**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 | This is one of the most important principles to follow when designing a program. It is important to acknowledge that not every user may use the program as it was designed either intentionally or unintentionally, so it is important to safe guard your programs against malicious or negligent user activity. At every possible user input, it is important to ensure that the user input is valid and safe for your program to use. This will help prevent buffer overflows and sql injection attacks. Implement try/catch blocks, if statements, and set limits regarding input size and data type to help validate user input before it can be stored and incorporated into the program. |
| 1. Heed Compiler Warnings | Do not disregard compiler warnings when building a program. The compiler is set up to warn the programmer about code fallacies and vulnerabilities for a reason. It is foolish to disregard them and can lead to bugs and security vulnerabilities within your program down the road. It is also important to note that just because the compiler is not warning you about a possible issue does not mean your code is safe. Please ensure to thoroughly debug your program regarding its syntax and its logic. |
| 1. Architect and Design for Security Policies | It is important to ensure that your program meets not only our company’s aesthetic design regarding our program’s GUI, but also follows our code design requirements. Code must be thoroughly commented, and variables, classes, methods, and modules must have names that are descriptive to their purpose. Please follow the wordWord or word\_Word syntax when naming methods and variables and avoid one letter variables like n or i. It is important to make sure that someone, who is unfamiliar with your code, can read it and understand what is happening within it. Also, for security purposes, be sure to use the proper accesbility modifiers when naming classes and methods. Use only private and protected for most of your code and try to only make something public if it is a method or class that is directly used by the user. It is important to use setter and getter methods so class data can not be directly changed within the object, ensuring that any object changes are validated before they are implemented. |
| 1. Keep It Simple | This is probably the most important policy but the hardest to follow. “Simple is better than Complex” is a programming best practice and should always be on your mind while programming. Logic and syntax bugs can get easily buried in overly-complex code which will not only make your code a nightmare to debug but will make it harder to ensure the security of your code before it is released. This will waste company resources and will not be tolerated. So, after writing each method or every class, go through your code and try to find ways to simplify it. This will not only make your code more efficient at run-time but it will also make your code more secure as well. |
| 1. Default Deny | Implementing default deny permissions within your program, is a security practice that must be adhered to. By denying everything outside of a program’s set use will help keep your program more secure. This can apply to user access or to other programs within the same computer. Limiting a program’s accessibility will also help keep your program’s security maintainable. This way you do not have to try to figure what or who cannot access your program but only who or what can. Only give permission when it is explicitly needed for your program to run properly. |
| 1. Adhere to the Principle of Least Privilege | Adhering to the principle of least privilege ensures that no-user or outside system has access to more than what is necessary for that user or system to perform the tasks that it needs to. It is easy to give a permission if necessary, it is much harder to rein in privileges or to perform damage control resulting from a malicious user given too much access. Only give permissions to a user or other system that is explicitly needed for them to perform their given role. Always limit permissions and privileges as much as possible: always airing on the side of less and not more. |
| 1. Sanitize Data Sent to Other Systems | Data breaches do not only destroy a company’s reputation but can also be financially devastating for a company. By sanitizing data before it is sent to other systems, you ensure that sensitive data within your program will not fall into the wrong hands. Always sanitize user passwords, birthdates, credit card numbers, emails, basically any type of personal user data must be sanitized before it is sent to or shared with another system, even if that system is within our company. This also applies to sensitive company data as well as employee data. |
| 1. Practice Defense in Depth | While designing your program it is important to ensure that the defense in depth methodology is implemented. Please ensure that you apply multiple security layers to your program. Use security measures like validating user input along with implementing the principle of least privilege as well as adhering to our code design and architecture security practices. It is better to have more security than less, so make sure that your program’s potential vulnerabilities are secured under multi layered security. If you apply all our company’s security and design principles that are listed within this document, your program will be protected in multiple ways, thus practicing security in depth. |
| 1. Use Effective Quality Assurance Techniques | Quality Assurance plays an important role in our company and every program written needs to be thoroughly debugged and tested before it can be released or even considered for release. It is important to implement unit tests and test coverage analysis to your code to ensure that your code is properly vetted. We strive for a 98 to 100% test coverage at our company and anything less than 98% coverage will not be accepted. Please also use “assert throws” when testing exceptions and errors within your program. Write clean and well commented code to ensure your code’s quality and testability. It is also important to implement a form of source code analysis as well as running automated unit tests. |
| 1. Adopt a Secure Coding Standard | As a programmer, you should have already adopted a set of secure coding standards, but if you have not it is important that you do. Having a set of rules and guidelines to help you develop secure and efficient code will be vital to your success at this company as well as a programmer in general. This document will help guide you on our secure coding standards and should be referenced often to ensure that your code meets our guidelines. For more information about why it is important to adopt a secure coding standard as well as some programming best practices go to <https://www.whitehatsec.com/glossary/content/secure-coding-standards> and <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices> |

