**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 encompasses the identification and validation of data sources. Data sources are defined as but are not limited to user-controlled files, command-line arguments, environment variables and network interfaces. When dealing with input data it should be validated to protect against exception conditions such as overflow. It also covers specifications and the need to address limits such as minimum and maximum length of values. (Seacord, R. C., 2013) |
| 1. Heed Compiler Warnings | Out of the box the compiler supports type-checking but it leaves much to be desired. It is recommended that you configure the compiler to detect more warnings. Depending upon your development environment you can use flags to tell the compiler to look for common exploits. If you’re using gcc you might use -Wstrict-prototypes -o2 which would detect warning that are only visible when data flow analysis is performed. (Seacord, R. C., 2013) |
| 1. Architect and Design for Security Policies | The architecture and design of a system are crucial to its security. If you have flaws in parts of your systems design or architecture these may be indicators of possible areas that can be easily exploited. When necessary, systems should be broken down in to subsystems that can communicate with one another. Allowing only certain subsystems to access higher privilege sets. (Seacord, R. C., 2013) |
| 1. Keep It Simple | The greater the complexity in the design the greater the risk for exploitation. Whenever possible try and keep the design small and simple. This will not only make the system more secure with less room for error but it will also make troubleshooting easier if something goes wrong. (Seacord, R. C., 2013) |
| 1. Default Deny | All inbound and outbound traffic should be blocked unless it has been expressly permitted. From a networking standpoint this may be defined at the firewall. Access should be set with permissions in mind instead of exclusions. It’s significantly easier to find and fix access denial when it was supposed to be granted over finding scenarios where access is granted but should have been denied. (Seacord, R. C., 2013) |
| 1. Adhere to the Principle of Least Privilege | When it comes to the system executing processes it should be accomplished with the least number of privileges possible. If elevated privileges are required, they should only be granted for a small amount of time. (Seacord, R. C., 2013) |
| 1. Sanitize Data Sent to Other Systems | You must ensure that data being sent to another system conforms to the standards of said system. It is your responsibility to sanitize the data before passing it to another function that may not understand the context. One popular method of data sanitization is known as blacklisting. Cleansing any string of known bad characters such as a “;” can protect against unexpected behavior. Other approach known as whitelisting is when you supply a list of acceptable values and only allow those values to be used as input. (Seacord, R. C., 2013) |
| 1. Practice Defense in Depth | Defense in Depth at its core is managing risk through multiple defensive strategies across multiple layers of the application. In theory if one layer fails another layer can step in and thwart an attack. This is not only useful in preventing issues at runtime but also for identifying changes that might cause unintended behavior during development. (Seacord, R. C., 2013) |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques may vary but it is an extremely important step in the product development lifecycle. Using a framework such as the Team Software Process for Secure Software Development can help ensure defects are detected and removed from the code at various points during the software development lifecycle. (Seacord, R. C., 2013) |
| 1. Adopt a Secure Coding Standard | Coding standards can help guide programmers to follow a set of rules while developing code for the system. Setting and following a standard can greatly reduce the introduction of known code-related vulnerabilities. An often-overlooked outcome from adopting coding standards is that the code becomes easier to read. I find that the code begins to read in a similar manner even if it performs vastly different operations across files. Please refer to CERT <https://www.sei.cmu.edu/about/divisions/cert/> (Seacord, R. C., 2013) |

## 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] | Use only explicitly signed or unsigned char type for numeric values |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below the variable c of type char can either be signed or unsigned. Due to this the print statement might return 5 or -17. It is hard to determine whether or not the program is executing currently or not without knowing if c is supposed to be signed or unsigned. |
| char c = 200;  int i = 1000;  printf("i/c = %d\n", i/c); |

| **Compliant Code** |
| --- |
| In the compliant example below the variable c of type unsigned char makes the division operation predictable. Meaning we know that 200 / 1000 should return 5. |
| unsigned char c = 200;  int i = 1000;  printf("i/c = %d\n", i/c); |

|  |
| --- |
| **Principles(s):** This standard relates to the Keep It Simple principle. By using unsigned variables, we eliminate the need to implement complex logic to verify number wrapping or otherwise unexpected results. This approach will keep our methods that perform mathematical equations significantly less complex and easier to test. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 |  | Supported indirectly via MISRA C:2012 rules 10.1, 10.3 and 10.4. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-INT07 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | LANG.TYPE.IOT | Inappropriate operand type |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this recommendation. In particular, it flags any instance of a variable of type char (without a signed or unsigned qualifier) that appears in an arithmetic expression |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Prevent Buffer Overflow by Validating Input |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below the user input is not validated before being assigned to a variable which only accepts 20 characters. This can lead to a Buffer Overflow. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";  std::cin >> user\_input;  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

