**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_heading=h.gjdgxs)

[Purpose 2](#_heading=h.30j0zll)

[Scope 2](#_heading=h.1fob9te)

[Module Three Milestone 2](#_heading=h.3znysh7)

[Ten Core Security Principles 2](#_heading=h.4d34og8)

[C/C++ Ten Coding Standards 3](#_heading=h.2s8eyo1)

[Coding Standard 1 4](#_heading=h.3rdcrjn)

[Coding Standard 2 5](#_heading=h.26in1rg)

[Coding Standard 3 6](#_heading=h.lnxbz9)

[Coding Standard 4 7](#_heading=h.35nkun2)

[Coding Standard 5 8](#_heading=h.1ksv4uv)

[Coding Standard 6 9](#_heading=h.2jxsxqh)

[Coding Standard 7 10](#_heading=h.z337ya)

[Coding Standard 8 11](#_heading=h.28h4qwu)

[Coding Standard 9 13](#_heading=h.nmf14n)

[Coding Standard 10 14](#_heading=h.37m2jsg)

[Defense-in-Depth Illustration 15](#_heading=h.1mrcu09)

[Project One 15](#_heading=h.46r0co2)

[1.](#_heading=h.2lwamvv) Revise the C/C++ Standards 15

[2.](#_heading=h.111kx3o) Risk Assessment 15

[3.](#_heading=h.3l18frh) Automated Detection 15

[4.](#_heading=h.206ipza) Automation 15

[5.](#_heading=h.4k668n3) Summary of Risk Assessments 16

[6.](#_heading=h.2zbgiuw) Create Policies for Encryption and Triple A 16

[7.](#_heading=h.1egqt2p) Map the Principles 17

[Audit Controls and Management 18](#_heading=h.3ygebqi)

[Enforcement 18](#_heading=h.2dlolyb)

[Exceptions Process 18](#_heading=h.sqyw64)

[Distribution 19](#_heading=h.3cqmetx)

[Policy Change Control 19](#_heading=h.1rvwp1q)

[Policy Version History 19](#_heading=h.4bvk7pj)

[Appendix A Lookups 19](#_heading=h.2r0uhxc)

[Approved C/C++ Language Acronyms 19](#_heading=h.1664s55)

## 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 | Input validation is the technique of properly testing any input supplied by a user or application in order to resist improperly formed data from entering an information system. It helps prevent malicious attempts to confuse a system, such as SQL injections or buffer overflows. Input validation also requires proper exception handling to ensure that the system can safely mitigate these threats. |
| 1. Heed Compiler Warnings | Even though a warning is not an error, it should be treated as such. Avenues of attack can come in many unexpected ways. If a vulnerability is as easily detected by a software’s compiler, it can just as easily be detected by a hacker. Benign warnings may not break your code, but it can spark ideas for how a user might. |
| 1. Architect and Design for Security Policies | By establishing a strong foundation for the development of a system’s framework, you can prevent problems before they happen. When code becomes intricate and verbose, the process of identifying bugs and potential weaknesses can become a tedious endeavor. By defining what principles and techniques will be implemented in a software’s design, developers can focus on the system’s security throughout the development process. |
| 1. Keep It Simple | There is beauty in simplicity. By ensuring code is not overly complex, tasks such as maintenance or debugging become significantly less arduous. The more intricate software becomes, the more potential vectors of attack are available for hackers. Simplicity can also imply efficiency. |
| 1. Default Deny | If a system admin really needs to access sensitive information, they will have a way. It is far safer to assume that a user does not have certain privileges, rather than vice versa. Under no circumstances should a system risk offering privileges to users who are trying to access features which require privileges in the first place. |
| 1. Adhere to the Principle of Least Privilege | Hacking into a system doesn’t always happen from a back door. If a hacker acquires admin credentials, the system will never know any different. It is imperative that the hierarchy of access is as minimal as possible to prevent hackers from walking into a system using perfectly valid credentials. By keeping roles and duties distinct and separate, it also promotes organization and accountability in logging events that take place during a system’s development. |
| 1. Sanitize Data Sent to Other Systems | Once data is sent from one system to another, it’s imperative for the receiving system to check and validate what it’s receiving before using it. In the same way that packages can be opened or stolen in the US Postal Service, injections can happen in a similar fashion when it comes to data networking. The story of the Trojan Horse comes to mind. |
| 1. Practice Defense in Depth | One can never be too safe when it comes to protecting valuable data. If we use an analogy such as a medieval castle, they didn’t just settle for walls. They also had moats. The idea is that proper defense requires backup plans for the circumstances in which primary fortifications are breached. This idea is very applicable in software development. |
| 1. Use Effective Quality Assurance Techniques | Just because software moves from A to B as intended, doesn’t necessarily imply that it happened as efficiently as possible. Effectiveness and efficiency are the goals of proper quality assurance, and by ensuring that input, output, and everything in between is behaving as smoothly as possible, you will successfully ensure that your program is also safe from points of invasion. |
| 1. Adopt a Secure Coding Standard | By adopting secure coding standards, developers can heavily ease the work of testers. Not only does it help patch weaknesses before they can even happen, development overall will be streamlined. This means less debugging, faster release, and a much more secure final product. |