## 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** | DCL53-CPP | Do not write syntactically ambiguous declarations. |

| **Noncompliant Code** |
| --- |
| In line 8 of the code below, the compiler does not know whether this is a function declaration of a function called date that returns an object of type MyDates, or if an object of the class MyDates is being created using an anonymous instance of the Date class. This will cause the compiler to have to pick an option which will lead to potential bugs upon execution. |
| using namespace std;  Date{}    class MyDates{    Date newDate;  MyDate(Date d){    newDate = d;  }  }  int main(){  MyDates date(Date());  cout << date.getDate()<< endl;    } |

| **Compliant Code** |
| --- |
| Within the compliant version of this code, the intent of the implementation of the MyDates class is clear. It is clear to the compiler in this case that a new object of type MyDates is being created using an instance of the Date class. |
| Using namespace std;  Date{}  class MyDates{  Date newDate;  MyDate(Date d){    newDate = d;  }  }  int main(){  Date newDate();  MyDates date(newDate);  cout << date.getDate()<< endl;    } |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP Check | 2.4.1 | XML Analysis | Automatically searches for errors and logic bugs that the compile and/or IDE misses. |
| JetBrains IDE | 2021 | IntelliJ Smart Debugger | Searches code as it is being written for logic/syntax bug as well as ambiguous code. Note: It can not be used in C++, only Java. |
| PVS-Studio | 2019 | Static Analysis | Similar to CPP Check as it also performs static analysis on C++ code. The main difference between the two is that PVS can work in conjunction with Visual Studios. |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT32-C | Protect your program from buffer overflows when performing mathematical operations. |

| **Noncompliant Code** |
| --- |
| When performing mathematical operations in C++ it is very important that the correct data type is assigned for each variable because if the data type is too small to hold the variable a buffer overflow can occur. It is important to remember that the storage capacity for signed integers is -127 to 127 and -250 to 250 for unsigned integers. It is extremely important that you safeguard your program from buffer over and under flow errors by checking the values before performing any type of mathematical operation. |
| int getSum(int numA, int numB) {  sum = numA + numb;  return sum; |

| **Compliant Code** |
| --- |
| As you can see within this code block, the programmer safeguarded their program against potential buffer overflow errors and ensures that a value is not stored outside of its data type’s limits. |
| int getSum(int numA, int numB) {  int sum;    if (((numB > 0) && (numA> (INT\_MAX - numB))) ||  ((numB < 0) && (numA < (INT\_MIN - numB)))) {  throw overflow\_error();  } else {  sum = numA + numb;  }  return sum;    } |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP Check | 2.4.1 | XML Static Analysis | Automatically searches for errors and logic bugs that the compile and/or IDE misses. |
| PVS-Studio | 2019 | Static Analysis | Similar to CPP Check as it also performs static analysis on C++ code. The main difference between the two is that PVS can work in conjunction with Visual Studios. |
| Unit Tests | Latest | J-Unit or Google Tests can be used | Develop Unit Tests using either the J-Unit framework or Google Tests depending on the language, for each method created. It is also important when testing intertwined classes to create unit tests that test the classes’ ability to function with each other. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR38-C | Do not confuse wide string functions with string functions. Wide strings can contain null characters which can cause a string function to behave unpredictably upon execution. |

