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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Data entering the system from external sources should be cleansed, be of proper size and type for variables within the system. Validated input can eliminate a large range of software vulnerabilities. |
| 1. Heed Compiler Warnings | Project/Application compilers should be set at highest warning level and code modified to correct warnings. Static and dynamic analysis tools should be use to find and remove additional security flaws. |
| 1. Architect and Design for Security Policies | Design your system to implement and enforce security policies. |
| 1. Keep It Simple | Increase system complexity will increase the chance of errors in implementation, configuration, and use. Keeping it simple can help insure reduced errors and a higher ability to secure the system. |
| 1. Default Deny | Access to systems should be denied by default and access only provided once the protection scheme identifies proper conditions are met. |
| 1. Adhere to the Principle of Least Privilege | Every process and users should only have enough permissions to perform their required task/job. |
| 1. Sanitize Data Sent to Other Systems | [Sanitize all data passed to complex subsystems [C STR02-A] such as command shells, relational databases, and commercial off-the-shelf (COTS) components. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Risk can be mitigated through the use of multi-layer security practices. Multiple layers help insure if one layer was to fall another is there to help prevent a flaw from becoming exploitable. |
| 1. Use Effective Quality Assurance Techniques | Quality Assurance can help identify and eliminate vulnerabilities. Good techniques should be in place; such as, penetration testing, source code audits, and security reviews. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

### 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** | **Do not declare variables inside a switch statement before the first case label** |
| --- | --- | --- |
| **Data Type** | STD-001-DCL | If a programmer declares variables, initializes them before the first case statement, and then tries to use them inside any of the case statements, those variables will have scope inside the switch block but will not be initialized and will consequently contain indeterminate values. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example declares variables and contains executable statements before the first case label within the switch statement |
| #include <stdio.h>    extern void f(int i);    void func(int expr) {  switch (expr) {  int i = 4;  f(i);  case 0:  i = 17;  /\* Falls through into default code \*/  default:  printf("%d\n", i);  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the statements before the first case label occur before the switch statement |
| #include <stdio.h>    extern void f(int i);    int func(int expr) {  /\*  \* Move the code outside the switch block; now the statements  \* will get executed.  \*/  int i = 4;  f(i);    switch (expr) {  case 0:  i = 17;  /\* Falls through into default code \*/  default:  printf("%d\n", i);  }  return 0;  } |

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

| **Principles(s):** Keep It Simple: This standard falls under the principle as we look at standard coding best practices of where to initialize variables. |
| --- |

**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=87152428) | 20.10 | **switch-skipped-code** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL41** | Fully implemented |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | -Wsometimes-uninitialized |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **MISRA C 2004 Rule 15.0**  **MISRA C 2012 Rule 16.1** | Implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.3 | **C2008, C2882, C3234** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.4 | **CERT.DCL.SWITCH.VAR\_BEFORE\_CASE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **385 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-DCL41-a** | A switch statement shall only contain switch labels and switch clauses, and no other code |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **527** | Assistance provided |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule DCL41-C](https://www.mathworks.com/help/bugfinder/ref/certcruledcl41c.html) | Checks for ill-formed switch statements (rule partially covered) |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **3234** **2008** **2882** | Partially implemented |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.17 | [V622](https://pvs-studio.com/en/docs/warnings/v622/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 20.10 | **switch-skipped-code** | Fully checked |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **initialisation** | Exhaustively detects undefined behavior |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Data Value** | STD-002-INT | This behavior is more informally called unsigned integer wrapping. Unsigned integer operations can wrap if the resulting value cannot be represented by the underlying representation of the integer. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in an unsigned integer wrap during the addition of the unsigned operands ui\_a and ui\_b. If this behavior is unexpected, the resulting value may be used to allocate insufficient memory for a subsequent operation or in some other manner that can lead to an exploitable vulnerability. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution performs a precondition test of the operands of the addition to guarantee there is no possibility of unsigned wrap: |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum;  if (UINT\_MAX - ui\_a < ui\_b) {  /\* Handle error \*/  } else {  usum = ui\_a + ui\_b;  }  /\* ... \*/  } |

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

| **Principles(s):** Validate Input Data: Here when need to make sure we use the proper data types and techniques for values of changing size. |
| --- |

