**DGreen 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 | Validating input data from all untrusted data sources can helps in eliminating the most of software vulnerabilities. It is good idea to validate all external source data including command line arguments, network interfaces, environmental variables, and user-controlled files, etc. |
| 1. Heed Compiler Warnings | Compiling code using highest warning level can help to detect all the bad coding practices and prevent it from possible vulnerabilities. Static and dynamic analysis tools can be used to detect and eliminate additional security flaws in our code. |
| 1. Architect and Design for Security Policies | Always create a best possible software architecture and design your software to implement and enforce security policies. For instance, if our system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystem, each with appropriate privilege set. |
| 1. Keep It Simple | Keeping simple and small design possibly helps in reducing error during development phase and increase the effort required to achieve an appropriate level assurance when security mechanisms become more complex. |
| 1. Default Deny | Unless you want something special in your program, deny everything that does not match the standard. For instance, you want the quantity of item in integer data type then deny all the input that does not match the data type. |
| 1. Adhere to the Principle of Least Privilege | Every process in your system should execute with the least set of privileges necessary to complete the job. This will reduce the possibilities for hacker to execute arbitrary code with elevated privileges (Saltzer, 74). |
| 1. Sanitize Data Sent to Other Systems | Always sanitize the data before sending it to complex subsystem, databases, and COTS components. Attacker may use SQL injection or other command to invoke unused functionalities in the component. |
| 1. Practice Defense in Depth | Use multiple defensive layers of security in your system, if one fails another will be operative. For instance, we can use a firewall to prevent access via non-web port, encrypt password in database, check to prevent buffer overflow while writing web-server, password authentication, etc. |
| 1. Use Effective Quality Assurance Techniques | Use high quality assurance techniques to identify and eliminate vulnerability in your system. For instance, we can do fuzz testing, penetration testing, and source code review and audit to identify vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Always develop and follow the secure coding standard for your chosen programming languages and platform. |

## 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-001-cpp] | Always validate user input data to make sure right input is inputted. |

| **Noncompliant Code** |
| --- |
| It is good practice to validate the user input data before processing it. Here the variable user\_input is declare to hold only twenty characters and the code miss to validate the input from user. If user input more than twenty characters which the variable cannot hold, as a result buffer overflow occur and the program crash. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];    std::cout << "Enter a value: ";    std::cin>> user\_input;    std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

| **Compliant Code** |
| --- |
| In this compliant code, user input is validated to take less than twenty character and even below it check if the length of variable is less than twenty of not. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];    std::cout << "Enter a value: ";    std::cin.width(20); // Take less than twenty character from user input  std::cin>> user\_input;    if (strlen(user\_input) <= 20) {  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl;  } |

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

|  |
| --- |
| ValidateInput Data**:** It is important to make sure that you are getting right input from user. So, for that always validate the input data before processing it. Otherwise, it may have adverse effect on your program. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure |  |
| Code Sonar | 6.0p0 | **LANG.MEM.BO** | Buffer overrun |
| Parasoft C/C++ test | 2020.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the range. |
| Polyspace bug finder | R2020a | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Check for array access out of bound. |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [DTV-002-CPP] | Make sure that data value is within the range it can hold. |

| **Noncompliant Code** |
| --- |
| To avoid numeric overflow always make sure that data value is within the range that data type can hold. In this example, if we go on incrementing the result at some point value of result run out of the max range that it can hold. This results in numeric overflow and return wrong answer. |
| template <typename T>  T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  result += increment;    }  return result;  } |

| **Compliant Code** |
| --- |
| The example below check the value of result before incrementing if it could go out of range while incrementing and throws error if result could overflow else return the correct answer. |
| template <typename T>  T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  // Stop the loop before overflow happen  if (increment > std::numeric\_limits<T>::max() - result) {  throw std::overflow\_error("Numeric Overflow"); // Throws an overflow error  }  else {  result += increment;  }    }  return result;  } |