| **Noncompliant Code** |
| --- |
| In this piece of code the strings may not be equal due to the fact that one string is a wide string data type. Wide strings can contain null values which can cause string methods such as strcmp or strcpy to behave unpredictably. Keep track of your data types while you are writing code to prevent hidden syntax bugs from arising. |
| import<iostream>  Using namespace std;    string:: stringOne = “string”;  wstring:: stringTwo = “string”; // wide string    bool stringCompare = strcmp(stringOne, stringTwo);  cout<< “These strings are equal: “;  cout<< stringCompare<< endl; |

| **Compliant Code** |
| --- |
| Use the same data types when performing operations on more than one variable. Do not try to combine doubles with ints or wide strings with strings. Even if you are sure that it will work, it is best to avoid completely because bugs arising from coding this way can be unpredictable and may not be detected during testing. |
| import<iostream>  Using namespace std;    string:: stringOne = “string”;  string:: stringTwo = “string”; // wide string    bool stringCompare = strcmp(stringOne, stringTwo);  cout<< “These strings are equal: “;  cout<< stringCompare<< endl; |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP Check | 2.4.1 | XML Static Analysis | Automatically searches for errors and logic bugs that the compile and/or IDE misses. |
| PVS-Studio | 2019 | Static Analysis | Similar to CPP Check as it also performs static analysis on C++ code. The main difference between the two is that PVS can work in conjunction with Visual Studios. |
| Unit Tests | Latest | J-Unit or Google Tests can be used | Develop Unit Tests using either the J-Unit framework or Google Tests depending on the language, for each method created. It is also important when testing intertwined classes to create unit tests that test the classes’ ability to function with each other. |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | SQL1-C | Validate and check all user input against SQL key words to prevent SQL injection attacks. |

| **Noncompliant Code** |
| --- |
| Within this piece of noncompliant code, the developer does not check user input against known SQL key words. This lapse in judgment leaves the program wide open to SQL injection attacks. |
| <Code>  boo takeInput(){  cout<<”Please enter your name and password using (n,pw) format.””;<<endl;  string userInput;  cin>> userInput;  sqlite3\_exec(db, userInput.c\_str(), callback, &records, &error\_message) != SQLITE\_OK);  //Note there are no checks against user input  <Code> |

| **Compliant Code** |
| --- |
| In order to make the above code compliant, you must implement a safe guard against sql injection attacks. In the code below a simple method was created to check for the OR 1=1 injection attack. This code is very limited and to protect completely against SQL attacks the method would need to be expanded to cover other SQL key words. It is key to realize that all user input must be validated before it is incorporated into your program. |
| bool check\_query(const std::string& sql)  {  std::size\_t orLoc = sql.find\_last\_of("OR"); // finds position of OR  std::size\_t orLoc2 = sql.find\_last\_of('or');  std::size\_t equalLoc = sql.find\_last\_of("="); // finds position of =  if ((orLoc != std::string::npos || orLoc2 != std::string::npos) && (equalLoc > orLoc || equalLoc > orLoc2))  // if or is before equal sql injection is blocked  {  return false;  }  return true;  }  <Code>  boo takeInput(){  cout<<”Please enter your name and password using (n,pw) format.””;<<endl;  string userInput;  cin>> userInput;  if(check\_query(userInput) == true)  {  sqlite3\_exec(db, userInput.c\_str(), callback, &records, &error\_message) != SQLITE\_OK);    }  else{  throw badSQLQuery()  }  <Code> |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SQL Inject Me | 2020 | Mozilla Firefox add-on Automated Tool | Tests a web-based program for SQL injection vulnerabilities. It is an easy to use, Mozilla Fire Fox Add on. |
| SQL Map | 2020 | Open Source SQL Injection Tool | Provides multiple automation tools as well as manual tools to help determine a program’s SQL vulnerabilities. |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM30-C | Do not access freed memory. Accessing a variable or piece of memory that has already been freed will result in null pointer exceptions and unpredictable behavior in your code. |

