**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 | Data received from a client should never be trusted in any scenario. All input should be sanitized to prevent attacks such as injections. Input can easily be written in such a way to check the system for vulnerabilities. |
| 1. Heed Compiler Warnings | Compiler warnings should not be ignored. Compiler warnings are useful for identifying potential issues or instability with software. If these are ignored there is a good chance of finding vulnerabilities or instability in the software. |
| 1. Architect and Design for Security Policies | A software’s design is very important when it comes to its security. Security principles such as these should be considered when designing software well before the coding process. Having good practice of these software security principles will help to stop future issues such as attacks or instability in a software or system. |
| 1. Keep It Simple | Sometimes a solution is as easy as you initially think it to be. Do not overthink a solution to a problem, look for solid straight forward solutions. |
| 1. Default Deny | Authorization should be approved after validation only! Access should never be given by default, so therefore always DENY by default. |
| 1. Adhere to the Principle of Least Privilege | Assume each user has the least amount of privilege when it comes to access or authorization. Each user could be an attacker and a good first defense is a good offense by limiting what the user can do before we know if they are a legitimate user. |
| 1. Sanitize Data Sent to Other Systems | As input is super important to sanitize and validate before any of it can be used in a system, its equally as important to sanitize again when being sent to other systems. |
| 1. Practice Defense in Depth | There is no such thing as a perfect defense! However, it’s important to be redundant as possible and to create many layers of security to at a minimum slow down a hacker or attacker. |
| 1. Use Effective Quality Assurance Techniques | How do you know if your software works properly? Testing through an effective quality assurance process is important for software development. For security purposes it is a good idea to also have those who test your software to attempt to act like an attacker would for the sake of finding vulnerabilities before real attackers do. |
| 1. Adopt a Secure Coding Standard | Secure coding starts with the programmers and the management. One way of increasing quality of code is by performing code reviews where quality can be checked to find problems. |

## 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** | [INT-33-C] | Ensure that division and remainder operations do not result in divide-by-zero errors |

| **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](https://wiki.sei.cmu.edu/confluence/display/c/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):** Validate Input Data – This standard extends the idea of validating the input to make sure its not going to cause errors down the line in your software. In this case you are validating that the input for the division operations does not result in a divide by zero operation. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Very Likely | Extremely Costly | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | Checking for various issues in the code and is more in depth than typical code analysis tools that come in IDEs. | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| Visual Studio Code Analysis | 16.9.0 | [Insert text.] | Performs a low level code analysis that can be helpful to find issues with code, however CPPCheck is preferred for this task. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [INT-32-C] | Ensure that operations on signed integers do not result in overflow |

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

| **Compliant Code** |
| --- |
| This compliant solution ensures that the addition operation cannot overflow, regardless of representation: |
| #include <limits.h>    void f(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum;    if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||        ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {      /\* Handle error \*/    } else {      sum = si\_a + si\_b;    }    /\* ... \*/  } |

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

|  |
| --- |
| **Principles(s):** Keep It Simple – issues like this are commonly caused by overthinking a solution, sometimes a programmer can overlook the code they just wrote because they are trying to finish quickly but end up causing an overflow issue such as this.  Heed Compiler Warnings – These types of warnings would be given by code analysis tools which are common through all of these standards. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Rare | Minor | Medium | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | [Insert text.] | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| Visual Studio Code Analysis | 16.9.0 | [Insert text.] | Performs a low level code analysis that can be helpful to find issues with code, however CPPCheck is preferred for this task. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STR-30-C] | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the char pointer str is initialized to the address of a string literal. Attempting to modify the string literal is [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior): |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. (See [STR11-C. Do not specify the bound of a character array initialized with a string literal](https://wiki.sei.cmu.edu/confluence/display/c/STR11-C.+Do+not+specify+the+bound+of+a+character+array+initialized+with+a+string+literal).) This code creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

