**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 | Any data coming from an external source or provided by a user must be validated. This data could be intentionally or unintentionally malicious to the code's operation. Validating data leads to software which is more reliable as well as more secure. |
| 1. Heed Compiler Warnings | While code will compile in most cases with active warnings, these warnings serve an important purpose that should not be ignored. Ignoring compiler warnings can lead to situations where the code will not be as secure or as reliable as is necessary for modern software. |
| 1. Architect and Design for Security Policies | Ensuring that a particular piece of software is architected from the ground up for security makes it easier for software creators to make the final piece of software more secure. For example, if as a coder I write a lot of extra code to account for a bad software design, the end application is more fragile and more likely to break or be compromised. Conversely, if the application is architected for security, it makes the coding of that security much easier and more robust. |
| 1. Keep It Simple | Simple mitigations to code vulnerability work best. Long complicated mitigations add complexity to the code and make it more likely that a security hole will exist. |
| 1. Default Deny | Ensuring that the default stance for any code that allows access is deny allows the developer the opportunity to vet the credentials and authorization of a particular user before granting any kind of access. This protects us against situations where a code vulnerability is used to access the software because without any kind of authorization the default action is deny. |
| 1. Adhere to the Principle of Least Privilege | Granting a particular user or external entity the least number of privileges needed keeps the software secure by protecting against account or access compromise. If the authorization of the user or external entity is narrow enough that compromised account cannot be used to perform activities which might have wide ranging consequences |
| 1. Sanitize Data Sent to Other Systems | Being a good steward of data means that other systems will have to work less hard to potentially sanitize any data they receive from this software. This reduces the overall load on the system and ensures that this software cannot be used as an attack vector to send malicious data. |
| 1. Practice Defense in Depth | Defense in depth is the practice of ensuring that we as security practitioners are securing the environment from many different angles using many different techniques. This in practicality creates a series of layered armor which individually would not be sufficient but taken together form a comprehensive and ant-fragile protection from malicious actors. |
| 1. Use Effective Quality Assurance Techniques | Adding quality assurance techniques to the security envelope surrounding the software and software development process is important for several reasons. First, adding things like static code analysis and vulnerability scanning ensures that the code being developed is free of known security issues. Second, adding QA techniques like unit and integration tests ensure that the final code product works and is as secure as it can be. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard ensures that all developers are on the same page when it comes to how to code securely. Developing a coherent standard also makes it possible to scan or test against this standard so that it is easier to find areas where the standard is not being followed. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Use safe data types where possible. Safe data types reduce the risk of overflow attacks. |

| **Noncompliant Code** |
| --- |
| Password entry prompt using a bounded char[] |
| std::char[20] password;  cin << “Enter password: “ << password; |

| **Compliant Code** |
| --- |
| Password entry prompt using a string. |
| std::string password;  cin << “Enter password: “ << password; |

| 1. **Principles(s):** Architect and Design for Security Policies – This principle applies because using the correct variable types needs to be planned for ahead of time. Swapping a variable out last minute can cause major re-work. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | 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 |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Values stored in variables should be checked for overflow conditions. This helps to ensure the code remains secure. |

| **Noncompliant Code** |
| --- |
| This code implements a function that adds two integers with no checking. This is vulnerable to overflow. |
| template <typename T>  T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  T max = std::numeric\_limits<T>::max();  T min = std::numeric\_limits<T>::min();    for (unsigned long int i = 0; i < steps; ++i)  {  result += increment;  }    return result;  } |

| **Compliant Code** |
| --- |
| This code implements a validation function which returns a null response when there is an overflow present. |
| template <typename T>  T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  T max = std::numeric\_limits<T>::max();  T min = std::numeric\_limits<T>::min();    for (unsigned long int i = 0; i < steps; ++i)  {  if (increment > 0 && result > max - increment) {  return null;  }  result += increment;  }    return result;  } |

