Date: September 7, 2015

Automated Source Code Security Measure

Beta 2

OMG Document Number: **ptc/2015-09-02**

Standard document URL:

http://www.omg.org/spec/ASCSM/20150902/AutomatedSourceCodeSecurityMeasure

Machine consumable files:

http://www.omg.org/spec/ASCSM/20150815/AutomatedSourceCodeSecurityMeasure.xmi

http://www.omg.org/spec/ASCSM/20150816/AutomatedSourceCodeSecurityMeasureSMM.xmi

---

OMG's Issue Reporting Procedure

Table of Contents

Preface

About the Object Management Group

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Meta-model); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Formal Specifications are available from this URL: *http://www.omg.org/spec*
Specifications are organized by the following categories:

Business Modeling Specifications Middleware Specifications

- CORBA/IIOP
- Data Distribution Services
- Specialized CORBA

IDL/Language Mapping Specifications

Modeling and Metadata Specifications

- UML, MOF, CWM, XMI
- UML Profile

Modernization Specifications

Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications

- CORBAServices

- CORBAFacilities

CORBA Embedded Intelligence Specifications

CORBA Security Specifications

OMG Domain Specifications

Signal and Image Processing Specifications

All of OMG‟s formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO/IEC standards. Please consult http://www.iso.org

Issues

The reader is encouraged to report and technical or editing issues/problems with this specification to http://www.omg.org

1. Scope

1.1 Purpose

The purpose of this specification is to establish a standard measure of security based on detecting violations of good architectural and coding practices that could result in unauthorized entry into systems, theft of confidential information, and the malicious compromise of system integrity. Establishing a standard for this measure is important because such measures are being used in outsourcing and system development contracts without having an approved international standard to reference. They are also critical to other software-intensive OMG initiatives such as the Internet of Things Consortium.

1.2 Overview of Software Quality Characteristic Measurement

Measurement of the internal or structural quality aspects of software has a long history in software engineering (Curtis, 1980). Software quality characteristics are increasingly being incorporated into development and outsourcing contracts as the equivalent of service level agreements. That is, target thresholds based on quality characteristic measures are being set in contracts for delivered software. Currently there are no standards for most of the software quality characteristic measures being used in contracts. ISO/IEC 25023 purports to address these measures, but only provides measures of external behavior and does not define measures that can be developed from source code during development. Consequently, providers are subject to different interpretations and calculations of common quality characteristics in each contract. This specification addresses one aspect of this problem by providing a specification for measuring one quality characteristic, Security, from the source code. This specification is one of four specifying source code level measures of quality characteristics. The other three specify quality characteristic measures for Security, Performance Efficiency, and Maintainability.

The most recent advance in measuring the structural quality of software is based on the analysis and measurement of violations of good architectural and coding practice that can be detected by statically analyzing the source code. The CWE/SANS 25 and OWASP Top Ten lists of security weaknesses are examples of this approach. These lists are drawn from the Common Weakness Enumeration (CWE) repository maintained by MITRE Corporation. CWE contains descriptions of over 800 weaknesses that represent violations of good architectural and coding practice in software that can be exploited to gain unauthorized entry into a system. The Software Assurance community has been a leader in this area of measurement by championing the detection of code weaknesses as a way of improving one aspect of software quality—software security.

Unfortunately there are no equivalent repositories of weaknesses for Reliability, Performance Efficiency, or Maintainability. Knowledge of these weaknesses is spread across software engineering textbooks, expert blogs, and information sharing sites such as github. An OMG standard for Reliability can fill the void for a consensus body of knowledge about the most egregious Security problems that should be detected and remediated in source code.

Using violations of good architectural and coding practices in software quality metrics presents several challenges for establishing baselines. Growth in the number of unique violations to be detected could continually raise the bar for measuring quality, reducing the validity of baseline comparisons. Further, different vendors will detect different sets of violations, making comparisons difficult across commercial software quality measurement offerings. One solution to this problem is to create a stable list of violations that are used for computing a baseline for each quality characteristic. The Automated Source Code Security Measure was developed by a team of industry experts to form the basis for a stable baseline measure.