|  |
| --- |
| **Principles(s):** Heed Compiler Warnings – These types of issues are found in code reviews and in code analysis tools. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Rare | Low | Minor | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | Checking for various issues in the code and is more in depth than typical code analysis tools that come in IDEs. | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| Visual Studio | 16.9.0 | [Insert text.] | Performs a low level code analysis that can be helpful to find issues with code, however CPPCheck is preferred for this task. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [IDS-00-J] | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows JDBC code to authenticate a user to a system. The password is passed as a char array, the database connection is created, and then the passwords are hashed.  Unfortunately, this code example permits a SQL injection attack by incorporating the unsanitized input argument username into the SQL command, allowing an attacker to inject validuser' OR '1'='1. The password argument cannot be used to attack this program because it is passed to the hashPassword() function, which also [sanitizes](https://wiki.sei.cmu.edu/confluence/display/java/Rule+BB.+Glossary#RuleBB.Glossary-sanitize) the input. |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;    class Login {    public Connection getConnection() throws SQLException {      DriverManager.registerDriver(new              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"      return DriverManager.getConnection(dbConnection);    }      String hashPassword(char[] password) {      // Create hash of password    }      public void doPrivilegedAction(String username, char[] password)                                   throws SQLException {      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):** Architect and Design for Security Policies – It should be planned in a design to attempt to use prepared statements when working with databases especially when users are submitting input that interacts with a database. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Eclipse | ? | Code analysis can provide a way to find errors that you may have missed. | prerera code analysis to find potential issues with the code before those issues cause further problems. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [MEM-30-C] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| This example from Brian Kernighan and Dennis Ritchie [[Kernighan 1988](https://wiki.sei.cmu.edu/confluence/display/c/AA.+Bibliography#AA.Bibliography-Kernighan88)] shows both the incorrect and correct techniques for freeing the memory associated with a linked list. In their (intentionally) incorrect example, p is freed before p->next is executed, so that p->next reads memory that has already been freed. |
| #include <stdlib.h>    struct node {  **int** value;    struct node \*next;  };    void free\_list(struct node \*head) {    for (struct node \*p = head; p != NULL; p = p->next) {  **free**(p);    }  } |

| **Compliant Code** |
| --- |
| Kernighan and Ritchie correct this error by storing a reference to p->next  in q before freeing p: |
| #include <stdlib.h>    struct node {  **int** value;    struct node \*next;  };    void free\_list(struct node \*head) {    struct node \*q;    for (struct node \*p = head; p != NULL; p = q) {      q = p->next;  **free**(p);    }  } |

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

|  |
| --- |
| **Principles(s):** Heed Compiler Warnings – Compiler warnings are helpful and would point out these types of errors. Code analysis tools can also be more precise to find errors like these and give suggestions on how to fix them. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | Checking for various issues in the code and is more in depth than typical code analysis tools that come in IDEs. | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [ERR-06-C] | Understand the termination behavior of assert() and abort() |

| **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):** Keep it Simple – Sometimes we get involved in our coding process and make mistakes such as using a method that performs similarly to another method however can lead to issues. Take the time to write code properly if it’s necessary take breaks and come back when you can focus.  Heed Compiler Warnings – Compiler warnings and code analysis software can help you find common issues like this that can lead to larger problems. Code reviews are also a great source for finding code that can be troublesome. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Very Rare | None | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | Checking for various issues in the code and is more in depth than typical code analysis tools that come in IDEs. | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [ERR-56-CPP] | Guarantee exception safety |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. The function deallocates array and assigns the element counter, nElems, before allocating a new block of memory for the copy. As a result, if the new expression throws an exception, the function will have modified the state of both member variables in a way that violates the implicit invariants of the class. Consequently, such an object is in an indeterminate state and any operation on it, including its destruction, results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <cstring>    class IntArray {  **int** \*array;    std::**size\_t** nElems;  public:    // ...      ~IntArray() {      delete[] array;    }        IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {      if (this != &rhs) {        delete[] array;        array = nullptr;        nElems = rhs.nElems;        if (nElems) {          array = new **int**[nElems];          std::**memcpy**(array, rhs.array, nElems \* sizeof(\*array));        }      }      return \*this;    }      // ...  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the copy assignment operator provides the [strong exception safety](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-strongexceptionsafety) guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. In addition, by copying the array to the newly allocated storage before deallocating the existing array, the function avoids the test for self-assignment, which improves the performance of the code in the common case [[Sutter 2004](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-Sutter04)]. |
| #include <cstring>    class IntArray {  **int** \*array;    std::**size\_t** nElems;  public:    // ...      ~IntArray() {      delete[] array;    }      IntArray(const IntArray& that); // nontrivial copy constructor      IntArray& operator=(const IntArray &rhs) {  **int** \*tmp = nullptr;      if (rhs.nElems) {        tmp = new **int**[rhs.nElems];        std::**memcpy**(tmp, rhs.array, rhs.nElems \* sizeof(\*array));      }      delete[] array;      array = tmp;      nElems = rhs.nElems;      return \*this;    }      // ...  }; |