### C/C++ Ten Coding Standards

The following standards have been completed and reported per company philosophy. Please review them and be ready to ensure that all ideals are referenced during the development process.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Detect errors when converting a string to a number** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | When converting between data types, parsing isn’t 100% reliable in all situations. Parsing an integer or floating point number from a string can produce many errors. It’s always important to ensure that the transition from one type to another is as safe and secure as possible. |

| **Noncompliant Code** |
| --- |
| In this example, multiple numeric values are converted from the standard input stream. However, if the text received from the standard input stream cannot be converted into a numeric value that can be represented by an int, the resulting value stored into variables **i** and **j** might be unexpected. |
| #include <iostream>    void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, each converted value read from the standard input stream is tested for validity before reading the next value in the sequence, allowing error recovery on a per-value basis. It checks std::istream::fail() to see if the failure bit was set due to a conversion failure or whether the bad bit was set due to a loss of integrity with the stream object. |
| #include <iostream>  #include <limits>    void f() {  int i;  std::cin >> i;  if (std::cin.fail()) {  // Handle failure to convert the value.  std::cin.clear();  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), ' ');  }    int j;  ...  std::cin >> j;  if (std::cin.fail()) {  std::cin.clear();  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), ' ');  }    // ...  } |

| **Principles(s):** Validate Input Data, always ensure that user input is checked before assignment. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | cert-err34-c | Checked by clang-tidy |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | CERT\_C-ERR34-a | The library functions atof, atoi and atol from library stdlib.h shall not be used |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | CERT C: Rule ERR34-C | Checks for unsafe conversion from string to numeric value (rule fully covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that unsigned integers do not wrap** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Unsigned integer operations can wrap if the resulting value cannot be represented by the underlying representation of the integer. This behavior is more informally called unsigned integer wrapping. It is important to check that the value within the data type stays within the bounds to preserve accuracy and prevent errors. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in an unsigned integer wrap during the addition of the unsigned operands ui\_a and ui\_b. If this behavior is unexpected, the resulting value may be used to allocate insufficient memory for a subsequent operation or in some other manner that can lead to an exploitable vulnerability. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Complying with this standard can be approached with a precondition or a postcondition test. This compliant solution performs a precondition test of the operands of the addition to guarantee there is no possibility of unsigned wrap |
| #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):** Heed compiler warnings, many times the console warns against certain data types, despite the program running correctly. Use effective quality assurance techniques, because even though the code works now, doesn’t mean things aren’t vulnerable down the line. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | INTEGER\_OVERFLOW | Implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 493 S, 494 S | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | CERT\_C-INT30-a  CERT\_C-INT30-b  CERT\_C-INT30-c | Avoid integer overflows  Integer overflow or underflow in constant expression in '+', '-', '\*' operator  Integer overflow or underflow in constant expression in '<<' operator |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | unsigned overflow | Exhaustively verified |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not confuse narrow and wide character strings and functions** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Passing narrow string arguments to wide string functions or wide string arguments to narrow string functions can lead to unexpected and undefined behavior. Scaling problems are likely because of the difference in size between wide and narrow characters. It is important to keep string types consistent to preserve code integrity. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example incorrectly uses the strncpy() function in an attempt to copy up to 10 wide characters. However, because wide characters can contain null bytes, the copy operation may end earlier than anticipated, resulting in the truncation of the wide string. |
| #include <stddef.h>  #include <string.h>    void func(void) {  wchar\_t wide\_str1[] = L"0123456789";  wchar\_t wide\_str2[] = L"0000000000";    strncpy(wide\_str2, wide\_str1, 10);  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the proper-width functions. Using wcsncpy() for wide character strings and strncpy() for narrow character strings ensures that data is not truncated and buffer overflow does not occur. |
| #include <string.h>  #include <wchar.h>    void func(void) {  wchar\_t wide\_str1[] = L"0123456789";  wchar\_t wide\_str2[] = L"0000000000";  /\* Use of proper-width function \*/  wcsncpy(wide\_str2, wide\_str1, 10);    char narrow\_str1[] = "0123456789";  char narrow\_str2[] = "0000000000";  /\* Use of proper-width function \*/  strncpy(narrow\_str2, narrow\_str1, 10);  } |

