**Green Pace Developer: Security Policy Guide Template**



Green Pace Secure Development Policy

Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

# Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

# Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

# Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

# Module Three Milestone

## Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validate input data means to verify and confirm input from all untrusted sources. Proper input validation can eliminate the vast majority of 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 | Heed Complier Warnings means to compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Architect and Design for Security Policies is creating 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 It Simple means to 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 | Default Deny is to 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 | Adhere to the Principle of Least Privilege means 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 has to execute arbitrary code with elevated privileges |
| 1. Sanitize Data Sent to Other Systems | Sanitize Data Sent to Other Systems means to 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 through the use of 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 | Practicing Defense in Depth means to 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 | The Use of 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 | Adopt a Secure Coding Standard means to develop and/or apply a secure coding standard for your target development language and platform. |

## Reference:

<https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices>

## C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Ensure that unsigned integer operations do not wrap |

| **Noncompliant Code** |
| --- |
| This code can lead to a vulnerability due to the allocation of insufficient memory from the unsigned integer wrap. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| To prevent an unsigned wrap, perform a precondition test of the operands of the addition. |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum;  if (UINT\_MAX - ui\_a < ui\_b) {  /\* Handle error \*/  } else {  usum = ui\_a + ui\_b;  }  /\* ... \*/  } |

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

|  |
| --- |
| **Principles(s):** INT30-C Integer wrap can lead to buffer overflows and the execution of arbitrary code by an attacker. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | integer-overflow | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT30 | Implemented |
| CodeSonar | 6.1p0 | ALLOC.SIZE.IOFLOW | Integer overflow of allocation size |
| Coverity | 2017.07 | INTEGER\_OVERFLOW | Implemented |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Guarantees that container indices and iterators are within the valid range |

| **Noncompliant Code** |
| --- |
| This example shows an insert function that has two int’s but this data can be influenced by sources whom are untrusted. The function does not check the lower bound which can result in a write outside of the bounds of the memory that was referenced by table |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

| **Compliant Code** |
| --- |
| Negative arguments are not passed due to the parameter pos being declared as size\_t |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

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

|  |
| --- |
| **Principles(s):** CTR50-CPP Using an invalid array or container index can result in an arbitrary memory overwrite or abnormal program termination. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | LANG.MEM.BO | Buffer overrun |
| LDRA tool suite | 9.7.1 | 45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X | Partially implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2021a | CERT C++: CTR50-CPP | Checks for: Array access out of bounds. Array access with tainted index. Pointer dereference with tainted offset. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| A buffer overflow could occur due to an unbounded input |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The practice of using a std::string instead of a bounded array guards against buffer overflows |
| #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):** STR50-CPP A buffer overflow can result if copied string data to a buffer that isn’t large enough to hold the data. Attackers can use this vulnerability to execute arbitrary code |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | LANG.MEM.BO | Buffer overrun |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-STR50-c | Avoid overflow when writing to a buffer |
| Polyspace Bug Finder | R2021a | CERT C++: STR50-CPP | Checks for:  Use of dangerous standard function  Missing null in string array  Buffer overflow from incorrect string format specifier  Destination buffer overflow in string manipulation |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| An undefined behavior occurs due to delete attempting to free memory that wasn’t returned from new |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    delete s1;  } |

| **Compliant Code** |
| --- |
| In this example the call to ::operator delete() is removed, instead explicitly calling s1's destructor. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    s1->~S();  } |

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

|  |
| --- |
| **Principles(s):** MEM51-CPP Passing a pointer value to a deallocation function that was not previously obtained by the matching allocation function results in undefined behavior, which can lead to exploitable vulnerabilities. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | ALLOC.TM | Type mismatch |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-MEM51-b | Always provide empty brackets ([]) for delete when deallocating arrays |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Not fully implemented |
| Polyspace Bug Finder | R2021a | CERT C++: MEM51-CPP | Checks for:  Invalid deletion of pointer  Invalid free of pointer  Deallocation of previously deallocated pointer |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| An undefined behavior is produced due to ::operator delete() attempting to free memory that was not returned by ::operator new(). |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    delete s1;  } |

| **Compliant Code** |
| --- |
| This code is in compliance because it removes the call to ::operator delete() |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    s1->~S();  } |

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

|  |
| --- |
| **Principles(s):** MEM51-CPP An undefined behavior is produced due to passing a pointer value to a deallocation function that was not previously obtained by the matching allocation function |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 6.1p0 | ALLOC.TM | Type mismatch |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-MEM51-b | Always provide empty brackets ([]) for delete when deallocating arrays |
| Polyspace Bug Finder | R2021a | CERT C++: MEM51-CPP | Checks for:  Invalid deletion of pointer  Invalid free of pointer  Deallocation of previously deallocated pointer |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Do not declare more than one variable per declaration |

| **Noncompliant Code** |
| --- |
| This code is confusing as one might think both src and c are declared as char |
| char \*src = 0, c = 0; |

| **Compliant Code** |
| --- |
| Instead of being declared on a single line as above each variable is declared on its own line |
| char \*src; /\* Source string \*/  char c; /\* Character being tested \*/ |

