**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. Validate Input Data | Often considered the first level of defense for software security, validating input data helps ensure that correctly formatted data enters the system. This prevents malformed data to triggering system malfunctions. |
| 1. Heed Compiler Warnings | Compiler warnings inform developers of errors before the system is even up and running. These will inform the developer of bugs in the code and fixing them early can prevent a cascading set of errors. |
| 1. Architect and Design for Security Policies | It’s important to adhere and enforce the security policies that have been implemented. An example of this would be splitting your system into intercommunicating subsystems with different set privileges for each subsystem. |
| 1. Keep It Simple | Use as little code as needed and keep the software design simple. The more complex a system is the higher chance there are errors throughout the development process. |
| 1. Default Deny | Set the system up for access by permissions instead of exclusion. Access should be denied by default and the security identifies which access is allowed. |
| 1. Adhere to the Principle of Least Privilege | Principle of Least Privilege reduces risk of access by only giving permissions to users/actors that need it. This can also be enhanced by limiting the time allowed to access parts of the system. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data ensures that the data conforms to the requirements of the subsystem that it is being passed to. This ensures that the data conforms to the security requirements when being passed to different systems. |
| 1. Practice Defense in Depth | Make sure to manage risk with multiple defense strategies. Multiple layers of defense have a higher chance of catching errors and finding vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques can help identify and eliminate vulnerabilities. Multiple techniques, such as source code audits and penetration testing, should be used to catch any errors. Using external reviewers is also beneficial as they can provide an independent perspective. |
| 1. Adopt a Secure Coding Standard | Developing and adopting a secure coding standard as this helps ensure the security policy is being used by the whole company. It ensures that everyone working on a project is building to the same level of security. |

### 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-001-CPP] | Declarations and Initialization  Use proper data type declarations and be aware of conversions that could introduce errors. |

| **Noncompliant Code** |
| --- |
| Initializing incorrect data types or not declaring the data type. |
| x = 10; |

| **Compliant Code** |
| --- |
| Includes type specifier to identify the variables data type. |
| unsigned int x = 10; |

| **Principles(s):** 1. Validate Input Data  This standard makes sure that we are defining data, which fits principle 1 to make sure that correctly formatted data enters the system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading | Partially checked |
| Code Sonar | 8.1p0 | LANG.STRUCT.DEF.FDH LANG.STRUCT.DEF.ODH | Function defined in header file Object defined in header file |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| RuleChecker | 22.10 | type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure that integer conversions don’t result in lost data. |

| **Noncompliant Code** |
| --- |
| Range errors can occur when converting from an unsigned integer to a signed integer. |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Validate ranges when converting from unsigned to signed. |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  if (u\_a <= SCHAR\_MAX) {  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  } else {  /\* Handle error \*/  }  } |

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

| **Principles(s):** 1. Validate Input Data  This standard makes sure that we are converting data correctly, which fits principle 1 to make sure that correctly formatted data enters the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT  LANG.CAST.COERCE LANG.CAST.VALUE  ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value  Truncation of allocation size Truncation of size  Tainted buffer access |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Parasoft C/C++test | 2024.1 | CERT\_C-INT31-a CERT\_C-INT31-b CERT\_C-INT31-c CERT\_C-INT31-d CERT\_C-INT31-e CERT\_C-INT31-f CERT\_C-INT31-g CERT\_C-INT31-h CERT\_C-INT31-i CERT\_C-INT31-j CERT\_C-INT31-k CERT\_C-INT31-l CERT\_C-INT31-m CERT\_C-INT31-n CERT\_C-INT31-o CERT\_C-INT31-p | An expression of essentially Boolean type should always be used where an operand is interpreted as a Boolean value An operand of essentially Boolean type should not be used where an operand is interpreted as a numeric value An operand of essentially character type should not be used where an operand is interpreted as a numeric value An operand of essentially enum type should not be used in an arithmetic operation Shift and bitwise operations should not be performed on operands of essentially signed or enum type An operand of essentially signed or enum type should not be used as the right hand operand to the bitwise shifting operator An operand of essentially unsigned type should not be used as the operand to the unary minus operator The value of an expression shall not be assigned to an object with a narrower essential type The value of an expression shall not be assigned to an object of a different essential type category Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category The second and third operands of the ternary operator shall have the same essential type category The value of a composite expression shall not be assigned to an object with wider essential type If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type If a composite expression is used as one (second or third) operand of a conditional operator then the other operand shall not have wider essential type Avoid data loss when converting between integer types Avoid value change when converting between integer types |
| RuleChecker | 23.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee there is sufficient space for string storage. |