| **Compliant Code** |
| --- |
| In the compliant example below the user input is limited to the size of the variable therefore never allowing the user to cause a Buffer Overflow. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";  std::cin.getline(user\_input, 20);  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

|  |
| --- |
| **Principles(s):** This standard maps directly to the ValidateInput Data principle. By validating the length of the expected value, we eliminate the need to further validate the input data for length. Allowing us to turn our attention to further validation. We may want to verify that the input provided meets other acceptance criteria before ultimately allowing our system to use the input. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | C++2835, C++2836, C++2839, C++5216 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.1 | NNTS.MIGHT  NNTS.TAINTED |  |
| [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 |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not pass a non-null-terminated character sequence to a library function that expects a string |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below the character sequence will not be null-terminated when it is passed to printf() |
| #include <stdio.h>    void func(void) {  char c\_str[3] = "abc";  printf("%s\n", c\_str);  } |

| **Compliant Code** |
| --- |
| In the compliant example below by omitting the array bound the compiler allocates the required storage to store the provided string as well as the null terminating character. |
| #include <stdio.h>    void func(void) {  char c\_str[] = "abc";  printf("%s\n", c\_str);  } |

|  |
| --- |
| **Principles(s):** We can map this standard to the Sanitize Data Sent to Other Systems principle. By sanitizing our string before calling a library function using this standard exemplifies the principle. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 |  | Supported  Astrée supports the implementation of library stubs to fully verify this guideline. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-STR32 | Partially implemented: can detect some violation of the rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | MISC.MEM.NTERM.CSTRING | Unterminated C String |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | STRING\_NULL | Fully implemented |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL Injection by using a data dictionary |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below executes the provided statement without verifying that the provided columns exist in the database. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

| **Compliant Code** |
| --- |
| In the compliant example below the SQL statement is checked for a where clause. We then use a data dictionary to determine if the provided criteria columns exist in the data dictionary. |
| std::set<std::string> validColumns{ "ID", "NAME", "PASSWORD" };  …  bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  records.clear();  std::string localCopy(sql);  std::transform(localCopy.begin(), localCopy.end(), localCopy.begin(), ::tolower);  char equal = '=';  // loop through characters of the provided sql statement  for (int i = 0; i < localCopy.size(); i++) {  // if we find an equal sign validate the left-hand side of the equal sign  // against our list of validColumns.  if (localCopy[i] == equal) {  std::string statement = localCopy.substr(0, i);  std::string lhs = statement.substr(statement.find\_last\_of(' ') + 1);  std::transform(lhs.begin(), lhs.end(), lhs.begin(), ::toupper);  if (std::find(validColumns.begin(), validColumns.end(), lhs.c\_str()) == validColumns.end()) {  std::cout << "Possible SQLInjection detected." << std::endl;  std::cout << "Prevented execution of statement: " << sql.c\_str() << std::endl;  return false;  }  }  }    char\* error\_message;  if (sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

|  |
| --- |
| **Principles(s):** This standard applies to two of the stated principles ValidateInput Data and Sanitize Data Sent to Other Systems. With this standard we validate not only that the supplied input values are correct but we are also ensuring that what we send to the database is sanitized first. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/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](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | TAINTED\_STRING | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.1 | NNTS.TAINTED  SV.TAINTED.INJECTION |  |

### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| In the noncompliant example below the local variable space is passed to the new operator as an expression. This will cause the operator delete() to try and free memory that was not returned by the 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** |
| --- |
| In the compliant example below calls s1’s destructor directly removing a call to the 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();  } |

|  |
| --- |
| **Principles(s):** Applying this standard will help us ensure we are implementing the principle Adopt a Secure Coding Standard. Memory allocation and deallocation should be handled safely in order to not open our system up to possible exploits. In ensuring we allocate and deallocate accordingly we are less likely to open up unexpected exploits. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | invalid\_dynamic\_memory\_allocation  dangling\_pointer\_use |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MEM51 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM | Free non-heap variable  Double free  Type mismatch |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below assert is used to assert a memory-mapped structure. This can cause the code to behave in an unexpected manner. |
| #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 the compliant example below a preprocessor conditional statement is used to evaluate assertions at compile time. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

|  |
| --- |
| **Principles(s):** The following standard applies the Architect and Design for Security Policies principle. By using the approach of evaluating assertions at compile time we avoid any unexpected behavior due to the way in which we design our assertions. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Fully implemented |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL03 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | CC2.DCL03 | Fully implemented |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not reference base classes or class data members in a constructor or destructor function-try-block handler |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below the constructor uses a try-block which can result in undefined behavior. |
| #include <string>    class C {  std::string str;    public:  C(const std::string &s) try : str(s) {  // ...  } catch (...) {  if (!str.empty()) {  // ...  }  }  }; |