**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) | 20.10 | **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) | 6.2p0 | **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 |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.3 | **C2910, C2911, C2912, C2913, C3383, C3384, C3385, C3386**  **C++2910, C++2911, C++2912, C++2913** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.4 | [NUM.OVERFLOW](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [CWARN.NOEFFECT.OUTOFRANGE](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [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) | 2021.2 | **CERT\_C-INT30-a** **CERT\_C-INT30-b** **CERT\_C-INT30-c** | Avoid integer overflows 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) | R2021a | [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. |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **2910 [C], 2911 [D], 2912 [A],**  **2913 [S], 3383, 3384, 3385, 3386** | Partially implemented |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2910, 2911, 2912, 2913** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.17 | [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** | **Cast characters to unsigned char before converting to larger integer sizes** |
| --- | --- | --- |
| **String Correctness** | STD -003-STR | Signed character data must be converted to unsigned char before being assigned or converted to a larger signed type. This rule applies to both signed char and (plain) char characters on implementations where char is defined to have the same range, representation, and behaviors as signed char. |

| **Noncompliant Code** |
| --- |
| This problem can be repaired by explicitly declaring the c\_str variable as unsigned char |
| static int yy\_string\_get(void) {  register unsigned char \*c\_str;  register int c;  c\_str = bash\_input.location.string;  c = EOF;    /\* If the string doesn't exist or is empty, EOF found \*/  if (c\_str && \*c\_str) {  c = \*c\_str++;  bash\_input.location.string = c\_str;  }  return (c);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the result of the expression \*c\_str++ is cast to unsigned char before assignment to the int variable c |
| static int yy\_string\_get(void) {  register char \*c\_str;  register int c;  c\_str = bash\_input.location.string;  c = EOF;    /\* If the string doesn't exist or is empty, EOF found \*/  if (c\_str && \*c\_str) {  /\* Cast to unsigned type \*/  c = (unsigned char)\*c\_str++;  bash\_input.location.string = c\_str;  }  return (c);  } |

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

| **Principles(s):** Validate Input Data: Here when need to make sure we use the proper data types and techniques for values of changing size. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **char-sign-conversion** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR34** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **MISC.NEGCHAR** | Negative Character Value |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule when checking for violations of [INT07-C. Use only explicitly signed or unsigned char type for numeric values](https://wiki.sei.cmu.edu/confluence/display/c/INT07-C.+Use+only+explicitly+signed+or+unsigned+char+type+for+numeric+values) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **MISRA C 2012 Rule 10.1**  **MISRA C 2012 Rule 10.2**  **MISRA C 2012 Rule 10.3**  **MISRA C 2012 Rule 10.4** | Implemented  Essential type checkers |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.STR34** | Fully implemented |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 2.95 and later | [-Wchar-subscripts](http://gcc.gnu.org/onlinedocs/gcc-3.0.4/gcc_3.html#IDX139) | Detects objects of type char used as array indices |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.3 | **C2140, C2141, C2143, C2144, C2145, C2147, C2148, C2149, C2151, C2152, C2153, C2155**  **C++3051** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.4 | **CXX.CAST.SIGNED\_CHAR\_TO\_INTEGER** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **434 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-STR34-b** **CERT\_C-STR34-c** **CERT\_C-STR34-d** | Cast characters to unsigned char before assignment to larger integer sizes An expressions of the 'signed char' type should not be used as an array index Cast characters to unsigned char before converting to larger integer sizes |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **571** | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule STR34-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr34c.html) | Checks for misuse of sign-extended character value (rule fully covered) |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **2140, 2141, 2143, 2144,**  **2145, 2147, 2148, 2149,**  **2151, 2152, 2153, 2155** | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-SQL | SQL injection vulnerabilities arise in applications where elements of a SQL query originate from an untrusted source. Without precautions, the untrusted data may maliciously alter the query, resulting in information leaks or data modification. The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures. |

| **Noncompliant Code** |
| --- |
| Unfortunately, this code example permits a SQL injection attack by incorporating the unsanitized input argument username into the SQL command, allowing an attacker to inject validuser' OR '1'='1. The password argument cannot be used to attack this program because it is passed to the hashPassword() function, which also sanitizes the input. |
| public void doPrivilegedAction(String username, char[] password)  throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);    String sqlString = "SELECT \* FROM db\_user WHERE username = '"  + username +  "' AND password = '" + pwd + "'";  Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sqlString);    if (!rs.next()) {  throw new SecurityException(  "User name or password incorrect"  );  }  // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a parametric query with a ? character as a placeholder for the argument. This code also validates the length of the username argument, preventing an attacker from submitting an arbitrarily long user name. |
| try {  String pwd = hashPassword(password);    // Validate username length  if (username.length() > 8) {  // Handle error  }    String sqlString =  "select \* from db\_user where username=? and password=?";  PreparedStatement stmt = connection.prepareStatement(sqlString);  stmt.setString(1, username);  stmt.setString(2, pwd);  ResultSet rs = stmt.executeQuery();  if (!rs.next()) {  throw new SecurityException("User name or password incorrect");  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  } |