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

|  |
| --- |
| Adopt a Secure Coding Standard**:** It is always better to follow secure coding habit, otherwise our program may show unusual behabior. In the compliant code above, the variable “result” gets overflow at some point and produce unusual result. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure |  |
| Code Sonar | 6.0p0 | **LANG.MEM.BO** | Buffer overrun |
| Parasoft C/C++ test | 2020.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the range. |
| Polyspace bug finder | R2020a | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Check for array access out of bound. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STR-003-CPP] | Make sure that storage for strings has sufficient space for character data. |

| **Noncompliant Code** |
| --- |
| If buffer is not large enough to hold the data copied then it results in buffer overflow. In this example, char array name can only hold 15 character. If user input more than 15 characters then it results in buffer overflow. |
| void f() {  **char** name[15];  std::cout << “Enter your name” << std::endl;     std::cin >> name;    std::cout << “Your name is “ << name << std::endl;  } |

| **Compliant Code** |
| --- |
| To ensure not to truncate your data, it is best to use std::string instead of a bounded array. This example used std::string instead of using char array to hold name. |
| void f() {     std::string name;  std::cout << “Enter your name” << std::endl;     std::cin >> name;    std::cout << “Your name is “ << name << std::endl;  } |

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

|  |
| --- |
| ValidateInput Data**:** In the noncompliant above char array is declared to hold only 15 characters. If user input more than 15 characters then the array could not hold all that characters so buffer overflow occur. In order to prevent it, we must validate the user input. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | LANG.MEM.BO | Buffer overrun |
| LDRA tool suite | 2018 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| Parasoft C/C++ test | 2020.2 | **CERT\_CPP-STR50-e** | Avoid overflow when writing to a buffer |
| Polyspace bug finder | R2020a | CERT C++:STR50-CPP | Buffer overflow from incorrect string format specifier |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [SQL-004-CPP] | Use Technique to detect SQL injection. |

| **Noncompliant Code** |
| --- |
| The noncompliant code below fail to detect the injected code (eg: OR 1=1) in the sql statement. As a result run\_query function() return all the records present in the table. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  // clear any prior results  records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

| **Compliant Code** |
| --- |
| The compliant code below used technique to detect the injected code in sql statement. When it find the position of injected code, it will remove those code and use the cleaned SQL as parameter in sqlite3\_exec() function. This prevent program from being hacked using SQL injection |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  // clear any prior results  records.clear();  char\* error\_message;  std::string injectedSQL(sql);  std::string cleanSQL(sql);  int pos = find\_match\_position(sql); // find the position of injected code  if (pos > 0) {  // Erase the injected code  cleanSQL = injectedSQL.erase(pos, cleanSQL.length() - pos);  }  if(sqlite3\_exec(db, cleanSQL.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

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

|  |
| --- |
| Sanitize Data Sent to Other Systems: When sending data to complex subsystem or database, it is important to sanitize the data. Otherwise, attacker may use SQL injection or other command to invoke unused functionalities in the component. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 6.7 | S2077 | Executing SQL queries is security sensitive. |
|  |  | S3649 | SQL queries should not be vulnerable to injection attacks. |

### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| In this example int array is created using new[](size) operator without checking the result of allocation. This function can throw an exception in case of allocation fails resulting abnormal termination of the program. |
| 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** |
| --- |
| Example below used std::nothrow, doing this returns either a null pointer or a allocate the space to the pointer. It is always important to check if the pointer is null before referencing the pointer. The example below handles the error appropriately. |
| void f(const **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = new (std::nothrow) **int**[size];    if (!copy) {      // Handle error      return;    }    std::**memcpy**(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

