**ValidGreen Pace Developer: Security Policy Guide**



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 | To have valid inputs and making sure that values from external data sources are handled correctly. |
| 1. Heed Compiler Warnings | To use static and dynamic analysis tolls to detect and eliminate any flaws withing the system. Use the highest warning level available for the compiler and reduce warnings in your code as well. |
| 1. Architect and Design for Security Policies | Have an architecture and design that enforces security policies. Dividing the system into distinct subsystems with different privileges. |
| 1. Keep It Simple | Having a simple design reduces the likelihood of errors during implementation and configuration. Simpler systems also require less security mechanisms. |
| 1. Default Deny | Access is denied throughout, and the scheme identifies condition in which access is granted. |
| 1. Adhere to the Principle of Least Privilege | Letting the process execute with the minimum number of privileges needed. Any permission should only need for the time it takes to require a specific task. This reduces opportunities for attackers. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all the data that is passes to complex subsystems including command shells relational databases, and commercial off-the-shelf components. Sending secured data should always be taken seriously. |
| 1. Practice Defense in Depth | Have multiple layers of defense, more layers of defense can help a system defend from all types of attacks and lets them identify different security penetrations based on the layers. |
| 1. Use Effective Quality Assurance Techniques | Have multiple different types of testing including fuzz testing, penetration testing, and source code audits. Having independent reviews as well can lead to a more secure system. |
| 1. Adopt a Secure Coding Standard | Develop a standard for the development language or platform you are using. Helps better understand the different standards of the language and common errors that occur. |

## 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** | STD-031-INT | Data Type Conversion (INT)  Transitioning from one type of data value to another to assure no lost or misinterpreted data. |

| **Noncompliant Code** |
| --- |
| The code results in an error with the negative number misinterpreted as a positive number. |
| signed int u = -10;  unsigned int t = (unsigned int)u; //eliminates warning  printf(t); |

| **Compliant Code** |
| --- |
| The code results in no error and it also detects if there is a negative number |
| signed int u = - 10;  unsigned int t;  if (u < 0){  printf(“Error, negative integer”);  }  else{  t = (unsigned int) u;  printf(t);  } |

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

|  |
| --- |
| **Principles(s):** 1, making sure the data entered will be used for the variable type with no missing information. |

**Threat Level 4**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2017.07 | memsetValueOutOfRange | The second argument to memset () cannot be represented as unsigned char |
| Astree | 20.10 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| TrustInSoft Analyzer | 1.38 | Signed\_downcast | Exhaustively verified |
| Polyspace Bug Finder | R2021a | CERT C: Rule INT31-C | Checks for: integer conversion overflow, call to memset with unintended value, sign change integer conversion overflow, and more |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-032-INT | Fixed Width Data Value (INT)  To make sure the value in the data type does not exceed a maximum or minimum value causing an overflow or underflow. |

| **Noncompliant Code** |
| --- |
| The code results in an unsigned integer wrap during addition causing ui to result in a negative value. |
| Unsigned int ui = UINT\_MAX;  ui++;  printf(ui); |

| **Compliant Code** |
| --- |
| The code will result in an overflow, ui will become a negative number and |
| unsigned int ui = UINT\_MAX;  unsigned int t = 10;  if((UINT\_MAX – t)<ui){  printf(“Overflow occurred”);  break;  }  else{  printf(ui + t);  } |

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

|  |
| --- |
| **Principles(s):** 1, making sure that the variable is declared to the right type. |

**Threat Level 5**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially Implemented checks for overflow and underflow |
| Parasoft C/C++test | 2021.1 | CERT\_C-INT32-a, CERT\_C-INT32-b, CERT\_C-INT32-c | Avoid integer overflows |
| PolySspace Bug Finder | R2021a | CERT C: Rule INT32-C | Checks for: integer overflow, tainted division operand, tainted modulo operand |
| CodeSonar | 6.1p0 | ALLOC.SIZE.ADDOFLOW, ALLOC.SIZE.IOFLOW, ALLOC.SIZE.MULOFLOW, ALLOC.SIZE.SUBUFLOW, and more | Addition overflow of allocation size, integer overflow of allocation size, multiplication overflow if allocation size, subtraction underflow of allocation size, and more |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-031-STR | String Correctness Storage (STR)  Guarantee that there is sufficient space for strings character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| Due to the input being unbound, this could result in a buffer overflow. |
| void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| This code ensure data will not be truncated and it guards against buffer overflow. |
| void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

|  |
| --- |
| **Principles(s):** 1, making sure that the string will fit in the correct data type. |