1.3 Development of the Automated Source Code Security Measure

The Consortium for IT Software Quality (CISQ) was formed as a special interest group of OMG to create specifications for automating standard measures of software quality attributes and submit them to OMG for approval. The Objective of the Consortium for IT Software Quality (CISQ) is to develop specifications for automated measures of software quality characteristics taken on source code. These measures were designed to provide international standards for measuring software structural quality that can be used by IT organizations, IT service providers, and software vendors contracting, developing, testing, accepting, and deploying software applications. Executives from the member companies that joined CISQ prioritized Reliability, Security, Performance Efficiency, and Maintainability to be developed as measurement specification.

The original 24 CISQ member companies decided to base the security measure on an existing security community body of knowledge concerning exploitable weaknesses. This specification defines a method for automating the measurement of Security from violations of secure architectural and coding practice in source code. These violations were drawn from the Common Weakness Enumeration (CWE) maintained by Mitre Corporation, a cyber-security community repository of over 800 known weaknesses in software that can be exploited for unauthorized intrusion into a system. This specification was developed from the CWE/SANS Institute Top 25 most commonly exploited weaknesses, nitwenty-two of which can be detected in source code. The CWE/SANS Top 25 Most Dangerous Software Errors provides a list of the 25 most widespread and commonly exploited security anti-patterns and associated rules that can be found at http://cwe.mitre.org/top25/#Listing.

1.4 Structure of the Automated Source Code Security Measure

ISO/IEC 25010 defines a quality characteristic as being composed from several quality sub-characteristics. This framework for software product quality is presented in Figure 1 for the eight quality characteristics presented in 25010. The quality characteristics and their sub-characteristics selected for source code measurement by CISQ are indicated in blue.

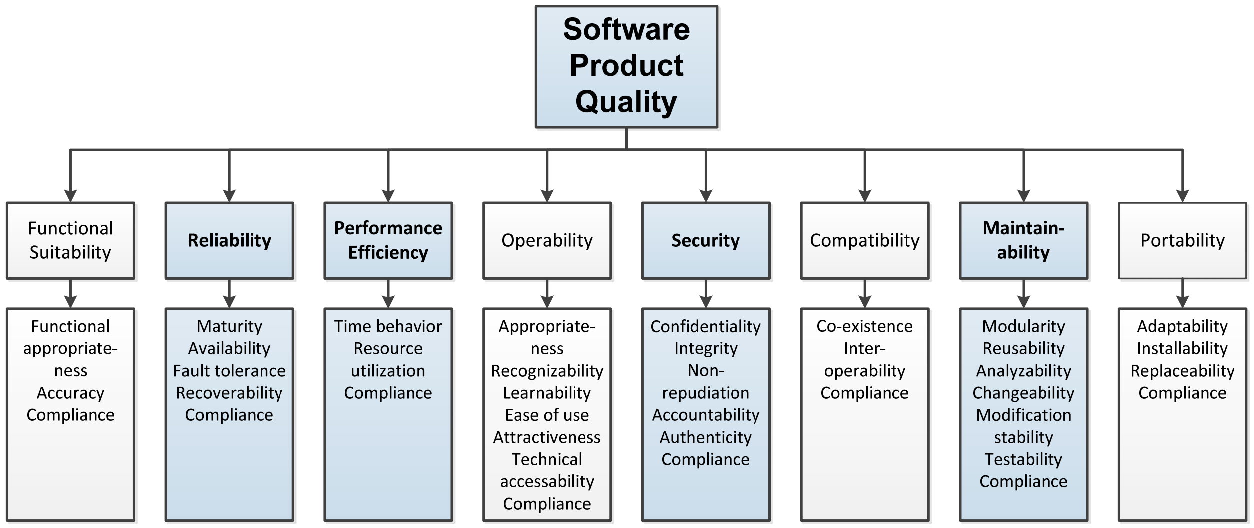
Software Product Quality
Functional Suitability
Reliability
Performance Efficiency
Operability
Security
Compatibility
Maintain-ability
Portability
Functional appropriate-ness Accuracy Compliance
Maturity Availability Fault tolerance Recoverability Compliance
Time behavior Resource utilization Compliance
Appropriate-ness Recognizability Learnability Ease of use Attractiveness Technical accessability Compliance
Confidentiality Integrity Non-repudiation Accountability Authenticity Compliance
Co-existence Inter-operability Compliance
Modularity Reusability Analyzability Changeability Modification stability Testability Compliance
Adaptability Installability Replaceability Compliance

Figure 1. Software Quality Characteristics from ISO/IEC 25010 with CISQ focal areas highlighted.

