**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 | Validating input data is important because hackers can break your system by inputting the right or wrong input. So, you need to be able to validate inputs before they even touch any important systems. |
| 1. Heed Compiler Warnings | Heed Compiler Warnings, allow your code to build and take errors, without crashing your system. It simply throws an error, letting you know something went wrong. |
| 1. Architect and Design for Security Policies | Designing a system in such a way to protect any private or secure data is not accessible unless you have the correct credentials. |
| 1. Keep It Simple | Security is good, however security at the detriment of the customers satisfaction. To much security can bog down a system and make it agonizingly slow. |
| 1. Default Deny | Basically, if data coming through your system wasn’t requested by it or is listed to be allowed into the system, it doesn’t get through no matter what. |
| 1. Adhere to the Principle of Least Privilege | Clients should never have access to data that isn’t required. By giving clients and other viewers access to more data than they should have, you open yourself up to exploitation. By limiting what they have access to, you can better manage what hackers can do to your system. |
| 1. Sanitize Data Sent to Other Systems | Makes sure that any data being sent from a system meet the guidelines to security to minimize risk of private data from being leaked. |
| 1. Practice Defense in Depth | Making Multiple layers of protecting on certain input to prevent multiple types of attacks. And having multiple different levels of the same or different checks at different points along the journey to ensure security. |
| 1. Use Effective Quality Assurance Techniques | Using other programs to check for additional vulnerabilities. Not just from your code. But also, vulnerabilities from the programing language you chose. As well as any additional system that help your system run. |
| 1. Adopt a Secure Coding Standard | **Class Declaration**: |

### 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** | **Validating Inputs** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | When validating input, it is important to make sure you are getting the correct data type. So, if you ask for a number, your user should give you a number and not a string. |

| **Noncompliant Code** |
| --- |
| Here we have code that simply takes an input and then inserts it into an int variable. If a user inserts a word string or character the code will error out and break. So, we have to make sure the user inserts a number. |
| int main()  {  int userinput;  std::cout << "Enter a number: ";  std::cin >> userinput;  std::cout << "You entered: " << userinput << std::endl;  } |

| **Compliant Code** |
| --- |
| In this example we put the user input into a while loop that will only end if they enter a number. If they enter anything else, they will be given an error and be asked to try again. |
| int main()  {  int userinput;  std::cout << "Enter a number: ";  while (1) {  try{  std::cin >> userinput;  break;  }  catch () {  // handle error  }  }  std::cout << "You entered: " << userinput << std::endl;  } |

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

| **Principles(s):**  **Validate Input Data:** This code validates the input that comes in because the only thing that should be entered is a number.  **Default Deny:** The code will not progress further until a number is inputted. The while loop makes sure nothing outside it will happen unless these conditions are met. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.13.0 | CPP | A tool that checks CPP code for potential error or problems |
| Coverity | 2023.9.0 | CPP, C, JAV | A tool designed to scan multiple coding languages for potential problems |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Prevention of Overflow** |
| --- | --- | --- |
| **Data Value** | SRD-002-CPP | When handling data there exists the possibility that a system can handle too much data. So it’s important, to prevent such error from occur before they happen |

| **Noncompliant Code** |
| --- |
| In this example, this code will increment “result until it reaches the number of steps listed. However there is the chance that result will get too big and will overflow and cause errors with in the code. |
| 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** |
| --- |
| Here we can see that we set up a “max” and before we increment the result, we check to see if the resulting increment would exceed our max, if so we return an error code. |
| template <typename T>  T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  T max = std::numeric\_limits<T>::max();  for (unsigned long int i = 0; i < steps; ++i)  {  if (increment > 0 && result > max - increment) {  // throw std::overflow\_error("An Overflow Occured")  return -99999999999;  }  else {  result += increment;  }  }  return result;  } |

