**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 | Validating and verifying all data inputted into a system helps to mitigate or prevent vulnerabilities caused by malicious inputs. Input validation helps to ensure that only legitimate and expected data is accepted, reducing the risk of SQL injections and other attacks. Data is deemed safe and private for use and storage. |
| 1. Heed Compiler Warnings | Compiler gives warning messages that flag potential vulnerabilities in code during the development process. Developers understand and fix these warnings to identify and fix security flaws before code releases into production. |
| 1. Architect and Design for Security Policies | Create architecture that considers potential security risks and design their systems with security in mind. Have clear rules and policies that help to outline the best security practice. |
| 1. Keep It Simple | Simple code reduces complexity and increases understanding and thought process. Comments may help. Additionally, simple code often contains less vulnerabilities than complex code. Simple leads to less errors and warnings with complex security design. |
| 1. Default Deny | Denying access to all resources by default and only granting access to those that are explicitly authorized. This approach helps to reduce the risk of unauthorized access and prevents security breaches. This limits roles to the bare minimum for any new user. |
| 1. Adhere to the Principle of Least Privilege | When a new user gets assigned a role, that user receives the minimum level of access required to perform their tasks. In the event of security breach or a compromised account, minimal damage can occur. The administrator bares the most responsibility. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such as command shells, relational databases, configuration files, and so on. Sanitizing any data sent to other systems to ensure that it cannot be exploited by attackers. |
| 1. Practice Defense in Depth | This principle involves implementing multiple layers of security controls to protect against different types of attacks. An example is multi-factor authentication (MFA), which uses phone and password to identify an individual. Multi-layer defenses gives reassurance and protection if one of the layers is compromised. |
| 1. Use Effective Quality Assurance Techniques | Excellent quality assurance techniques such as testing and code review identify and mitigate potential hazardous code. Routine testing ensure that a vulnerability reveals itself. Using multiple testing phases, independent security reviews, employable gray or white hat hackers, and external security reviews provide extra security measures. |
| 1. Adopt a Secure Coding Standard | A secure coding standard is a set of guidelines and best practices that developers should follow to ensure the security of their code. A coding standard gives implementation, guidelines, and practical use in the work. Be sure to familiarize for each language of choice. |

### 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** | DCL12-C | Implement abstract data types using opaque types |

| **Noncompliant Code** |
| --- |
| string\_mx type is fully visible to the user of the data type which has potential for manipulation. Data encapsulation or hiding of code remains to apply to string\_mx. |
| 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** |
| --- |
| String\_mx is a private type, which hides implementation from the user. Developers create a string\_m.h header file which contains the encapsulation idea that string\_mx is private in the header file. |
| 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);  /\* ... \*/  /\*place in header file\*/  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):** 1. Validate Input Data gives privacy to code. 7. Sanitize Data Sent to Other System creates private field variables that pass through other architectural components such as database. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL12 |  |
| LDRA tool suite | 9.7.1 | 104 D | Partially implemented |
| Parasoft C/C++test | 2022.2 | 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 | R2023a | CERT C: Rec. DCL12-C | 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** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT31-C | Ensure that integer conversions do not result in lost or misinterpreted data |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data, loss of sign, can occur when converting from a value of an unsigned integer type to a value of a signed integer type. |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Add if statement or coding solution that validates from unsigned to signed. This compliant solution can be used to convert a value of unsigned long int type to a value of signed char type |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  if (u\_a <= SCHAR\_MAX) {  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  } else {  /\* Handle error \*/  }  } |