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

|  |
| --- |
| **Principles(s):** Heed Compiler Warnings – Watch for these warnings and pay attention to them when they do come up. Warnings are there for a reason.  Architect and Design for Security Policies – These types of issues can happen due to a lack of design. Think ahead when a system or software directly interacts with the overall security of a program or system. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Very | High | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | Checking for various issues in the code and is more in depth than typical code analysis tools that come in IDEs. | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Uninitialized Memory | [STD-33-C] | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the set\_flag() function is intended to set the parameter, sign\_flag, to the sign of number. However, the programmer neglected to account for the case where number is equal to 0. Because the local variable sign is uninitialized when calling set\_flag() and is never written to by set\_flag(), the comparison operation exhibits [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior) when reading sign. |
| void set\_flag(**int** number, **int** \*sign\_flag) {    if (NULL == sign\_flag) {      return;    }      if (number > 0) {      \*sign\_flag = 1;    } else if (number < 0) {      \*sign\_flag = -1;    }  }    **int** is\_negative(**int** number) {  **int** sign;    set\_flag(number, &sign);    return sign < 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution trivially repairs the problem by accounting for the possibility that number can be equal to 0.  Although compilers and [static analysis](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-staticanalysis) tools often detect uses of uninitialized variables when they have access to the source code, diagnosing the problem is difficult or impossible when either the initialization or the use takes place in object code for which the source code is inaccessible. Unless doing so is prohibitive for performance reasons, an additional defense-in-depth practice worth considering is to initialize local variables immediately after declaration. |
| void set\_flag(**int** number, **int** \*sign\_flag) {    if (NULL == sign\_flag) {      return;    }      /\* Account for number being 0 \*/    if (number >= 0) {      \*sign\_flag = 1;    } else {      \*sign\_flag = -1;    }  }    **int** is\_negative(**int** number) {  **int** sign = 0; /\* Initialize for defense-in-depth \*/    set\_flag(number, &sign);    return sign < 0;  } |

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

|  |
| --- |
| **Principles(s):** Heed Compiler Warnings – Watch for these warnings and pay attention to them when they do come up. Warnings are there for a reason.  Architect and Design for Security Policies – These types of issues can happen due to a lack of design. Think ahead when a system or software directly interacts with the overall security of a program or system. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | Checking for various issues in the code and is more in depth than typical code analysis tools that come in IDEs. | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Derefence Null Pointers | [STD-34-C] | Do not dereference Null Pointers |