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

| **Principles(s):**  **Practice Defense in Depth:** This method can be used to prevent files that are too big from being uploaded, their by preventing buffer overflow errors.  **Use Effective Quality Assurance Techniques:** This problem throws a unique code telling the system that a buffer overflow error has occurred, which then through a message detailing the problem. This allows people using the code to identify what is going on and right the mistake. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.13.0 | CPP | A tool that checks CPP code for potential error or problems |
| Coverity | 2023.9.0 | CPP, C, JAV | A tool designed to scan multiple coding languages for potential problems |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Properly converting Strings to numbers** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | When handling data you might need to store a number as a string and then convert it back to a int late. So, its important to hand when conversion fails. |

| **Noncompliant Code** |
| --- |
| This code doesn’t have proper check to confirm that the conversion will work properly. So has a high probability that it will cause problems. |
| void func(const char \*buff) {    int matches;    int si;      if (buff) {      matches = sscanf(buff, "%d", &si);      if (matches != 1) {        /\* Handle error \*/      }    } else {      /\* Handle error \*/    }  } |

| **Compliant Code** |
| --- |
| Here we have the conversion go through multiple check like: is this really a number, does it have extra characters, is it too long/ too short etc. |
| void func(const char \*buff) {    char \*end;    int si;      errno = 0;      const long sl = strtol(buff, &end, 10);      if (end == buff) {      (void) fprintf(stderr, "%s: not a decimal number\n", buff);    } else if ('\0' != \*end) {      (void) fprintf(stderr, "%s: extra characters at end of input: %s\n", buff, end);    } else if ((LONG\_MIN == sl || LONG\_MAX == sl) && ERANGE == errno) {      (void) fprintf(stderr, "%s out of range of type long\n", buff);    } else if (sl > INT\_MAX) {      (void) fprintf(stderr, "%ld greater than INT\_MAX\n", sl);    } else if (sl < INT\_MIN) {      (void) fprintf(stderr, "%ld less than INT\_MIN\n", sl);    } else {      si = (int)sl;        /\* Process si \*/    }  } |

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

| **Principles(s):**  **Default Deny:** This code prevents anything other than the correct variable to be handled by the code.  **Use Effective Quality Assurance Techniques/Practice Defense in Depth:** By having multiple if/else if statements we can do a series of quick checks to cover for multiple types of potential problems. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Low | Medium | 6 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.13.0 | CPP | A tool that checks CPP code for potential error or problems |
| Coverity | 2023.9.0 | CPP, C, JAV | A tool designed to scan multiple coding languages for potential problems |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Precheck inputs for SQL Injections** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Most input options in programming is susceptible to SQL Injections. So it so its important to validate input to prevent the accidental handling of SQL injections |

| **Noncompliant Code** |
| --- |
| This code simply takes an input and only checks it for SQL error and not Injection attacks. And if it doesn’t find an error, send back an all-clear sign. |
| 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** |
| --- |
| Meanwhile this code properly validates the string by checking it for a 1=1 attack, or similar attacks. By checking the string directly in a broad scope we are able to prevent SQL attacks from occurring. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  // checks if string contains "or" and ";"  if (sql.find("or ") != std::string::npos && sql.find(";", sql.find("or ")) !=  std::string::npos){  std::cout << "An SQL injection has been detected!" << sql << std::endl;  return false;  }  // checks if stirng contains an "="  if (sql.find("=") != std::string::npos) {  std::cout << "An SQL injection has been detected!: " << sql << std::endl;  return false;  }  // 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;  } |

