**Green Pace Developer: Security Policy Guide – Francisco Ortega**

**Project One**



# 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.

## Principles, Standards and Policies

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Input validation is a paramount task for any data coming into a system from an external source, regardless of whether the source is new or previously trusted. The goal of these efforts is to ensure that malicious code has not been injected or that unintended side effects do not occur from some expected input to prevent attacks like SQL injection or buffer overflows. The system should remain resilient in the instances of invalid data. |
| 1. Heed Compiler Warnings | A program that executes is not secure. Compiler warnings provide an early indication of potential vulnerabilities that could be exposed. Actively acknowledging and making efforts to fix these warnings, rather than suppressing them, allows the developer to consider hard to spot weak points in the system that could be exploited by malicious actors. |
| 1. Architect and Design for Security Policies | Security for be a formative and rigorous component during the software development lifecycle from beginning to end rather than being an afterthought or “nice-to-have.” Security policies should be well-defined, accessible, and readily applicable to any new subprocess or component. The architecture and design of a system should facilitate its own goals without compromising any upstream or downstream systems. |
| 1. Keep It Simple | Less code, less problems. The design of any system should be as simple as possible. Complexity, especially unnecessary or avoidable complexity, makes a system more difficult to manage and secure. Avoiding complexity minimizes vulnerabilities. |
| 1. Default Deny | The default for any system should be denial of access to any resource or data unless explicit permissions have been granted for the relevant task. This principle aligns with the principle of least privilege and enforces the idea that all actions are intended and accounted for. |
| 1. Adhere to the Principle of Least Privilege | This principle ensures that all measurable tasks can be completed with the minimum exposure of the system to unwarranted risks. A user should only have the required permissions needed for the purposes the system intended for them to achieve. For example, a routine user on a social media platform should not have access to the database as this could result in breaches of data. |
| 1. Sanitize Data Sent to Other Systems | The process of transporting data between systems is prone to producing unintended consequences without proper maintenance. Like input validation, extensive efforts should be made to conform to the requirements of other systems without exposing unnecessary data outside of the current system. Sanitization should also ensure that malicious code does not make it from one secure boundary to another. |
| 1. Practice Defense in Depth | Multiple layers of security controls should be in place and actively enforced to ensure protection for a system at each level regardless of how many layers have previously been breached. For example, if there are physical barriers to entering a building, the malicious actor should not have direct access to the codebase simply because they made it past this initial layer. This practice can also extend to protecting other systems and containing the surface area of an existing attack to the compromised system. |
| 1. Use Effective Quality Assurance Techniques | Having a standard for techniques like required unit testing and rigorous code reviews are straightforward QA practices that can help ensure the code or process being added has as few issues early in development. Making use of QA tools like those for static analysis enhances this effort to reduce revisions in later iterations. |
| 1. Adopt a Secure Coding Standard | Minimizing the surface area of attack for a system depends on the quality of the code being written. A secure coding standard like CERT or OWASP ensures that the development team adheres to standardized guidelines for preventing vulnerabilities. |

### 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** | [DCL-053-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL53-CPP.+Do+not+write+syntactically+ambiguous+declarations) | Do not write syntactically ambiguous declarations |

| **Noncompliant Code** |
| --- |
| In the following example, the declaration of function f is ambiguous an will return int as opposed to evaluating the intended multiplication called in the main function. |
| int x = 5;  int y = 10;  int f(int());  int main() {  f(x \* y);    return 0;  } |

| **Compliant Code** |
| --- |
| In the following example, the declaration of function f is not explicitly defined such that the parameter is obvious. Therefore, when the function is called with the intended multiplication in the main function, the appropriate actions will be evaluated accordingly without ambiguity. |
| int f(int value) {  return value;  }  int main() {  int x = 5;  int y = 10;  int result = f(x \* y);    /\* more logic \*/  return 0;  } |

