**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_heading=h.gjdgxs)

[Purpose 2](#_heading=h.30j0zll)

[Scope 2](#_heading=h.1fob9te)

[Module Three Milestone 2](#_heading=h.3znysh7)

[Ten Core Security Principles 2](#_heading=h.4d34og8)

[C/C++ Ten Coding Standards 3](#_heading=h.2s8eyo1)

[Coding Standard 1 4](#_heading=h.3rdcrjn)

[Coding Standard 2 5](#_heading=h.26in1rg)

[Coding Standard 3 6](#_heading=h.lnxbz9)

[Coding Standard 4 7](#_heading=h.35nkun2)

[Coding Standard 5 8](#_heading=h.1ksv4uv)

[Coding Standard 6 9](#_heading=h.2jxsxqh)

[Coding Standard 7 10](#_heading=h.z337ya)

[Coding Standard 8 11](#_heading=h.28h4qwu)

[Coding Standard 9 13](#_heading=h.nmf14n)

[Coding Standard 10 14](#_heading=h.37m2jsg)

[Defense-in-Depth Illustration 15](#_heading=h.1mrcu09)

[Project One 15](#_heading=h.46r0co2)

[1.](#_heading=h.2lwamvv) Revise the C/C++ Standards 15

[2.](#_heading=h.111kx3o) Risk Assessment 15

[3.](#_heading=h.3l18frh) Automated Detection 15

[4.](#_heading=h.206ipza) Automation 15

[5.](#_heading=h.4k668n3) Summary of Risk Assessments 16

[6.](#_heading=h.2zbgiuw) Create Policies for Encryption and Triple A 16

[7.](#_heading=h.1egqt2p) Map the Principles 17

[Audit Controls and Management 18](#_heading=h.3ygebqi)

[Enforcement 18](#_heading=h.2dlolyb)

[Exceptions Process 18](#_heading=h.sqyw64)

[Distribution 19](#_heading=h.3cqmetx)

[Policy Change Control 19](#_heading=h.1rvwp1q)

[Policy Version History 19](#_heading=h.4bvk7pj)

[Appendix A Lookups 19](#_heading=h.2r0uhxc)

[Approved C/C++ Language Acronyms 19](#_heading=h.1664s55)

## 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 input data is one of the most crucial security principles. Any time the user submits information, we must verify that what they submitted is within the bounds of what data type is being used. For example, if the required data type is an integer, the user should not be able to enter a string, and they must not break the bounds of the integer through an underflow or overflow. Not validating input data could lead to buffer overflows, allowing the user to access potentially sensitive areas of the software. |
| 1. Heed Compiler Warnings | When the compiler tells the developer that something is wrong, in most cases, it may be for good reason. For example, popular modern compilers such as GCC and MSVC automatically enable buffer overflow protection and will issue warnings when it may occur in code. If the developer does not heed these warnings, they risk the security and stability of their application. |
| 1. Architect and Design for Security Policies | Writing standards for security practices through security policies is a good way of ensuring that throughout the development process, said security practices are followed and referenced. |
| 1. Keep It Simple | By making a function more or overly complex, it risks the developer and future developers not being able to tell what exactly is happening behind many layers of abstraction, and could potentially lead to memory safety issues. It is akin to building a Jenga tower. |
| 1. Default Deny | In the world of network security, default deny is a practice where traffic is denied by default unless the operator explicitly allows traffic given certain rules. The same concept can be applied to software security by restricting the execution of pieces of software. |
| 1. Adhere to the Principle of Least Privilege | The Principle of Least Privilege is where the software is given the least amount of privileges necessary to complete the task it needs to. For example, a piece of software that manages a single file does not need access to the user’s home folder–it ONLY needs access to that single file. Likewise, if an application does not require administrator or root access, it should not be given those special privileges lest there be a security hole. |
| 1. Sanitize Data Sent to Other Systems | There may be cases where other systems that were assumed to be safe are compromised, and as such, it is not a good idea to take whatever data is being given at face value. This is especially true for systems where any user can interact with it, such as a web portal. |
| 1. Practice Defense in Depth | Having layered protections such as a firewall, antivirus, and antimalware solutions, as well as keeping a system up-to-date, is an example of defense in depth where multiple solutions are deployed to cover the largest attack vectors possible. |
| 1. Use Effective Quality Assurance Techniques | Bugs such as invalidated input and buffer overflows that are not caught by the compiler may be caught by effective quality assurance techniques. For example, talking through the logic of a function of an application may allow multiple developers to think of edge cases for input validation, memory safety problems, and more. |
| 1. Adopt a Secure Coding Standard | A secure coding standard is a reference for the security of an ecosystem of software development. An example is the SEI CERT C++ Coding Standard which is used by the wide secure coding community. It outlines rules to follow when writing software. |