| **Noncompliant Code** |
| --- |
| In this code, a linked list is implemented; however, when the programmer wrote the readList method, they attempted to access freed memory which will result in a null pointer exception or unpredictable behavior at run time. Within the method, the programmer set the node on the list to null before setting the pointer to the next node on the list. This will cause the compiler to try to access information that no longer exists. |
| using namespace std;  class LinkedList{  public:  string word;  Node \* next;  };    void readList( Node \*item) {    while(item!=NULL){    cout<< item -> data << endl;  item=NULL;  item = item -> next;  }  // item is set to null before the next pointer is set! |

| **Compliant Code** |
| --- |
| In this compliant code block example, you can see that the next pointer is set before the item is set to null. This prevents a null pointer exception from occurring. |
| using namespace std;  class LinkedList{  public:  string word;  Node \* next;  };    void readList( Node \*item) {    while(item!=NULL){    cout<< item -> data << endl;  item = item -> next;  item=NULL;  }  // item is set to null after the next pointer is set! |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Not Very likely | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP Check | 2.4.1 | XML Static Analysis | Automatically searches for errors and logic bugs that the compile and/or IDE misses. |
| PVS-Studio | 2019 | Static Analysis | Similar to CPP Check as it also performs static analysis on C++ code. The main difference between the two is that PVS can work in conjunction with Visual Studios. |
| Unit Tests | Latest | J-Unit or Google Tests can be used | Develop Unit Tests using either the J-Unit framework or Google Tests depending on the language, for each method created. It is also important when testing intertwined classes to create unit tests that test the classes’ ability to function with each other. |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | ASRT01 | Avoid using assertions for anything other than debugging purposes! |

| **Noncompliant Code** |
| --- |
| Assertions are extremely helpful for writing tests and debugging your program but they should never be reachable by the user. If an assertion is reachable by the user it can leave your program vulnerable to exploitation. For example assert\_throws in java is very helpful in ensuring that the correct error is thrown during testing. However, if assertions are reachable by the user, it can cause the wrong errors to throw as well as cause unpredictable behavior at runtime. In this example a program is retrieving an user’s social security number and an assertion is used to ensure that the variable is not null. However, if the variable is null because it was not entered by the user correctly to begin with, it can cause an unchecked exception to be thrown which can cause the program to crash unexpectedly. It is better to use try/catch code blocks or use if statement followed by throw instead of using an assertion. |
| <Code>    string socSec = database.getUserSocialSecurity();  assert socSec != NULL;  <Code> |

| **Compliant Code** |
| --- |
| In the compliant version of this example, instead of using an assertion, the programmer uses an if statement to check if the variable is null and throws the correct exception if true. This will help ensure that the correct errors are thrown during runtime and will ensure that the program behaves exactly as it should at runtime, limiting possible bugs. |
| <Code>    string socSec = database.getUserSocialSecurity();  if(socSec == NULL){  throw NullPtrExc();  }    <Code> |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP Check | 2.4.1 | XML Static Analysis | Automatically searches for errors and logic bugs that the compile and/or IDE misses. |
| PVS-Studio | 2019 | Static Analysis | Similar to CPP Check as it also performs static analysis on C++ code. The main difference between the two is that PVS can work in conjunction with Visual Studios. |
| Unit Tests | Latest | J-Unit or Google Tests can be used | Develop Unit Tests using either the J-Unit framework or Google Tests depending on the language, for each method created. It is also important when testing intertwined classes to create unit tests that test the classes’ ability to function with each other. |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR33-C | Detect and handle standard library errors correctly. |

| **Noncompliant Code** |
| --- |
| In this code example the programmer has created a simple remove file method. Although the method’s syntax is correct, the programmer has not implemented an occurrence for if the remove method should fail for any reason. For example, if the file name entered does not exist on the computer the remove method will return null but the removeFile method will return true. This is an issue because it the exception from the remove method is left unchecked which can result in the program failing or behaving unpredictably. Depending on the program, this could be a difficult bug to find because the method will always return true. So, it is important to build in exception handling for built in methods as well as for custom methods. |
| import<iostream>;  using namespace std;  bool removeFile( char file[ ] ) {  remove(file);  return true;  } |