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

| **Principles(s):** Validate Input Data to ensure that assigned containers are receiving data of appropriate length, Keep It Simple and make sure that non-static variables aren’t always assigned very strict type definitions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | Very High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-STR38 | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | CERT\_C-STR38-a | Do not confuse narrow and wide character strings and functions |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule STR38-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr38c.html) | Checks for misuse of narrow or wide character string (rule fully covered) |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | pointer arithmetic | Partially verified |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Exclude user input from format strings** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Never call a formatted I/O function with a format string containing a tainted value . An attacker who can fully or partially control the contents of a format string can crash a vulnerable process, view the contents of the stack, view memory content, or write to an arbitrary memory location. Consequently, the attacker can execute arbitrary code with the permissions of the vulnerable process |

| **Noncompliant Code** |
| --- |
| The incorrect\_password() function in this noncompliant code example is called during identification and authentication to display an error message if the specified user is not found or the password is incorrect. The function accepts the name of the user as a string referenced by the user. This is an example of untrusted data that originates from an unauthenticated user. The function constructs an error message that is then output to stderr using the C Standard fprintf() function. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fprintf(stderr, msg);  free(msg);  } |

| **Compliant Code** |
| --- |
| This compliant solution fixes the problem by replacing the fprintf() call with a call to fputs(), which outputs msg directly to stderr without evaluating its contents: |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fputs(msg, stderr);  free(msg);  } |

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

| **Principles(s):** Validate User Input to check for any unexpected symbols or operators, Sanitize Data Sent to Other Systems to keep data secure and systems protected, Architect and Design for Security Policies to ensure that developers keep as many potential vulnerabilities patched, Practice Defense in Depth by readying protocol for the event that an attempted injection is successful |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | CERT\_C-FIO30-a  CERT\_C-FIO30-b  CERT\_C-FIO30-c | Avoid calling functions printf/wprintf with only one argument other than string constant  Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable  Never use unfiltered data from an untrusted user as the format parameter |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 592 | Partially supported: reports non-literal format strings |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule FIO30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulefio30c.html) | Checks for tainted string format (rule partially covered) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Evaluating a pointer into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that have been deallocated are dangling pointers, and accessing one can result in exploitable vulnerabilities. When memory is freed, all pointers into it become invalid, and its contents might either be returned to the operating system, making the freed space inaccessible, or remain intact and accessible. As a result, the data at the freed location can appear to be valid but change unexpectedly. Consequently, memory must not be written to or read from once it is freed. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the 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):** Heed compiler warnings and minimize nullptrs , Adopt a Secure Coding Standard to emphasize proper techniques and handle deallocated memory |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 51 D, 484 S, 112 D | Partially implemented |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule MEM30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem30c.html) | Checks for use of previously freed pointer (rule partially covered) |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | dangling\_pointer | Exhaustively verified |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions, in general, demand validating data as it is being compiled. Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. To prevent such errors, you must either limit copies through truncation or, preferably, validate that the destination is of sufficient size to hold the character data to be copied and the null-termination character. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example demonstrates an off-by-one error [Dowd 2006]. The loop copies data from src to dest. However, because the loop does not account for the null-termination character, it may be incorrectly written 1 byte past the end of dest. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the loop termination condition is modified to account for the null-termination character that is appended to dest: |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n - 1); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

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

| **Principles(s):** Adopt a secure coding standard which practices assertion statements, Validate Input Data with the use of assertion statements to maintain software safety and security |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | 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  Rule partially covered. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | All exceptions thrown by an application must be caught by a matching exception handler. Even if the exception cannot be gracefully recovered from, using the matching exception handler ensures that the stack will be properly unwound and provides an opportunity to gracefully manage external resources before terminating the process. A program that encounters an unrecoverable exception may explicitly catch the exception and terminate, but it may not allow the exception to remain uncaught. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, 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** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| 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):** Adopt a secure coding standard which practices exception handling with strategies such as try-catch, Use Effective Quality Assurance Techniques which employs the best possible practices in ensuring exceptions are not present at the end of development, Validate User Input to ensure that exceptions are not raised unexpectedly by the user |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | main-function-catch-all  early-catch-all | Partially checked |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 527 S | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not write syntactically ambiguous declarations** |
| --- | --- | --- |
| Syntax | [STD-008-CPP] | It is possible to devise syntax that can ambiguously be interpreted as either an expression statement or a declaration. Do not write a syntactically ambiguous declaration. With the advent of uniform initialization syntax using a braced-init-list, there is now syntax that unambiguously specifies a declaration instead of an expression statement. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an attempt is made to declare a local variable, w, of type Widget while executing the default constructor. However, this declaration is syntactically ambiguous where the code could be either a declaration of a function pointer accepting no arguments and returning a Widget or a declaration of a local variable of type Widget. |
| #include <iostream>    struct Widget {  Widget() { std::cout << "Constructed" << std::endl; }  };    void f() {  Widget w();  } |