| **Principles(s):**  **4** - Keep It Simple)  **9** - Use Effective Quality Assurance Techniques  Ambiguous declarations create unreadable and unmaintainable code that requires more effort from the developer to understand how or why something is working or not working. Simple, digestible, and easy to follow code ensures that there is one strict interpretation of the code with expected behavior. This ensures that the code can be readily reviewed and tested for quality by others who might not have worked with the code directly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.DECL.FNEST** | Nested Function Declaration |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++1109, C++2510** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **CERT.DCL.AMBIGUOUS\_DECL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **296 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL53-a** **CERT\_CPP-DCL53-b CERT\_CPP-DCL53-c** | Parameter names in function declarations should not be enclosed in parentheses Local variable names in variable declarations should not be enclosed in parentheses Avoid function declarations that are syntactically ambiguous |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: DCL53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl53cpp.html) | Checks for declarations that can be confused between:   * Function and object declaration * Unnamed object or function parameter declaration   Rule fully covered. |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wvexing-parse |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3468**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3468) |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [INT-030-C](https://wiki.sei.cmu.edu/confluence/display/c/INT30-C.+Ensure+that+unsigned+integer+operations+do+not+wrap) | Ensure that unsigned integer operations do not wrap |

| **Noncompliant Code** |
| --- |
| In the following example, there is no check before the addition of the two unsigned integers resulting in the new value wrapping around to 0. This explicit example will always wrap around because a is set to the max value for unsigned int types even if the core logic could work for most cases of addition, this code fails to account for unexpected results. |
| #include <iostream>  #include <limits>  int main() {  unsigned int a = std::numeric\_limits<unsigned int>::max();  unsigned int b = 1;  unsigned int c = a + b;  std::cout << "a + b = " << c << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| In the following example, a check between the current value of a and the max value that a could have been made to ensure that the wraparound to 0 does not occur. This addition will still always overflow because of the non-practical max value being set on b but the checking logic can be applied to conventional instances of addition to ensure a program operates as expected. |
| #include <iostream>  #include <limits>  int main() {  unsigned int a = std::numeric\_limits<unsigned int>::max();  unsigned int b = 1;  if (a > std::numeric\_limits<unsigned int>::max() - b) {  std::cerr << "Overflow detected!" << std::endl;  } else {  unsigned int c = a + b;  std::cout << "a + b = " << c << std::endl;  }  return 0;  } |

| **Principles(s):**  **1** - Validate Input Data  **3** - Architect and Design for Security Policies  **9** - Use Effective Quality Assurance Techniques  This standard ensures that operations and logic do not produce unexpected results. Checking for a potentially overflow before it happens allows the application to validate input and gracefully handle any exceptions that arise. Input validation ensures that only sanitized data enters the system. Quality is improved by being aware of these potential outcomes as more edge case unit tests can be written before a system is tested by QA teams so that they can focus on more pressing matters. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **integer-overflow** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-INT30** | Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size. Integer overflow of allocation size. Multiplication overflow of allocation size. Subtraction underflow of allocation size. Addition overflow of size. Unreasonable size argument Multiplication overflow of size. Subtraction underflow of size. |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule by ensuring that operations are checked for overflow before being performed (Be mindful of exception INT30-EX2 because it excuses many operations from requiring [validation](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validation), including all the operations that would validate a potentially dangerous operation. For instance, adding two unsigned ints together requires validation involving subtracting one of the numbers from UINT\_MAX, which itself requires no validation because it cannot wrap.) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **INTEGER\_OVERFLOW** | Implemented |
| [Cppcheck Premium](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck+Premium) | 24.9.0 | **premium-cert-int30-c** | Partially implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.2 | **C2910, C3383, C3384, C3385, C3386**  **C++2910**  **DF2911, DF2912, DF2913,** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.2 | **NUM.OVERFLOW** **CWARN.NOEFFECT.OUTOFRANGE** **NUM.OVERFLOW.DF** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **493 S, 494 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-INT30-a** **CERT\_C-INT30-b** **CERT\_C-INT30-c** | Avoid wraparounds when performing arithmetic integer operations Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule INT30-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint30c.html) | Checks for:   * Unsigned integer overflow * Unsigned integer constant overflow   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.33 | [**V658**](https://pvs-studio.com/en/docs/warnings/v658/)**,**[**V1012**](https://pvs-studio.com/en/docs/warnings/v1012/)**,**[**V1028**](https://pvs-studio.com/en/docs/warnings/v1028/)**,**[**V5005**](https://pvs-studio.com/en/docs/warnings/v5005/)**,**[**V5011**](https://pvs-studio.com/en/docs/warnings/v5011/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **unsigned overflow** | Exhaustively verified. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STR-053-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access) | Range check element access |