ISO/IEC 25023 establishes a framework of software quality characteristic measures wherein each quality sub-characteristic consists of a collection of quality attributes that can be quantified as quality measure elements. A quality measure element quantifies a unitary measurable attribute of software, such as the violation of a quality rule. Figure 2 presents an example of the ISO/IEC 25023 quality measurement framework using a partial decomposition for the Automated Source Code Security Measure.

The non-normative portion of this specification begins by listing the security issues that can plague software developed with poor architectural and coding practices. Quality rules written as architectural or coding practices are conventions that avoided the problem described in the security issue. These quality rules were then transformed into software quality measure elements by counting violations of these architectural and coding practices and conventions.

The normative portion of this specification represents each quality measure element developed from a security rule using the Structured Patterns Meta-model Standard (SPMS). The code-based elements in these patterns are represented using the Knowledge Discovery Meta-model (KDM). The calculation of the Automated Source Code Security Measure from its quality measure elements is then represented in the Structured Metrics Meta-model (SMM). This calculation is presented as the simple sum of quality measure elements without being adjusted by a weighting scheme.

There are several weighting schemes that can be applied in aggregating violation counts into structural quality measures. The most effective weighting often depends on the measure's use such as assessing operational risk or estimating maintenance costs. The quality measure elements included in this specification were considered to be severe violations of secure architectural and coding practices that would need to be remediated. Therefore, weightings based on severity would add little useful information to the measure since the variance among weights would be small. In order to support benchmarking among applications, this specification includes a measure of the violation density. This measure is created by dividing the total number of violations detected by a count of Automated Function Points (Object Management Group, 2014).

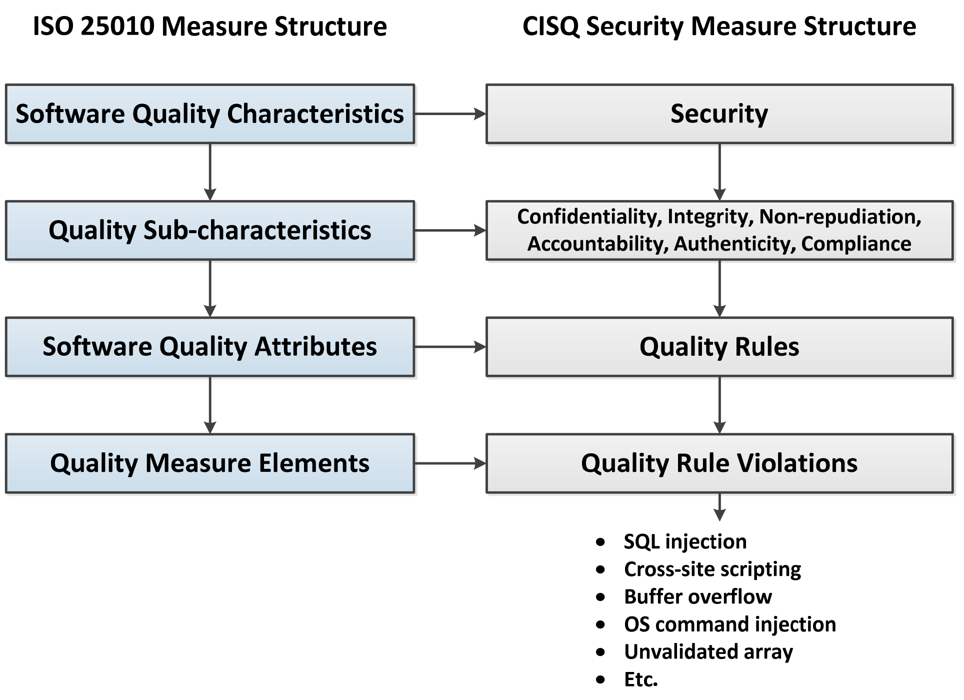
ISO 25010 Measure Structure
CISQ Security Measure Structure
Software Quality Characteristics
Security
Quality Sub-characteristics
Confidentiality, Integrity, Non-repudiation, Accountability, Authenticity, Compliance
Software Quality Attributes
Quality Rules
Quality Measure Elements
Quality Rule Violations
SQL injection
Cross-site scripting
Buffer overflow
OS command injection
Unvalidated array
Etc.