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

| **Principles(s):** 2.Validate Input data accepts data in assigned type. 4.Keep It simple prevents errors and warnings within the code repository |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 |  | |  | | --- | | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 | |
| CodeSonar | 7.3p0 | LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT  LANG.CAST.COERCE LANG.CAST.VALUE  ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value  Truncation of allocation size Truncation of size  Tainted buffer access |
| Compass/ROSE | [Insert text.] |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |
| Coverity\* | 2017.07 | NEGATIVE\_RETURNS  REVERSE\_NEGATIVE  MISRA\_CAST | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior  Can find instances where a negativity check occurs after the negative value has been used for something else  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Helix QAC | 2023.1 | C2850, C2855, C2890, C2895, C2900, C2905,  C++2850, C++2855, C++2890, C++2895, C++2900, C++2905,  C++3000, C++3010  DF2851, DF2852, DF2853,  DF2856, DF2857, DF2858, DF2891, DF2892, DF2893, DF2896, DF2897, DF2898, DF2901, DF2902, DF2903, DF2906, DF2907, DF2908 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR30-C | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| Char str is initialized to the address of a string literal. Str is unable to be modified. |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| Using an array initializer of str allows the initial values of characters in an array to modify character or size of array of characters. Str is able to 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):** 2.Heed Compiler Warnings may show incorrect use of characters. 4.Keep It Simple checks for |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | string-literal-modfication write-to-string-literal | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| Compass/ROSE |  |  | Can detect simple violations of this rule |
| Polyspace Bug Finder | R2023a | CERT C: Rule STR30-C | Checks for writing to const qualified object (rule fully covered) |
| Splint | 3.1.1 |  |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STR02-C | Sanitize data passed to complex subsystems |

| **Noncompliant Code** |
| --- |
| Unknown data can pass through the works of system or subsystem. John Viega and Matt Messier provide an example of an application that inputs an email address to a buffer and then uses this string as an argument in a call to system() [Viega 2003]. Rate values may pass into the validation system. |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| Whitelisting is an approach to data sanitization. A list of acceptable characters manage the input values. Non-agreeable, dangerous characters are removed. |
| 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):** 7.Sanitize Data Sent to Other Systems directly names to the goal of secured subsystems. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 |  | Supported by stubbing/taint analysis |
| CodeSonar | 7.3p0 | 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 | 6.5 | TAINTED\_STRING | Fully implemented |
| Klocwork | 2023.1 | NNTS.TAINTED SV.TAINTED.INJECTION |  |
| LDRA tool suite | 9.7.1 | 108 D, 109 D | Partially implemented |
| Parasoft C/C++test | 2022.2 | 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 | R2023a | CERT C: Rec. STR02-C | 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** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | CON32-C | Prevent data races when accessing bit-fields from multiple threads |

| **Noncompliant Code** |
| --- |
| Adjacent bit fields may be stored in a single memory location. Modification of bit-fields in different threads results in undefined behavior. |
| struct multi\_threaded\_flags {  unsigned int flag1 : 2;  unsigned int flag2 : 2;  };    struct multi\_threaded\_flags flags;    int thread1(void \*arg) {  flags.flag1 = 1;  return 0;  }    int thread2(void \*arg) {  flags.flag2 = 2;  return 0;  } |

| **Compliant Code** |
| --- |
| All flags have access protection, which prevents data race or odd behaviors in different threading. |
| #include <threads.h>    struct multi\_threaded\_flags {  unsigned int flag1 : 2;  unsigned int flag2 : 2;  };    struct mtf\_mutex {  struct multi\_threaded\_flags s;  mtx\_t mutex;  };    struct mtf\_mutex flags;    int thread1(void \*arg) {  if (thrd\_success != mtx\_lock(&flags.mutex)) {  /\* Handle error \*/  }  flags.s.flag1 = 1;  if (thrd\_success != mtx\_unlock(&flags.mutex)) {  /\* Handle error \*/  }  return 0;  }    int thread2(void \*arg) {  if (thrd\_success != mtx\_lock(&flags.mutex)) {  /\* Handle error \*/  }  flags.s.flag2 = 2;  if (thrd\_success != mtx\_unlock(&flags.mutex)) {  /\* Handle error \*/  }  return 0;  } |

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

| **Principles(s):** 1.Validate Input Data handles the data access through transmission. 3.Architect and Design for Security Policies implement threads to handle multiple data streams. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | read\_data\_race  write\_data\_race | Supported by sound analysis (data race alarm) |
| Axivion Bauhaus Suite | 7.2.0 | CertC-CON32 |  |
| CodeSonar | 7.3p0 | CONCURRENCY.DATARACE CONCURRENCY.MAA | Data race Multiple Accesses of Atomic |
| Coverity | 2017.07 | MISSING\_LOCK | Partially implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCL03-C | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| Assert() has a property that concerns memory -mapped structure |
| #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** |
| --- |
| Static assumptions allow assumptions that are checked during compile time. No runtime cost in space or time is incurred. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),  "Structure must not have any padding"); |