| **Noncompliant Code** |
| --- |
| This noncompliant code example is derived from a real-world example taken from a vulnerable version of the libpng library as deployed on a popular ARM-based cell phone [[Jack 2007](https://wiki.sei.cmu.edu/confluence/display/c/AA.+Bibliography#AA.Bibliography-Jack07)]. The  libpng library allows applications to read, create, and manipulate PNG (Portable Network Graphics) raster image files. The libpng library implements its own wrapper to malloc() that returns a null pointer on error or on being passed a 0-byte-length argument.  This code also violates [ERR33-C. Detect and handle standard library errors](https://wiki.sei.cmu.edu/confluence/display/c/ERR33-C.+Detect+and+handle+standard+library+errors). |
| #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 compliant solution ensures that the pointer returned by png\_malloc() is not null. It also uses the unsigned type size\_t to pass the length parameter, ensuring that negative values are not passed to func().  This solution also ensures that the user\_data pointer is not null. Passing a null pointer to memcpy() would produce undefined behavior, even if the number of bytes to copy were 0.  The user\_data pointer could be invalid in other ways, such as pointing to freed memory. However there is no portable way to verify that the pointer is valid, other than checking for 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 \*/    }    chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);    if (NULL == chunkdata) {      /\* Handle error \*/    }    if (NULL == user\_data) {      /\* 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 – Watch for these warnings and pay attention to them when they do come up. Warnings are there for a reason.  Architect and Design for Security Policies – These types of issues can happen due to a lack of design. Think ahead when a system or software directly interacts with the overall security of a program or system. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | High | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | Checking for various issues in the code and is more in depth than typical code analysis tools that come in IDEs. | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Selection Statements | [STD-45-C] | Do not perform assignments in selection statements |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an assignment expression is the outermost expression in an if statement: |
| if (a = b) {    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| When the assignment of b to a is not intended, the conditional block is now executed when a is equal to b: |
| if (a == b) {    /\* ... \*/  } |

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

|  |
| --- |
| **Principles(s):** [Heed Compiler Warnings – Watch for these warnings and pay attention to them when they do come up. Warnings are there for a reason.  Code reviews and code analysis tools would help to find these types of issues in your code! These types of problems are common when a programmer is moving quickly through their code and makes a typo but it doesn’t result in an a error and its harder to find unless you utilize these types of tools or procedures to stop these issues. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Very Likely | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.4.1 | Checking for various issues in the code and is more in depth than typical code analysis tools that come in IDEs. | CPPCheck provides a code analysis to find potential issues with the code before those issues cause further problems. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## 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.

For each stage of the dev ops process illustrated by the image is a set of common and standard procedures that create each layer of protection of the overall project or program that is being developed by Green Pace. During pre-production is a cycle that decreases the likelihood of security breaches being a potential threat and this is due to proper planning. This cycle transforms into a different type when the project moves into production. For the production cycle its less about the design and planning aspect and more about the maintenance aspect. Each stage is designed to assess and respond if issues are detected to maintain a stabilized operation.

## 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 |
| [INT-33-C] | High | Very Likely | High | High | 5 |
| [INT-32-C] | Low | Rare | Minor | Medium | 1 |
| [STR-30-C] | Low | Rare | Low | Minor | 1 |
| [IDS-00-J] | High | Unlikely | High | High | 5 |
| [MEM-30-C] | Medium | Likely | Medium | Medium | 2 |
| [ERR-06-C] | Low | Very Rare | None | Low | 1 |
| [ERR-56-CPP] | High | Very | High | High | 4 |
| [STD-33-C] | High | High | Low | High | 4 |
| [STD-34-C] | High | High | High | High | 3 |
| [STD-45-C] | Low | Very Likely | Medium | Medium | 3 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## 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 | Prevents an attacker from accessing unencrypted data by ensuring all data is encrypted when stored. |
| Encryption at flight | Prevents an attacker from accessing unencrypted data by ensuring all data is encrypted when its being transmitted. |
| Encryption in use | Prevents an attacker from accessing unencrypted data by ensuring all data is encrypted when the data is in active runtime memory. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Each user needs to provide authentication to be a valid user. |
| Authorization | Each user needs to be given authorization to be given access to proper areas of a software or system. |
| Accounting | All user accounts should be logged for what actions are done on the network such as user logins, changes to a database, creation of a new user and what files were accessed by each user. |

**\***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 | 04/08/2021 | Design Update | Colton Thompson |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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