| **Compliant Code** |
| --- |
| This compliant solution shows two equally compliant ways to write the declaration. The first way is to elide the parentheses after the variable declaration, which ensures the syntax is that of a variable declaration instead of a function declaration. The second way is to use a braced-init-list to direct-initialize the local variable. |
| #include <iostream>    struct Widget {  Widget() { std::cout << "Constructed" << std::endl; }  };    void f() {  Widget w1; // Elide the parentheses  Widget w2{}; // Use direct initialization  } |

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

| **Principles(s):** Adopt a Secure Coding Standard which practices proper syntax and function declarations, Heed Compiler Warnings which often detect any undefined variables or functions, Keep it Simple to prevent developers from using any unnecessarily complex or overly specific functions/libraries |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 296 S | Partially Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b  CERT\_CPP-DCL53-c | Parameter names in function declarations should not be enclosed in parentheses  Local variable names in variable declarations should not be enclosed in parentheses  Avoid function declarations that are syntactically ambiguous |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl53cpp.html) | Checks for declarations that can be confused between:   * Function and object declaration * Unnamed object or function parameter declaration   Rule fully covered. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3468**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3468) | N/A |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Never hard code sensitive information** |
| --- | --- | --- |
| Sensitive Information | [STD-009-CPP] | Hard coding sensitive information, such as passwords or encryption keys can expose the information to attackers. Anyone who has access to the executable or dynamic library files can examine them for strings or other critical data, revealing the sensitive information. Hard coding sensitive information also increases the need to manage and accommodate changes to the code. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example must authenticate to a remote service with a code, using the authenticate() function declared below. It passes the authentication code to this function as a string literal. The authentication code exists in the program's binary executable and can be easily discovered. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  if (!authenticate("correct code")) {  printf("Authentication error\n");  return -1;  }    printf("Authentication successful\n");  // ...Work with system...  return 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution requires the user to supply the authentication code, and securely erases it when done, using the memset\_s() function, provided by C11. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  #define CODE\_LEN 50  char code[CODE\_LEN];  printf("Please enter your authentication code:\n");  fgets(code, sizeof(code), stdin);  int flag = authenticate(code);  memset\_s(code, 0, sizeof(code));  if (!flag) {  printf("Access denied\n");  return -1;  }  printf("Access granted\n");  // ...Work with system...  return 0;  } |

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

| **Principles(s):** Architect and Design for Security Policies to ensure sensitive data is handled with measures such as cryptography, Adhere to the Principle of Least Privilege to minimize the number of ways sensitive data can be accessed, Default Deny any user who is unrecognized, Defense in Depth to be ready for hackers to attempt to breach multiple layers |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | HARDCODED.AUTH  HARDCODED.DNS  HARDCODED.KEY  HARDCODED.SALT | Hardcoded Authentication  Hardcoded DNS Name  Hardcoded Crypto Key  Hardcoded Crypto Salt |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | CERT\_C-MSC41-a | Do not hard code string literals |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 2460 | Assistance provided: reports when a literal is provided as an argument to a function parameter with the ‘noliteral’ argument Semantic; several Windows API functions are marked as such and the ‘-sem’ option can apply it to other functions as appropriate |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule MSC41-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemsc41c.html) | Checks for hard coded sensitive data (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Obey the one-definition rule** |
| --- | --- | --- |
| Strict Definitions | [STD-010-CPP] | The one-definition rule (ODR) model is defined by C++ Standard. C++ restricts named object definitions to ensure that linking will behave deterministically by requiring a single definition for an object across all translation units. “Every program shall contain exactly one definition of every non-inline function or variable that is odr-used in that program; no diagnostic required”. Do not violate the one-definition rule; violations result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, two different translation units define a class of the same name with differing definitions. Although the two definitions are functionally equivalent (they both define a class named S with a single, public, nonstatic data member int a), they are not defined using the same sequence of tokens. |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| The correct mitigation depends on programmer intent. If the programmer intends for the same class definition to be visible in both translation units because of common usage, the solution is to use a header file to introduce the object into both translation units, as shown in this compliant solution. |
| // S.h  struct S {  int a;  };    // a.cpp  #include "S.h"    ...  // b.cpp  #include "S.h" |