| **Noncompliant Code** |
| --- |
| In this example, an element is accessed without checking if the index is within the valid range for the defined string. |
| #include <iostream>  #include <string>  int main() {  std::string s = "Hi";    char c = s[2];  std::cout << c << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| In this example, the at function is used to safely check that the index being accessed is within the bounds of the range. Otherwise, the method will throw an std::out\_of\_range exception that can be caught an handled appropriately. |
| #include <iostream>  #include <string>  int main() {  std::string s = "Hi";    try {  char c = str.at(2);  std::cout << c << std::endl;  } catch (const std::out\_of\_range& e) {  std::cerr << "Index out of range: " << e.what() << std::endl;  }    return 0;  } |

| **Principles(s):**  **1** - Validate Input Data  **8** - Practice Defense in Depth  **9** - Use Effective Quality Assurance Techniques  Range checking extends the idea of input validation and prevents the application from unexpectedly crashing due to any easily resolvable check. Buffer Overflow attacks can be used to gain unauthorized access to a system, so ensuring that each access is checked can mitigate this vulnerability as an added layer of security. This is another QA technique that minimizes the amount of tech debt that is accumulated once a bug is found late in development. |
| --- |

**Threat Level**

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

**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/cplusplus/CodeSonar) | 8.1p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++3162, C++3163, C++3164, C++3165** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [IDS-000-J](https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection) | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| In the following code, login information for a user is fetched based on the username that is passed in. This value is simply concatenated onto the query string with no validation. This code makes it easy for a malicious actor to pass in SQL injection attacks that can return unintended results from the database. |
| #include <iostream>  #include <string>  #include <sqlite3.h>  int execute\_sql(sqlite3\* db, const std::string& username) {  std::string sql = "SELECT username,password FROM users WHERE username = '" + username + "';";  char\* err = 0;  int result = sqlite3\_exec(db, sql.c\_str(), 0, 0, &err);  if (result != SQLITE\_OK) {  std::cerr << "Error executing query: " << err << std::endl;  } else {  std::cout << "Query executed successfully" << std::endl;  }    return result;  } |

| **Compliant Code** |
| --- |
| In the following code, the same data will be returned based on the provided username but there are more checks in place to ensure that only the relevant data is returned if it exists. Specifically, this example makes use of prepared statements to ensure that the parameterized values coming into the query are sanitized and acceptable. Relevant checks allow the execution of the query to fail before a call to the database is even made if something is not properly structured. |
| #include <iostream>  #include <string>  #include <sqlite3.h>  int execute\_sql(sqlite3\* db, const std::string& username) {  std::string sql = "SELECT username,password FROM users WHERE username = ?;”;  sqlite3\_stmt\* prepared;  int result = sqlite3\_prepare\_v2(db, sql.c\_str(), -1, &prepared, nullptr);  /\* Check prepared statement \*/  sqlite3\_bind\_text(prepared, 1, username.c\_str(), -1, SQLITE\_STATIC);  result = sqlite3\_step(prepared);  /\* Check for results of query from prepared statement if any \*/    return result;  } |

| **Principles(s):**  **1** - Validate Input Data  **7** - Sanitize Data Sent to Other Systems  **3** - Architect and Design for Security Policies  This coding standard aligns with the similar vulnerability that buffer overflow attacks present. SQL injection attacks can be readily prevented by having proper input validation and sanitization. Only properly sanitized data should be sent between systems to mitigate the surface area of a potential breach. A properly architected system will account for this common attack pattern and have monitoring in check that can alert of potentially faults in the system or attempts to use SQL Injection. |
| --- |

