**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 | When looking to validate input data there are a few things that need to be kept in mind. The first think is to think that all data input is vulnerable. This helps the developer to be able to have a wide view on the situation to ensure that the data is safe especially if there are multiple applications reading from the same database. The next point is to think that the user has a malicious intent. From here this allows the developer to be able to determine and set boundaries on what a trusted user would be inputting and using the data for while a malicious user would not use it for this purpose. This would allow for the developer to be able to set a logic to the application to be able to look for those things and flag them if detected. This should also check the data from any untrusted sources, network interfaces, and environmental variables. |
| 1. Heed Compiler Warnings | When compiling the code in order to make the code be able to work proficiently as possible you must not only fix the errors but as well as the warnings. Most warnings can be cleared by simple edits to the code. Over time these warnings when the applications are up and running could eventually become vulnerable or weak spots within the code. |
| 1. Architect and Design for Security Policies | This is a policy for keeping the software secure. Each program is different in purpose, design, intent and security privileges. One of the best suggestions for this section is to break up the project into sections and ensure that the security measures are met in each section instead of waiting to the end, and leaving possible room for vulnerability. |
| 1. Keep It Simple | Simply do not over complicate the code. The more complicated the code the more likely there are to be errors. The more complex the code the more the security privileges and ensuring that the code is secure is going to increase. |
| 1. Default Deny | Instead of denying access based on excluding people base it off of permissions. When looking at who in the company would need the permissions it would be anyone that would need access to that information in order to be able to do their daily tasks. |
| 1. Adhere to the Principle of Least Privilege | An process within the system should execute the least amount of privileges in the least amount of time possible to execute the task. This helps to reduce any errors, least amount of permissions (while this will be validate at the beginning, it will not have to validate more than once in the process). The least amount of time would not allow for much time if hacked for the hacker to be able to gain access to that information. |
| 1. Sanitize Data Sent to Other Systems | This deals with the calling process, not necessarily input data. As data is being shared from system to system this would be using a calling process. This could be information such as SQl command texts. This works to prevent injection attacks. |
| 1. Practice Defense in Depth | In a sense manage risk through multiple security defense strategies. This helps to protect the data if the first layer of protection gets broken through, has a vulnerability or weak point, or becomes in adequate. The more sensitive information the more defense strategies the application is more than likely going to have. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance techniques could help to reduce the risk of vulnerabilities. When looking at external review which allows for a independent review by correcting any invalid assumptions. This can help lead to more secure systems. |
| 1. Adopt a Secure Coding Standard | This is creating a standard for the platform and language that you are working on. This will help to ensure that the system is the same all the way through. |

## 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** | **Declarations and Initialization** |
| --- | --- | --- |
| **Data Type** | [STD-001-DCL] | This can be broken up into two parts. The first part is the declarations, the recommendation for this is one declaration per line – this encourages comments which allows for the code to be easier followed. The next part is initialization this is referring to initialing local variables after they have been declared not later in the code. This can cause the user to not initialize the variable. The initialization will determine what the initial value is. |

| **Noncompliant Code** |
| --- |
| For this block this is adding in two different variables on the exact same line. As well as two different data types on the same line. This makes reading the code tricky, as well as does not encourage comments. |
| // Assuming same data type but different variables. No comments added.  int height, length;  // Two different data types on the same line  int foo, fooarray[]; |

| **Compliant Code** |
| --- |
| In order for the code to be compliant the code should have one variable declared per line, with comments. As well as any variable that illudes to a starting value should be initialized soon after the variable is declared. |
| // Same data type with comments easier to understand what is happening in the code. As well as the variables are initialized  int height; // Height variable  height == 8;  int length; // length  length == 3;  // Two different data types on the same line  int foo; // name of object  int fooarray[]; // object array |

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

|  |
| --- |
| **Principles(s):** Keep it simple. This maps to the standard by not making the data types over complex and difficult to read. Also by keeping it simple it encourages comments which help to make the code easy to follow. This will also help the developer to know if the local variable has been declared or not. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High - 12 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2020a | CERT C++: DCL50-CPP | Checks for function definition with ellipsis notation |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-DCL50-a | Functions shall not be defined with a variable number of arguments. |