### 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 | Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses an unscoped enumeration but provides a fixed underlying int type allowing any value from the parameter to be converted to a valid enumeration value. |
| enum EnumType : int {  First,  Second,  Third  };    void f(int intVar) {  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 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 7.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2022.3 | C++3013 |  |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-INT50-a | 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 | 7.20 | V1016 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Guarantee that container indices and iterators are within the valid range. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows a function, insert\_in\_table(), that has two int parameters, pos and value, both of which can be influenced by data originating from untrusted sources. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the parameter pos is declared as size\_t, which prevents the passing of negative arguments. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

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

| **Principles(s):**  Validate input  Heed compiler warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | overflow\_upon\_dereference |  |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.STRUCT.PARITH | Buffer overrun  Buffer underrun  Type overrun  Type underrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object  Pointer Arithmetic |
| Helix QAC | 2022.3 | C++2891, C++3139, C++3140 |  |
| Klocwork | 2022.3 | ABV.ANY\_SIZE\_ARRAY  ABV.GENERAL  ABV.GENERAL.MULTIDIMENSION  ABV.STACK  ABV.TAINTED  SV.TAINTED.ALLOC\_SIZE  SV.TAINTED.CALL.INDEX\_ACCESS  SV.TAINTED.CALL.LOOP\_BOUND  SV.TAINTED.INDEX\_ACCESS |  |
| LDRA tool suite | 9.7.1 | 45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X | Partially implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2022b | CERT C++: CTR50-CPP | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |
| PRQA QA-C++ | 4.4 | 2891, 3139, 3140 |  |
| PVS-Studio | 7.20 | V781 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Do not attempt to create a std::string from a null pointer. |

| **Noncompliant Code** |
| --- |
| A std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| The rules from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    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):**  Heed compiler warnings  Use effective quality assurance techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | assert\_failure |  |
| CodeSonar | 7.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Helix QAC | 2022.3 | C++4770, C++4771, C++4772, C++4773, C++4774 |  |
| Klocwork | 2022.3 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finger | R2022b | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule fully covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. |
| #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):**  Architect and design for security policies  Use effective quality assurance techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| CodeSonar | 7.1p0 | ALLOC.UAF | Use after free |
| Compass/ROSE |  |  |  |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Helix QAC | 2022.3 | C4866, C4867, C4868, C4871, C4872, C4873  C++3339, C++4303, C++4304, C++4866, C++4867, C++4868, C++4871, C++4872, C++4873 |  |
| Klocwork | 2022.3 | UFM.DEREF.MIGHT  UFM.DEREF.MUST  UFM.FFM.MIGHT  UFM.FFM.MUST  UFM.RETURN.MIGHT  UFM.RETURN.MUST  UFM.USE.MIGHT  UFM.USE.MUST |  |
| LDRA tool suite | 9.7.1 | 51 D, 484 S, 112 D | Partially implemented |
| Parasoft C/C++test | 2022.1 | CERT\_C-MEM30-a | Do not use resources that have been freed |
| Parasoft Insure++ |  |  | Runtime analysis |
| PC-lint Plus | 1.4 | 449, 2434 | Fully supported |
| Polyspace Bug Finder | R2022b | CERT C: Rule MEM30-C | Checks for:   * Accessing previously freed pointer * Freeing previously freed pointer   Rule partially covered. |
| PRQA QA-C | 9.7 | 2731, 2732, 2733 |  |
| PRQA QA-C++ | 4.4 | 3339, 4303, 4304 |  |
| PVS-Studio | 7.20 | V586, V774 |  |
| Splint | 3.1.1 |  |  |
| TrustInSoft Analyzer | 1.38 | dangling\_pointer | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/0d556bb8)). |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Detect and handle memory allocation errors. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program. |
| #include <cstring>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new int[size];  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new (std::nothrow) int[size];  if (!copy) {  // Handle error  return;  }  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

