**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

# Written by: Leah Marshall

## Contents

[Overview 2](#_gjdgxs)

[Purpose 2](#_30j0zll)

[Scope 2](#_1fob9te)

[Module Three Milestone 2](#_3znysh7)

[Ten Core Security Principles 2](#_2et92p0)

[C/C++ Ten Coding Standards 3](#_3dy6vkm)

[Coding Standard 1 4](#_1t3h5sf)

[Coding Standard 2 5](#_4d34og8)

[Coding Standard 3 6](#_2s8eyo1)

[Coding Standard 4 7](#_17dp8vu)

[Coding Standard 5 8](#_3rdcrjn)

[Coding Standard 6 9](#_26in1rg)

[Coding Standard 7 10](#_lnxbz9)

[Coding Standard 8 11](#_35nkun2)

[Coding Standard 9 13](#_1ksv4uv)

[Coding Standard 10 14](#_44sinio)

[Defense-in-Depth Illustration 15](#_2jxsxqh)

[Project One 15](#_z337ya)

[1.](#_3j2qqm3) Revise the C/C++ Standards 15

[2.](#_1y810tw) Risk Assessment 15

[3.](#_4i7ojhp) Automated Detection 15

[4.](#_2xcytpi) Automation 15

[5.](#_1ci93xb) Summary of Risk Assessments 16

[6.](#_3whwml4) Create Policies for Encryption and Triple A 16

[7.](#_2bn6wsx) Map the Principles 17

[Audit Controls and Management 18](#_qsh70q)

[Enforcement 18](#_3as4poj)

[Exceptions Process 18](#_1pxezwc)

[Distribution 19](#_49x2ik5)

[Policy Change Control 19](#_2p2csry)

[Policy Version History 19](#_147n2zr)

[Appendix A Lookups 19](#_3o7alnk)

[Approved C/C++ Language Acronyms 19](#_23ckvvd)

## 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 | This principle includes making sure only the correct data enters the information system. Malicious code or any other security threats are made sure to be properly mitigated. |
| 1. Heed Compiler Warnings | Compiler warnings are there to notify a developer of any potential error. Though warnings don’t prevent code from compiling, it’s still important to heed these warnings as they can potentially become a security risk. |
| 1. Architect and Design for Security Policies | When designing a system it’s important to take into consideration security policies at every stage of the development process. For instance, taking into account different authorization levels, sub systems, etc. This will allow for potential vulnerabilities or issues to be noticed a lot earlier. |
| 1. Keep It Simple | The more complex a security system is, the more likely there will be errors and security vulnerabilities. By minimizing the “fluff”, you’re creating a system that’s easier to implement and more likely to be effective. |
| 1. Default Deny | By default, access should be denied and permissions should only be given when explicitly stated. This minimizes the risk of sensitive data being accessed by the wrong party. |
| 1. Adhere to the Principle of Least Privilege | Similar to default deny, users should only be granted the minimum level of access necessary to complete their task. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data before it’s passed to other systems allow potential issues such as calls made out of context, SQL injection attacks, or unused functions to be cleared to further avoid these being used as security vulnerabilities. |
| 1. Practice Defense in Depth | Several layers of security should be setup in order for the system to have as much protection as possible. If one layer if compromised, there are still other layers there to help minimize damage done. |
| 1. Use Effective Quality Assurance Techniques | Penetration testing, code reviews, security audits, and vulnerability scanning are all QA techniques that allow the organization to find and handle potential risks before attackers can. |
| 1. Adopt a Secure Coding Standard | Adopting secure coding standards allow developers to reduce the risk of introducing risks into the system and thus improving security as a whole. Secure coding standards such as input validation and secure authentication are just a few that promote secure systems. |