| **Compliant Code** |
| --- |
| In this example, the programmer implemented error handling in the case that the remove method returned NULL instead of 0. Thus, preventing a unchecked exception from occurring. When the method is called and returns false the program or user will know that the remove method failed unlike in the previous example where the method always returned true. |
| import<iostream>;  using namespace std;  bool removeFile( char file[ ] ) {  if(remove(file) == NULL)  {  return false;  }  else  return true;  } |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP Check | 2.4.1 | XML Static Analysis | Automatically searches for errors and logic bugs that the compile and/or IDE misses. |
| PVS-Studio | 2019 | Static Analysis | Similar to CPP Check as it also performs static analysis on C++ code. The main difference between the two is that PVS can work in conjunction with Visual Studios. |
| Unit Tests | Latest | J-Unit or Google Tests can be used | Develop Unit Tests using either the J-Unit framework or Google Tests depending on the language, for each method created. It is also important when testing intertwined classes to create unit tests that test the classes’ ability to function with each other. |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| User Access | CLA01 | Do not allow the user to directly modify your program without proper input checks |

| **Noncompliant Code** |
| --- |
| Within this example the programmer has allowed direct access to class variables and this is leaving the program extremely vulnerable to user exploitation. Since there is nothing stopping unwanted input the user can put anything they want in for the class’s variables. In the case that this class was not storing dog information but personal user information, creating a class like this would allow the user direct access to manipulate sensitive user information. |
| class Dogs{  private string breed;  private string name;  public Dogs(string brd, string nme){  breed = brd;  name = nme;  }  int main() {    string breed;  string name;  cin >> breed;  cin >> name;  Dogs marley(breed, name);  } |

| **Compliant Code** |
| --- |
| Within this coding example, the programmer safeguards their program against user exploitation by ensuring that the user does not have direct access to class variables by implementing a set method for each of the class’s variables. |
| class Dogs{  private string breed;  private string name;  public Dogs(string brd, string nme){  try{  breed = setBreed(brd);  name = setName(nme);  }  catch(invalidInputExc)  {  <Do something with error>  }  }  string setBreed(string brd){  if(sizeof(brd)>11 || brd ==NULL)  {  throw invalidInputExc();  }  else {  this.breed = brd;  }  String setName(string nme){  if(sizeof(nme)>11 || nme==NULL)  {  throw invalidInputExc();  }  else {  this.name = nme;  }  }  int main() {    string breed;  string name;  cin >> breed;  cin >> name;  Dogs marley(breed, name);  } |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Peer Reviewing | n/a | Peer Analysis | Although this is not an automated tool, automated programs can not check against every programing best practice. So, for some coding standards a peer can be your best asset available. |
| CPP Check | 2.4.1 | XML Static Analysis | Automatically searches for errors and logic bugs that the compile and/or IDE misses. |
| PVS-Studio | 2019 | Static Analysis | Similar to CPP Check as it also performs static analysis on C++ code. The main difference between the two is that PVS can work in conjunction with Visual Studios. |
| Unit Tests | Latest | J-Unit or Google Tests can be used | Develop Unit Tests using either the J-Unit framework or Google Tests depending on the language, for each method created. It is also important when testing intertwined classes to create unit tests that test the classes’ ability to function with each other. |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Clarity | CLR01 | Use proper variable names and comment your code |