**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 |
| [Fortify](https://wiki.sei.cmu.edu/confluence/display/java/Fortify) | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/java/Klocwork) | 2024.2 | **SV.DATA.DB** **SV.SQL** **SV.SQL.DBSOURCE** | Implemented |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2024.1 | **CERT.IDS00.TDSQL** | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 9.9 | [**S2077**](https://rules.sonarsource.com/java/RSPEC-2077)  [**S3649**](https://rules.sonarsource.com/java/RSPEC-3649) | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649) |
| [SpotBugs](https://wiki.sei.cmu.edu/confluence/display/java/SpotBugs) | 4.6.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** **SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [MEM-050-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory) | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| In the following example, a pointer attempts to access memory that has already been freed leading to what is known as a dangling pointer. This code can result in unwarranted vulnerabilities since the pointer attempts to access a dereferenced and deallocated address in memory. |
| #include <iostream>  int main() {  int\* s = new int(2);  /\* Additional business logic \*/  delete s;  std::cout << \*s << std::endl;    return 0;  } |

| **Compliant Code** |
| --- |
| The following example, provides compliant access to the allocated memory. All business logic, including the direct access to the allocated memory via pointer, occur before the memory is deallocated. Additional logic can be added to following the freed memory statement to safely attempt to access this point in memory but this example highlights the significance in avoiding access to such memory after deallocation. |
| #include <iostream>  int main() {  int\* s = new int(2);  /\* Additional business logic \*/  std::cout << \*s << std::endl;  delete s;    return 0;  } |

| **Principles(s):**  **3** - Architect and Design for Security Policies  **9** - Use Effective Quality Assurance Techniques  **10** - Adopt a Secure Coding Standard  Attempts to access freed memory can result in unintended outcomes. A well-designed system should involve rigorous analysis of the code to make sure these cases are minimized and contained if they are overlooked. Memory management is a QA focus point that can be reinforced through static analysis and resilient testing strategies. Having a secure coding standard allows a team to actively check for these potential faults in code reviews as the entire team will be aware of the issue. |
| --- |

**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 |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++4303, C++4304** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.2 | **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.33 | [**V586**](https://pvs-studio.com/en/docs/warnings/v586/), [**V774**](https://pvs-studio.com/en/docs/warnings/v774/) |  |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Splint) | 5.0 |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [DCL-003-C](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression) | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| This example defines a memory-mapped structure that is required for the code to function correctly. A use of the runtime assert() macro is used to validate this structure, but this assertions happens at runtime and can cause a silent malfunction or runtime error and depends on the execution path to actually diagnose anything. |
| #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 code block makes use of static\_assert to validate the data structure without having to be nested within a function like assert. This ensures that the check for the constant expression happens at compile time and can produce more meaningful diagnostic information. |
| #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  **9** - Use Effective Quality Assurance Techniques  Compile time warnings reduce the overhead from discovering a bug if and only when a runtime error occurs. If code was not properly or thoroughly tested for edge cases, these bugs can go overlooked. Static assertions during development can be used to catch these issues early but their use should align with QA standards and ensure that a refactor is not required to remove the assertions later in development. |
| --- |

**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 |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [ERR-051-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions) | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In the following example, there are nested calls from the main entry point down to the function that will ultimately throw an exception but there is no management of potential exceptions in either the main or calling\_func functions. Exception will propagate up the chain and terminate the program abruptly. |
| void throwing\_func() noexcept(false);    void calling\_func() {  throwing\_func();  }    int main() {  calling\_func();  } |

| **Compliant Code** |
| --- |
| In the following example, the same nested structure exists but the calling\_func is wrapped within a try-catch block that ensures all exceptions down the chain are handled gracefully without the potential leaks or crashes seen in the previous noncompliant example. Additional try-catch blocks could be added to the nested functions for more robust error handling but this example satisfies the concern with the previous example. |
| void throwing\_func() noexcept(false);    void calling\_func() {  throwing\_func();  }    int main() {  try {  calling\_func();  } catch (...) {  // Handle error  }  } |

| **Principles(s):**  **8** - Practice Defense in Depth  **9** - Use Effective Quality Assurance Techniques  Handling all exceptions ensures that a program will persist a response to the user without crashing. Unhandled exceptions can lead to security vulnerabilities but gracefully handling exceptions promotes defense in depth as failures are defaulted against. Allowing an exception to bubble up to the calling source reinforces quality assurance as bugs can be identified as opposed to having an unexpected stack track to identify when a system should be tested for performance rather than basic functionality. |
| --- |