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

|  |
| --- |
| Adopt a Secure Coding Standard: Memory management is one of the important things to consider while programming in C/C++. Improper allocation of memory can crash the program at some point leaving the vulnerability. So, it is critical to follow secure coding standard for proper allocation of memory. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2020.2 | MEM52-a | Do not allocate resources in function argument list |
| Polyspace Bug finder | R2020a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [ASR-006-CPP] | Make a right use of assert. |

| **Noncompliant Code** |
| --- |
| It is better to use assert if the condition is true, if the condition is false then the program terminated abnormally. The noncompliant code below use assert to test if the variable myInt is null or not. For the second call of display\_number(third\_ptr) the condition is false. As a result program terminate abnormally. |
| #include <iostream>  #include <cassert>  using namespace std;  void display\_number(int\* myInt) {     assert(myInt!=NULL);     cout<<"myInt contains value" << " = "<<\*myInt<<endl;  }  int main ()  {     int myptr=5;     int \* second\_ptr = NULL;     int \* third\_ptr = NULL;     second\_ptr=&myptr;     display\_number (second\_ptr);     display\_number (third\_ptr);     return 0;  } |

| **Compliant Code** |
| --- |
| The compliant code below used if statement instead of assert. This prevent program terminating abnormally even the condition is false. |
| #include <iostream>  #include <cassert>  using namespace std;  void display\_number(int\* myInt) {     if(myInt!=NULL){     cout<<"myInt contains value" << " = "<<\*myInt<<endl;  }  }  int main ()  {     int myptr=5;     int \* second\_ptr = NULL;     int \* third\_ptr = NULL;     second\_ptr=&myptr;     display\_number (second\_ptr);     display\_number (third\_ptr);     return 0;  } |

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

|  |
| --- |
| **Default Deny:** In the noncompliant code above use of assert does not match the coding standard. So, it is better to deny the use of non-standard code. Assert can be use for true statement, but in this cast the statement can be false. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| N/A | N/A | N/A | No tools found. |
| [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-007-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In the example code below, both the function f() and main() does not catch the exceptions thrown by the throwing\_func(). Because of this not, control does not transfer to the nearest handler to search for exception thrown. As a result std::terminate() will be called. |
| void throwing\_func() noexcept(false);    void f() {     throwing\_func();  }    **int** main() {     f();  } |

| **Compliant Code** |
| --- |
| The example below used the try catch block inside the main() function to catch the exception thrown. Here the main entry point handles all the exceptions allowing the fraceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {     throwing\_func();  }    **int** main() {     try {       f();     } catch (...) {       // Handle error     }  } |

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

|  |
| --- |
| Use Effective Quality Assurance Techniques: Always use try catch block to catch all the unhandled exceptions. It is the effective way of catching all the unhandled exceptions. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | **main-function-catch-all early-catch-all** | Partially checked |
| Parasoft C/C++ test | 2020.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| Polyspace Bug Finder | R2020a | CERT C++: ERR51-CPP | Checks for unhandled exceptions |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| File Input Output | [FIO-008-CPP] | Close files when they are no longer needed. |

| **Noncompliant Code** |
| --- |
| In the example code below, std::fstream object file is constructed. The constructor for std::fsteam calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. As a result, the underlying std::basic \_filebuf<T> object maintained by the object is not properly closed. |
| void f(const std::string &fileName) {     std::fstream file(fileName);     if (!file.is\_open()) {       // Handle error       return;     }     // ...     std::terminate();  } |

| **Compliant Code** |
| --- |
| In the example below, std::fstream::close() is called before std::terminate() is called. This ensure that the file resources are properly closed. |
| void f(const std::string &fileName) {     std::fstream file(fileName);     if (!file.is\_open()) {       // Handle error       return;     }       file.close();     if (file.fail()) {       // Handle error     }     std::terminate();  } |