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

| **Principles(s):** Validate Input Data: SQL Injection is when a malicious user attempts to access the system through additional input into value fields outside the expected data for the field.  Defense in Depth: Through encryption of passwords another layer of protection is added to the system. Converting Plaintext for input fields into an encrypted value to match against in the DB prevents SQL Injection by changing the content of the initial user input. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | 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) | 6.2p0 | **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) |  | **SV.DATA.BOUND** **SV.DATA.DB** **SV.HTTP\_SPLIT** **SV.PATH** **SV.PATH.INJ** **SV.SQL** | Implemented |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2021.2 | **CERT.IDS00.TDSQL** | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 6.7 | [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) | 4.5.3 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** **SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Free dynamically allocated memory when no longer needed** |
| --- | --- | --- |
| **Memory Protection** | STD-005-MEM | Before the lifetime of the last pointer that stores the return value of a call to a standard memory allocation function has ended, it must be matched by a call to free() with that pointer value. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the object allocated by the call to malloc() is not freed before the end of the lifetime of the last pointer text\_buffer referring to the object: |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }  return 0;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the pointer is deallocated with a call to free(): |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }    free(text\_buffer);  return 0;  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 |  | Supported, but no explicit checker |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM31** | Can detect dynamically allocated resources that are not freed |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **ALLOC.LEAK** | Leak |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **RESOURCE\_LEAK**  **ALLOC\_FREE\_MISMATCH** | Finds resource leaks from variables that go out of scope while owning a resource |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | leakReturnValNotUsed | Doesn't use return value of memory allocation function |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.3 | **C2706, C2707, C2708**  **C++2706, C++2707, C++2708, C++3337, C++3338** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.4 | **CL.FFM.ASSIGN**  **CL.FFM.COPY**  **CL.SHALLOW.ASSIGN**  **CL.SHALLOW.COPY**  **FMM.MIGHT**  **FMM.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **50 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-MEM31-a** | Ensure resources are freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) |  |  | Runtime analysis |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **429** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule MEM31-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem31c.html) | Checks for memory leak (rule fully covered) |
| [PRQA QA-C](https://www.securecoding.cert.org/confluence/display/c/PRQA+QA-C) | 9.7 | **2706, 2707, 2708** |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | STD-006-DCL | Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities (see MSC11-C. Incorporate diagnostic tests using assertions). The runtime assert() macro has some limitations, however, in that it incurs a runtime overhead and because it calls abort(). Consequently, the runtime assert() macro is useful only for identifying incorrect assumptions and not for runtime error checking. As a result, runtime assertions are generally unsuitable for server programs or embedded systems. |

| **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** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: Assertions are a staple for Unit testing techniques when validating the functionality and therefore quality of the code. |
| --- |

**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) | 6.2p0 | **(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 |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-ERR | When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. |

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

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

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

| **Principles(s):** Heed Compiler Warnings: Exceptions can occur during both build and runtime. The build exceptions will appear during compile and should be reviewed then. |
| --- |

**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) | 20.10 | **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** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.4 | **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) | 2021.2 | **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) | R2021b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4035, 4036, 4037** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **main-function-catch-all** **early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Converting a pointer to integer or integer to pointer** |
| --- | --- | --- |
| Data Value | STD-008-INT | Although programmers often use integers and pointers interchangeably in C, pointer-to-integer and integer-to-pointer conversions are implementation-defined.  Conversions between integers and pointers can have undesired consequences depending on the implementation. |

| **Noncompliant Code** |
| --- |
| The size of a pointer can be greater than the size of an integer, such as in an implementation where pointers are 64 bits and unsigned integers are 32 bits. This code example is noncompliant on such implementations because the result of converting the 64-bit ptr cannot be represented in the 32-bit integer type |
| void f(void) {  char \*ptr;  /\* ... \*/  unsigned int number = (unsigned int)ptr;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Any valid pointer to void can be converted to intptr\_t or uintptr\_t and back with no change in value. (See INT36-EX2.) The C Standard guarantees that a pointer to void may be converted to or from a pointer to any object type and back again and that the result must compare equal to the original pointer. Consequently, converting directly from a char \* pointer to a uintptr\_t, as in this compliant solution, is allowed on implementations that support the uintptr\_t type. |
| #include <stdint.h>    void f(void) {  char \*ptr;  /\* ... \*/  uintptr\_t number = (uintptr\_t)ptr;  /\* ... \*/  } |