| **Noncompliant Code** |
| --- |
| It is clear from this code block, that the programmer did not implement descriptive variable names and methods into their code. The programmer also failed to comment their code which makes it even more difficult for an outside programmer to understand how this code functions and what its purpose is. By not using complete words for variable names, another programmer does not know if “bd” means breed, bed, or big dog. From the way the programmer designed their code and their lack of comments it will be very difficult for another programmer to come in and expand this class or to even debug it. Coding in this fashion, not only wastes other programmers’ time, it also is a security threat because the true meaning of the code has not been explained through comments or proper variable names. Teamwork is the foundation in which this company was built, so it is extremely important that your code is conducive to a team environment. |
| protected class Dogs{  private string bd;  private string n;  protected Dogs(string br, string nm){  try{  bd = setBd(br);  n = setNm(nm);  }  catch(invalidInputExc)  {  <Do something with error>  }  }  protected string setBd(string br){  if(sizeof(br)>11 || br ==NULL)  {  throw invalidInputExc();  }  else {  this.bd = br;  }  protected string setNm(string nm){  if(sizeof(nm)>11 || nm==NULL)  {  throw invalidInputExc();  }  else {  this.n = nm;  } |

| **Compliant Code** |
| --- |
| While writing this block of code, the programmer adequately commented the code so that other programmers would understand how the program functions. Also, the programmer implemented descriptive variable names and methods so that their purpose is known to those who are unfamiliar with the program. This is an essential coding requirement because it ensures that multiple programmers are able to work on the same program as well as allows work to be shared throughout the company. Since code is created within a development team, it is essential that you are not the only one who understands the syntax of your code. |
| protected class Dogs{  private string breed;  private string name;  protected Dogs(string brd, string nme){  try{  breed = setBreed(brd);  name = setName(nme);  }  // will try to set Breed and name, and will catch exception if the methods fail  catch(invalidInputExc)  {  <Do something with error>  }  }  // checks user input before setting the breed variable  protected string setBreed(string brd){  if(sizeof(brd)>11 || brd ==NULL)  {  throw invalidInputExc();  }  else {  this.breed = brd;  }  //checks user input before setting the name variable  protected string setName(string nme){  if(sizeof(nme)>11 || nme==NULL)  {  throw invalidInputExc();  }  else {  this.name = nme;  } |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Peer Reviewing | n/a | Peer Analysis | Although this is not an automated tool, automated programs cannot check against every programing best practice. So, for some coding standards a peer can be your best asset available, especially regarding using proper naming conventions and commenting practices. |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| User Access | CLA02 | Do not make inner classes and methods public and accessible to the user |

| **Noncompliant Code** |
| --- |
| In this compliancy example, the programmer made this inner class as well as its methods public. Thus, giving any user or program complete access and control of the program’s inner class and variables. This is a major security concern for our company and coding in this way is not tolerated. It is never a good idea to give a user or another program complete and unlimited access to your program’s inner workings; and, the security consequences that result from coding in this way are limitless. |
| public class Dogs{  public string breed;  public string name;  public Dogs(string brd, string nme){  try{  breed = setBreed(brd);  name = setName(nme);  }  catch(invalidInputExc)  {  <Do something with error>  }  }  public string setBreed(string brd){  if(sizeof(brd)>11 || brd ==NULL)  {  throw invalidInputExc();  }  else {  this.breed = brd;  }  public string setName(string nme){  if(sizeof(nme)>11 || nme==NULL)  {  throw invalidInputExc();  }  else {  this.name = nme;  } |

| **Compliant Code** |
| --- |
| Within this example, the programmer used the protected modifier in order to safeguard their program’s inner classes and methods. By using the protected modifier, a program or user outside of the source code’s package cannot access the class or its methods. Doing this helps ensure only the right programs and users have access to the class. Also, by making the class variables private, the programmer ensures that the class’s variables cannot be directly accessed or modified by a user or another program. This style of programming follows the principle of least privilege because it ensures that users and other programs have the least possible privileges to the class and its variables. |
| protected class Dogs{  private string breed;  private string name;  protected Dogs(string brd, string nme){  try{  breed = setBreed(brd);  name = setName(nme);  }  catch(invalidInputExc)  {  <Do something with error>  }  }  protected string setBreed(string brd){  if(sizeof(brd)>11 || brd ==NULL)  {  throw invalidInputExc();  }  else {  this.breed = brd;  }  protected string setName(string nme){  if(sizeof(nme)>11 || nme==NULL)  {  throw invalidInputExc();  }  else {  this.name = nme;  } |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Peer Reviewing | n/a | Peer Analysis | Although this is not an automated tool, automated programs can not check against every programing best practice. So, for some coding standards a peer can be your best asset available. |
| CPP Check | 2.4.1 | XML Static Analysis | Automatically searches for errors and logic bugs that the compile and/or IDE misses. |
| PVS-Studio | 2019 | Static Analysis | Similar to CPP Check as it also performs static analysis on C++ code. The main difference between the two is that PVS can work in conjunction with Visual Studios. |
| Unit Tests | Latest | J-Unit or Google Tests can be used | Develop Unit Tests using either the J-Unit framework or Google Tests depending on the language, for each method created. It is also important when testing intertwined classes to create unit tests that test the classes’ ability to function with each other. |