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

| **Principles(s):**  Validate input  Heed compiler warnings  Architect and design for secure policies  Use effective quality assurance techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  |  |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| Klocwork | 2022.3 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2022b | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| PRQA QA-C++ | 4.4 | 3225, 3226, 3227, 3228, 3229, 4632 |  |
| PVS-Studio | 7.20 | V522, V668 |  |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #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** |
| --- |
| This portable compliant solution uses static\_assert. |
| #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):**  Use effective quality assurance techniques |
| --- |

**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 | 7.1p0 | (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** | STD-007-CPP | Do not rely on indeterminate values of errno. |

| **Noncompliant Code** |
| --- |
| The handler() function in this noncompliant code example attempts to restore default handling for the signal indicated by signum. If the request to set the signal to default can be honored, the signal() function returns the value of the signal handler for the most recent successful call to the signal() function for the specified signal. Otherwise, a value of SIG\_ERR is returned and a positive value is stored in errno. Unfortunately, the value of errno is indeterminate because the handler() function is called when an external signal is raised, so any attempt to read errno (for example, by the perror() function) is undefined behavior: |
| #include <signal.h>  #include <stdlib.h>  #include <stdio.h>    typedef void (\*pfv)(int);    void handler(int signum) {  pfv old\_handler = signal(signum, SIG\_DFL);  if (old\_handler == SIG\_ERR) {  perror("SIGINT handler"); /\* Undefined behavior \*/  /\* Handle error \*/  }  }    int main(void) {  pfv old\_handler = signal(SIGINT, handler);  if (old\_handler == SIG\_ERR) {  perror("SIGINT handler");  /\* Handle error \*/  }    /\* Main code loop \*/    return EXIT\_SUCCESS;  } |

| **Compliant Code** |
| --- |
| This compliant solution does not reference errno and does not return from the signal handler if the signal() call fails: |
| #include <signal.h>  #include <stdlib.h>  #include <stdio.h>    typedef void (\*pfv)(int);    void handler(int signum) {  pfv old\_handler = signal(signum, SIG\_DFL);  if (old\_handler == SIG\_ERR) {  abort();  }  }    int main(void) {  pfv old\_handler = signal(SIGINT, handler);  if (old\_handler == SIG\_ERR) {  perror("SIGINT handler");  /\* Handle error \*/  }    /\* Main code loop \*/    return EXIT\_SUCCESS;  } |

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

| **Principles(s):**  Validate input  Heed compiler warnings  Architect and design for secure policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ERR32 |  |
| Compass/ROSE |  |  | Could detect violations of this rule by looking for signal handlers that themselves call signal(). A violation is reported if the call fails and the handler therefore checks errno. A violation also exists if the signal handler modifies errno without first copying its value elsewhere |
| Coverity | 2017.07 | MISRA C 2012 Rule 22.8  MISRA C 2012 Rule 22.9  MISRA C 2012 Rule 22.10 | Implemented |
| Helix QAC | 2022.3 | C2031, C4781, C4782, C4783  C++4781, C++4782, C++4783 |  |
| Klocwork | 2022.3 | MISRA.INCL.SIGNAL.2012  MISRA.STDLIB.SIGNAL |  |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced enforcement |
| Parasoft C/C++test | 2022.1 | CERT\_C-ERR32-a | Properly use errno value |
| Polyspace Bug Finder | R2022b | CERT C: Rule ERR32-C | Checks for misuse of errno in a signal handler (rule fully covered) |
| PRQA QA-C | 9.7 | 2031 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Arrays** | STD-008-CPP | Do not form or use out-of-bounds pointers or array subscripts. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the function f() attempts to validate the index before using it as an offset to the statically allocated table of integers. However, the function fails to reject negative index values. When index is less than zero, the behavior of the addition expression in the return statement of the function is undefined behavior (Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that does not point into, or just beyond, the same array object). On some implementations, the addition alone can trigger a hardware trap. On other implementations, the addition may produce a result that when dereferenced triggers a hardware trap. Other implementations still may produce a dereferenceable pointer that points to an object distinct from table. Using such a pointer to access the object may lead to information exposure or cause the wrong object to be modified. |
| enum { TABLESIZE = 100 };    static int table[TABLESIZE];    int \*f(int index) {  if (index < TABLESIZE) {  return table + index;  }  return NULL;  } |

| **Compliant Code** |
| --- |
| One compliant solution is to detect and reject invalid values of index if using them in pointer arithmetic would result in an invalid pointer. |
| enum { TABLESIZE = 100 };    static int table[TABLESIZE];    int \*f(int index) {  if (index >= 0 && index < TABLESIZE) {  return table + index;  }  return NULL;  } |

