**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 | To validate input from all data sources, because proper input can alleviate a lot of software vulnerabilities. |
| 1. Heed Compiler Warnings | Be sure to compile code using the highest level of warning in the compiler, as this helps to purge warnings. This also includes using dynamic and static analysis tools to detect and get rid of other security defects. |
| 1. Architect and Design for Security Policies | Design software and create an architecture that imposes security policies. |
| 1. Keep It Simple | Avoid complex designs and keep it as simple and efficient as possible. Complexity can bring an increase of errors during configuration and use. |
| 1. Default Deny | Implement access on permissions, so that by default access is denied and the protection structure takes effect. |
| 1. Adhere to the Principle of Least Privilege | Execution should occur with the least amount of privilege necessary. Higher levels of permissions should only be allowable for the minimum time required to complete the privileged task. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all of the data that is passed on to complex subsystems such as: relational db, shells, and COTS. SQL, command, and other injection attacks could occur if not sanitized before summoning the subsystem. |
| 1. Practice Defense in Depth | Manage risk with multiple layers of defensive strategies to prevent exploitable vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | QA techniques should be in effect to help eliminate vulnerabilities. This could inclide fuzz testing, source code audits, and penetration testing. |
| 1. Adopt a Secure Coding Standard | Develop a secure coding standard for the specified language and platform. |

## C/C++ Ten Coding Standards

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

### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [INT-113-CPP] | Do not cast an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| A value of integral or enumeration type can be converted to an enumeration type. The value is unchanged if the original value is within the range of enumeration values. The resulting value is unspecified and might not be in that range.  Attempts to check whether a given value is within range of acceptable enumeration values. However, this cats to the enumeration type, but may not be able to represent the given integer value; which results in unspecified behavior. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| Bounds check: checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. |
| enum EnumType {    First,    Second,    Third  };  void f(**int** intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

|  |
| --- |
| **Principles(s):** Validate Input & Keep it Simple. It is possible for unspecified values to result in a buffer overflow, leading to the execution of arbitrary code by an attacker. However, because enumerators are rarely used for indexing into arrays or other forms of pointer arithmetic, it is more likely that this scenario will result in data integrity violations rather than arbitrary code execution. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-INT113-CPP | [Insert text.] |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-INT113-CPP** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PRQA QA-C++ | 4.4 | 3013 |  |
| PVS-Studio | 7.07 | V1016 |  |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [INT-220-CPP] | Ensure that unsigned integer operations do not wrap |

| **Noncompliant Code** |
| --- |
| A computation involving unsigned operands can never overflow, because a result that cannot be represented by the resulting unsigned integer type is reduced below the number that is one greater than the largest value that can be represented by the resulting type.  Unsigned integer wrap during the addition of the unsigned operands ui\_a and ui\_b. |
| void func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum = ui\_a + ui\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Performs a precondition test of the operands of the addition to guarantee there is no possibility of unsigned wrap. |
| 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):** Validate Input. Practice Defense in Depth, Adopt a Secure Coding Standard. Integer wrap can lead to buffer overflow and the execution of arbitrary code by attackers |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Ineger-overflow | Fully checked |
| CodeSonar | 6.0p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| Compass/ROSE |  |  | Can detect violations of this rule by ensuring that operations are checked for overflow before being performed (Be mindful of exception INT30-EX2 because it excuses many operations from requiring [validation](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validation), including all the operations that would validate a potentially dangerous operation. For instance, adding two unsigned ints together requires validation involving subtracting one of the numbers from UINT\_MAX, which itself requires no validation because it cannot wrap.) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rule INT220-CPP](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** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STR-007-CPP] | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| A character string literal is a sequence of zero or more multibyte characters enclosed in double-quotes, as in “xyz”. A UTF-8 string literal is the same, except prefixed by u8. A wide string literal is the same, except prefixed by the letter L, u, or U. |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| Creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

|  |
| --- |
| **Principles(s):** Practice Defense in Depth. Undefined behavior which can lead to integer overflow. Modifying strong literals can lead to DoS attacks. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2020.2 | **CERT\_C-STR007-a CERT\_C-STR007-b** | A string literal shall not be modified Do not modify string literals |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **PW** | Deprecates conversion from a string literal to "char \*" |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect simple violations of this rule |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rule STR30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr30c.html) | Checks for writing to const qualified object |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [FIO-000-CPP] | Exclude user input from format strings |