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

| **Principles(s):** Adopt a Secure Coding Standard which practices proper nomenclature, Keep it Simple when defining variables and functions |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 286 S, 287 S | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL60-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl60cpp.html) | Checks for inline constraints not respected (rule partially covered) |

### 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.

At Green Pace, it is important to maintain the policies and structure that has allowed this company to flourish. That being said, it is also time to move forward in updating and upgrading this very pipeline to facilitate the transition to a new age of cybersecurity. The transition will focus less of changing, and more on reassessing the nature of what each step in the process will entail. From the assessing and planning stage to the design stage, Green Pace will now hold our development team responsible for incorporating these new philosophies into the DevSecOps toolchain. We must ensure that every developer is on the same page, and understands that risk mitigation begins at the genesis of a new project. Within the building phase, testers will work alongside developers to utilize some of the many automated programs available to increase the synergy and efficiency of both teams. By reducing the scope of potential issues, it will allow for the verification and testing phase to reallocate their focus in isolating any severe vulnerabilities.

By attacking the issue early, and establishing a security groundwork at the instantiation of our product idea, our final deliverable will reach a higher level of security that will benefit both our maintenance teams and our company clients. Defense in depth requires our techniques to be present at every stage of development.

### 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 | Medium | Unlikely | Medium | Low | 2 |
| STD-002-CPP | High | Likely | High | Medium | 4 |
| STD-003-CPP | High | Likely | Low | Very High | 4 |
| STD-004-CPP | High | Likely | Medium | High | 5 |
| STD-005-CPP | High | Likely | Medium | High | 5 |
| STD-006-CPP | High | Likely | Medium | High | 5 |
| STD-007-CPP | Low | Probable | Medium | Low | 2 |
| STD-008-CPP | Low | Unlikely | Medium | Low | 1 |
| STD-009-CPP | High | Probable | Medium | Medium | 4 |
| STD-010-CPP | High | Unlikely | High | 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 | When sensitive data has been stored under cyber lock and key on a disk or relevant storage device, this is considered data at rest. All company information must be encrypted using the appropriate cipher and algorithm to mitigate any form of data breach. |
| Encryption at flight | Sensitive data is sensitive for a purpose, and is often required by company administrators and active clients. However, moving data typically involves prior decryption. When decrypted data is in flight and moving from one system to another, it is vulnerable much like an armored vehicle moving large sums of currency from one bank to another. Company policy will require all sensitive data management to use the correct protocol, such as a secure web browser, and also transmission through channels which provide transport layer security (TLS) data encryption. |
| Encryption in use | Cyber attacks aren’t always carried out while developers are away from their desk. If data is being used by systems or administrators, it is neither in rest nor in flight. We must employ in-use encryption to ensure that our information is safe throughout its entire lifecycle. Company data being actively processed will require developers to become familiar with techniques such as hashing, and also ensuring that their execution environment is trusted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | System authentication is the method by which we can safely verify that the user logging in is the appropriate one. All company associates will be required to maintain a valid username and password which will allow our system to grant access per the account credentials. All accounts will have their credentials maintained within a secure database, and login attempts will allow access only once the user has provided the username and password which matches our database entry. New users must contact a system administrator to receive their initial account information and begin setup. |
| Authorization | Authorization is the active security protocol once a user has been successfully authenticated. If a valid user has gained access and wishes to perform a certain task, their account will be checked for the relevant privilege associated with the task. Essentially, it is the enforcement of company security policies by restricting individuals to their respective roles. Each user will have a level of access associated with their account status. Please contact a company administrator if certain privileges are required. |
| Accounting | Accounting is the final stage of our Triple A Framework, and involves the oversight of all system traffic. Session statistics and usage information will be logged, and files accessed by users will be reported. This will help us maintain authorization control, billing, trend analysis, and resource utilization. Any unexpected system tasks will be noted and addressed immediately to ensure that our new system is as safe and secure as possible for our userbase. |

**\***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 | 01/24/2022 | First Milestone | Ryan Mitchell | Trevor Hodde |
| 1.2 | 02/12/2022 | Completed Policy | Ryan Mitchell |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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