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

|  |
| --- |
| Adopt a Secure Coding Standard: As per the secure Coding Standard, it is always important to close file once you are done with file operation. Otherwise, file can we modified by hacker at any point. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2020.2 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| Polyspace Bug Finder | R2020a | CERT C++: FIO51-CPP | Checks for resource leak |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integer Division | [INT-009-CPP] | Ensure that division and remainder operations do not result in divide by zero errors. |

| **Noncompliant Code** |
| --- |
| The example code below ensures that operations on signed integer do not result in overflow, but it fails to prevent a divisible by zero error during division of the signed operands val1 and val2. |
| void divide (**signed** **long** val1, **signed** **long** val1) {  **signed** **long** result;     if ((val1 == LONG\_MIN) && (val2 == -1)) {       /\* Handle error \*/     } else {       result = val1 / val2;     }     /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The example code below handled both divisible by zero error and signed overflow. |
| void divide (**signed** **long** val1, **signed** **long** val2) {  **signed** **long** result;     if ((val2 == 0) || ((val1 == LONG\_MIN) && (val2 == -1))) {       /\* Handle error \*/     } else {       result = val1 / val2;     }     /\* ... \*/  } |

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

|  |
| --- |
| Adopt a Secure Coding Standard**:** It is important to check the possible error that could occur while doing mathematical calculation in programming. So, always follow secure coding standard to check the divisible by zero error. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| N/A | N/A | N/A | Divisible by zero error can be catch using if statement. |
| [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** |
| --- | --- | --- |
| Arrays | [ARR-010-CPP] | Do not form or use out-of-bounds pointers or array subscripts. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code below, the function f() attempt to validate the index before using it as an offset to the statically allocated table of integers, but it fails to reject the negative index values. This negative index value might produce result that does not point into or just beyond the same array object. |
| enum { TABLESIZE = 100 };    static **int** table[TABLESIZE];    **int** \*f(**int** index) {     if (index < TABLESIZE) {       return table + index;     }     return NULL;  } |

| **Compliant Code** |
| --- |
| The compliant code below used an unsigned type to avoid having to check for negative index value. It also reject out-of-bound positive values of index. |
| enum { TABLESIZE = 100 };    static **int** table[TABLESIZE];    **int** \*f(**size\_t** index) {     if (index < TABLESIZE) {       return table + index;     }     return NULL;  } |

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

|  |
| --- |
| Default Deny: Unless you want something special in your program, deny everything that does not match the standard. In the noncompliant code above, we do not need the int data type for index. So, it is better to avoid using non-standard coding habit. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | **CERT\_C-ARR30-a** | Partially checked  Can detect all accesses to invalid pointers as well as array index out-of-bounds accesses and prove their absence. |
| Parasoft C/C++ testing | 2020.2 | **CERT\_C-ARR30-a** | Avoid accessing arrays out of bounds |
| Polyspace Bug Finder | R2020a | CERT C: RULE ARR30-CPP | Array access out of bounds  Pointer access out of bounds |

## 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.

DevSecOps in short means development, security, and operations. Mainly it is design to make everyone accountable for security with the objectives of implementing security decisions and actions at the same scale and speed as development and operation. The life cycle of DevSecOps is mainly categorized into two phases, pre-production and production. Pre-production phase focuses in developing secure system or software. It is again divided into four phases Assess and plan, design, build, and verify and test.

Plan: This is the phase where team plans on managing time, cost, quality, risk and issues. These activities, include business-need assessment, project plan creation, feasibility analysis, risk analysis, business requirements gathering, business process creation, and ecosystem instantiation, etc.

Design: In this phase team focuses in designing secure system or software architecture using test-driven design and best practices like OWASP. It used tools to support the development and convert the requirement into source code.

Build: In this phase, build tools perform the task of building and packaging applications services, and microservices into artifacts. This includes secure build, trusted repositories, and secure open-source usages.

Test: In this phase, system or software built is tested to detect vulnerabilities, errors, and bugs using different technique available. This may include activities such as unit test, functional test, integration test, system test, regression test, acceptance test, performance test, and variety of security test.

Once the pre-production phase is completed, it moves to the production phase which involves transition and health check, monitor and detect, respond, and maintain and stabilize.