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

| **Principles(s):** Validate Input Data – This one is pretty cut and dried. The aim is to sanitize input data to ensure that variables are not overflowed or underflowed. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Low | High | L15 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | 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 |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Ensure that data types which are vulnerable to buffer overflows (like arrays and character arrays) are being checked for an overflow condition or that a safer data type is used. |

| **Noncompliant Code** |
| --- |
| This code uses a bounded character array to provide user input. This data type is vulnerable to buffer overflow. |
| std::char[20] user\_input;  std::cout << "Enter a value: ";  std::cin >> user\_input; |

| **Compliant Code** |
| --- |
| This code uses a string type for user input. String types are not vulnerable to buffer overflow. |
| std::string user\_input;  std::cout << "Enter a value: ";  std::cin >> user\_input; |

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

| **Principles(s):** Adopt a secure coding standard – By making a standard to validate inputs and protect against overflows we’re protecting against arbitrary memory locations becoming overwritten and potentially causing a security hole. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | MISC.MEM.NTERM  LANG.MEM.BO LANG.MEM.TO | No space for null terminator  Buffer overrun Type overrun |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Any code which interfaces with an external SQL system and also relies on non-validated external inputs should be sanitized before being sent to the SQL server. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  } |

| **Compliant Code** |
| --- |
| std::regex pattern("\\bor\\b\\s\*(\\w+)\\s\*=\\s\*(\\w+)");  std::smatch match;  // Analyze the incoming c string sql query for a pattern match.  // If matched, then a SQL injection attack is likely.  std::cout << sql.c\_str() << std::endl;  if (std::regex\_search(sql, match, pattern)) {  std::cout << "SQL Injection likely. Aborting query..." << std::endl;  return false;  }  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  } |

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

| **Principles(s):** Sanitize data sent to other systems – The SQL query created using external input will be sent to the SQL system. It is important to sanitize these access requests so that the system is not compromised. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| OWASP ASST | Beta | Automated Software Security Toolkit | A Novel Open Source Web Security Scanner. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| s is dereferenced after it has been deallocated. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** Adopt a secure coding standard – Accessing freed memory needs to be codified as a standard so that programmers do not code freed memory access into the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use assertions to verify variables which should not be null |

| **Noncompliant Code** |
| --- |
| The sk pointer is initialized to tun->sk before checking if tun is a null pointer |
| static unsigned int tun\_chr\_poll(struct file \*file, poll\_table \*wait) {  struct tun\_file \*tfile = file->private\_data;  struct tun\_struct \*tun = \_\_tun\_get(tfile);  struct sock \*sk = tun->sk;  unsigned int mask = 0;    if (!tun)  return POLLERR;    DBG(KERN\_INFO "%s: tun\_chr\_poll\n", tun->dev->name);    poll\_wait(file, &tun->socket.wait, wait);    if (!skb\_queue\_empty(&tun->readq))  mask |= POLLIN | POLLRDNORM;    if (sock\_writeable(sk) ||  (!test\_and\_set\_bit(SOCK\_ASYNC\_NOSPACE, &sk->sk\_socket->flags) &&  sock\_writeable(sk)))  mask |= POLLOUT | POLLWRNORM;    if (tun->dev->reg\_state != NETREG\_REGISTERED)  mask = POLLERR;    tun\_put(tun);  return mask;  } |

| **Compliant Code** |
| --- |
| This solution uses asserts to validate that file, sk, and sk->socket are non-null. |
| static unsigned int tun\_chr\_poll(struct file \*file, poll\_table \*wait) {  assert(file);  struct tun\_file \*tfile = file->private\_data;  struct tun\_struct \*tun = \_\_tun\_get(tfile);  struct sock \*sk;  unsigned int mask = 0;    if (!tun)  return POLLERR;  assert(tun->dev);  sk = tun->sk;  assert(sk);  assert(sk->socket);  /\* The remaining code is omitted because it is unchanged... \*/  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – Catching errors effectively is a form of Quality Assurance through ensuring that errors due to external inputs are being caught and handled effectively. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | LANG.FUNCS.ASSERTS | Not enough assertions |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions |

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

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard – Not handling all exceptions will allow the code to run just fine, until there is an issue. Having a standard which forces exceptions to be handled allows for more secure software. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Aqua Security | Latest | SAST | SAST code scanning allows code to be scanned against a specific set of policies. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | [STD-008-CRG] | Wrap functions that can fail spuriously in a loop |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, reorganize\_data\_structure() is to be used as an argument to thrd\_create(). After reorganizing, the function attempts to replace the head pointer so that it points to the new version. If no other thread has changed the head pointer since it was originally loaded, reorganize\_data\_structure() is intended to exit the thread with a result of true, indicating success. Otherwise, the new reorganization attempt is discarded and the thread is exited with a result of false. However, atomic\_compare\_exchange\_weak() may fail even when the head pointer has not changed. Therefore, reorganize\_data\_structure() may perform the work and then discard it unnecessarily. |
| #include <stdatomic.h>  #include <stdbool.h>    struct data {  struct data \*next;  /\* ... \*/  };    extern void cleanup\_data\_structure(struct data \*head);    int reorganize\_data\_structure(void \*thread\_arg) {  struct data \*\_Atomic \*ptr\_to\_head = thread\_arg;  struct data \*old\_head = atomic\_load(ptr\_to\_head);  struct data \*new\_head;  bool success;    /\* ... Reorganize the data structure ... \*/    success = atomic\_compare\_exchange\_weak(ptr\_to\_head,  &old\_head, new\_head);  if (!success) {  cleanup\_data\_structure(new\_head);  }  return success; /\* Exit the thread \*/  } |

| **Compliant Code** |
| --- |
| To recover from spurious failures, a loop must be used. However, atomic\_compare\_exchange\_weak() might fail because the head pointer changed, or the failure may be spurious. In either case, the thread must perform the work repeatedly until the compare-and-exchange succeeds, as shown in this compliant solution: |
| #include <stdatomic.h>  #include <stdbool.h>  #include <stddef.h>    struct data {  struct data \*next;  /\* ... \*/  };    extern void cleanup\_data\_structure(struct data \*head);    int reorganize\_data\_structure(void \*thread\_arg) {  struct data \*\_Atomic \*ptr\_to\_head = thread\_arg;  struct data \*old\_head = atomic\_load(ptr\_to\_head);  struct data \*new\_head = NULL;  struct data \*saved\_old\_head;  bool success;    do {  if (new\_head != NULL) {  cleanup\_data\_structure(new\_head);  }  saved\_old\_head = old\_head;    /\* ... Reorganize the data structure ... \*/    } while (!(success = atomic\_compare\_exchange\_weak(  ptr\_to\_head, &old\_head, new\_head  )) && old\_head == saved\_old\_head);  return success; /\* Exit the thread \*/  } |

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

| **Principles(s): Defense in depth – By wrapping the code in a loop we’re creating another level of protection against failure.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Aqua Security | Latest | SAST | SAST code scanning allows code to be scanned against a specific set of policies. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Environment | [STD-009-CRG] | Do not call system() |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the system() function is used to execute any\_cmd in the host environment. |
| #include <string.h>  #include <stdlib.h>  #include <stdio.h>    enum { BUFFERSIZE = 512 };    void func(const char \*input) {  char cmdbuf[BUFFERSIZE];  int len\_wanted = snprintf(cmdbuf, BUFFERSIZE,  "any\_cmd '%s'", input);  if (len\_wanted >= BUFFERSIZE) {  /\* Handle error \*/  } else if (len\_wanted < 0) {  /\* Handle error \*/  } else if (system(cmdbuf) == -1) {  /\* Handle error \*/  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the call to system() is replaced with a call to execve(). The exec family of functions does not use a full shell interpreter, so it is not vulnerable to command-injection attacks, such as the one illustrated in the noncompliant code example. |
| #include <sys/types.h>  #include <sys/wait.h>  #include <unistd.h>  #include <errno.h>  #include <stdlib.h>    void func(char \*input) {  pid\_t pid;  int status;  pid\_t ret;  char \*const args[3] = {"any\_exe", input, NULL};  char \*\*env;  extern char \*\*environ;    /\* ... Sanitize arguments ... \*/    pid = fork();  if (pid == -1) {  /\* Handle error \*/  } else if (pid != 0) {  while ((ret = waitpid(pid, &status, 0)) == -1) {  if (errno != EINTR) {  /\* Handle error \*/  break;  }  }  if ((ret == 0) ||  !(WIFEXITED(status) && !WEXITSTATUS(status))) {  /\* Report unexpected child status \*/  }  } else {  /\* ... Initialize env as a sanitized copy of environ ... \*/  if (execve("/usr/bin/any\_cmd", args, env) == -1) {  /\* Handle error \*/  \_Exit(127);  }  }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Cert-env33-c | Checked by clang-tidy |
| Coverity | 2017.07 | DONT\_CALL | Implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Ouput | [STD-010-CRG] | Exclude user input from format strings |