| **Compliant Code** |
| --- |
| In the compliant example below the constructor parameter is inspected instead of the class data member like in the noncompliant example. This allows us to avoid undefined behavior. |
| #include <string>    class C {  std::string str;    public:  C(const std::string &s) try : str(s) {  // ...  } catch (...) {  if (!s.empty()) {  // ...  }  }  }; |

|  |
| --- |
| **Principles(s):** This standard maps to the Architect and Design for Security Policies principle. By designing our software to avoid undefined behavior we are architecting and designing our software with security policies in mind. A key to protecting our software from security exploits is to design our code in such a way that all code executes a specific behavior and does not open itself up to unexpected behavior. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-ERR53 |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | [CERT C++: ERR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr53cpp.html) | Checks for constructor or destructor function-try-block handler referencing base class or class data member (rule fully covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | exception-handler-member-access | Fully checked |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | exception-handler-member-access | Fully checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Do not leak resources when handling exceptions |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below pst is never released properly. Therefore, when an exception is thrown a resource leak will occur. |
| #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 the compliant example below pst is freed within the exception handler preventing a resource leak on exception. |
| #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;  } |

|  |
| --- |
| **Principles(s):** This standard maps to the principle of Adhere to the Principle of Least Privilege. You should always assume the user even if an Administrator does not require detailed exception information. Leaking detailed exception information can help a malicious actor exploit your application in unexpected ways. Leaving this detailed information to be sent to a logging or monitoring system and instead returning a generic failure message is a much more secure approach. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | **C++4756, C++4757, C++4758** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **50 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | **CERT\_CPP-ERR57-a** | Ensure resources are freed |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Avoid using default operator new for over-aligned types |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below the default operator new is used to obtain storage in order to construct and object that exceeds the alignment of most implementations. |
| struct alignas(32) Vector {  char elems[32];  };    Vector \*f() {  Vector \*pv = new Vector;  return pv;  } |

| **Compliant Code** |
| --- |
| In the compliant example below an overloaded version of the operator new is used to obtain properly sized storage. |
| #include <cstdlib>  #include <new>    struct alignas(32) Vector {  char elems[32];  static void \*operator new(size\_t nbytes) {  if (void \*p = std::aligned\_alloc(alignof(Vector), nbytes)) {  return p;  }  throw std::bad\_alloc();  }  static void operator delete(void \*p) {  free(p);  }  };    Vector \*f() {  Vector \*pv = new Vector;  return pv;  } |

|  |
| --- |
| **Principles(s):** This standard maps to the principle Use Effective Quality Assurance Techniques. By ensuring we are allocating the appropriate storage we avoid opening ourselves to common quality assurance issues down the road. Memory leaks can be extremely dangerous and we should avoid them wherever possible. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | **C++3129** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | **CERT\_CPP-MEM57-a** | Avoid using the default operator 'new' for over-aligned types |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | [CERT C++: MEM57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem57cpp.html) | Checks for situations where operator new is not overloaded for possibly overaligned types (rule fully covered) |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| In the noncompliant example below the first iterator will eventually be incremented beyond the end element range resulting in undefined behavior. This is due to the iterators being passed in improper order. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.end(), c.begin(), [](int i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| In the noncompliant example below the iterators are passed in proper order therefore not causing undefined behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.begin(), c.end(), [](int i) { std::cout << i; });  } |

|  |
| --- |
| **Principles(s):** Using valid iterator ranges maps directly to the Keep It Simple principle. In doing so we will ensure we do not cause undefined behavior. Lowering the number of possible exploits within our software. It also will cause less confusion as a developer will not be left trying to understand why we may want to iterate beyond the defined range. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | Overflow\_upon\_dereference |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | C++3802 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-CTR53-a  CERT\_CPP-CTR53-b | Do not use an iterator range that isn't really a range  Do not compare iterators from different containers |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.07 | V539, V662, V789 |  |

## 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.

With Green Pace’s already well-established DevOps process and infrastructure implementing the many automation options will be rather easy. First, we should ensure that we have a continuous integration and continuous deployment (CI/CD) process where much of our automation can be run. I foresee this encompassing much of the pre-production process. Creating a CI\CD pipeline using an application such as Jenkins will allow us to build jobs that run through the various standards that have been proposed within this document.