Figure 2. ISO/IEC 25010 Framework for Software Quality Characteristics Measurement

1.5 CWE/SANS Top 25 Weaknesses

The foundation for this specification is the CWE/SANS Institute Top 25 Most Dangerous Software Errors that provides a list of the 25 most widespread and frequently exploited security weaknesses in software. The anti-patterns and associated rules that constitute these weaknesses can be found in the Common Weakness Enumeration accessible at http://cwe.mitre.org/top25/#Listing. This specification is developed from nineteen of the CWE/SANS Top 25 which can be detected and counted in source code. The CWE is a widely used industry source (http://cwe.mitre.org/community/citations.html) that provides a foundation for an ITU and ISO/IEC standard, in addition to 2 ISO/IEC technical reports:

- SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY Cybersecurity information exchange – Vulnerability/state exchange - Common weakness enumeration (CWE)
- ISO/IEC 29147:2014 Information Technology -- Security Techniques -- Vulnerability Disclosure"
- ISO/IEC TR 24772:2013 Information technology -- Programming languages -- Guidance to avoiding vulnerabilities in programming languages through language selection and use
- ISO/IEC Technical Report is ISO/IEC TR 20004:2012 Information Technology -- Security Techniques -- Refining Software Vulnerability Analysis under ISO/IEC 15408 and ISO/IEC 18045

The Automated Source Code Security Measure is a correlated measure rather than an absolute measure. That is, since it does not measure all possible security-related weaknesses it does not provide an absolute measure of security. However, since it includes counts of what industry experts have determined to be the top 25 known weaknesses, it provides a strong indicator of security that will be highly correlated with the absolute security of a software system and with the probability that it can be breached.

Since the CWE is recognized as the primary industry repository of security weaknesses (Lewis, 2010), it is supported by the majority of vendors providing tools and technology in the software security domain (http://cwe.mitre.org/compatible/compatible.html), such as Coverity, HP Fortify, Klockwork, IBM, CAST, Veracode, and others. These vendors already have capabilities for detecting many of the CWE/SANS Top 25 security weaknesses. Consequently, CWE/SANS Top 25 provides the best source for developing a measure that can be common among the majority of vendors in the software security domain. Industry experts who developed the CWE purposely worded the CWEs to be language and application agnostic in order to allow vendors to develop detectors specific to a wide range of languages and application types beyond the scope that could be covered in the CWE. Since some of the CWE/SANS Top 25 may not be relevant in some languages, the reduced opportunity for anti-patterns in those cases will be reflected in the scores.

Since the impact and frequency of specific violations in the CWE/SANS Top 25 could change over time, this approach allows specific violations to be included, excluded, amplified, or diminished over time in order to support the most effective benchmarking, diagnostic, and predictive use. This specification will be adjusted through controlled OMG processes to reflect changes in the threat environment while retaining the ability to compare baselines. Measurement vendors can compute this standard baseline measure, as well as their own extended measures that include other security anti-patterns.

1.6 Using and Improving This Measure

The Automated Source Code Security Measure is a correlated measure rather than an absolute measure. That is, since it does not measure all possible security-related weaknesses it does not provide an absolute measure of security. However, since it includes counts of what industry experts considered high severity security weaknesses, it provides a strong indicator of security that will be highly correlated with the absolute security of a software system and with the probability that it can experience unauthorized penetrations, data theft, malicious internal damage, and related problems.

Since the impact and frequency of specific violations in the Automated Source Code Security Measure could change over time, this approach allows specific violations to be included, excluded, amplified, or diminished over time in order to support the most effective benchmarking, diagnostic, and predictive use. This specification will be adjusted through controlled OMG specification revision processes to reflect changes in security engineering while retaining the ability to compare baselines. Vendors of static analysis and measurement technology can compute this standard baseline measure, as well as their own extended measures that include other security weaknesses not included as measure elements in this specification.

2. Conformance

Implementations of this specification should be able to demonstrate the following attributes in order to claim conformance—automated, objective, transparent, and verifiable.