### 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] | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| Two different translation units define a class of the same name with differing definitions. |
| // a.cpp  struct S {  int a;  };  // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| Use a header file to introduce the object into both translation units. |
| // S.h  struct S {  int a;  };  // a.cpp  #include "S.h"  // b.cpp  #include "S.h" |

| **Principles(s):** (4) Keep it Simple; (3) Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially Checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully implemented |
| Polyspace Bug Finder | R2023b | CERT C++: DCL60-CPP | Checks for inline constraints not respected (rule partially covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized memory. |

| **Noncompliant Code** |
| --- |
| Uninitialized local variable is evaluated as part of an expression to print its value, resulting in undefined behavior. |
| # include <iostream>  void f() {  int i;  std::cout<<i;  } |

| **Compliant Code** |
| --- |
| The object is initialized prior to printing its value. |
| # include <iostream>  void f() {  int i = 0;  std::cout<<i;  } |

| **Principles(s):** (1) Validate Input Data; (4) Keep it Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | uninitialized-read | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.STRUCT.RPL LANG.MEM.UVAR | Return pointer to local  Uninitialized variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 |  |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| Because std::getenv() returns a nullptr on failure, the code can lead to undefined behavior when the variable does not exist. |
| # include <cstdlib>  #include <string>  void f() {  std::string tmp(std::entenv(“TMP”);  if (!tmp.empty()){  //…  }  } |

| **Compliant Code** |
| --- |
| The results from std::getenv() are checked for null before the std::string is constructed. |
| # include <cstdlib>  # include <string>  void f() {  const char \*tmpPtrVal = std::getenv(“TMP”);  std::string tmp(tmpPtrVal ? tmpPtrVal : “”);  if (!tmp.empty()){  //…  }  } |

| **Principles(s):** (2) Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | assert\_failure |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | DF4770, DF4771, DF4772, DF4773, DF4774 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.1 | 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 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| Untrusted data may maliciously alter the query, resulting in information leaks or data modification. |
| String pwd = hashPassword(password);  String sqlString = "select \* from db\_user where username=" +  username + " and password =" + pwd;  PreparedStatement stmt = connection.prepareStatement(sqlString); |

| **Compliant Code** |
| --- |
| Using a parametric query with a ? character as a placeholder for the argument. The length of the username is also checked, which prevents an attacker from submitting a super long username. |
| // Validate username length  **if** (username.length() > 8) {  // Handle error  }    String sqlString =  "select \* from db\_user where username=? and password=?"; |

| **Principles(s):** (1) Validate Input Data; (7) Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_ FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| The s is dereferenced after being deallocated. If this access results in write after free, the vulnerability can be exploited to run arbitrary code. |
| #include <new>  **struct** S {  **void** f();  };  **void** g() noexcept(**false**) {  S \*s = **new** S;  // ...  **delete** s;  // ...  s->f();  } |

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

| **Principles(s):** (2) Heed Compiler Warnings; (9) 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](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | dangling\_pointer\_use |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MEM50 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| The assert() macro asserts 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** |
| --- |
| static\_assert allows for incorrect assumptions to be diagnosed at compile time instead of a silent malfunction. |
| #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"); |

| **Principles(s):** (2) Heed Compiler Warnings; (3) Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL03 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| The call to f() may result in a call to std::terminate() because throwing\_func() may throw an exception |
| #include <cstdlib>  **void** throwing\_func() noexcept(**false**);  **void** f() { // Not invoked by the program except as an exit handler.  throwing\_func();  }  **int** main() {  **if** (0 != std::**atexit**(f)) {  // Handle error  }  // ...  } |

| **Compliant Code** |
| --- |
| f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>  **void** throwing\_func() noexcept(**false**);  **void** f() { // Not invoked by the program except as an exit handler.  **try** {  throwing\_func();  } **catch** (...) {  // Handle error  }  }  **int** main() {  **if** (0 != std::**atexit**(f)) {  // Handle error  }  // ...  } |

| **Principles(s):** (9) Use Effective Quality Assurance Techniques; (10) Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | stdlib-use | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | BADFUNC.ABORT BADFUNC.EXIT | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | C++5014 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | MISRA.TERMINATE  CERT.ERR.ABRUPT\_TERM |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Initializing** | [STD-008-CPP] | Write a constructor memory initializers in the canonical order. |