### Coding Standard 2

| **Coding Standard** | **Label** | **Expressions** |
| --- | --- | --- |
| **Data Value** | [STD-002-EXP] | This is referring to using symbolic names instead of literal values. This helps to make the code more manageable and readable. |

| **Noncompliant Code** |
| --- |
| Noncompliant code would be code that uses literal values instead of symbolic names throughout the code. |
| If (10 < 23)  {  Price = unitsOrdered \* 23;  }  Else  {  Price= unitsOrdered \* 17  } |

| **Compliant Code** |
| --- |
| The compliant code will set the symbolic name and the value at the beginning of the code equal to each other. Then uses the symbolic name through out the code. |
| //Quantity = user input  kDiscount == 23; // Discount associated  KStandard == 17 // normal price range  If (Quantity < KDiscount)  {  Price = unitsOrdered \* KDiscount;  }  Else  {  Price= unitsOrdered \* KStandard;  } |

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

|  |
| --- |
| **Principles(s):** Keep it simple – this maps to the standard by making the value understandable to what it is defining such as you do not want to use the numbers as you may forget and use the wrong on later within the code. While having a variable makes things a lot simpler.  Using Effective Quality Assurance Techniques – this maps to the standard as by a external reviewer they would be able to understand what that value is, and why it is being used. While a just a value with nothing attached can make it very confusing and allow for assumptions to be made that could be harmful to the application.  Heed Compiler Warnings – this maps to the standard as it could throw flags that there is a problem that was detected, or that there is multiple values or data types at the same time. This could happen if the value is associated to the same name. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required. |
| Coverity | V7.5.0 | EVALUATION\_ORDER | Can detect the specific instance where a statement contains multiple side effects on the same value with an undefined evaluation order because, with different complier flags or different compilers or platforms, the statement may behave differently. |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP\_EXP50-a  CERT\_CPP\_EXP50-b  CERT\_CPP\_EXP50-c  CERT\_CPP\_EXP50-d  CERT\_CPP\_EXP50-e  CERT\_CPP\_EXP50-f | The value of an expression shall be the same under any order of evaluation that the standard permits  Don’t write code that depends on the order of evaluation of function arguments  Don’t write code that depends on the order of evaluation of function designator and function arguments  Don’t write code that depends on the order of evaluation of expression that involves a function call  Between sequence points an object shall have its stored value modified at most once by the evaluation of an expression  Don’t write code that depends on the order of evaluation of function calls. |
| PolySpace Bug Finder | R2020a | CER C++: EXP50-CPP | Checks for situations where expression value depends on order o evaluation |

### Coding Standard 3

| **Coding Standard** | **Label** | **Integers** |
| --- | --- | --- |
| **String Correctness** | [STD-003-INT] | This is ensuring that a unsigned integer is not involving with operations that could cause an overflow on the code. This would cause the number to be greater than what it is actually supposed to be |

| **Noncompliant Code** |
| --- |
| The noncompliant code can result in a integer wrap with the addition of a unsigned operation. The resulting value may be used to allocate insufficient memory for a subsequent operation which could lead to vulnerability. |
| Void func(unsigned int ui\_b, unsigned int ui\_c)  {  Unsigned int usum = ui\_b + ui\_c;  /\*…\*/  } |

| **Compliant Code** |
| --- |
| The compliant code would preform a precondition test of the of the operands to guarantee there is no possibility of unsigned wrap |
| #include <limits.h>  Void func(unsigned in ui\_b, unsigned in ui\_c)  {  Unsigned int usum;  If (UINT\_MAX – ui\_b < ui\_c)  {  /\*Handle error\*/  }  Else  {  Usum = ui\_ b + ui\_c;  }  /\*..\*/  } |

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

|  |
| --- |
| **Principles(s):** Heed Compiler Warnings – this maps to the standard as this would cause an overflow. This would come up once the code is compiled. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  MISC.MEM.SIZE.ADDOFLOW  MISC.MEM.SIZE.BAD  MISC.MEM.SIZE.MULOFLOW  MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size  Integer overflow of allocation size  Multiplication overflow of allocation size  Subtraction underflow of allocation size  Addition overflow of size  Unreasonable size argument  Multiplication overflow of size  Subtraction underflow of size |
| Parasoft C/C++test | 2020.2 | CERT\_C-INT30-a  CERT\_C-INT30-b  CERT\_C-INT30-c | Avoid integer overflows  Integer overflow or underflow in constant expressions in ‘+’, ‘-‘. ‘\*’ operator  Integer overflow or underflow in constant expression in ‘<<’ operator |
| Polyspace Bug Finder | R2020a | CERT C: Rile INT30-C | Checks for  Unsigned integer overflow  Unsigned integer constant overflow |