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

| **Principles(s):**  **Validate Input Data:** Even if data is successfully inputted, it doesn’t mean that what’s in the is safe.  **Practice Defense in Depth:** Check to see if strings that will be used for commands/database inputs have SQL injections are crucial. A simple string can do a lot of damage.  **Sanitize Data Sent to Other Systems:** Any call that retrieve data from a SQL database should be checked for SQL injections |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 10 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SQLMap | 1.8.4 | JAV | A tool to help potential area vulnerable to SQL Injection attacks |
| jsql injection | 11 | JAV | A tool to help scan for database information leaks. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Protection from Buffer Overflow** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | You need to prevent user from intentionally or accidentally overflowing memory off arrays. Failure to do this can cause major problems within your system. |

| **Noncompliant Code** |
| --- |
| Here we see code that simply allows the user to input anything they want into a 20-space character array. So is someone, intentionally or otherwise, inputs more than 20 characters the code will crash. |
| int main()  {  std::cout << "Buffer Overflow Example" << std::endl;  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** |
| --- |
| Here we see the code is first validate to be shorter than 20 characters by putting it first in a string. Then when it is confirmed to be shorter than 20, converts it into the 20-space character array. |
| int main()  {  std::cout << "Buffer Overflow Example" << std::endl;  const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::string temp;  std::cout << "Enter a value: ";  // temp variable to help with the prevention of overflow  std::cin >> temp;  if (temp.length() > 20) {  // if temp is to long, throw an error  std::cout << "Buffer Overflow detected. Please only enter a max of 20 characters." << std::endl;  }  else {  // else, convert string to character array and continue  strcpy\_s(user\_input, temp.c\_str());  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.**

| **Principles(s):**  **Validate Input Data:** Prevention of inserting to many characters into a limited character array  **Heed Compiler Warnings:** Problems like this can be overlooked, and only be seen through things like manually running the code or Unit testing can expose errors like these. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.13.0 | CPP | A tool that checks CPP code for potential error or problems |
| Coverity | 2023.9.0 | CPP, C, JAV | A tool designed to scan multiple coding languages for potential problems |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Proper use of number validation** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | When using validation of number and you are looking for a specific number. Its better to use assert to more quickly analyze numbers instead of if else. |

| **Noncompliant Code** |
| --- |
| Here we have a lot of code says the number must be even, however it can be done easier. |
| int main() {  int even\_num = 3;  if (even\_num % 2 == 0) {  // cout even number  }  else {  // cout out must be an even number  }  return 0;  } |

| **Compliant Code** |
| --- |
| Here we use a simple asset statement to check if the number is even. If not, it throws an error. |
| int main() {  int even\_num = 3;  // asserts the value of even\_num must be even  assert((even\_num % 2 == 0));  return 0;  } |

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

| **Principles(s):**  **Keep It Simple:** You should always look for simpler ways to code things. It makes reanalyzing the code easier when coming back later or when someone new is learning it. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.13.0 | CPP | A tool that checks CPP code for potential error or problems |
| Coverity | 2023.9.0 | CPP, C, JAV | A tool designed to scan multiple coding languages for potential problems |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Exceptions in Conversion** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | When handling converting data. You need to be able to catch when a conversion will fail. If not, your code will simple crash. |

| **Noncompliant Code** |
| --- |
| This code has not case for if the conversion fail. So if the sring has anything other than a number, the code will crash. |
| int main()  {    string str1 = "45 inches";      int myint1 = stoi(str1);      cout << "stoi(\"" << str1 <<            "\") is " << myint1 << '\n';      return 0;  } |

| **Compliant Code** |
| --- |
| Here we contain the conversion in a try catch method so if any errors happen, the code will immediately stop the conversion and go to the catch. |
| int main()  {    string str1 = "45 inches";  try {  int myint1 = stoi(str1);  cout << "stoi(\"" << str1 <<            "\") is " << myint1 << '\n';      return 0;  }  catch () {  // exception message  }        cout << "stoi(\"" << str1 <<            "\") is " << myint1 << '\n';      return 0;  } |