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

| **Principles(s):** 4.Keep It Simple brings no program errors or warnings. Running memory efficiently prevents attacks of memory leakage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.3p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/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 |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | SIG35-C | Do not return from a computational exception signal handler |

| **Noncompliant Code** |
| --- |
| Division operation may perform an error or unexpected behavior if denom is 0. |
| #include <errno.h>  #include <limits.h>  #include <signal.h>  #include <stdlib.h>    volatile sig\_atomic\_t denom;    void sighandle(int s) {  /\* Fix the offending volatile \*/  if (denom == 0) {  denom = 1;  }  }    int main(int argc, char \*argv[]) {  if (argc < 2) {  return 0;  }    char \*end = NULL;  long temp = strtol(argv[1], &end, 10);    if (end == argv[1] || 0 != \*end ||  ((LONG\_MIN == temp || LONG\_MAX == temp) && errno == ERANGE)) {  /\* Handle error \*/  }    denom = (sig\_atomic\_t)temp;  signal(SIGFPE, sighandle);    long result = 100 / (long)denom;  return 0;  } |

| **Compliant Code** |
| --- |
| Apply abort(), quick\_exit(), or \_Exit() to handle computation exception signal. |
| #include <errno.h>  #include <limits.h>  #include <signal.h>  #include <stdlib.h>    int main(int argc, char \*argv[]) {  if (argc < 2) {  return 0;  }    char \*end = NULL;  long denom = strtol(argv[1], &end, 10);    if (end == argv[1] || 0 != \*end ||  ((LONG\_MIN == denom || LONG\_MAX == denom) && errno == ERANGE)) {  /\* Handle error \*/  }    long result = 100 / denom;  return 0;  } |

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

| **Principles(s):**4.Keep It Simple applies that code will be freed from errors and exceptions. 9. Use Effective Quality Assurance Techniques applies helpful identifiers, which may detect exceptions during testing. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-SIG35 |  |
| CodeSonar | 7.3p0 | LANG.STRUCT.RFCESH | Return from Computational Exception Signal Handler |
| Helix QAC | 2023.1 | DF4846, DF4847, DF4848 |  |
| Klocwork | 2023.1 | CERT.STDLIB.SIGNAL |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | PRE30-C | Do not create a universal character name through concatenation |

| **Noncompliant Code** |
| --- |
| Incorrect way to produce a universal character name by token concatenation. |
| #define assign(uc1, uc2, val) uc1##uc2 = val    void func(void) {  int \u0401;  /\* ... \*/  assign(\u04, 01, 4);  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Universal character name created without token concatenation. |
| #define assign(ucn, val) ucn = val    void func(void) {  int \u0401;  /\* ... \*/  assign(\u0401, 4);  /\* ... \*/  } |

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

| **Principles(s):** 4. Keep It Simple prevents errors and warning within the program. 1. Validate Input Data is an area of concern to check correct input |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | universal-character-name-concatenation | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-PRE30 | Fully implemented |
| CodeSonar | 7.3p0 | LANG.PREPROC.PASTE LANG.PREPROC.PASTEHASH | Macro uses ## operator ## follows # operator |
| Helix QAC | 2023.1 | C0905  C++0064,C++0080 |  |
| Klocwork | 2023.1 | MISRA.DEFINE.SHARP |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | EXP46-C | Do not use a bitwise operator with a Boolean-like operand |

| **Noncompliant Code** |
| --- |
| A bitwise & operator is used with the results of an equality-expression |
| if (!(getuid() & geteuid() == 0)) {  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The && operator for the logical operation within the conditional expression |
| if (!(getuid() && geteuid() == 0)) {  /\* ... \*/  } |