## 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.

Explanation of Automation:

Some of the tools that will be used to aide in enforcing this company’s coding standard include: CPP Check, Jet Brain’s IntelliJ IDE automated debugging tools, Unit Tests, CPP Check, built-in tools within Visual Studios, PVS Studios, SQL Inject Me, and SQL Inject Me. All of the tools other than the SQL injection prevention tools will be used mainly during the build and verify/test phases. There is a major emphasis on the importance of unit testing here at DevOps and we expect our programmers to have line coverage within the unit tests of 98 to 100%. Enforcing a line coverage rule ensures that each line of code written at Dev Ops has been properly tested before being released. Using tools like CPP check and PVS studios, will be employed heavily during the building as well as testing phases to ensure that all possible bugs have been remedied. Dev Ops also stresses the importance of following compiler warnings, and code that disables compiler warnings will not be tolerated unless the developer has received written permission to do so. In regard to the SQL injection tools, they will be used in the building and testing phases as well as during the Monitor and Health Check Phases. Creating secure and sound code is of the utmost importance here at Dev Ops and developing code that “cuts-corners” or is considered to be insecure in regard to our programming standards and practices will not be tolerated. The image above lays out our production process and what to expect during each step. Any deviation from the steps laid out in the image must be consented to by management first.

## 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 |
| DCL53-CPP | Medium | Likely | Medium | High | 2 |
| INT32-C | High | Likely | Medium | High | 3 |
| STR38-C | Low | Likely | Low | Medium | 1 |
| SQL1-C | Very High | Very Likely | Very High | Very High | 5 |
| MEM30-C | Medium | Not Very Likely | Low | Medium | 3 |
| ASRT01 | Low | Likely | Low | Medium | 2 |
| ERR33-C | High | Likely | Medium | High | 4 |
| CLA01 | Very High | Very Likely | Very High | Very High | 5 |
| CLR01 | Low | Likely | Low | Medium | 1 |
| CLA02 | High | Likely | High | High | 4 |

## 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 in rest, is data that is encrypted once it is no longer actively in use and is encrypted before it is stored in a system’s memory or on a server. Examples of data at Dev Ops that is encrypted in rest is our employee passwords and employee personal data. Dev Ops wants to ensure that our employee information is does not fall into the wrong hands so before it is stored it is encrypted for its protection. |
| Encryption at flight | Encryption at flight, is data that is being encrypted before being sent to another network or server. Basically, this type of encryption occurs anytime sensitive data is being sent or transferred. At Dev Ops, we encrypt all sensitive data before sending it anywhere, even within our own company. This practice will ensure that our data is always in a protected state before being sent for other usages. One example of encryption in-flight is our use of only HTTPS web traffic. |
| Encryption in use | Encryption in use, is data that remains encrypted while it is in use. Examples of how Dev Ops employs encryption in use is while processing financial data and password verification. These types of user data must be protected at all times, meaning it must be encrypted at all times. There is a zero tolerance policy here at Dev Ops for misuse of sensitive data and not following the proper encryption protocols regarding sensitive information. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a one of the most important security policies, as it is the first shield of protection that Dev Ops employs. Authentication is the act of verifying and validating all user and employee input data. All usernames, passwords, emails, and security questions must be authenticated before granting access to a Dev Ops public or private program or system. In the case of remote use, the network that is requesting access must be fully authenticated before being granted access. Without proper authentication, Dev Ops is vulnerable to any type of malicious access, so not employing the proper authentication protocols will not be tolerated. |
| Authorization | Authorization is another important protocol that is strictly enforced here at Dev Ops. A user must provide authorization before having access to any private or public Dev Ops system or program. Internal authorization can only be granted by upper management and Human Resources depending on the level of authorization. We follow the principle of least privilege here at Dev Ops, which ensures that no one is granted access for more than what is required to do their necessary tasks. This protocol protects Dev Ops against improper usage as well as ensures that any sensitive data can not be used illegally or maliciously. |
| Accounting | Accounting is an important protocol that is also strictly enforced at Dev Ops. Accounting ensures that the right people are granted access while the wrong people are not: each user and employee account is accounted for and have the correct level of access. A strict accounting policy allows Dev Ops to ensure that each account is for legitimate use. |