| **Noncompliant Code** |
| --- |
| Never call a formatted I/O function with a format string containing a tainted value.  The incorrect\_password() function is noncompliant, because it is called during identification and authentication to display an error message if the specified user is not found or the password is incorrect. |
| 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** |
| --- |
| Replaces the fprint() call with a call to fputs(), which outputs msg directly to stderr without evaluating its contents |
| 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 Input, Practice Defense in Depth, Architect and Design for Security Policies. Failing to execute user input from formatted specifiers can allow attackers to crash vulnerable processes, and view the memory stack, content or write arbitrary code with the permissions of the vulnerable process. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | **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 |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect violations of this rule when the -Wformat-security flag is used |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **IO.INJ.FMT** **MISC.FMT** | Format string injection Format string |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [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** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [MEM-759-CPP] | Allocate sufficient memory for an object. |

| **Noncompliant Code** |
| --- |
| Integer types used as size arguments to malloc(), calloc(), realloc(), or aligned\_alloc() must have sufficient range to represent the size of the objects to be stored. If size arguments are incorrect, buffer overflow may occur.  Inadequate space is allocated for a struct tm object because the size of the pointer is being used to determine the size of the pointed-to object |
| struct **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {    struct **tm** \*tmb;    tmb = (struct **tm** \*)**malloc**(sizeof(tmb));    if (tmb == NULL) {      return NULL;    }    \*tmb = (struct **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };    return tmb;  } |

| **Compliant Code** |
| --- |
| The correct amount of memory is allocated for the struct tm object. |
| struct **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {    struct **tm** \*tmb;    tmb = (struct **tm** \*)**malloc**(sizeof(\*tmb));    if (tmb == NULL) {      return NULL;    }    \*tmb = (struct **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };    return tmb;  } |

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

|  |
| --- |
| **Principles(s):** Practice Defense in Depth, Adopt a Secure Coding Standard. If size arguments are incorrect, this can lead to buffer overflow, and for attackers to write arbitrary code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **malloc-size-insufficient** | Partially checked  direct rule violations, all undefined behavior resulting from invalid memory accesses is reported by Astrée. |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rule MEM35-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem35c.html) | Checks for:  Pointer access out of bounds  Memory allocation with tainted size  Rule fully covered. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | **CERT\_C-MEM35-a** | Do not use sizeof operator on pointer type to specify the size of the memory to be allocated via 'malloc', 'calloc' or 'realloc' function |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **ALLOC.SIZE.ADDOFLOW** **ALLOC.SIZE.IOFLOW** **ALLOC.SIZE.MULOFLOW** **ALLOC.SIZE.SUBUFLOW** **ALLOC.SIZE.TRUNC** **IO.TAINT.SIZE** **MISC.MEM.SIZE.BAD** | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [DCL-44-CPP] | Use a static assertion to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| Assertions are used for finding and eliminating defects that may result in vulnerabilities.  This code uses assert() to assert a property concerning a memory-mapped structure that is essential for the code to run properly |
| 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** |
| --- |
| A preprocessor conditional statement is used |
| 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 |

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

|  |
| --- |
| **Principles(s):** Heed Compiler Warnings, Practice Defense in Depth, Adopt a Secure Coding Standard Static assertion is a valuable diagnostic tool for finding and removing defects that may result in vulnerabilities at compile time. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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 |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [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** | [ERR-556-CPP] | Handle all exceptions. |

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

| **Compliant Code** |
| --- |
| 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, Practice Defense in Depth, Use Effective Quality Assurance Techniques. Allowing the app to abnormally terminate can lead to resources not being closed. This is a perfect vector for DoS attacks. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **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) | R2020a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [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 |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations & Initialization | [DCL-777-CPP] | Do not modify the standard namespaces |

| **Noncompliant Code** |
| --- |
| The declaration of x is added to the namespace std, causing undefined behavior |
| namespace std {  **int** x;  } |

| **Compliant Code** |
| --- |
| namespace nonstd {  **int** x;  } |
| This assumes the intention was to place the declaration of x into a namespace to prevent collisions with other global identifiers. Instead of placing declaration into namespace std, it is placed into one without a reserved name. |