## 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 |
| --- | --- | --- | --- | --- | --- |
| INT-001-cpp | High | likely | Medium | P18 | L1 |
| DTV-002-CPP | High | Likely | Medium | P18 | L1 |
| STR-003-CPP | High | Likely | Medium | P18 | L1 |
| SQL-004-CPP | High | Likely | Medium | P12 | L1 |
| MEM-005-CPP | High | Likely | Medium | P18 | L1 |
| ASR-006-CPP | High | Likely | medium | P18 | L1 |
| ERR-007-CPP | Low | Likely | Medium | P4 | L3 |
| FIO-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| INT-009-CPP | low | Likely | Medium | P4 | L3 |
| ARR-010-CPP | High | Likely | High | P9 | L2 |

## Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption of data when it is in rest position is known as encryption at rest. For instance, when the data is stored on disk or database. It is designed to prevent the attacker from accessing the unencrypted data. Even though attacker obtain encrypted data, they still need encryption key to decrypt the data. So, attacker cannot read the data easily even after they stole it from our system. |
| Encryption at flight | The process of encrypting data while it is being transmitted is known as encryption at flight. It is important to encrypt data at flight because attacker could potentially sniff the network traffic and gain access to the data as it traverses the network. It uses complex algorithms to scramble the data being sent. When it is received, the data can be decrypted using a key provided by the originator of the message. |
| Encryption in use | Encryption in use is a capability that lets us run our computation on encrypted data or encrypted application. It also means protecting data in memory. Intel’s SGX or ARM TrustZone is a good example of encryption in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of identifying a user, typically by having login credential such as username and password before access is granted. Normally, authentication is done by identifying the login credential. In more secure system, user is even asked to answer some security question or to verify the security code sent over email or phone. Applying authentication policy helps in identifying the right user and granting access to the system securely. |
| Authorization | Authorization is the process of identifying the role of the user in the system. When a user is logged into a system, he/she tries to perform some activities. The authorization process determines whether the user is permitted to do that activity or not. It prevents user to do unauthorized activity inside the system. |
| Accounting | Accounting is the process of measuring the resources a user consumes during access. It tracks the amount of data user sent and/or received or the activity performed during a session. It is carried out by logging of session statistics and usage information and is used for authorization control, billing, trend analysis, resource utilization, and capacity planning activities. |

**\***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 | 03/18/2021 | Modified Template | Tek Bista |  |
| 1.1 | 03/18/2021 | Final modified template | Tek Bista |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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

# References

[Defense, D. o. (2019, 09 12).](Defense, D. o. (2019, 09 12). DoD Enterprise DevSecOps Reference Design. Retrieved from Department of Defence: https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdfSEI CERT C++ Coding Standard. (2021, 03 21). Retrieved from Confluence: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682) *[DoD Enterprise DevSecOps Reference Design.](Defense, D. o. (2019, 09 12). DoD Enterprise DevSecOps Reference Design. Retrieved from Department of Defence: https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdfSEI CERT C++ Coding Standard. (2021, 03 21). Retrieved from Confluence: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682)* [Retrieved from Department of Defence: https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0\_Public%20Release.pdf](Defense, D. o. (2019, 09 12). DoD Enterprise DevSecOps Reference Design. Retrieved from Department of Defence: https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdfSEI CERT C++ Coding Standard. (2021, 03 21). Retrieved from Confluence: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682)

*[SEI CERT C++ Coding Standard.](Defense, D. o. (2019, 09 12). DoD Enterprise DevSecOps Reference Design. Retrieved from Department of Defence: https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdfSEI CERT C++ Coding Standard. (2021, 03 21). Retrieved from Confluence: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682)* [(2021, 03 21). Retrieved from Confluence: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682](Defense, D. o. (2019, 09 12). DoD Enterprise DevSecOps Reference Design. Retrieved from Department of Defence: https://dodcio.defense.gov/Portals/0/Documents/DoD%20Enterprise%20DevSecOps%20Reference%20Design%20v1.0_Public%20Release.pdfSEI CERT C++ Coding Standard. (2021, 03 21). Retrieved from Confluence: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682)