**\***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.

|  |  |  |
| --- | --- | --- |
| **Principle** | **Standard** | **Connection** |
| 1. ValidateInput Data | CLA01 : Validate User Input | The connection between this standard and this principle is that they both regard the importance of validating user input during a program’s execution. |
| 2. Heed Compiler Warnings | DCL53-CPP : Do Not Write Syntactically Ambiguous Code | The connection between this standard and this principle is that ambiguous code is normally caught with newer IDEs and the warnings from the compiler should always be followed, especially in the cases of ambiguous code. |
| 3. Architect and Design for Security Policies | CLR01: Use proper names and commenting procedures | The connection between this standard and this principle is that they both regard the importance of following Dev Ops design protocols. Dev Ops wants their programs designed to their standards, aesthetically as well as internally. |
| 4. Keep It Simple | DCL53-CPP : Do Not Write Syntactically Ambiguous Code  CLR01: Use proper names and commenting procedures | The connection between these standards and this principle is that they both deal with how Dev Ops wants their programs written. Dev Ops values clear easy to read code that can easily be shared among team members, so it is important to use proper names and write clear and unambiguous code. |
| 5. Default Deny | CLA01: Do not allow user input that has not been verified | The connection between this standard and this principle is that they both follow a policy of default denial. In regard to the standard user input is denied until it has been properly verified. Thus, following the Dev Ops Default Deny principle. |
| 6. Adhere to the Principle of Least Privilege | CLA02: Do not allow user access to inner methods and classes | The connection between this standard and this principle is that both adhere to the idea that in the case of user access less is more. The standard limits user access to only what is needed to run the program, thus, adhering to the principle of least privilege. |
| 7. Sanitize Data Sent to Other Systems | SQL1-C : Validate all SQL user-input | The connection between this standard and this principle is that all Dev Ops data in rest should be encrypted, to prevent malicious access during a SQL injection attack. By sanitizing the data sent to a database ensures that no data will be improperly released. |
| 8. Practice Defense in Depth | SQL1-C : Validate all SQL user-input  CLA02: Do not allow user access to inner methods and classes  CLA01: Do not allow user input that has not been verified | The connection between these standards and this principle is that implementing these standards in conjunction with each other is practicing defense in depth. Each of these standards provides a layer of protection against malicious user input ensuring that the program is protected from multiple angles, thus practicing defense in depth. |
| 9. Use Effective Quality Assurance Techniques | All of the Dev Ops Coding Standards | The connection between this company’s coding standards and this principle is that following the coding standards laid out in this document is using an effective quality assurance technique. Each of these standards provides a way to further improve the quality of a program as well as prevent unpredictable behavior within a program. |
| Adopt a Secure Coding Standard | All of the Dev Ops Coding Standards | The connection between this company’s coding standards and this principle is that by following the coding standards laid out in this document you are adopting a secure coding standard. The standards laid out in this document safe guard a program from unpredictable behavior as well as malicious access thus providing a secure coding standard. |

**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 | 05/02/2020 | Company Practices | Jacqueline Woods | [Insert text.] |
| 3.0 | 05/21/2020 | Document Completion | Jacqueline Woods | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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