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

|  |
| --- |
| **Principles(s):**  Practice Defense in Depth, Sanitize Data Sent to Other Systems, Adopt a Secure Coding Standard. Altering the standard namespace can cause undefined behavior. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [CTR-333-CPP] | Guarantee that library functions do not overflow |

| **Noncompliant Code** |
| --- |
| Copying data into a container that is not large enough to hold it will result in a buffer overflow.  The std::copy() provides no bounds checking and can lead to a buffer overflow. The vecto of integers is copied from src to dest using std::copy(). |
| void f(const std::vector<**int**> &src) {    std::vector<**int**> dest;    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

| **Compliant Code** |
| --- |
| This solution is easiest to construct dest from src directly. |
| void f(const std::vector<**int**> &src) {    std::vector<**int**> dest(src);    // ...  } |

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

|  |
| --- |
| **Principles(s):** Practice Defense in Depth, Adopt a Secure Coding Standard, Use Effective Quality Assurance Techniques. Copying data to a buffer too small will result in a buffer overflow. Attackers can then write arbitrary code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **invalid\_pointer\_dereference** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **BADFUNC.BO.\***  **LANG.MEM.BO** | A collection of warning classes that report uses of library functions prone to internal buffer overflows.  Buffer Overrun |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-CTR52-a** | Do not pass empty container iterators to std algorithms as destinations |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| string | [STR-888-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| Std::string object is created from the results of a call to std::getenv(). This returns a null pointer on failure, which can lead to undefined behavior. |
| void f() {    std::string tmp(std::**getenv**("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The results from the all to std::getenv() are checked for null before std::string object is constructed. |
| void f() {    const **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

|  |
| --- |
| **Principles(s):** Practice Defense in Depth, Adopt a Secure Coding Standard, Sanitize Data Sent to Other Systems, Keep it Simple. Dereferencing a null pointer leads to undefined behavior, which can terminate a program. This leads attackers to execute arbitrary code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **assert\_failure** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |

## 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.

[Insert your written explanations here.]

## 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 |
| --- | --- | --- | --- | --- | --- |
| INT-113-CPP | Medium | Unlikely | Medium | High | 3 |
| INT-220-CPP | High | Likely | High | High | 2 |
| STR-007-CPP | Low | Likely | Low | High | 2 |
| FIO-000-CPP | High | Likely | Medium | High | 1 |
| MEM-759-CPP | High | Probable | High | Medium | 2 |
| DCL-44-CPP | Low | Unlikely | High | Low | 3 |
| ERR-556-CPP | Low | Probable | Medium | Low | 3 |
| DCL-777-CPP | High | Unlikely | Medium | Medium | 2 |
| CTR-333-CPP | High | Likely | Medium | High | 1 |
| STR-888-CPP | High | Likely | Medium | High | 1 |

## 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 | This is designed to prevent an attacker from accessing unencrypted data by ensuring the data is is encrypted when on disk. If an attacker gains access to a hard drive with encrypted data, but not the required keys, they can work to bypass the encryption to read the data. Due to this encryption at rest is of the highest recommendation and a high priority requirement. |
| Encryption at flight | This is where and how encrypted data is transferred from one point to another.The encrypted data is sent over a secure protocol using a data-encryption key. |
| Encryption in use | This is data that is being read, accessed, or processed by being transported between systems. To secure this data, this is where access is restricted by user roles, limiting system access, but also ensuring data is encrypted during its use/transfer. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the first step in having a secure identification system. The system should ensure that the person accessing the system is who they say they are. There are 3 categories to this method: What They Know: user logins, and/or security questions. Who They Are: biometric authentication. What They Have: 2-facto authentication to verify identity. |
| Authorization | User level of access which includes the type of access control. This can be done by one of three methods: Discretionary Access Control (DAC): file access by users is given permissions by owner/admin. Role-Based Access Control (RBAC): User role determined by organization. Mandatory Access Control (MAC): security level of users is granted based on security of the content being accessed. This also pertains to addition of new users, and any changes to the database itself. |
| Accounting | User access is monitored and logged for auditing purposes. This is to know which files a user is accessing, attempting to access, and whether more or less authorization is needed. |

**\***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 |