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

| **Principles(s):** 4.Keep It Simple applies good coding standard to simple and short coding, which leads to no or few errors or warnings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | inappropriate-bool | Supported indirectly via MISRA C:2012 Rule 10.1 |
| Axivion Bauhaus Suite | 7.2.0 | CertC-EXP46 |  |
| CodeSonar | 7.3p0 | LANG.TYPE.IOT | Inappropriate operand type |
| Coverity | 2017.07 | CONSTANT\_EXPRESSION\_RESULT | Partially implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | FIO47-C | Use valid format strings |

| **Noncompliant Code** |
| --- |
| Mismatch between arguments and conversions may result in unexpected behavior. Error\_type is an int but the parameter expects a string in a printf() function. |
| #include <stdio.h>    void func(void) {  const char \*error\_msg = "Resource not available to user.";  int error\_type = 3;  /\* ... \*/  printf("Error (type %s): %d\n", error\_type, error\_msg);  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Printf() function has matching arguments and conversions. |
| #include <stdio.h>    void func(void) {  const char \*error\_msg = "Resource not available to user.";  int error\_type = 3;  /\* ... \*/  printf("Error (type %d): %s\n", error\_type, error\_msg);    /\* ... \*/  } |

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

| **Principles(s):** 4. Keep It Simple provides code without errors and exceptions. 9 Use Effective Quality Assurance Techniques aid to build robust code with proven testing measures. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO47 | Fully implemented |
| CodeSonar | 7.3p0 | IO.INJ.FMT MISC.FMT MISC.FMTTYPE | Format string injection Format string Format string type error |
| Coverity | 2017.07 | PW | Reports when the number of arguments differs from the number of required arguments according to the format string |
| GCC | 4.3.5 |  | Can detect violations of this recommendation when the -Wformat flag is used |
| Helix QAC | 2023.1 | C0161, C0162, C0163, C0164, C0165, C0166, C0167, C0168, C0169, C0170, C0171, C0172, C0173, C0174, C0175, C0176, C0177, C0178, C0179, C0180, C0184, C0185, C0190, C0191, C0192, C0193, C0194, C0195, C0196, C0197, C0198, C0199, C0200, C0201, C0202, C0204, C0206, C0209  C++3150, C++3151, C++3152, C++3153, C++3154, C++3155, C++3156, C++3157, C++3158, C++3159 |  |
| Klocwork | 2023.1 | SV.FMT\_STR.PRINT\_FORMAT\_MISMATCH.BAD SV.FMT\_STR.PRINT\_FORMAT\_MISMATCH.UNDESIRED SV.FMT\_STR.PRINT\_IMPROP\_LENGTH SV.FMT\_STR.PRINT\_PARAMS\_WRONGNUM.FEW SV.FMT\_STR.PRINT\_PARAMS\_WRONGNUM.MANY SV.FMT\_STR.SCAN\_FORMAT\_MISMATCH.BAD SV.FMT\_STR.SCAN\_FORMAT\_MISMATCH.UNDESIRED SV.FMT\_STR.SCAN\_IMPROP\_LENGTH SV.FMT\_STR.SCAN\_PARAMS\_WRONGNUM.FEW SV.FMT\_STR.SCAN\_PARAMS\_WRONGNUM.MANY SV.FMT\_STR.UNKWN\_FORMAT |  |

### 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.

DevSecOps stands for development, security, operations. DevOps stands for development and operations. The difference is the security. In DevSecOps, many steps may implement added security in the pre-production and production life cycles. Starting at pre-production, Assess and Plan stage responds to new threats and potential security attacks. The architecture of the system provides components that may incorporate software or hardware addons. Design stage may add more security checks such as OWASP. In the build stage, secure coding and building must uphold strong security standards. In the Verify and test stage, certain security vulnerabilities are checked and made trusted. Following into production life cycle, transition and health check apply more rigorous testing and deploying. Monitor and detect stage use log collections, and analytics in order to catch security vulnerabilities. In the Respond stage, blocking attacks and limiting services help to add security. In the final stage of Maintain and stabilize, the application is reassessed for potential cyberattacks and held in stable state. This completes the stages of DevSecOps.