**Threat Level 4**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CERTC-SERT31 | Detects calls to unsafe string function that may cause buffer overflow |
| CodeSonar | 6.1p0 | LANG.MEM.BO, LANG.MEM.TO, MISC.MEM.NTERM, BADFUNC.BO | Buffer overrun, type overrun, no space for null terminator, a collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Parasoft C/C++ test | 2021.1 | CERT\_C-STR31-a-e | Avoid accessing arrays out of bounds, avoid overflow when writing to a buffer |
| Polyspace Bug Finder | R2021a | CERT C: Rule STR31-C | 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, and tainted mull or non-null-terminated string |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-030-FIO | SQL Injection Prevention (FIO)  Preventing from unwanted users entering the system. |

| **Noncompliant Code** |
| --- |
| This protects against users who are trying to use a 1=1 but the user just has to type OR “”=”” and the results will always be true. |
| uName = getRequestString(“username”);  uPass = getRequestString(“userpassword”);  sql = ‘SELECT \* FROM Users WHERE Name = “ ‘ + UNAME + ‘” AND Pss = “ ‘ + uPass +’ “ ‘ |

| **Compliant Code** |
| --- |
| With this code the parameters are added to the query at the execution time in a controlled manner. |
| userID = getRequestString(“UserId”);  sql = “SELECT \* FROM Users WHERE UserID = @0”;  db.Execute(sql, userID); |

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

|  |
| --- |
| **Principles(s):** 6 and 8, making sure the user has the right privileges in the system, and having multiple security layers prevents against SQL injection. |

**Threat Level 4**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | IO.INJ.FMT, MISC.FMT | Format string injection, format string |
| GCC | 4.3.5 |  | Can detect violations of this rule when the -Wformat-security flag is used |
| Parasoft C/C++ test | 2021.1 | CERT\_C-FIO030-a-c | Avoid calling functions prontf/wprintf with only one argument other than string constant |
| PC-lint Plus | 1.4 | 592 | Partially supported: reports non-literal format strings |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-035-MEM | Memory Allocation Protection (MEM)  Preventing the memory leakage helps defend against potential hackers and leakage of data. |

| **Noncompliant Code** |
| --- |
| Due to the memory allocations are passed as arguments to a function call, the exception is thrown because of the new declarations, this could result in a memory leak. |
| struct A { };  struct B { };  void g(A \*, B \*);  void f(){  g(new A, new B);  } |

| **Compliant Code** |
| --- |
| This code removes the memory allocation issue entirely and passes the objects by reference instead. |
| struct A { };  struct B { };  void g(A &a, B &b);  void f(){  A a;  B b;  g(a, b);  } |

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

|  |
| --- |
| **Principles(s):** 7, making sure the data being transferred is encrypted so if memory is leaked it would not be harmful. |

**Threat Level 4**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Atree | 20.10 | Malloc-size-insufficient | Besides direct rule violations, all undefined behavior resulting from invalid memory accesses is reported by Astree |
| CodeSonar | 6.1p0 | ALLOC.SIZE.ADDOFLOW, ALLOC.SIZE.IOFLOW, ALLOC.SIZE.MULOFLOW, ALLOC.SIZE.SUBUFLOW, and more | Addition overflow of allocation size, integer overflow of allocation size, multiplication overflow if allocation size, subtraction underflow of allocation size, and more |
| Coverity | 2017.07 | BAD\_ALLOC\_STRLEN, SIZECHECK (deprecated) | Can find instances where string length is miscalculated for memory allocation purposes. |
| Parasoft C/C++ test | 2021.1 | CERT\_C-MEM35-a | Do no use sizeof operator on pointer type to specify the size of the memory to be allocated via malloc, calloc or realloc function |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-ERR | Assertion Evaluation (ERR)  Using assert correctly can help detect bugs and issues withing code much faster and will result in less time reviewing mistakes. |

| **Noncompliant Code** |
| --- |
| This is bad practice as assert is not evaluated in the release version so x will have different values in the debug and release. |
| Int x = 3;  ASSERT (x ==5) |

| **Compliant Code** |
| --- |
| This solves an issues with release and debug. |
| assert(x);  x++; |

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

|  |
| --- |
| **Principles(s):** 9, having multiple testing methods can prevent security vulnerabilities. |

