**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 7](#_Toc52464061)

[Coding Standard 4 8](#_Toc52464062)

[Coding Standard 5 10](#_Toc52464063)

[Coding Standard 6 12](#_Toc52464064)

[Coding Standard 7 14](#_Toc52464065)

[Coding Standard 8 16](#_Toc52464066)

[Coding Standard 9 18](#_Toc52464067)

[Coding Standard 10 20](#_Toc52464068)

[Defense-in-Depth Illustration 22](#_Toc52464069)

[Project One 22](#_Toc52464070)

[1. Revise the C/C++ Standards 22](#_Toc52464071)

[2. Risk Assessment 22](#_Toc52464072)

[3. Automated Detection 22](#_Toc52464073)

[4. Automation 22](#_Toc52464074)

[5. Summary of Risk Assessments 23](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 23](#_Toc52464076)

[7. Map the Principles 24](#_Toc52464077)

[Audit Controls and Management 25](#_Toc52464078)

[Enforcement 25](#_Toc52464079)

[Exceptions Process 25](#_Toc52464080)

[Distribution 26](#_Toc52464081)

[Policy Change Control 26](#_Toc52464082)

[Policy Version History 26](#_Toc52464083)

[Appendix A Lookups 26](#_Toc52464084)

[Approved C/C++ Language Acronyms 26](#_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 | Validate input from all untrusted data sources. Proper input validation can eliminate most software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code [C MSC00-A, C++ MSC00-A]. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set. |
| 1. Keep It Simple | Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied, and the protection scheme identifies conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker must execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components using SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

## 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] | Make sure you declare identifiers before use. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the variable is not assigned a specifier. |
| extern foo; |

| **Compliant Code** |
| --- |
| In this compliant code, the variable is assigned a specifier. |
| extern int foo; |

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

|  |
| --- |
| **Principles(s):** Heed compiler warnings as not utilizing a type in this case would cause issues at the compilation level. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | type-specifier  function-return-type  implicit-function-declaration  undeclared-parameter | Fully checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC-DCL31 | Fully implemented |
| Clang | 3.9 | -Wimplicit-int |  |
| Coverity | 2017.07 | MISRA C 2012 Rule 8.1 | Implemented |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Make sure that signed integers do not overflow. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the combination of sign A and sign B are not checked to accumulate to be over the maximum number for the data type. |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  } |

| **Compliant Code** |
| --- |
| In this compliant code, the max is checked with an if statement before the sum is calculate which prevents a potential error and allows the developer to handle it accordingly. |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  } |

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

|  |
| --- |
| **Principles(s):** Heed compiler warnings as the compiler would handle the overflow manually if an error occurs. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tol** |
| --- | --- | --- | --- |
| Astrée | 20.10 | integer-overflow | Fully checked |
| 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 |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Helix QAC | 2021.1 | C2800, C2801, C2802, C2803, C2860, C2861, C2862, C2863  C++2800, C++2801, C++2802, C++2803, C++2860, C++2861, C++2862, C++2863 |  |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Don’t attempt to modify a string literal. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, a string literal is initialized to a specific address through a reference. |
| char \*str = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| In this compliant code, a string is initialized as a character array and thus can be manipulated using a specific address. |
| char str[] = "string literal";  str[0] = 'S'; |

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

|  |
| --- |
| **Principles(s):** Heed compiler warnings as the compiler would definitely throw a fit over attempting to change a specific element of a string pointer. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | string-literal-modfication  write-to-string-literal | Fully checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC-STR30 | Fully implemented |
| Compass/ROSE |  |  | Can detect simple violations of this rule |
| Coverity | 2017.07 | PW | Deprecates conversion from a string literal to "char \*" |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C] | Prevent SQL Injection. |

| **Noncompliant Code** |
| --- |
| Data sanitization requires an understanding of the data being passed and the capabilities of the subsystem. John Viega and Matt Messier provide an example of an application that inputs an email address to a buffer and then uses this string as an argument in a call to system() |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| It is necessary to ensure that all valid data is accepted, while potentially dangerous data is rejected or sanitized. Doing so can be difficult when valid characters or sequences of characters also have special meaning to the subsystem and may involve validating the data against a grammar. In cases where there is no overlap, whitelisting can be used to eliminate dangerous characters from the data. |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {  \*cp = '\_';  } |

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

|  |
| --- |
| **Principles(s):** Sanitize data as the issue is that the data can input SQL commands. Sanitization combats that directly. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 |  | Supported by stubbing/taint analysis |
| CodeSonar | 6.0p0 | IO.INJ.COMMAND  IO.INJ.FMT  IO.INJ.LDAP  IO.INJ.LIB  IO.INJ.SQL  IO.UT.LIB  IO.UT.PROC | Command injection  Format string injection  LDAP injection  Library injection  SQL injection  Untrusted Library Load  Untrusted Process Creation |
| Coverity | 6.5 | TAINTED\_STRING | Fully implemented |
| Klocwork | 2021.1 | NNTS.TAINTED  SV.TAINTED.INJECTION |  |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not leak resources when handling exceptions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pst is not properly released when process\_item throws an exception, causing a resource leak. |
| #include <new>    struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };    void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }    try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  throw;  }  delete pst;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the exception handler frees pst by calling delete. |
| #include <new>    struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.    void process\_item() noexcept(false);  };    void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }  try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  delete pst;  throw;  }  delete pst;  } |

