
| Helix QAC | 24.1 | C2800, C2860, C++2800, C++2860, DF2801, DF2802, DF2803, DF2861, DF2862, DF2863 |  |
| Coverity | 2017.07 | TAINTED\_SCALER, BAD\_SHIFT | implemented |
| Klocwork | 2024.1 | NUM.OVERFLOW, CWARN.NOEFFECT.OUTOFRANGE, NUM.OVERFLOW.DF |  |

#### 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 |

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 noncompliant code does not work in this case because while there is an assertation it does not work well because it is only ran during runtime and only if the code containing it is executed. |
| 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** |
| --- |
| The compliant code works because it makes the assertation run at compilation and the error results from it will be more informative. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),  "Structure must not have any padding"); |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – This principle is related because testing keep the quality of the program up because it will be tested for issues beforehand. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | 1 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| 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 abruptly terminate the program |

https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR50-CPP.+Do+not+abruptly+terminate+the+program

| **Noncompliant Code** |
| --- |
| The noncompliant code is an issue because the way that it works is if there is an error it just terminates the program right away. |
| void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  throwing\_func();  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

| **Compliant Code** |
| --- |
| The compliant code works in this case because it uses a try and catch statement to handle the error in a way that does not just abruptly terminate the program. |
| void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – This principle applies because having something so that users know why a program closes or to prevent the program from closing is important towards the quality of the program. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | 4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Stdlib-use | Partially checked |
| Helix QAC | 2024.1 | C++5014 |  |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Polyspace Bug Finder | R2023b | CERT C++:ERR50-CPP | Checks for implicit call to terminate() function |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Type | [STD-008-CPP] | Declare identifiers before using them |

https://wiki.sei.cmu.edu/confluence/display/c/DCL31-C.+Declare+identifiers+before+using+them

| **Noncompliant Code** |
| --- |
| The noncompliant code does not meet the standard because the identifier is not declared with its data type. |
| extern foo; |

| **Compliant Code** |
| --- |
| The compliant code meets the standard because the identifier is fully declared with its data type. |
| extern int foo; |

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

| **Principles(s):** Adopt a secure coding standard – Having a standard set like this keeps code consistent so that others can expect how the code will work even if someone else made it. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | 3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL31 | Fully implemented |
| Coverity | 2017.07 | MISRA C 2012 Rule 8.1 | Implemented |
| ÉCLAIR | 1.2 | CC2.DCL31 | Fully implemented |
| Helix QAC | 2024.1 | C0434, C2050, C2051, C3335 | Fully implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | [STD-009-CPP] | Free dynamically allocated memory when no longer needed |

https://wiki.sei.cmu.edu/confluence/display/c/MEM31-C.+Free+dynamically+allocated+memory+when+no+longer+needed

| **Noncompliant Code** |
| --- |
| The noncompliant code does not meet the standard because it allocates memory with the text\_buffer but does not free it after its use. |
| enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }  return 0;  } |

| **Compliant Code** |
| --- |
| The compliant code meets the standard because it still allocates memory with the text\_buffer but frees it later on after it has been used. |
| enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }    free(text\_buffer);  return 0;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – This principle applies because keeping memory usage to a reasonable level is important and can improve the quality of the program with better runtime. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | 8 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Cppcheck | 1.66 | leakReturnValNotUsed | Doesn't use return value of memory allocation function |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM31 | Can detect dynamically allocated resources that are not freed |
| Coverity | 2017.07 | RESOURCE\_LEAK  ALLOC\_FREE\_MISMATCH | Finds resource leaks from variables that go out of scope while owning a resource |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| String Correctness | [STD-010-CPP] | Use valid format strings |

https://wiki.sei.cmu.edu/confluence/display/c/FIO47-C.+Use+valid+format+strings

| **Noncompliant Code** |
| --- |
| The noncompliant code does not meet the standard because the format string is done wrong where the error\_type argument is matched with the s specifier instead of the d specifier. |
| void func(void) {  const char \*error\_msg = "Resource not available to user.";  int error\_type = 3;  /\* ... \*/  printf("Error (type %s): %d\n", error\_type, error\_msg);  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The compliant code because the format string is done right with the error\_type argument is matched with d instead of s. |
| void func(void) {  const char \*error\_msg = "Resource not available to user.";  int error\_type = 3;  /\* ... \*/  printf("Error (type %d): %s\n", error\_type, error\_msg);    /\* ... \*/  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – Using proper format stings keeps the quality of the program better. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool Suite | 9.7.1 | 486 S  589 S | Fully implemented |
| CodeSonar | 8.1p0 | IO.INJ.FMT  MISC.FMT  MISC.FMTTYPE | Format string injection  Format string  Format string type error |
| Coverity | 2017.07 | PW | Reports when the number of arguments differs from the number of required arguments according to the format string |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO47 | Fully implemented |

### 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.

Automation will be used in the process to check different security standards that need to be checked for all programs. They will be used while the programs are being created so that testing and coding are done at the same time. The current DevSecOps process can be modified by adding in planning for the tools that will be used in the design phase of the build. Then in the verifying and testing stage they will be used to check for errors and vulnerabilities. They can be used in the detect stage to detect any new issues that are found or come up. It can also be used in the maintain phase incase of any new vulnerabilities that are found and need to be tested for.

### 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 | Probable | Low | 6 | 2 |
| STD-002-CPP | High | Probable | High | 6 | 2 |
| STD-003-CPP | High | Likely | Medium | 18 | 1 |
| STD-004-CPP | High | Likely | Medium | 18 | 1 |
| STD-005-CPP | High | Likely | High | 9 | 2 |
| STD-006-CPP | Low | Unlikely | High | 1 | 3 |
| STD-007-CPP | Low | Probable | Medium | 4 | 3 |
| STD-008-CPP | Low | Unlikely | Low | 3 | 3 |
| STD-009-CPP | Medium | Probable | Medium | 8 | 2 |
| STD-010-CPP | High | Unlikely | Medium | 6 | 2 |

### 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 | Encryption at rest is making sure that data that is stationary such as on a stored disk is encrypted and protected. This is because this data could contain important data such as user information even when it is not being used. |
| Encryption in flight | Encryption in flight means to protect data as it is in transit because moving data is often targeted and needs to be kept safe. |
| Encryption in use | Encryption in use means to have the output of the data still be encrypted even if the system has decrypted and is using the data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is making sure to authenticate things such as users identity to prevent users from accessing areas they should not. This includes user logins and where new users are given access to. |
| Authorization | Authorization is making it so that different users are only able to interact with parts of the system that they need to, this keeps general users from reaching the backend of more important systems where they could cause issues. |
| Accounting | Accounting is keeping track of access of different parts of the system so any issues can be traced back. This includes things like tracking any ways that the database are changed to detect for a breach and files that are used by users to see if they are accessing ones they should not. |

### 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 | 04/14/2024 | Initial Document | Ethan Hutchison |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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