| **Noncompliant Code** |
| --- |
| Because the declaration order of the member variables does not match the member initializer order, attempting to read the value of someVal results in an unspecified value being stored into dependsOnSomeVal |
| **class** C {  **int** dependsOnSomeVal;  **int** someVal;    **public**:  C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| The declaration order of the class member variables is changed so that the dependency can be ordered properly. |
| **class** C {  **int** someVal;  **int** dependsOnSomeVal;    **public**:  C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Principles(s):** (4) Keep it Simple; (3) Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | initializer-list-order | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-OOP53 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreorder |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.STRUCT.INIT.OOMI | Out of Order Member Initializers |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-009-CPP] | Do not cast out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| If a value outside of enum’s range were passed to f(), the cast to EnumType would result in an unspecified 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** |
| --- |
| Checks if the value can be represented by the enumeration type before performing the conversion |
| **enum** EnumType {  First,  Second,  Third  };  **void** f(**int** intVar) {  **if** (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = **static\_cast**<EnumType>(intVar);  } |

| **Principles(s):** (3) Architect and Design for Security Policies; (10) Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | cast-integer-to-enum | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-INT50 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | C++3013 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations and Initialization | [STD-010-CPP] | Never Qualify a reference type with const or volatile. |

| **Noncompliant Code** |
| --- |
| The modification of p makes the program ill-formed |
| #include <iostream>    **void** f(**char** c) {  **const** **char** &p = c;  p = 'p'; // Error: read-only variable is not assignable  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Removes the const qualifier |
| #include <iostream>    **void** f(**char** c) {  **char** &p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Principles(s):** (3) Architect and Design for Security Policies; (1) Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-DCL52 |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | C++0014 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-DCL52-a | Never qualify a reference type with 'const' or 'volatile' |

### Defense-in-Depth Illustration

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



### Automation

This illustration provides a visual representation of the existing DevSecOps automation process.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. On the pre-production side, the process starts with assessing and planning. This includes examining potential vulnerabilities and hazards, ranking the tasks accordingly, and adjusting to the changes in regulation. Then, while in the design stage, adhering to OWASP principles and other best-practices are crucial. While building, verifying, and testing, validating against standards, checking for vulnerabilities, and implementing security practices should all be included. Then, while in the production stage, starting with the transition and health check, secure setups and penetration tests should be done before deploying. Then, after deploying, establishing log analysis and detection monitoring are important during the Monitor and detect step. Lastly, threats that pop up during monitoring should be assessed and responded to, thus allowing for the final step of maintaining and stabilizing.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Probable | Medium | High | 1 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CLG | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | Medium | 2 |
| STD-009-CPP | Medium | Unlikely | Medium | Medium | 2 |
| STD-010-CPP | Low | Unlikely | Low | Low | 3 |

### Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest protects data which is stationary. This might include hard drives, cloud assets, etc.. This protection can be done through disk encryption, implementing DLP protections, and extending loss prevention to the cloud. |
| Encryption in flight | Encryption at flight protects data which moves around. This can be between two devices within a network, or outside a network. This protection can be done through simple firewalls and authentication, automated controls, and email encryption. |
| Encryption in use | Encryption in use protects data which is being created, edited, etc.. Protection should start before use, identity management should be doubled down on, and the right to access should be heavily managed. |

**Source:** [**https://www.mimecast.com/blog/data-in-transit-vs-motion-vs-rest/**](https://www.mimecast.com/blog/data-in-transit-vs-motion-vs-rest/)

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is “the act of confirming the truth of an attribute of a single piece of data claimed true by an entity”. Authentication confirms a person’s identity using static passwords, one-time passwords, digital certificates, and biometric credentials. Confirming a person’s identity is important when working with sensitive information, as you only want the correct person to access it. |
| Authorization | After a person is authenticated and their identity is proven, they must receive authorization privileges to access certain information. Least Privilege is important here as you are ensuring that users are only accessing information that is necessary for them to perform their daily tasks. |
| Accounting | Accounting is “the process that keeps track of a user’s activity while attached to a system”. This information can be used to detect breaches, identify trending behaviors within the system, and for forensic investigation. In the case of a breach, cybersecurity professionals can trace back where the breach comes from, and what could have caused the incident. |

**Source:** [**https://www.ccsinet.com/blog/aaa-identity-management/**](https://www.ccsinet.com/blog/aaa-identity-management/)

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| [Insert text.] | 3/24/2024 | Completed all Milestone requirements | Leah Marshall | [Insert text.] |
| [Insert text.] | 4/14/2024 | Completed all Project Requirements | Leah Marshall | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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