**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 | **main-function-catch-all early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | [MSC-050-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC50-CPP.+Do+not+use+std%3A%3Arand%28%29+for+generating+pseudorandom+numbers) | Do not use std::rand() for generating pseudorandom numbers |

| **Noncompliant Code** |
| --- |
| The following example generates an ID with a numeric portion that is produced via the std::rand() function. This code is non-compliant, however, because there is limited randomness based on how the function seeds the generation and can lead to predictable results that can pose as vulnerabilities or unwarranted duplication. The resulting value can also have modulo bias. |
| #include <cstdlib>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID"  // followed by a random integer in the range [0-10000].  id += std::to\_string(std::rand() % 10000);  // ...  } |

| **Compliant Code** |
| --- |
| This updated code for the same ID generation leverages two mechanisms for pseudo random generation from the C++ library. Specifically, the Mersenne Twister algorithm is used as the engine for providing random values while the distribution is normalized via the density function. These two factors make the code compliant and remove the modulo bias to achieve more practical pseudorandom IDs. |
| #include <random>  #include <string>    **void** f() {    std::string id("ID");    std::uniform\_int\_distribution<**int**> distribution(0, 10000);    std::random\_device rd;    std::mt19937 engine(rd());    id += std::to\_string(distribution(engine));    // ...  } |

| **Principles(s):**  **3** - Architect and Design for Security Policies  **10** - Adopt a Secure Coding Standard  Using pseudorandom generation violates secure architecture and design policies as this leaves a backdoor for malicious actors to attempt to reverse engineering business logic. The severity of not aligning with this standard depends on the use case but should generally be avoided for other random generation methods that promote security. If teams adopt a secure coding standard, they can align on this as it will become easy to identify this in code reviews as a bad practice that could potentially be exploited. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 4.0 (prerelease) | cert-msc50-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **BADFUNC.RANDOM.RAND** | Use of rand |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.MSC30** | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++5028** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **CERT.MSC.STD\_RAND\_CALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **44 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MSC50-a** | Do not use the rand() function for generating pseudorandom numbers |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MSC50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc50cpp.html) | Checks for use of vulnerable pseudo-random number generator (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programing | [OOP-052-CCP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP52-CPP.+Do+not+delete+a+polymorphic+object+without+a+virtual+destructor) | Do not delete a polymorphic object without a virtual destructor |

| **Noncompliant Code** |
| --- |
| This example demonstrates the lack of a virtual destructor for b. Deletion of the object through a polymorphic pointer will result in undefined behavior for type without a virtual destructor. This occurs regardless of the implicit non-virtual destructor that is provided. |
| struct Base {  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

| **Compliant Code** |
| --- |
| This example resolves the implicit and unexpected behavior from the previous code by including an explicit declaration for a virtual destructor. This retains the polymorphic behavior across objects that are deleted. |
| struct Base {  virtual ~Base() = default;  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

| **Principles(s):**  **3** - Architect and Design for Security Policies  **10** - Adopt a Secure Coding Standard  Memory leaks from deleting a polymorphic object with no virtual destructor are security vulnerabilities as they can cause a system to crash or result in unexpected behavior that can be exploited by malicious actors. A team that aligns on a coding standard will be able to identify these bugs in their code and accurately test the system against such edge cases. Securely designed systems should appropriately handle object destruction or identify this issue early. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **non-virtual-public-destructor-in-non-final-class** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wdelete-non-virtual-dtor |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.DNVD** | delete with Non-Virtual Destructor |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++3402, C++3403, C++3404** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.2 | **CL.MLK.VIRTUAL** **CWARN.DTOR.NONVIRT.DELETE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **303 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-OOP52-a** | Define a virtual destructor in classes used as base classes which have virtual functions |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: OOP52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop52cpp.html) | Checks for situations when a class has virtual functions but not a virtual destructor (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.33 | [**V599**](https://pvs-studio.com/en/docs/warnings/v599/), [**V689**](https://pvs-studio.com/en/docs/warnings/v689/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **non-virtual-public-destructor-in-non-final-class** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S1235**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-1235) |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [FIO-051-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed) | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| The following example demonstrates working with a file but never explicitly closing it. This can become an issue where data or resources are leaked or not properly updated depending on the concurrent operations. Corruption of data or unexpected results can occur even though the file will be automatically closed when the ofstream object is out of scope. |
| #include <iostream>  #include <fstream>  int main() {  std::ofstream file("hello.txt");  if (file.is\_open()) {  file << "Hello!" << std::endl;  }  return 0;  } |