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

| **Principles(s):**  Validate input  Heed compiler warnings  Architect and design for secure policies  Use effective quality assurance techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | array-index-range  array-index-range-constant  null-dereferencing  pointered-deallocation  return-reference-local | Partially checked  Can detect all accesses to invalid pointers as well as array index out-of-bounds accesses and prove their absence.  This rule is only partially checked as invalid but unused pointers may not be reported. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ARR30 | Can detect out-of-bound access to array / buffer |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU  LANG.STRUCT.PARITH  LANG.STRUCT.PBB  LANG.STRUCT.PPE  BADFUNC.BO.\* | Buffer overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun  Pointer Arithmetic  Pointer before beginning of object  Pointer past end of object  A collection of warning classes that report uses of library functions prone to internal buffer overflows. |
| Compass/ROSE |  |  | Could be configured to catch violations of this rule. The way to catch the noncompliant code example is to first hunt for example code that follows this pattern:  for (LPWSTR pwszTemp = pwszPath + 2; \*pwszTemp != L'\\';  \*pwszTemp++;)  In particular, the iteration variable is a pointer, it gets incremented, and the loop condition does not set an upper bound on the pointer. Once this case is handled, ROSE can handle cases like the real noncompliant code example, which is effectively the same semantics, just different syntax |
| Coverity | 2017.07 | OVERRUN  NEGATIVE\_RETURNS  ARRAY\_VS\_SINGLETON  BUFFER\_SIZE | Can detect the access of memory past the end of a memory buffer/array  Can detect when the loop bound may become negative  Can detect the out-of-bound read/write to array allocated statically or dynamically  Can detect buffer overflows |
| Cppcheck | 1.66 | arrayIndexOutOfBounds, outOfBounds, negativeIndex, arrayIndexThenCheck, arrayIndexOutOfBoundsCond, possibleBufferAccessOutOfBounds | Context sensitive analysis of array index, pointers, etc.  Array index out of bounds  Buffer overflow when calling various functions memset,strcpy,..  Warns about condition (a[i] == 0 && i < unknown\_value) and recommends that (i < unknown\_value && a[i] == 0) is used instead  Detects unsafe code when array is accessed before/after it is tested if the array index is out of bounds |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Signals** | STD-009-CPP | Call only asynchronous-safe functions within signal handlers. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the C standard library functions fputs() and free() are called from the signal handler via the function log\_message(). Neither function is asynchronous-safe. |
| #include <signal.h>  #include <stdio.h>  #include <stdlib.h>    enum { MAXLINE = 1024 };  char \*info = NULL;    void log\_message(void) {  fputs(info, stderr);  }    void handler(int signum) {  log\_message();  free(info);  info = NULL;  }    int main(void) {  if (signal(SIGINT, handler) == SIG\_ERR) {  /\* Handle error \*/  }  info = (char \*)malloc(MAXLINE);  if (info == NULL) {  /\* Handle Error \*/  }    while (1) {  /\* Main loop program code \*/    log\_message();    /\* More program code \*/  }  return 0;  } |

| **Compliant Code** |
| --- |
| Signal handlers should be as concise as possible—ideally by unconditionally setting a flag and returning. This compliant solution sets a flag of type volatile sig\_atomic\_t and returns; the log\_message() and free() functions are called directly from main(). |
| #include <signal.h>  #include <stdio.h>  #include <stdlib.h>    enum { MAXLINE = 1024 };  volatile sig\_atomic\_t eflag = 0;  char \*info = NULL;    void log\_message(void) {  fputs(info, stderr);  }    void handler(int signum) {  eflag = 1;  }    int main(void) {  if (signal(SIGINT, handler) == SIG\_ERR) {  /\* Handle error \*/  }  info = (char \*)malloc(MAXLINE);  if (info == NULL) {  /\* Handle error \*/  }    while (!eflag) {  /\* Main loop program code \*/    log\_message();    /\* More program code \*/  }    log\_message();  free(info);  info = NULL;    return 0;  } |