| **Noncompliant Code** |
| --- |
| Unbound input could lead to buffer overflow. |
| #include <iostream>  void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Using <string> instead of a bounded array ensures data is not truncated. |
| #include <iostream>  #include <string>    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. Validate Input Data  This standard makes sure that we aren’t causing overflow errors, which fits principle 1 to make sure that correctly formatted data enters the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | 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 |
| Klocwerk | 2024.1 | SV.FMT\_STR.BAD\_SCAN\_FORMAT SV.UNBOUND\_STRING\_INPUT.FUNC |  |
| Parasoft C/C++test | 2023.1 | CERT\_C-STR31-a CERT\_C-STR31-b CERT\_C-STR31-c CERT\_C-STR31-d CERT\_C-STR31-e | Avoid accessing arrays out of bounds Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Avoid using unsafe string functions which may cause buffer overflows |
| TrustInSoft Analyzer | 1.38 | mem\_access | Exhaustively verified |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL Injection by sanitizing and validating queries. |

| **Noncompliant Code** |
| --- |
| No precautions allow for SQL Injection strings to cause errors in the query. |
| uName = getRequestString("username");  uPass = getRequestString("userpassword");    sql = “SELECT \* FROM Users WHERE Name = " + uName + " AND Pass = " +  uPass + ” |

| **Compliant Code** |
| --- |
| Validating input can catch errors that are part of a SQL Injection. |
| PreparedStatement pStmt = PreparedStatement();    std::cin >> username;  std::cin >> userpassword;    sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, username,  userpassword}; |

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

| **Principles(s):** 1. Validate Input Data, 7. Sanitize Data Sent to Other Systems  Prevent SQL injection by validating the incoming/sanitizing the data to make sure it hasn’t been tampered with and fits the requirements for incoming data into the system |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Coverity | 7.5 | SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_ FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SpotBugs | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Free dynamically allocated memory when no longer needed |

| **Noncompliant Code** |
| --- |
| Object allocation is not freed before end of lifetime of the pointer referring to the object. |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }  return 0;  } |

| **Compliant Code** |
| --- |
| Pointer is deallocated with a call to free() |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }    free(text\_buffer);  return 0;  } |

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

| **Principles(s):** 1. Validate Input Data  This standard makes sure that we are freeing up data, which fits principle 1 to make sure that correctly formatted data enters the system to prevent overflows. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Cppcheck | 1.66 | leakReturnValNotUsed | Doesn't use return value of memory allocation function |
| Parasoft C/C++test | 2023.1 | CERT\_C-MEM31-a | Ensure resources are freed |
| PC-lint Plus | 1.4 | 429 | Fully supported |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Don’t leave assertions in production code. |

| **Noncompliant Code** |
| --- |
| Assertions can be used to inject malicious code. Assert is left in. |
| void f()  {  std::vector<int> myVec\* = std::vector<int>(5);  assert(myVec != nullptr);  myVec.push\_back(1);  } |

| **Compliant Code** |
| --- |
| Removing assertions prevents later injections. |
| void f()  {  std::vector<int> myVec\* = std::vector<int>(5);  myVec.push\_back(1);  } |

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

| **Principles(s):** 9. Use Effective Quality Assurance Techniques  Make sure to removed assertions to create high quality code that is up to standard and expectations. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 6.7 | S3346 | Expressions used in "assert" should not produce side effects |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Use exceptions to catch errors. |

| **Noncompliant Code** |
| --- |
| No exception handler can lead to missed issues which terminates the program. |
| void raise\_exception() noexcept(false);    void target() {  raise\_exception();  }    int main() {  target();  } |

| **Compliant Code** |
| --- |
| Using an exception handler catches the error. |
| void raise\_exception() noexcept(false);    void target() {  raise\_exception();  }    int main() {  try {  target();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):** 2. Heed compiler warnings  Make sure to use exceptions to help find issues and catch errors before they happen. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | main-function-catch-all early-catch-all | Partially checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Polyspace Bug Finder | R2023b | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Management** | [STD-008-CPP] | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| Access resulting in write-after-free can lead to a vulnerability that can be exploited to run arbitrary code with correct permissions. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| Dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** 1. Validate Input Data  This standard makes sure that we are freeing up data, which fits principle 1 to make sure that correctly formatted data enters the system to prevent overflows. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | 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 | 2023.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| PVS-Studio | [Insert text.] | 7.30 |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Pointers** | [STD-009-CPP] | Do not attempt to create a string from a null pointer |