- ***Automated***—The analysis of the source code and the actual counting must be fully automated. The initial inputs required to prepare the source code for analysis include the source code of the application, the artifacts and information needed to configure the application for operation, and any available description of the architectural layers in the application.
- ***Objective***—After the source code has been prepared for analysis using the information provided as inputs, the analysis, calculation, and presentation of results must not require further human intervention. The analysis and calculation must be able to repeatedly produce the same results and outputs on the same body of software.
- ***Transparent***—Implementations that conform to this specification must clearly list all source code (including versions), non-source code artifacts, and other information used to prepare the source code for submission to the analysis.
- ***Verifiable***—Compliance with this specification requires that an implementation state the assumptions/heuristics it uses with sufficient detail so that the calculations may be independently verified by third parties. In addition, all inputs used are required to be clearly described and itemized so that they can be audited by a third party.

3. Normative References

3.1 Normative

The following normative documents contain provisions, which, through reference in this text, constitute provisions of this specification. For dated references, subsequent amendments to, or revisions of any of these publications do not apply.

- Structured Patterns Meta-model Standard, admtf/14-02-01
- Knowledge Discovery Meta-model, version 1.3 (KDM), formal/2011-08-04
- Structured Metrics Meta-model, version 1.0 (SMM), formal/2012-01-05
- MOF/XMI Mapping, version 2.4.1 (XMI), formal/2011-08-09
- Automated Function Points (AFP), formal/2014-01-03
- ISO/IEC 25010 Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models

4. Terms and Definitions

For the purposes of this specification, the following terms and definitions apply.

Automated Function Points—a specification for automating the counting of Function Points that mirrors as closely as possible the counting guidelines of the International Function Point User Group. (OMG, formal 2014-01-03)

Common Weakness Enumeration—a repository maintained by MITRE Corporation of known weaknesses in software that can be exploited to gain unauthorized entry into a software system. (cwe.mitre.org)

Cyclomatic Complexity—A measure of control flow complexity developed by Thomas McCabe based on a graph-theoretic analysis that reduces the control flow of a computer program to a set of edges, vertices, and their attributes that can be quantified. (McCabe, 1976)

Internal Software Quality—the degree to which a set of static attributes of a software product satisfy stated and implied needs for the software product to be used under specified conditions. This will be referred to as software structural quality, or simply structural quality in this specification. (ISO/IEC 25010)

Quality Measure Element—a measure defined in terms of a software quality attribute and the measurement method for quantifying it, including optionally the transformation by a mathematical function. (ISO/IEC 25010)

Software Quality Property—measurable component of software quality. (derived from ISO/IEC 25010)

Security—degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization. (ISO/IEC 25010)

Software Anti-pattern—also referred to as an anti-pattern, is a violation of good architectural or coding practice that can, based on historical evidence, cause problems in software development, maintenance, or operations.

Software Product—a set of computer programs, procedures, and possibly associated documentation and data. (ISO/IEC 25010)

Software Product Quality Model—a model that categorizes product quality properties into eight characteristics (functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability). Each characteristic is composed of a set of related sub-characteristics. (ISO/IEC 25010)

Software Quality—degree to which a software product satisfies stated and implied needs when used under specified conditions. (ISO/IEC 25010)

Software Quality Attribute—an inherent property or characteristic of software that can be distinguished quantitatively or qualitatively by human or automated means. (derived from ISO/IEC 25010)

Software Quality Characteristic—a category of software quality attributes that bears on software quality. (ISO/IEC 25010)

Software Quality Characteristic Measure—a software quality measure derived from measuring the attributes related to a specific software quality characteristic.

Software Quality Issue—architectural or coding practices that are known to cause problems in software development, maintenance, or operations and for which software quality rules can be defined that help avoid problems created by the issue.

Software Quality Measure—a measure that is defined as a measurement function of two or more values of software quality measure elements. (ISO/IEC 25010)

Software Quality Measure Element—a measure defined in terms of a software quality attribute and the measurement method for quantifying it, including optionally the transformation by a mathematical function. (ISO/IEC 25010)

Software Quality Measurement—(verb) a set of operations having the object of determining a value of a software quality measure. (ISO/IEC 25010)

Software Quality Model—a defined set of software characteristics, and of relationships between them, which provides a framework for specifying software quality requirements and evaluating the quality of a software product. (derived from ISO/IEC 25010)

Software Quality Property—measureable component of software quality. (derived from ISO/IEC 25010)

Software Quality Rule—an architectural or coding practice or convention that represents good software engineering practice and avoids problems in software development, maintenance, or operations. Violations of these quality rules produces software anti-patterns.