### 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 |
| --- | --- | --- | --- | --- | --- |
| DCL12-C | Low | Unlikely | High | P1 | L3 |
| INT31-C | High | Probable | High | P6 | L2 |
| STR30-C | Low | Likely | Low | P18 | Low |
| STR02-C | High | Likely | Medium | P18 | L1 |
| CON32-C | Medium | Probable | Medium | P8 | L2 |
| DCL03-C | Low | Unlikely | High | P1 | L3 |
| SIG35-C | Low | Unlikely | High | P1 | L3 |
| PRE30-C | Low | Unlikely | Medium | P2 | L3 |
| EXP46-C | Low | Likely | Low | P9 | L2 |
| FIO47-C | High | Unlikely | Medium | P6 | L2 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption in rest is encryption of data that is stored on disk or hard memory. Storage may be solid -sate drives, flash drives, backup media, cloud databases, and so on. Most of the data is meant to be private and describes the client’s information. As a business owner, laws and security practices need to prevent attacks in the event of a data breach when the data is stored. Using role-based accounts authentication and multi factor authentication help to secure data. |
| Encryption at flight | Encryption at flight is encryption of data that is in transit. Data moves fast all the time from client to server. Encryption must apply during data transfer; without security, hackers can view or steal data. A VPN is a tool that provides private, encrypted access to a network. Maintaining a secure network and proper security transfer protocol transfer data securely. |
| Encryption in use | Encryption in use is encryption that prevents data visibility when data is at rest or in motion. Encryption in use covers access to databases and websites, and cloud infrastructure. Security prevents attacks of SQL injections, invalid data entry, cloud encryption keys and services, and so on. Anomalies are detected on websites during encryption in use. Defense in depth is a strategy that adds multiple layers of security features to the customer. Data is then protected strongly even if a layer of defense fails. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication uses user logins, passwords, RSAs, network sign ins, fingerprinting, MFA, and so on to check the identity of the user. A unique set of parameters must match the user. Computer Ip addresses and locations and personal information may also be authenticated as a prerequisite to access. Users need to be identified before the access to sensitive or private data. |
| Authorization | Authorization gains the role to do certain tasks. Authorization limits the amount of input and commands that the user can run. Enforcing policies determines the activities, resources, and services that are in use. Certain security levels give the user the ability to change personal data in a CRUD (create read update delete) application. Users are authorized to view files and database details. The principle of least privileged applies to users so that minimal damage occurs if the user account has access. |
| Accounting | Accounting uses the features from authentication and authorization to keep records. The amount of system time or amount of data tracks the user during a session. Logging these statistics may prove to be an indicator of unauthorized use. Accounting gives a paper trail and evidence of changed work. For security, accounting is like another layer of protection. If authentication or authorization fail, accounting is another layer. |

**\***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 best practice.

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

* Operating system logs

1. Validate Input Data - Validate data that might harm operating system

3. Architect and Design for Security – Operating system need the specify software to encapsulate privacy

5.Default Deny- Operating system code permits those who can get operating system statistics and hardware

6.Adhere to the Principle of Least Privilege – Person who has the least role will command the state of the operating system

7.Sanitize Data Sent to Other Systems – Command shells and relational databases belong to the operating system

8. Practice Defense in Depth – Operating system may require additional security measures to prevent cyberattacks

* Firewall logs

1.Validate Input Data – Validate data that might pass firewall

3. Architect and Design for Security – Firewall acts an additional defense to prevent hackers

5.Default Deny- Firewall permits those who can pass through the firewall

6.Adhere to the Principle of Least Privilege – IP Addresses may track web traffic that access internal sensitive data

7.Sanitize Data Sent to Other Systems – Firewall is part of the sanitize of defense

8. Practice Defense in Depth – Firewall is one step in the many layers of defense, which create the idea of defense in depth

* Anti-malware logs

1.Validate Input Data- Validate data that might be traced in the anti-malware

2.Heed Compiler Warnings – Program errors need to correct to protect against malware

4. Keep It Simple – Code needs to implement clean and secure code

5. Default Deny- Code denies unauthorized entry to everyone unless he has proven identity

7. Sanitize Data Sent to Other Systems – Anti-malware will detect transmissions of malicious code or intent

8. Practice Defense in Depth – Anti-malware is one step in the many layers of defense, which create the idea of defense in depth

9. Use Effective Quality Assurance Techniques – Antimalware is part of the quality standard that identifies and eliminates security vulnerabilities

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 |  |
| 2.0 | 3/13/2023 | 10 Core Security Intro, 10 C++ Coding Standards, Intro | Richard Chang | Richard Chang |
| 3.0 | 4/4/2023 | Risk Assessment, Automated Detection, Automation, Summary of Risk Assessment, Policies of encryption, Map of Principles | Richard Chang | Richard Chang |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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