| **Noncompliant Code** |
| --- |
| Creating a string by calling a function that returns a null pointer can lead to undefined behavior. |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| Check for null before the string object is constructed. |
| #include <cstdlib>  #include <string>    void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** 1. Validate Input Data  This standard makes sure that we aren’t causing null errors, which fits principle 1 to make sure that correctly formatted data enters the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | assert\_failure |  |
| CodeSonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023b | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Integer Functions** | [STD-010-CPP] | Ensure that division and remainder operations do not result in divide-by-zero errors. |

| **Noncompliant Code** |
| --- |
| Preventing overflow but not preventing divide-by-zero errors during division can cause errors. |
| #include <limits.h>    void func(signed long s\_a, signed long s\_b) {  signed long result;  if ((s\_a == LONG\_MIN) && (s\_b == -1)) {  /\* Handle error \*/  } else {  result = s\_a / s\_b;  }  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Test division operation to guarantee no divide-by-zero errors or signed overflow. |
| #include <limits.h>    void func(signed long s\_a, signed long s\_b) {  signed long result;  if ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {  /\* Handle error \*/  } else {  result = s\_a / s\_b;  }  /\* ... \*/  } |

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

| **Principles(s):** 1. Validate Input Data  This standard makes sure that we aren’t causing division errors, which fits principle 1 to make sure that correctly formatted data enters the system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | int-division-by-zero  int-modulo-by-zero | Fully checked |
| CodeSonar | 8.1p0 | LANG.ARITH.DIVZERO LANG.ARITH.FDIVZERO | Division by zero Float Division By Zero |
| Cppcheck | 1.66 | zerodiv zerodivcond | Context sensitive analysis of division by zero Not detected for division by struct member / array element / pointer data that is 0 Detected when there is unsafe division by variable before/after test if variable is zero |
| Parasoft C/C++test | 2023.1 | CERT\_C-INT33-a | Avoid division by zero |

### 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.

DevOps is a successful development strategy for applications, but the development of DevSecOps increases the security of the applications being built and maintained by incorporating security measures in the process. Using automation we can add in security checks throughout the development process to help prevent security breaches in the software that could be detrimental to the company and its clients. The left half of the diagram will include checking on potential security threats that could affect the software before production has even begun, including deciding what tools to use during automation that will catch errors as the development process begins. Then during initial testing, track any discovered security issues that can be addressed in the production phase (the right side of the figure) and find ways to fix those security issues. Many of the tools listed in the risk assessments standards above can help with this automation.

Once this automation is secured, it is important to also add security measures using defense in depth practices. This includes checking access points for the software or system and have a way to track who, what, where, and when part of the system is accessed, training employees in security measures, keeping updated with standards set in the industry, along with physical security measures at the company like security footage and employee id badges. Incorporating these security measures will make sure the whole system, from company to product, is secure.

### 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 | P3 | L3 |
| STD-002-CPP | High | Probable | High | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-006-CPP | Low | Unlikely | Low | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Likely | Medium | P18 | L1 |
| STD-009-CPP | High | Likely | Medium | P18 | L1 |
| STD-010-CPP | Low | Likely | Medium | P6 | 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 at rest | Encryption at rest is an encryption strategy used to secure data being stored, be it on a server or hard disk. This encryption is important as it protects data if a database is accessed without authorization. Encryption at rest generally requires brute force to get into, which is difficult and time consuming for hackers. |
| Encryption in flight | Encryption in flight protects data as it is being transferred between systems. Encrypting data like emails and using strategies like firewalls or a VPN and authentication can be used to protect data in flight. |
| Encryption in use | Encryption in use is an encryption strategy to protect data while it is being accessed. This type of encryption/decryption can be done in real time and only from the defined region of memory that it exists in, preventing the data from being used elsewhere. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a way to check that a user is meant to be able to access information. Each employee should have a login so that someone who isn’t an employee doesn’t have access to the system. User login with a secure password can help prevent unauthorized access as well as incorporating measures like two-factor-authentication. There should be a database that keeps track of all usernames and passwords and allows for the creation of new users by people who have higher authorization. |
| Authorization | Authorization checks to make sure that, once a user logs in with correct authentication, they are allowed access to certain areas of the system. This includes principle of least privilege which makes sure that employees are only able to access the parts of the system that they need in order to do their jobs. |
| Accounting | Accounting is making sure that there is a log of who accessed which files at what time and from what location. This can help track any issues that occur so security issues can be pinpointed quickly and fixed accordingly. |

**\***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 | 3/16/2024 | Milestone | Jonathan Way | David Buksbaum |
| 1.2 | 4/14/2024 | Project 1 | Jonathan Way | David Buksbaum |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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