### Coding Standard 4

| **Coding Standard** | **Label** | **Containers** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CTR] | This section deals with containers. A container is a class, a data structure, or an abstract data type. An SQL injection finds the vulnerabilities and exploits them. |

| **Noncompliant Code** |
| --- |
| Any non compliant code would not have any checks to ensure that the user is who they say that they are. |
| Mysql > use sqllab\_users;  Database changed  Mysql > show tables; |

| **Compliant Code** |
| --- |
| The compliant code shows a series of autheitation in order to be able to the changes are being completed by the appropriate user. |
| $input\_uname = $\_GET(‘username’);  $input\_pwd = $\_GET(‘Password’);  $hashed\_pwd = shal($input\_pwd);  …  $sql = “SELECT id, name, eid, ssalaray, birth, ssn, address, emall, Paswword  FROM credential  WHERE name = ‘$input\_uname’ and Password=’$hashed\_pwd’”;  $result = $conn -> query($sql); |

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

|  |
| --- |
| **Principles(s):** Default Deny – This maps to the standard because if the user is not an appropriate user then it would deny access. This would prevent the sql injection, with authentication. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.MEM.PBB  LANG.MEM.PPE | Buffer overrun  Buffer underrun  Type overrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object |
| Parasoft C/C++test | 2020.2 | CERT C++-CTR50-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2020a | CERT C++: CTR50-CPP | Checks for:  Array access out of bounds  Array access with tainted index  Pointer dereference with tainted offset |

### Coding Standard 5

| **Coding Standard** | **Label** | **Characters and Strings** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-STR] | When representing characters ensure that the appropriate type is being used. With strings adopting and implementing a consistent plan. This can help when modifying, adding strings as they can become over complicated in the memory quickly. |

| **Noncompliant Code** |
| --- |
| This noncompliant code demonstrates an off-by-one error. This is because the loop does not account of the null-termination character. |
| Void copy(size\_t n, char src[n], char dest[n]  {  Size\_t i;  For (I = 0; src[i] && (I < dest[i] = src[i];  }  Dest[i] = “\0’;  } |

| **Compliant Code** |
| --- |
| For this compliant code the loop termination condition is modified to be able to account for the null termination that is appended to dest. |
| Void copy(size\_t n, char src[n], char dest[n])  {  Size\_t I;  For (I = 0; src[i] && ( I < n-1); ++i)  {  Dest[i] = src[i];  }  Dest[i] =’\0’;  } |

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

|  |
| --- |
| **Principles(s):** Architect and Design for Security Policies – This is creating a policy and a plan in order to be able to stay constant as adding the string can become over complicated in the memory. This maps to the standard of helping to keep it over complex. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Parasoft C/C++test | [Insert text.] | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the ‘char’ buffer to store input from std::cin |
| Polyspace Bug Finder | R2020a | CERT C++ STR50-CPP | Checks for:  Use of dangerous standard function  Missing null in string array  Buffer overflow from incorrect string format specifier  Destination buffer overflow in string manipulation |

### Coding Standard 6

| **Coding Standard** | **Label** | **Memory Management** |
| --- | --- | --- |
| **Assertions** | [STD-006-MEM] | CWE entries are partially or fully elimated. This can cause memory to have vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The code below shows a local pointer to return an int however an int is not allocated |
| Int\* local\_pointer()  {  Int temp = 100;  //return a point to the local int  return(&temp);  }  Int main(0  {  It\* ptr = local\_pointer();  Return 0;  } |

| **Compliant Code** |
| --- |
| This is an example of storing memory on a local storage, which will result in local variables. |
| //Local storage example  Int Square(int number)  {  Int result;  Result = number \* number;  Return result;  } |

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

|  |
| --- |
| **Principles(s):** Use effective quality assurance techniques – this maps to the to help reduce risk of vulnerabilities. The technique could be to full get rid of anything in the memory fully. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | ALLOC.UAF | Use after free |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Polyspace Bug Finder | R2020a | CERT c++: MEM50-CPP | Checks for:  Pointer access out of bounds  Deallocation of previously deallocated pointer  Use of previously freed pointer |

### Coding Standard 7

| **Coding Standard** | **Label** | **Input Output** |
| --- | --- | --- |
| **Exceptions** | [STD-007-FIO] | For the input you do not want to include the user input within the strings. The contents of a formatted string is more likely to crash with a formatted string and is easier for a hacker to be able to access. |

| **Noncompliant Code** |
| --- |
| For the noncompliant code the users input is already located within the string. This leaves a vulnerability within the code |
| Int main()  {  Int 10;  Int 12;  Int sum;  Sum = 10 + 12;  Print(sum); |

| **Compliant Code** |
| --- |
| In order to be able to read what the user input the input must be called and then can be printed out within the output. Th |
| Int main()  {  Int a;  Int b;  Int sum;  Scanf(“%d, %d”, &a,&b);  Sum = a+b)  Printf((“%d”, sum);  Return 0; |