**Threat Level 2**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  | Can detect some violations of this rule, but only abort() will be detected because assert() is a macro |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced enforcement |
| Parasoft C/C++ test | 2021.1 | CERT\_C-ERR06-a | Do not use assertions |
| PC-Init Plus | 1.4 | 586 | Fully supported |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-051-ERR | Handle Exceptions (ERR)  Throwing an exception will be caught by a matching handler, even if the exception cannot recover successfully the handler can use external resources to solve the problem. |

| **Noncompliant Code** |
| --- |
| The following code has an issues with the thread\_start() as it does not match exceptions thrown by thowing\_func(), once an exception is thrown the std::terminate() is called. |
| void throwing\_func() noexcept(false);  void thread\_start(){  throwing\_func();  }  void f() {  std::thread t(thread\_start);  t.join();  } |

| **Compliant Code** |
| --- |
| In this code the thread\_start() handles all exceptions and does not rethrow so the thread can terminate as designed. |
| void throwing\_func() noexcept(false);  void thread\_start(void){  try{  throwing\_func();  }catch(…){  //handle of error  }  }  void f() {  std::thread t(thread\_start);  t.join();  } |

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

|  |
| --- |
| **Principles(s):** 2, using warnings can help detect if the user is missing vital information that could cause issues if not addressed. |

**Threat Level 1**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Main-function-catch-all, early catch-all | Partially checked |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-ERR51-a | Always catch exceptions, each exception thrown in the code shall have a handler of a compatible type |
| Polyspace Bug Finder | R2021a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 20.10 | Main0function-catch-all, early-catch-all | Partially checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | STD-052-OOP | OOP Polymorphic Deletion (OOP)  Deleting an object through a pointer to a type results in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Due to the default deleter for std::unique\_ptr calls delete on the internal pointer value the result is underfined behavior. |
| struct Base{  virtual void f();  };  struct Derived : Base {};  void f() {  std::unique\_ptr<Base> b = std::make\_unique<Derived()>();  } |

| **Compliant Code** |
| --- |
| The destructor for the Base has declared virtual destructor ensuring that the polymorphic delete operation results correctly. |
| struct Base{  virtual ~Base() = default;  virtual void f();  };  struct Derived : Base {};  void f() {  Base \*b = new Derived();  //…  Delete b;  } |

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

|  |
| --- |
| **Principles(s):** 2, using warnings can help prevent against a misuse of data types and operations in the system. |

**Threat Level 1**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Non-virtual-public-destructor-in-non-final-class | Partially checked |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-OOP52-a | Define a virtual destructor in classes used as base classes which have virtual functions |
| Polyspace Bug Finder | R2021a | CERT C++: OOP52-CPP | Checks for situations when a class has virtual functions but not a virtual destructor |
| LDRA tool suite | 9.7.1 | 303 S | Partially implemented |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | STD-041-MSC | Sensitive Information (MSC)  Hard coding sensitive information exposes the information to hackers and anyone with access to the executables or dynamic library can accidently reveal sensitive information. |

| **Noncompliant Code** |
| --- |
| The code must authenticate to a remote service with code, this passes the code to a function as a string literal and this can be easily discovered in the programs binary executables. |
| int authentication(const char\* code);  int main(){  if (!authenticate(“correct code”)){  printf(“Authentication error\n”);  return -1;  }  Printf(“Authentication successful\n”);  //working with system  return 0;  } |

| **Compliant Code** |
| --- |
| This code securely erases the information once the user supplies the authentication code. |
| int authentication(const char\* code);  int main() {  char Code[CODE\_LEN];  printf(“Please enter your authentication code\n”);  fgets(code, sizeof(code), stdin);  int flag = authenticate(code);  memset\_s(code, 0, sizeof(code));  if (!flag){  printf(Access denied\n”);  return -1;  }  printf(“Access granted\n”);  //work with system  return 0;  } |

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

|  |
| --- |
| **Principles(s):** 7 and 8, having sensitive information encrypted so that it is not exposed to hackers as well as making sure there are multiple layers of defense before the encrypted information is accessed. |