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

|  |
| --- |
| **Principles(s):** Adopt a secure coding standard as this is language dependent. Memory leaks are avoidable and wreak havoc on systems. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2021.1 |  |  |
| LDRA tool suite | 9.7.1 | 50 D | Partially implemented |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-ERR57-a | Ensure resources are freed |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C] | Use static assertions to test constant expression’s values |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the developer uses assert as a means of testing a constant expression. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| In this compliant code, the developer uses static\_assert as a means of testing a constant expression. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),  "Structure must not have any padding"); |

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

|  |
| --- |
| **Principles(s):** Heed compiler warnings as above doesn’t utilize proper static assertion. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CertC-DCL03 |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.0p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Honor exception specifications. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the noexcept conflicts with the fact that there may not be enough memory to perform the statement. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) noexcept(true) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| In this compliant code, the noexcept is removed to allow for exceptions. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) {  v.resize(s); // May throw, but that is okay  } |

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

|  |
| --- |
| **Principles(s):** Use effective quality assurance as you want to throw the error to test the system. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | unhandled-throw-noexcept | Partially checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-ERR55 |  |
| Helix QAC | 2021.1 |  |  |
| LDRA tool suite | 9.7.1 | 56 D | Partially implemented |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008-CPP] | Don’t delete a polymorphic object without a virtual destructor. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, base is has no access to a virtual destructor, thus results in unpredictable behavior. |
| struct Base {  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

| **Compliant Code** |
| --- |
| In this compliant code, a virtual destructor is defined and thus the delete b; statement within f() acts as expected. |
| 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):** Heed compiler warnings as the unpredictable behavior would be accounted for within the noncompliant code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | non-virtual-public-destructor-in-non-final-class | Partially checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-OOP52 |  |
| Helix QAC | 2021.1 |  |  |
| Klocwork | 2021.1 | CL.MLK.VIRTUAL  CWARN.DTOR.NONVIRT.DELETE |  |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory | [STD-009-CLG] | Do not read from uninitialized memory. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the integer is not initialized with a zero thus can lend itself to unpredictable behavior. |
| #include <iostream>    void f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| In this compliant code, the integer is correctly initialized with a zero. |
| #include <iostream>    void f() {  **int** i = 0;    std::cout << i;  } |

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

|  |
| --- |
| **Principles(s):** Keep it simple and always initialize variables before use. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | uninitialized-local-read  uninitialized-variable-use | Fully checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC-EXP33 |  |
| CodeSonar | 6.0p0 | LANG.MEM.UVAR | Uninitialized variable |
| Compass/ROSE |  |  | Automatically detects simple violations of this rule, although it may return some false positives. It may not catch more complex violations, such as initialization within functions taking uninitialized variables as arguments. It does catch the second noncompliant code example, and can be extended to catch the first as well |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-C] | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| A common, but noncompliant, practice is to choose a reserved name for a macro used in a preprocessor conditional guarding against multiple inclusions of a header file. (See also PRE06-C. Enclose header files in an include guard.) The name may clash with reserved names defined by the implementation of the C standard library in its headers or with reserved names implicitly predefined by the compiler even when no C standard library header is included. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    /\* Contents of <my\_header.h> \*/    #endif /\* \_MY\_HEADER\_H\_ \*/ |

| **Compliant Code** |
| --- |
| This compliant solution avoids using leading underscores in the macro name of the include guard. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    /\* Contents of <my\_header.h> \*/    #endif /\* MY\_HEADER\_H \*/ |

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

|  |
| --- |
| **Principles(s):** Keep it simple with consistent naming schema that does not tread into reserved identifiers. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | future-library-use  language-override  language-override-c99  reserved-declaration  reserved-declaration-c99  reserved-identifier | Partially checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC-DCL37 | Fully implemented. Reserved identifiers, as in DCL37-C-EX3, are configurable. |
| CodeSonar | 6.0p0 | LANG.STRUCT.DECL.RESERVED | Declaration of reserved name |
| Coverity | 2017.07 |  |  |

## 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.

Within the diagram above, development is done on the left portion with the assess and plan, design, build and verify cycle. On the left portion, operations are being performed such as transition, monitor, respond and maintain. Throughout the entirety of the diagram, security is being integrated to ensure the health of the system and a constant focus on actively tacking security issues.

## 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 | Low | Unlikely | Low | Medium | 3 |
| STD-002-CPP | High | Likely | High | High | 2 |
| STD-003-CPP | Low | Likely | Low | Low | 2 |
| STD-004-C | High | Likely | Medium | High | 1 |
| STD-005-CPP | Low | Probable | High | Low | 3 |
| STD-006-C | Low | Unlikely | High | Medium | 2 |
| STD-007-CPP | Low | Likely | Low | High | 2 |
| STD-008-CPP | Low | Likely | Low | Medium | 2 |
| STD-009-C | High | Probable | Medium | Medium | 1 |
| STD-010-C | Low | Unlikely | Low | Low | 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 | Focuses on preventing on disc data compromising scenarios through encryption. |
| Encryption at flight | Focuses on preventing data comprise scenarios while the data is being transferred through encryption. |
| Encryption in use | Focuses on preventing data comprise scenarios while the data is in memory through encryption. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication focuses on ensuring a user is who they say they are through valid credentials. This is important because you want to be able to identify what an individual should have access to. |
| Authorization | Authorization focuses on whether an individual has access to what they should have access to. This is important because you wouldn’t want a regular user to have administrative abilities for a website as they could manipulate the backend data. |
| Accounting | Accounting is important as a line of defense to account for what a user might be doing. In case of failure in the previous categories, a log consisting of what users do would inform an administrator of a user utilizing privileges they ought not have. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [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 |