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

|  |
| --- |
| **Principles(s):** Validate Input data – this maps the standard as it works to it validates the users input however, does not store the input in the string. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format String |
| Parasoft C/C++test | 2020.2 | CERT\_FIO30-a  CERT\_FIO30-b  CERT\_FIO30-C | Avoid calling function printf/wprintf with only one argument other than string constant  Avoid using function fprint/fwprintf with only two parameters when second parameter is a variable  Never use unfiltered data from an untrusted user as the format parameter |
| Polyspace Bug Finder | R202a | CERT C: Rule FIO30-c | Checks for tainted string format |

### Coding Standard 8

| **Coding Standard** | **Label** | **Exceptions and Error Handling** |
| --- | --- | --- |
| Error Reporting | [STD-008-ERR] | Throwing exceptions will help to determine if the code has failed. The code is looking at all the objects individually could be failed if they are empty. The fail could turn it into a vulnerability. |

| **Noncompliant Code** |
| --- |
| For the error code, this code works only up to 32 bytes of data before a vulnerability is seen. When the error occurs it would be located within the subsystem. This set of code does not look for any exceptions within the code, if anyone of the functions fails they will all fail. |
| // Error Handling  If (obj1.Function() == ERROR)  {  HandleError();  }  If(obj2.Function() == ERROR)  {  HandleEroor();  }  If(obj3.Function() == EROOR)  {  HandleEroor();  } |

| **Compliant Code** |
| --- |
| This set of code is looking for any exceptions. This will look at all three of the objects prior to the code failing. |
| Try  {  Obj1.Function();  Obj2.Function();  Obj3.Function();  }  Catch (const std::exception &e)  {  HandleError();  } |

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

|  |
| --- |
| **Principles(s):** Heed Complier Warnings – This maps the standard as it is looking for any sections of code that could be empty. In most cases this will show up as a warning as it can detect anything. |

**Threat Level**

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

**Automation**

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

### Coding Standard 9

| **Coding Standard** | **Label** | **Object Oriented Programming** |
| --- | --- | --- |
| Abstract models | [STD-009-OOP | Object Oriented programming deals with creating classes, indenting whitespace, and naming conventions. Some of the best practices is using something practical in order to best understand the class as well as being able to pull from the class later in the application without odd naming, which could pull from the wrong class or data. Giving out the wrong information. |

| **Noncompliant Code** |
| --- |
| This code does not use good naming conventions, and could cause wrong data to be pulled. |
| Interface DogInterface  {  Public function meow(); // This naming convention could make you think that you are calling a cat method or class.  Public function eatFoo($amount);  }  Class BorderCollie implements Doginterface  {  Protected $foodEaten = 1;  Public function meow()  {  Return t(‘Bark Bark!’);  }  Public function eatFood($amount)  {  $this->foodEaten += $amount;  }  } |

| **Compliant Code** |
| --- |
| This code uses a good naming convention that matches what the class is being created for. |
| Interface DogInterface  {  Public function bark();  Public function eatFood($amount);  }  Class BorderCollie implements Doginterface  {  Protected $foodEaten = 1;  Public function bark()  {  Return t(‘Bark Bark!’);  }  Public function eatFood($amount)  {  $this->foodEaten += $amount;  }  } |

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

|  |
| --- |
| **Principles(s):** Keep it simple – this maps to the standard as by keeping it simple it matches the class and is simple to recall later on. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | Low - 2 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-OOP50-a  CERT\_CPP-OOP50-b  CERT\_CPP-OOP50-c  CERT\_CPP-OOP50-d | Avoid calling virtual functions from constructors  Avoid calling virtual functions from destructors  Do not invoke class virtual functions from any of its constructors  Do nor invoke classes virtual functions from it destructor |

### Coding Standard 10

| **Coding Standard** | **Label** | **Concurrency** |
| --- | --- | --- |
| Threads | [STD-010-CON] | The most common is that the algorithms can be executed sequential can be parallel. |

| **Noncompliant Code** |
| --- |
| For this noncompliant the code runs fine, until it is running a muti thread algorithm to which it is no longer protected and becomes vulnerable. |
| Double cached\_computation (double a)  {  Static double cached\_a = 0.0; //1  Static double cached\_result = COMPUTATION\_OF\_ZERO; //2  Double result;  If (cached\_a == a) //1  Return cached\_result //2  Result = computation(a);  Cached\_a = a; //1  Cached\_result = result; //2  Return result;  } |