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

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

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

| **Principles(s):** Validate Input Data – By ensuring that user inputted code is not blindly included in format strings, we’re validating. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | TAINTED\_STRING | Implemented |
| CodeSonar | 7.3p0 | IO.INJ.FMT.MISC.FMT | Format string injection  Format String |

### 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.

The DevSecOps methodology forms a continuous feedback loop to constantly improve security of the coding and deployment pipleline. By incorporating lessons-learned as well as active security issues back into the pipeline through additional security tools or processes ensures that the pipeline and the associated software remains secure through the lifecycle of the software.

Including code scanning as part of the code submission process ensures that as code is being submitted it is also being tested through static code analysis as well as unit tests so that no bad code gets through the code commit process.

The second level of gates should include some kind of end-to-end or integration testing for the software either as part of a pull request or as part of an automated suite of tests after code is committed. Pull request reviews should be implemented to allow other engineers to examine the code before it is allowed to progress from a higher level branch such as a feature or hotfix branch to a lower level one, like main.

In addition to the tests for code completeness and security, operational security tools should be put into place to ensure that the code remains secure while running. Utilizing a holistic logging solution in tandem with a security tool which can analyze those logs for evidence of a breach is an important part of the DevSecOps feedback loop.

### 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 | L2 |
| STD-002-CPP | High | High | Low | High | L15 |
| STD-003-CPP | High | High | Low | High | L15 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CRG | Low | Unlikely | Medium | P2 | L3 |
| STD-009-CRG | High | Probable | Medium | P12 | L1 |
| STD-010-CRG | High | Likely | Medium | P18 | 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 | Encryption at rest is the state of data when it is not being used and is inactively residing on whatever storage media is used for the application. Ensuring that data is encrypted at rest is important from a physical security perspective. If any malicious actor were able to exfiltrate the physical drives that the data resides on, they would be able to read the data in plaintext if it is not encrypted. |
| Encryption at flight | Encryption in-flight refers to the state of data as it is moving from one system to another over a network layer. Because the data often travels over public networks, encrypting the data is of utmost importance. For example, an HTTPS SSL or TLS connection between a webserver and a web page in your browser encrypts the traffic from end to end. Secure online shopping would not be possible without encryption in flight. |
| Encryption in use | Encryption in use refers to the encrypting of data even when it is actively being utilized by an application. Only users which have the correct level of authorization and are appropriately authenticated can decrypt the data to utilize it. This prevents attacks in which an insider threat has direct access to the application environment. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Without authentication it is impossible to know whether a user is who they say they are. Authentication is the process of doing just this, verifying a user identity. This can be done in a number of ways and the most secure is to include another factor of authentication other than a password. Two Factor Authentication uses another method which identifies the user such as a cell phone or e-mail address to add defense in depth to the authentication process.  Logins to the application will require two factor authentication.  Passwords will be enforced to have at least 8 characters and include special characters.  Passwords will be rotated on a 90 day basis. |
| Authorization | Authorization is what happens after a user has been authenticated. Based on the users authentication result, that user can then be authorized to perform certain functions within an application. The concept of least privilege applies here as users should be given the least amount of privilege that their job requires. Only applying the least privilege ensures that even if an account is compromised the blast radius of the compromise can be reduced.  The principle of least privilege will be applied to user roles. |
| Accounting | Once a user is authenticated and authorized, they need to perform actions in the software. Properly auditing or logging the users activity ensures a ‘paper trail’ so that if anything were to be compromised the proper user can be attributed to the compromise. Activity logging can also be used with external tooling to look for common patterns of compromise and provide an early warning of malicious activity.  All user activity will be logged:   * New user creation. * All file access. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 06/11/2023 | Updated policies and explanations | Michael Ross |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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