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

| **Principles(s):** Validate Input Data: Here when need to make sure we use the proper data types and techniques for values of changing size. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **pointer-integral-cast**  **pointer-integral-cast-implicit**  **function-pointer-integer-cast**  **function-pointer-integer-cast-implicit** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-INT36** | Fully implemented |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | -Wint-to-pointer-cast, -Wint-conversion | Can detect some instances of this rule, but does not detect all |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **LANG.CAST.PC.CONST2PTR** **LANG.CAST.PC.INT** | Conversion: integer constant to pointer Conversion: pointer/integer |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **PW.POINTER\_CONVERSION\_LOSES\_BITS** | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.3 | **C0303, C0305, C0306, C0309, C0324, C0326, C0360, C0361, C0362**  **C++3040, C++3041, C++3042, C++3043, C++3044, C++3045, C++3046, C++3047, C++3048** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.4 | [MISRA.CAST.OBJ\_PTR\_TO\_INT.2012](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **439 S, 440 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-INT36-b** | A conversion should not be performed between a pointer to object type and an integer type other than 'uintptr\_t' or 'intptr\_t' |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **4287** | Partially supported: reports casts from pointer types to smaller integer types which lose information |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule INT36-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint36c.html) | Checks for unsafe conversion between pointer and integer (rule partially covered) |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **0303, 0305, 0306, 0309, 0324,**  **0326, 0360, 0361, 0362** | Partially implemented |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **3040, 3041, 3042, 3043, 3044,**  **3045, 3046, 3047, 3048** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.17 | [V527](https://pvs-studio.com/en/docs/warnings/v527/), [V528](https://pvs-studio.com/en/docs/warnings/v528/), [**V542**](https://pvs-studio.com/en/docs/warnings/v542/), [**V566**](https://pvs-studio.com/en/docs/warnings/v566/), [V601](https://pvs-studio.com/en/docs/warnings/v601/), [V647](https://pvs-studio.com/en/docs/warnings/v647/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 20.10 | **pointer-integral-cast**  **pointer-integral-cast-implicit**  **function-pointer-integer-cast**  **function-pointer-integer-cast-implicit** | Fully checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151949) | 3.11 | [S1767](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-1767) | Partially implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Ensure size arguments for variable length arrays are in a valid range** |
| --- | --- | --- |
| Data Value | STD-009-ARR | Variable length arrays (VLAs), a conditionally supported language feature, are essentially the same as traditional C arrays except that they are declared with a size that is not a constant integer expression and can be declared only at block scope or function prototype scope and no linkage. When supported, a variable length array can be declared where the integer expression size and the declaration of vla are both evaluated at runtime. If the size argument supplied to a variable length array is not a positive integer value, the behavior is undefined. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a variable length array of size size is declared. The size is declared as size\_t in compliance with INT01-C. Use rsize\_t or size\_t for all integer values representing the size of an object. |
| #include <stddef.h>    extern void do\_work(int \*array, size\_t size);    void func(size\_t size) {  int vla[size];  do\_work(vla, size);  } |

| **Compliant Code** |
| --- |
| This compliant solution ensures the size argument used to allocate vla is in a valid range (between 1 and a programmer-defined maximum); otherwise, it uses an algorithm that relies on dynamic memory allocation. The solution also avoids unsigned integer wrapping that, given a sufficiently large value of size, would cause malloc to allocate insufficient storage for the array. |
| #include <stdint.h>  #include <stdlib.h>    enum { MAX\_ARRAY = 1024 };  extern void do\_work(int \*array, size\_t size);    void func(size\_t size) {  if (0 == size || SIZE\_MAX / sizeof(int) < size) {  /\* Handle error \*/  return;  }  if (size < MAX\_ARRAY) {  int vla[size];  do\_work(vla, size);  } else {  int \*array = (int \*)malloc(size \* sizeof(int));  if (array == NULL) {  /\* Handle error \*/  }  do\_work(array, size);  free(array);  }  } |