| **Compliant Code** |
| --- |
| The following example demonstrates the explicit closing of the file that was being worked with. This is a best practice in ensuring that resources are not corrupted or leaked when they are not being used. As previously mentioned, this is something that will eventually happen automatically but in longer running programs with multiple file operations and no explicit file closures, the side effects can become more pronounced and harder to trace down. |
| #include <iostream>  #include <fstream>  int main() {  std::ofstream file("hello.txt");  if (file.is\_open()) {  file << "Hello!" << std::endl;  file.close();  }  return 0;  } |

| **Principles(s):**  **3** - Architect and Design for Security Policies  **9** - Use Effective Quality Assurance Techniques  **10** - Adopt a Secure Coding Standard  Not closing files can result in resource leaks or corruption of data. A properly architected system will ensure that all files and resources that are touched are appropriately and explicitly handled. A secure coding standard might include this as an explicit requirement for passing code reviews even when the file will be implicitly closed. Quality assurance can be used to validate that the input and output of files before and after handling produces the anticipated outcome with any faults in the data being flagged for review and refactor. Of course, this is something that the developers themselves should be preemptively testing in their unit tests. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **DF4786, DF4787, DF4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **RH.LEAK** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

### Defense-in-Depth Illustration

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



### Automation



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 existing DevOps process and infrastructure promotes an efficient methodology for building and deploying new apps and features but can leave security as an afterthought. DevSecOps improves on the DevOps framework by integrating security from the beginning and at each step, identifying security as a shared responsibility. However, the tendency to forgo security is exacerbated by the manual nature of reviewing and validating that each principle has been followed. Automation is an integral solution that can help validate compliance with the standards defined earlier throughout the DevSecOps lifecycle.

Green Pace can automate security checks without a lapse in their continuous development or continuous deployment pipelines. Specifically, tools like SonarQube and other static code analysis tools can be used during development to detect potentially overlooked issues like buffer overflows or uncaught exceptions. Code that is pushed into a remote repository can then be checked to ensure that tests were written for new code and that compliance with code coverage or image builds were successful before anyone else even reviews the code. The unit tests themselves can be integrated with automation tools like the ones identified for each coding standard to help ensure that all code aligns with a relevant coding standard.

Automation can help integrate security into each step of development and gives each person an opportunity to develop an awareness of common issues and how they can be minimized by following best practices. This integration is rather seamless with the existing DevOps architecture and promotes the shared responsibility model for security.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| [IDS-000-J](https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection) | High | Likely | Medium | **P18** | **L1** |
| [DCL-003-C](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression) | Low | Unlikely | High | **P1** | **L3** |
| [INT-030-C](https://wiki.sei.cmu.edu/confluence/display/c/INT30-C.+Ensure+that+unsigned+integer+operations+do+not+wrap) | High | Likely | High | **P9** | **L2** |
| [MEM-050-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory) | High | Likely | Medium | **P18** | **L1** |
| [MSC-050-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC50-CPP.+Do+not+use+std%3A%3Arand%28%29+for+generating+pseudorandom+numbers) | Medium | Unlikely | Low | **P6** | **L2** |
| [ERR-051-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions) | Low | Probable | Medium | **P4** | **L3** |
| [FIO-051-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed) | Medium | Unlikely | Medium | **P4** | **L3** |
| [OOP-052-CCP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP52-CPP.+Do+not+delete+a+polymorphic+object+without+a+virtual+destructor) | Low | Likely | Low | **P9** | **L2** |
| [STR-053-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access) | High | Unlikely | Medium | **P6** | **L2** |
| [DCL-053-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL53-CPP.+Do+not+write+syntactically+ambiguous+declarations) | Low | Unlikely | Medium | **P2** | **L3** |