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

| **Principles(s):**  Keep it simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | signal-handler-unsafe-call | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-SIG30 |  |
| CodeSonar | 7.1p0 | BADFUNC.SIGNAL | Use of signal |
| Compass/ROSE |  |  | Can detect violations of the rule for single-file programs |
| Helix QAC | 2022.3 | C2028, C2030 |  |
| LDRA tool suite | 9.7.1 | 88 D, 89 D | Partially implemented |
| Parasoft C/C++test | 2022.1 | CERT\_C-SIG30-a | Properly define signal handlers |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Concurrency** | STD-010-CPP | Clean up thread-specific storage. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, each thread dynamically allocates storage in the get\_data() function, which is then associated with the global key by the call to tss\_set() in the add\_data() function. This memory is subsequently leaked when the threads terminate. |
| #include <threads.h>  #include <stdlib.h>    /\* Global key to the thread-specific storage \*/  tss\_t key;  enum { MAX\_THREADS = 3 };    int \*get\_data(void) {  int \*arr = (int \*)malloc(2 \* sizeof(int));  if (arr == NULL) {  return arr; /\* Report error \*/  }  arr[0] = 10;  arr[1] = 42;  return arr;  }    int add\_data(void) {  int \*data = get\_data();  if (data == NULL) {  return -1; /\* Report error \*/  }    if (thrd\_success != tss\_set(key, (void \*)data)) {  /\* Handle error \*/  }  return 0;  }    void print\_data(void) {  /\* Get this thread's global data from key \*/  int \*data = tss\_get(key);    if (data != NULL) {  /\* Print data \*/  }  }    int function(void \*dummy) {  if (add\_data() != 0) {  return -1; /\* Report error \*/  }  print\_data();  return 0;  }    int main(void) {  thrd\_t thread\_id[MAX\_THREADS];    /\* Create the key before creating the threads \*/  if (thrd\_success != tss\_create(&key, NULL)) {  /\* Handle error \*/  }    /\* Create threads that would store specific storage \*/  for (size\_t i = 0; i < MAX\_THREADS; i++) {  if (thrd\_success != thrd\_create(&thread\_id[i], function, NULL)) {  /\* Handle error \*/  }  }    for (size\_t i = 0; i < MAX\_THREADS; i++) {  if (thrd\_success != thrd\_join(thread\_id[i], NULL)) {  /\* Handle error \*/  }  }    tss\_delete(key);  return 0;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, each thread explicitly frees the thread-specific storage returned by the tss\_get() function before terminating: |
| #include <threads.h>  #include <stdlib.h>    /\* Global key to the thread-specific storage \*/  tss\_t key;    int function(void \*dummy) {  if (add\_data() != 0) {  return -1; /\* Report error \*/  }  print\_data();  free(tss\_get(key));  return 0;  }    /\* ... Other functions are unchanged \*/ |

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

| **Principles(s):**  Architect and design for secure policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | [Insert text.] | Supported, but no explicit checker |
| CodeSonar | 7.1p0 | ALLOC.LEAK | Leak |
| Coverity | 2017.07 | ALLOC\_FREE\_MISMATCH | Partially implemented, correct implementation is more involved |
| Helix QAC | 2022.3 | C1780, C1781, C1782, C1783, C1784 |  |
| Parasoft C/C++test | 2022.1 | CERT\_C-CON30-a | Ensure resources are freed |
| Polyspace Bug Finder | R2022b | CERT C: Rule CON30-C | Checks for thread-specific memory leak (rule fully covered) |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[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 |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-008-CPP | High | Likely | High | P9 | L2 |
| STD-009-CPP | High | Likely | Medium | P18 | L1 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

### 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 a form of encryption where the encrypted data is at rest, such as in a database or cold storage. One example of this is paid invoices with a customer’s credit card information, where in the event of a server break-in, that data remains safe because it is encrypted. |
| Encryption at flight | “Encryption in Flight” is a form of encryption where the encrypted data is actively being used, such as in an HTTPS request between the server and client. If a client were to connect to a bank account from an unsecure network (public), an attacker may intercept that traffic to siphon credentials. Actively encrypting data between the client and server prevents this from happening. |
| Encryption in use | “Encryption in Use” is where encryption is applied at every stage of the process, never becoming insecure. This can be done through hardware-based environments or, more recently, software-based environments (https://security.stackexchange.com/a/245984). |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is where the identity of the user is proven. The policy is applied to ensure that the wrong person does not have the credentials of someone else. |
| Authorization | Authorization is the process of granting the authenticated person access to systems they only need access for. This is to prevent abuse and damage of systems. |
| Accounting | Accounting is the process of auditing and logging authentication and authorization to ensure policies are being applied correctly. |

**\***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 | 09/14/2022 | 3-2 Milestone: Coding Standards | Andrew Black | Ivan Gappy |
| 1.2 | 10/08/2022 | 6-2 Project One: Security Policy | Andrew Black |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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