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

| **Principles(s):** Heed Compiler Warning: Failing to follow this standard will create undefined behavior in the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **REVERSE\_NEGATIVE** | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.3 | **C1051** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.4 | **MISRA.ARRAY.VAR\_LENGTH.2012** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **621 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-ARR32-a** | Ensure the size of the variable length array is in valid range |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **9035** | Assistance provided |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule ARR32-C](https://www.mathworks.com/help/bugfinder/ref/certcrulearr32c.html) | Checks for:   * Memory allocation with tainted size * Tainted size of variable length array   Rule fully covered. |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **1051** | Partially implemented |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **negativeArraySize** | Context sensitive analysis Will warn only if given size is negative |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **alloca\_bounds** | Exhaustively verified. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| Memory Protection | STD-010-MEM | Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities. According to the C Standard, using the value of a pointer that refers to space deallocated by a call to the free() or realloc() function is undefined behavior. Reading a pointer to deallocated memory is undefined behavior because the pointer value is indeterminate and might be a trap representation. Fetching a trap representation might perform a hardware trap (but is not required to). |

| **Noncompliant Code** |
| --- |
| P is freed before p->next is executed, so that p->next reads memory that has already been freed. |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  for (struct node \*p = head; p != NULL; p = p->next) {  free(p);  }  } |

| **Compliant Code** |
| --- |
| Correct this error by storing a reference to p->next in q before freeing p |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  struct node \*q;  for (struct node \*p = head; p != NULL; p = q) {  q = p->next;  free(p);  }  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **dangling\_pointer\_use** | Supported  Astrée reports all accesses to freed allocated memory. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM30** | Detects memory accesses after its deallocation and double memory deallocations |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **ALLOC.UAF** | Use after free |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/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](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.3 | **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](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2021.4 | [UFM.DEREF.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [UFM.DEREF.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [UFM.FFM.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [UFM.FFM.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [UFM.RETURN.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [UFM.RETURN.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [UFM.USE.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [UFM.USE.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **51 D, 484 S, 112 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-MEM30-a** | Do not use resources that have been freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) |  |  | Runtime analysis |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **449, 2434** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule MEM30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem30c.html) | Checks for use of previously freed pointer (rule partially covered) |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **2731, 2732, 2733** |  |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **3339, 4303, 4304** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.17 | [**V586**](https://pvs-studio.com/en/docs/warnings/v586/), [V774](https://pvs-studio.com/en/docs/warnings/v774/) |  |

### 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.

Automation enforcement of standards should occur between the “Build” and “Verify and Test” phases. This is spot allows for validation of our written code after build in which we can do things like “Heed complier warnings” and run “Quality Assurance” test. We can check for security vulnerabilities in our libraries and use our automation application that can check for our standards. Following that will be or functionality test through verify and test.

### 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-DCL | Medium | Unlikely | Medium | P2 | L3 |
| STD-002-INT | High | Likely | High | P9 | L2 |
| STD-003-STR | Medium | Probable | Medium | P8 | L2 |
| STD-004-SQL | High | Probable | Medium | P12 | L1 |
| STD-005-MEM | Medium | Probable | Medium | P8 | L2 |
| STD-006-DCL | Low | Unlikely | High | P1 | L3 |
| STD-007-ERR | Low | Probable | Medium | P4 | L3 |
| STD-008-INT | Low | Probable | High | P2 | L3 |
| STD-009-ARR | High | Probable | High | P6 | L2 |
| STD-010-MEM | High | Likely | Medium | P18 | L1 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | This is the storing of data using encryption methods so that if someone attempts to access the data they will be unable to use it without the proper encryption/decryption algorithms and keys. This data is usually protected through several layers of security such as firewalls and anti-virus software |
| Encryption at flight | This is the process of using encrypted communications levels that the data will travel along. Such as HTTPS or SSL/TLS. |
| Encryption in use | This is the process of transforming the data before transit. This would include hashing passwords on input and comparing the hash password to the store data before providing access to a system. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the process of validating a user or application. It’s making sure the person/app is who they say they are. Methods include usernames and password or additional levels such as 2 Factor Authentication and using something you know(username/password) with something you have (application the provides an randomly generated access key that the system also knows) |
| Authorization | Authorization is giving access to only what a user/application need. Here we using things such as roles and policies to determine if a user has the right authority to access a piece of the system, whether it’s a database, or functions that perform specific actions. |
| Accounting | This is the process of logging and auditing how your systems are used. It allows for someone with the right authorization to see who is doing what and if there are any signs of abuse to the system. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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