### 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** refers to the application of encryption strategies for data that is stored and inactive. (Dinic, 2024) This can range from information in a database, hard drive, or other long-term storage solution. Encryption algorithms can be leveraged to prevent unauthorized access from potential malicious actors.  The policy for **Encryption at Rest** applies for data that will be retained for a significant or indeterminate amount of time. All locally stored database contents, backup information, or files should be encrypted with the **AES-256** encryption standardto prevent unauthorized access in the instance that the data at rest is compromised. |
| Encryption in flight | **Encryption in Flight** refers to the application of encryption strategies for data that is being transmitted over a given network from one location to another. (Dinic, 2024) This may include requests and responses between a client and server or any public communication channels. This data is arguably more insecure than data at rest as it is exposed to the Internet or Intranet and can be readily intercepted and/or modified.  The policy for **Encryption in Flight** applies to all information that will be moving externally or internally between source and destination. Sensitive information should be encrypted using standards like **AES-256** and relevant protocols like **SSL/TLS**. These protocols can be applied to web applications, email, and APIs among other services. |
| Encryption in use | **Encryption in Use** refers to the application of encryption strategies for data that is actively being utilized. This can be information that is being read, changed, or processed. Because of this active interaction data is most vulnerable at this point. (Dinic, 2024) Encryption strategies can be used to promote resiliency and safeguard against malicious actors.  The policy for **Encryption in Use** applies to all information that will be accessed by a system or user. The protection of this vulnerable data begins by enforcing the principle of least privilege to ensure that mistakes do not corrupt the data. Proper authentication standards should be used with strong encryption protocols to decrypt the data while needed and encrypt it after. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | **Authentication** refers to the concept of validating the user for whom they say they are. This ensures that only verified users and/or applications can access the protected system. (Martinez, 2024) Reinforced authentication mechanisms like **Multi-Factor Authentication (MFA)** can be used to confirm the access request. Following best practices like strong passwords can reduce the risk of a malicious actor authenticating like a trusted user.  The policy of **Authentication** should be applied by securely hashing passwords that are used for logging into the system. This level of protection should extend to all protected or potentially vulnerable resources to prevent unauthorized access from an overlooked weak spot. User logins should be enforced with time windows when inactivity is detected to minimize the surface area and risk of attacks. |
| Authorization | **Authorization** controls the level of access a user has to a system. This process general follows Authentication and determines the level of access a given user needs to complete a given task. This mechanism ensures that even authenticated users are prevented from having complete access to an entire system. (Martinez, 2024)  The policy of **Authorization** should be reinforced through the **Principle of least Privilege**, or the idea that the least required amount of access should be granted. The principle of **Default Deny** can also extend this to better protect resources and actions that might not have been explicitly accounted for. Role-based permissions should be implemented for all processes with particular precaution for those that engage in sensitive systems and resources, such as changing the database or files accessed by users. Actions like adding new users should be strictly regulated as these kinds of actions can have unwarranted outcomes if misused. |
| Accounting | **Accounting** refers to the logging of user activity and/or processes within a system. (Martinez, 2024) Example use cases include changes to database items and tracking the user and activity when a protected resource is accessed. Metrics can be aggregated into reports for efficient incident tracking.  The policy of **Accounting** applies to all transactions and interactions with sensitive data or resources in the system. By being judicious about the logs and metrics that are tracked it becomes easier to develop a paper trail for unusual activities. This allows incidents to be readily assessed to determine the point of failure. |

**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 | 09/22/2024 | Principles and Standards outlined | Francisco Ortega | Ahlam Alhweiti |
| 1.2 | 10/13/2024 | Security Policy Completed | Francisco Ortega | Ahlam Alhweiti |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

## References

Dinic, M. (2024, April 29). *Understanding encryption - data at rest, in motion, in use*. Jatheon Technologies Inc. https://jatheon.com/blog/data-at-rest-data-in-motion-data-in-use/

Martinez, J. (2024, September 27). *What is AAA Security? authentication, authorization, and accounting*. StrongDM. https://www.strongdm.com/blog/aaa-security