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

| **Principles(s):**  **Validate Input Data**: Prevent code from crashing all together is paramount. Throwing a error message is better than having something close or lock up.  **Heed Compiler Warnings:** Exception throwing is a good way to help easily diagnose what problem is happening. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Very Likely | High | Medium | 7 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.13.0 | CPP | A tool that checks CPP code for potential error or problems |
| Coverity | 2023.9.0 | CPP, C, JAV | A tool designed to scan multiple coding languages for potential problems |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Storing data** |
| --- | --- | --- |
| Data protection | STD-008-CPP | When using important data or access private data it is important to make sure that certain data cannot be accessed by the user in any way this can be achieve by storing data in a private function. |

| **Noncompliant Code** |
| --- |
| Here we have a simple password verification system. However this data could easily be access by some simply looking at the source code. |
| int main()  {  std::string = password = 1234;  std::string = user\_input;    std::cout << "Enter password: ";  std::cin >> user\_input;  if (password == user\_input){  std::cout << "Your account id is 7890" << std::endl;  }  else {  std::cout << "Incorrect password" << std::endl;  }  } |

| **Compliant Code** |
| --- |
| Here we call a private function only accessible via the code. |
| class Verify {  private:      int password = 1234;      void check(int input)      {        if (password == input){  std::cout << "Your account id is 7890" << std::endl;  }  else {  std::cout << "Incorrect password" << std::endl;  }   }    };  int main()  {  std::string = password = 1234;  std::string = user\_input;    std::cout << "Enter password: ";  std::cin >> user\_input;  check(user\_input);  } |

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

| **Principles(s):**  **Adhere to the Principle of Least Privilege**: preventing the main from accessing private data is very important when handling private data. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | High | 8 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.13.0 | CPP | A tool that checks CPP code for potential error or problems |
| Coverity | 2023.9.0 | CPP, C, JAV | A tool designed to scan multiple coding languages for potential problems |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Using Uppercase when writing Classes** |
| --- | --- | --- |
| Class Declaration | STD-009-CPP | When working with multiple variables it is important to be able to quickly identify what is a local variable and what is function. Making all classes have uppercase will allow you to understand what is and is not a function quickly |

| **Noncompliant Code** |
| --- |
| If the name class is ever referenced in code, it will be confusing if there are other variable if they look the same. |
| class name{  } |

| **Compliant Code** |
| --- |
| It’s better to make it upper case to make it ide |
| class Name{  } |

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

| **Principles(s):**  **Keep It Simple:** Make sure your naming convention for classes, variables, etc., is easy to understand and apply |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio | 17.9.2 | CPP, C, JAVA | A good coding software can help make sure that you are being consistent with naming schemes. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Encryption | STD-010-CPP | When handling files that contain sensitive or private information you can’t simply just save them on a hard drive and call it done. If the file is access by a third party that could cause a major data leak |

| **Noncompliant Code** |
| --- |
| In this code we can see the private file is simply read and then saved with no protection whatsoever. So, someone only need to find the name of the file and they have access to all that private data. |
| int main()  {  const std::string file\_name = "inputdatafile.txt";  const std::string source\_string = read\_file(file\_name);  // get the student name from the data file  const std::string student\_name = get\_student\_name(source\_string);  // save string to file  save\_data\_file(file\_name, student\_name, key, source\_string);  } |

| **Compliant Code** |
| --- |
| Here we see the code first encrypt the file contents before saving the document. Even a little bit of encryption can prevent disaster. |
| std::string encrypt\_decrypt(const std::string& source, const std::string& key)  {  // get lengths now instead of calling the function every time.  // this would have most likely been inlined by the compiler, but design for perfomance.  const auto key\_length = key.length();  const auto source\_length = source.length();  // assert that our input data is good  assert(key\_length > 0);  assert(source\_length > 0);  std::string output = source;  // loop through the source string char by char  for (size\_t i = 0; i < source\_length; ++i) {  // transform each character based on an xor of the key modded constrained to key length using a mod  output[i] = source[i] ^ key[i % key\_length];  }  // our output length must equal our source length  assert(output.length() == source\_length);  // return the transformed string  return output;  }  int main()  {  const std::string file\_name = "inputdatafile.txt";  const std::string encrypted\_file\_name = "encrypteddatafile.txt";  const std::string decrypted\_file\_name = "decrytpteddatafile.txt";  const std::string source\_string = read\_file(file\_name);  const std::string key = "password";  // get the student name from the data file  const std::string student\_name = get\_student\_name(source\_string);  // encrypt sourceString with key  const std::string encrypted\_string = encrypt\_decrypt(source\_string, key);  // save encrypted\_string to file  save\_data\_file(encrypted\_file\_name, student\_name, key, encrypted\_string);  std::cout << "Read File: " << file\_name << " - Encrypted To: " << encrypted\_file\_name << std::endl;  } |