**Threat Level 3**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | HARDCODED.AUTH, HARDCODE.DNS, HARDCODED.KEY, HARDCODED.SALT | Hardcoded authentication, DNS name, crypto key, and crypto salt |
| Parasoft C/C++ test | 2021.1 | CERT\_C-MSC41-a | Do no hard code string literals |
| PC-Init Plus | 1.4 | 2460 | Reports when a literal is provided as an argument to a function parameter with the noliteral argument semantic |
| Polyspace Bug Finder | R2021a | CERT C: Rule MSC41-C | Checks for hard coded sensitive data |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Arrays | STD-036-ARR | Array Pointer Subtraction (ARR)  When two pointers are subtracted, both must point to elements of the same array object or just one past the last element otherwise the operation results in undefined behavior. |

| **Noncompliant Code** |
| --- |
| The program incorrectly assumes that nums array is adjacent to end variable in memory, the compiler is permitted to insert padding bits between the two variables or even reorder the memory causing incorrect values. |
| enum {SIZE = 32};  void func(void){  int nums[SIZE];  int end;  int \*next\_num\_ptr = nums;  size\_t free\_elements;  //increment next\_num\_ptr as array fills  free\_elements = &end – next\_num\_ptr;  } |

| **Compliant Code** |
| --- |
| This computes the subtraction correctly. |
| enum {SIZE = 32};  void func(void){  int nums[SIZE];  int \*next\_num\_ptr = nums;  size\_t free\_elements;  //increment next\_num\_ptr as array fills  free\_elements = &(nums[SIZE]) – next\_num\_ptr;  } |

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

|  |
| --- |
| **Principles(s):** 2, using warnings can help prevent against a misuse of data types and operations in the system. |

**Threat Level 2**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ARR36 | Can detect operations on pointers that are unrelated |
| LDRA tool suite | 9.7.1 | 437 S, 438 S | Fully implemented |
| Parasoft C/C++ test | 2021.1 | CERT\_C-ARR36-a-b | Don not subtract two pointer that do not address elements of the same array, do not compare two unrelated pointers |
| TrustInSoft Analyzer | 1.38 | differing\_blocks | Exhaustively verified |

## 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.

Automation in programming helps develop code that meets standards and helps prevent against corruption of the program and its files. When to use automation tools in programming can help or hinder a project, implementing the tools too soon will only test a handful of information and potentially miss key elements while testing too late in the process will result in complete redo’s or an expensive and time-consuming fix. The best time to use the automation tools is during the test phase of the program as it will find errors caused by the developers or point out areas of future concern. This allows the developers to fix any issues without an expensive and extensive rework or the project.

## 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-031-INT | High | Probable | High | P6 | L2 |
| STD-032-INT | High | Likely | High | P9 | L2 |
| STD-031-STR | High | Likely | Medium | P18 | L1 |
| STD-030-FIO | High | Likely | Medium | P18 | L1 |
| STD-035-MEM | High | Probable | High | P6 | L2 |
| STD-006-ERR | Medium | Unlikely | Medium | P4 | L3 |
| STD-051-ERR | Low | Likely | Low | P9 | L2 |
| STD-052-OOP | Low | Likely | Low | P9 | L2 |
| STD-041-MSC | High | Probable | Medium | P12 | L1 |
| STD-036-ARR | Medium | Probable | Medium | P8 | L2 |

## Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption of data not being transferred, using an algorithm and a key they can easily encrypt the data and not have it be vulnerable. Having sensitive information encrypted is vital and when it is not being transferred or used it still needs to be secure. |
| Encryption at flight | Encryption of data that is being transmitted, using an algorithm and a key will encrypt and decipher the data. Sending secure information from one source to another is exposed through the internet and having an encryption on it makes sure that the information will not be leaked or taken during transmission. |
| Encryption in use | Encryption in use is taking plain text and scrambling it into an unreadable format. Using algorithms and kays allows the information to be encrypted and deciphered once delivered to the correct location. Protects confidentiality of data either stored on a system or transmitted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | To verify that the user accessing the system is who they say they are, having the correct password, using security questions, making sure the user is allowed to access the system and not allow unauthenticated users into the system. |
| Authorization | Determine the type of access the user has in a system. Making sure they have the right privileges, if the user is only needed for one portion, they shouldn’t have access to the whole system. Mandatory access control is used, the level of security a person is granted is related to the security of the content being accessed. |
| Accounting | Monitoring activity in the system, logging user activity with what files they are accessing or attempting to access can inform the type of authorization needed. This can also produce suspicious activities of user, using auditing and monitoring tools can help track and identify issues and unauthorized access. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

## Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

# Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

# Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

# Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

# Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

# Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

# Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 07/18/2021 | Security Principles and Coding Standard | Nicholas Wood |  |
| 1.2 | 08/07/2021 | Summary of Risk, Encryptions, and Principles | Nicholas Wood |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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