In doing so we can cover much of the Assess and plan, Verify and test, Build, and Design pre-production steps. Running our code against things such as Coverity and other automation tools to verify our code is compliant with our defined standards. I believe these tools should also be available at a local level for execution prior to code submission. Every developer should be responsible for checking their code is compliant before allowing the automation to take over.

Finally, as part of the production process this is where our continuous deployment can help. If the code meets our defined standards and principles, we can safely deploy our new changes as part of the production process. We can further automate the process by using the Monitor and detect processes already in place to proactively patch our software instead of reactively.

## 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 | Probable | Medium | P8 | L2 |
| STD-002-CPP | High | Likely | Medium | P18 | L1 |
| STD-003-CPP | High | Probable | Medium | P12 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-008-CPP | Low | Probable | High | P2 | L3 |
| STD-009-CPP | Medium | Unlikely | Low | P6 | L2 |
| STD-010-CPP | High | Probable | High | 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 | In order to protect our data while at rest we must implement data classifications for all of our data and secure it based upon its level of classification. For example, sensitive information such as credentials would have a classification of classified. Any data labeled classified shall not be stored in plain text instead they it be encrypted while at rest. |
| Encryption at flight | While data is in flight i.e. being transferred across the enterprise network. We must rely on firewalls and network access controls to ensure our data is protected. These can help aide us against malware attacks or possible network intrusions. |
| Encryption in use | Data may be decrypted while in use for such purposes as reporting or visualization. You cannot transfer any company data to a personal device. Nore can any printed company information may be removed from a company location or brought to a location that is not approved. When data is in use it is at its most exploitable state and must be treated as such. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is an important step in ensuring a user is who they say they are. With this is mind users will be added and managed through active directory. This will allow us to control access to any and all systems through one service. This will also allow us to monitor all user activity from one location. |
| Authorization | Users will be authorized to access systems through role-base access controls. These will be based on the data in which the user must access in order to perform their respective job responsibilities. This will allow us to control who gets to access different applications, shared storage spaces, and accessing and applying changes to the database. |
| Accounting | In order to monitor user usage, we will employ a security information and event management tool. Ideally the tool will supply us with detailed reports around user usage which we can in turn use to scan for possible suspicious activity. |

**\***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 | 05/23/2021 | Initial Standards | Joe Andersen |  |
| 1.2 | 06/13/2021 | Final Iteration | Joe Andersen |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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

References

Seacord, R. C. (2013). Secure Coding in C and C. [MBS Direct]. Retrieved from <https://mbsdirect.vitalsource.com/#/books/9780132981972/>

Confluence. INT07-C. Use only explicitly signed or unsigned char type for numeric values - SEI CERT C Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/c/INT07-C.+Use+only+explicitly+signed+or+unsigned+char+type+for+numeric+values>.

Confluence. STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string - SEI CERT C Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/c/STR32-C.+Do+not+pass+a+non-null-terminated+character+sequence+to+a+library+function+that+expects+a+string>.

Confluence. MEM51-CPP. Properly deallocate dynamically allocated resources - SEI CERT C++ Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources>.

Confluence. DCL03-C. Use a static assertion to test the value of a constant expression - SEI CERT C Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression>.

Confluence. ERR53-CPP. Do not reference base classes or class data members in a constructor or destructor function-try-block handler - SEI CERT C++ Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR53-CPP.+Do+not+reference+base+classes+or+class+data+members+in+a+constructor+or+destructor+function-try-block+handler>.

Confluence. ERR57-CPP. Do not leak resources when handling exceptions - SEI CERT C++ Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR57-CPP.+Do+not+leak+resources+when+handling+exceptions>.

Confluence. MEM57-CPP. Avoid using default operator new for over-aligned types - SEI CERT C++ Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM57-CPP.+Avoid+using+default+operator+new+for+over-aligned+types>.

Confluence. CTR53-CPP. Use valid iterator ranges - SEI CERT C++ Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterator+ranges>.

Confluence. STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator - SEI CERT C++ Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator>.

Confluence. STR02-C. Sanitize data passed to complex subsystems - SEI CERT C Coding Standard - Confluence. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems>.

Bowman, M. from B., Bowman, B., Adam Strange | 2 days ago, Kyle Marchini | 3 days ago, Mike Nelson | 3 days ago, Richi Jennings | 2 days ago, Richi Jennings | 4 days ago, &amp; 07, R. J. | J. (2019, March 18). The AAA Framework for Identity Access Security. Security Boulevard. <https://securityboulevard.com/2019/03/the-aaa-framework-for-identity-access-security/#:~:text=The%20AAA%20Framework%20is%20a,for%20Authenticate%2C%20Authorize%20and%20Account>.