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

|  |
| --- |
| **Principles(s):** DCL04-C Declaring no more than one variable per declaration can make code easier to read and eliminate confusion. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | LANG.STRUCT.DECL.ML | Multiple Declarations on Line |
| ECLAIR | 1.2 | CC2.DCL04 | Fully implemented |
| Parasoft C/C++test | 2021.1 | CERT\_C-DCL04-a | Each variable should be declared in a separate declaration statement |
| PC-lint Plus | 1.4 | 9146 | Partially supported: exceptions not supported |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this example of code neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| All exceptions are handled by the main entry point which ensures the the stack is unwound up to the main() function |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

|  |
| --- |
| **Principles(s):** ERR51-CPP If the application is terminated abnormally it could lead to resources not freed up or closed. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | main-function-catch-all | Partially checked |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-ERR51-b | Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| Polyspace Bug Finder | R2021a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | STD-008-CPP | Use valid references, pointers, and iterators to reference elements of a container |

| **Noncompliant Code** |
| --- |
| After the first call to insert() pos is invalidated |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| This code is compliant because pos is assigned a valid iterator on each insertion, which prevents undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

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

|  |
| --- |
| **Principles(s):** CTR51-CPP Undefined behavior is the result of using invalid references, pointers, or iterators to reference elements of a container. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | overflow\_upon\_dereference | Overflow |
| Helix QAC | 2021.2 | C++4746, C++4747, C++4748, C++4749 |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-CTR51-a | Do not modify container while iterating over it |
| PVS-Studio | 7.13 | V783 |  |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| OOP | STD-009-CPP | Honor replacement handler requirements |

| **Noncompliant Code** |
| --- |
| An infinite loop will occur with the default implementation of ::operator new() during low memory conditions |
| #include <new>    void custom\_new\_handler() {  // Returns number of bytes freed.  extern std::size\_t reclaim\_resources();  reclaim\_resources();  }    int main() {  std::set\_new\_handler(custom\_new\_handler);    // ...  } |

| **Compliant Code** |
| --- |
| If custom\_new\_handler() returns 0 there won’t be enough memory for operator new to to succeed throwing an exception as it should. |
| #include <new>    void custom\_new\_handler() noexcept(false) {  // Returns number of bytes freed.  extern std::size\_t reclaim\_resources();  if (0 == reclaim\_resources()) {  throw std::bad\_alloc();  }  }    int main() {  std::set\_new\_handler(custom\_new\_handler);    // ...  } |

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

|  |
| --- |
| **Principles(s):** OOP56-CPP Failing to meet the required behavior for a replacement handler results in undefined behavior. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++4776, C++4777, C++4778, C++4779 |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-OOP56-a | Properly define terminate handlers |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-OOP56-b | Properly define unexpected handlers |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-OOP56-c | Properly define new handlers |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | STD-010-CPP | Ensure actively held locks are released on exceptional conditions |

| **Noncompliant Code** |
| --- |
| If an exception should occur while manipulating the shared date the mutex will remain locked. |
| #include <mutex>    void manipulate\_shared\_data(std::mutex &pm) {  pm.lock();    // Perform work on shared data.    pm.unlock();  } |

| **Compliant Code** |
| --- |
| In this example the code catches any exceptions thrown when performing work on the shared data and unlocks the mutex before rethrowing the exception. |
| #include <mutex>    void manipulate\_shared\_data(std::mutex &pm) {  pm.lock();  try {  // Perform work on shared data.  } catch (...) {  pm.unlock();  throw;  }  pm.unlock(); // in case no exceptions occur  } |

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

|  |
| --- |
| **Principles(s):** CON51-CPP If an exception occurs while a mutex is locked, deadlock may result. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++5018 |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-CON51-a | Do not call lock() directly on a mutex |
| PRQA QA-C++ | 4.4 | 5018 |  |

## 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.

When looking at the current DevOps process my recommendation on where to modify the process to automate enforcement of the standards in this policy would be in the design phase of the loop. Doing this allows code to be tested and verified as it’s being written instead of vulnerabilities being caught two steps down in the verify and test stage.

## 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 | Likely | High | P9 | L2 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Probable | High | P6 | L2 |
| STD-009-CPP | Low | Probable | High | P2 | L3 |
| STD-010-CPP | Low | Probable | Low | 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 in rest | Encryption in rest refers to encrypted data that is being stored in a database or on a device such as a computer hard drive, usb drive or another device. This data is not currently being accessed or transferred. The policy should be used because this data is still susceptible to un-authorized users and having it encrypted keeps attackers from successfully compromising it and the owner of the data having a data breach even if the data is stolen. |
| Encryption at flight | Encryption at flight refers to protecting your data while its being transferred. Its possible data can be intercepted by attackers while being sent through email or other file transfer utilities. In encryption at flight data is encrypted before being sent and the end user’s computer is authenticated and then the data is decrypted upon arrival. |
| Encryption in use | Encryption is use is the ability run your computation on encrypted data or run an encrypted application. This can be performed either with software or hardware. An example of this is password verification and its protection via hashing. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of validating the identity of a user or a device to confirm that they have access to a system. This is like asking “who are you.” This can be accomplished via user login prompts or other forms of authentication like receiving a specific access code on an unrelated device. This should be used because it keeps just anyone from accessing data |
| Authorization | Authorization is the process of allowing users access to certain bits of information and not just opening up the entire system to the user. The system is basically asking “what are you allowed to access” This is performed by access levels or rules set by an administrator |
| Accounting | Accounting is the tracking of resources used, when they were used, who accessed them, and any commands that were issued. This is important to let auditors know if data was accessed when it shouldn’t have been or if a data breach occurred. This also helps to see if a user is trying to access data when they shouldn’t be. |

**\***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 | 8/8/2021 | Completed principles and coding standards | Corey Roth |  |
| 1.2 | 8/10/2021 | Completed rest of template | Corey Roth |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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