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

| **Principles(s):**  **Sanitize Data Sent to Other Systems**: Encrypting file before storing minimizes the risk of data leaks or accidental access to private information.  **Adopt a Secure Coding Standards:** Encryption should be assumed when handling private/ sensitive data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | high | 9 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Botan3 | 3.4.0 | CPP | A cryptography tool to help better encrypt data for modern C++ programs. |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

The DevOps process is a solid solution to help build a well-maintained product that will allow us to upgrade as time moves forward and as new technologies emerge. However, I would like to add a few new areas to the existing order of production and pre-production to better help integrate the new coding standards introduce in this document.

Firstly, I would like to add a new section to the pre-production side. That being “Check-up and Final Inspection”. In the “verify and test” section we check to see if we are compliant with functionality and standards we set ourselves to. This leaves the possibility that we do not meet those requirements and potentially must redo broken or no compliant code, which can take time. Having a dedicated slot for this will allow us to make time to adjust the code to those standards and give it a final look over to make sure we built it right.

Finally, I would like to add in a section for “Research” to the section after “respond” but before “Maintain and stabilize”. In “Research” we will look into common problems and either not them for the next pre-production cycle, or look into enhancements for our software that would benefit the customer that we would like to add.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | Medium | Likely | Medium | Medium | 4 |
| STD-003-CPP | Medium | Likely | Low | Medium | 6 |
| STD-004-CPP | High | Likely | High | High | 10 |
| STD-005-CPP | High | Unlikely | Low | High | 5 |
| STD-006-CPP | Low | Likely | Medium | Low | 3 |
| STD-007-CPP | Medium | Very Likely | High | Medium | 7 |
| STD-008-CPP | High | Medium | Medium | High | 8 |
| STD-009-CPP | Low | High | Low-High | Low | 1 |
| STD-010-CPP | High | Unlikely | High | high | 9 |

### 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 when sensitive data is kept in a single location like a database. Data will be stored in an encrypted format that can only be decrypted by a secure algorithm file. Access to this file will be restricted to IT and support staff only. This is put in place to make sure if our database is leak or accessed by an unauthorized party, our data will still be secure because it will appear as a garbled mess. |
| Encryption in flight | Encryption in flight is when sensitive data is secured as it is being moved from one location to another via the internet or other such mediums. To secure this data we will make sure secure data will only be transferred via a secure and reliable VPN, that will protect our private data from being accessed by anyone other than the intended party. This is to make sure any data transit watcher never finds anything on our network. |
| Encryption in use | Encryption in use is how we will decrypt the encrypted at-rest data and in flight. This process actively encrypts and decrypts data for both transit and rest. As mentioned before only IT and support staff will have access to this algorithm |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is when users are assigned a set of credentials that detail what data they have access to. What these do is identify who the person is and what their role. This will be assigned to all personnel when they create a company account. |
| Authorization | Authorization details what systems/data a person’s credentials will have access to. These allows us to control who has access to what data based on there role, making sure the person only has access to what they need to do their job. |
| Accounting | Accounting is when IT and support staff uphold the policies in this document to the letter. Their job will be to make sure all members follow and maintain their section of this document’s coding standards and policies. Making sure that no one has access to anything they shouldn’t, and all standards are met and upheld. |

**\***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 |  |
| 2.0 | 03/23/2024 | Coding standards and Principles | Matthew Johnson | Joseph Conlan |
| 3.0 | 04/13/2024 | Risk Assessment, Automation, etc. | Matthew Johnson | Joseph Conlan |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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