| **Compliant Code** |
| --- |
| The compliant code protects the critical data. This is important especially during multi threading. As this would protect the data that had once been exposed. |
| Std::mutea m\_a;  Std::mutea\_ mresult;  Double cached\_computation (double a)  {  Static double cached\_a = 0.0;  Static double cached\_result = COMPUTATION\_OF\_ZERO;  Double result;  {  Std::scooped\_lock(m\_a,m\_result);  If(cached\_a ==a)  {  Return cached\_result;  }  }  Result = computation(a);  {  Std::lock\_guard<std::mutea >lock(m\_a);  Cached\_a = a;  }  {  Std::lock\_guard<std::mutea > lck(m\_result);  Cached\_result = result;  }  Return result;  }  //2  Std::mutea m;  Double cached\_computation(double a)  {  Static double cached\_a = 0.0;  Static double cached\_result = COMPUTATION\_OF\_ZERO;  Double result;  {  Std::lock\_guard<std::mutea > lock(m);  If (cached\_a == a) return cached\_result;  {  Result = computation(a);  Cached\_a = a;  Cached-result = result;  }  }  Return result;  }  Std::mutea cachedComputationMutea;  {  //3  Std::lock\_guard<std::mutea > lck(cachedComputationMutea);  Auto cached = cached\_computation(3.33);  }  Double cached\_computation (double a)  {  Thread\_loacl double cached\_a = 0.0;  Thread\_local double cached\_result = COMPUTION\_OF\_ZERO;  Double result;  If (cached\_a ==a) return cached result;  {  Result = computation(a);  Cached\_a = a;  Cached\_result = result;  Return result;  }  } |

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

|  |
| --- |
| **Principles(s):** Adopt a Secure Coding standard – This helps to map this standard as it will ensure that it is done the same way throughout the entire application. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-CON51-a | Do not call lock() directly on a mutex |

## 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.

All of these were taken from: <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

## 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.

There are a couple of different locations that they can use the automation tools. The first one is during the build process. Any of the warnings that the tools give will be able to give Green Pace a chance to fix that issue and allow for it to not be a vulnerability. Verify and test during the configuration along with the tools it would allow for any problems that could be a vulnerability. Transition and health check during this time the developer can diagnosis any on going vulnerabilities with the tools that are still working on solutions. This is because not every new vulnerability has a solution. Monitor and detect the current code and detect in new issue using the appratie tools. Maintain and stabilize like the transition and health check this could be a routine to verify any changes or updates to the application to ensure that it is still running smoothly.

## 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-DCL | High | Probable | Medium | High - 12 | 1 |
| STD-002-EXP | Medium | Probable | Medium | Medium - 8 | 2 |
| STD-003-INT | Medium | Unlikely | Medium | Low - 4 | 3 |
| STD-004-CTR | High | Likely | High | Medium - 9 | 2 |
| STD-005-STR | High | Likely | Medium | High-18 | 1 |
| STD-006-MEM | High | Likely | Medium | High-18 | 1 |
| STD-007-FIO | Low | Likely | Medium | Medium - 6 | 2 |
| STD-008-ERR | Low | Probable | Medium | Low - 4 | 3 |
| STD-009-OOP | Low | Unlikely | Medium | Low-2 | 3 |
| STD-010-CON | Medium | Probable | High | Low - 4 | 3 |

## 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 | This is dealing with any data that is not being accessed and is being stored on a physical or logical medium. This would deal with the standard for memory protection. The standard shows what is need to be done in order to secure the data properly. |
| Encryption at flight | This is dealing with any data that is being transported between systems. The first part that this deals with is the principle for satanizing data as this could be data coming in from other systems. |
| Encryption in use | This is dealing with any information that is be processed, accessed or read. This is important to the policy as it will need to ensure that nothing is changed where it shouldn’t be. This would be dealing with the principle of least amount of privilege to ensure that the user is not accessing the wrong type of information. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the process of recognizing a user’s identity. This is important to ensure that it is the proper user. If it is not the correct user, if it not the correct user than changes could be made, or there is an issue with the security. This follows the principle of default deny. This also would deal with the principal of validating input. |
| Authorization | This is the function of specifying access rights/privileges to resources as well as access control in general. This would follow the principle of least privilege. This is important to the security policy because it will ensure that the users can only access what needs to within the application. This will also help to prevent any changes made to the application that was not supposed to be made. |
| Accounting | This helps to figure out what resources were accessed at what time, by whom, and what commands were used. This helps to ensure that only the proper that the proper people are accessing the proper data. This will keep users from accessing information or data that they are not supposed to. If they are this means that something security wise has gone wrong, and needs to be looked into why they accessed something that was off limits to them. This follows the principle of practice defense in depth. |

**\***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 | 04/11/2021 | Fully executed Security Policy | Cinnamon George | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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