Software Quality Sub-characteristic—a sub-category of a software quality characteristic to which software quality attributes and their software quality measure elements are conceptually related. (derived from ISO/IEC 25010)

Software Security— degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization. (ISO/IEC 25010)

Software Security Measure Element—a measure defined in terms of a quality attribute of software that affects it security and the measurement method for quantifying it, including optionally the transformation by a mathematical function. (adapted from ISO/IEC 25023)

Structural Element—a component of software code that can be uniquely identified and counted such as a token, decision, variable, etc.

Structural Quality—the degree to which a set of static attributes of a software product satisfy stated and implied needs for the software product to be used under specified conditions—a component of software quality. This concept is referred to as internal software quality in ISO/IEC 25010.

Violation—a pattern or structure in the code that is inconsistent with good architectural and coding practices and can lead to problems in operation or maintenance.

5. Symbols (and Abbreviated Terms)

CWE – Common Weakness Enumeration

CISQ – Consortium for IT Software Quality

KDM – Knowledge Discovery Meta-model

SPMS – Structured Pattern Meta-model Standard

SMM – Structured Metrics Meta-model

6. Additional Information (Informative)

6.1 Software Product Inputs

The following inputs are needed by static code analyzers in order to interpret violations of the software quality rules that would be included in individual software quality measure elements.

- The entire source code for the application being analyzed
- All materials and information required to prepare the application for production
- A list of vetted libraries that are being used to "neutralize" input data
- What routines/API calls are being used for remote authentication, to any custom initialization and cleanup routines, to synchronize resources, or to neutralize accepted file types or the names of resources
- The encryption algorithms that are being used

Static code analyzers will also need a list of the violations that constitute each quality element in the Automated Source Code Security Measure.

6.2 Automated Source Code Security Measure Elements

The violations of good architectural and coding practice incorporated into the Automated Source Code Security Measure are listed and describe in the following Table 1. Some of the CWEs from the Common Weakness Enumeration repository that are included in the Security measure are also defects that can cause security problems. In order to retain consistency across measurement specifications, the original CWE numbers and titles have been retained for these security measure elements. In this sub-clause and in Clause 7 each security measure element from Table 1 will be labeled as ASCSM-#, where # can be replaced by its CWE number.

Table 1. Security Patterns, Consequences, Objectives, and Measure Elements

Security Pattern	Consequence	Objective	Measure Element
ASCSM-CWE-22: Path Traversal Improper Input Neutralization	Software that is unaware of file path control incurs the risk of exposition of sensitive data, the risk of corruption of critical files, such as programs, libraries, or important data used in protection mechanisms	Avoid failure to sanitize user input in use in path manipulation operations	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the file path creation statement; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.
ASCSM-CWE-78: OS Command Injection Improper Input Neutralization	Software unaware of OS command control incurs the risk of unauthorized command execution, possibly used to disable the software, or possibly leading to unauthorized read and modify data access	Avoid failure to sanitize user input in use as operating system commands	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the in the platform action to be executed by the execution environment; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.
ASCSM-CWE-79: Cross-site Scripting	Software featuring weak output generation practices incurs the risk of arbitrary code	Avoid failure to sanitize user input in use in output	Number of instances where an external value is entered into the application through the user interface ReadsUI action,

Improper Input Neutralization	execution, the risk of sensitive data being compromised, and many other nefarious consequences	generation operations	transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the in the user interface WritesUI action; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.
ASCSM-CWE-89: SQL Injection Improper Input Neutralization	Software unaware of SQL command control incurs the risk of unauthorized read, modify, and delete access to sensitive data	Avoid failure to sanitize user input in use in SQL compilation operations	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the in the SQL compilation statement; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.
ASCSM-CWE-99: Name or Reference Resolution Improper Input Neutralization	Software unaware of resource identification control incurs the risk of unauthorized access to or modification of sensitive data and system resources, including configuration files and files containing sensitive information	Avoid failure to sanitize user input in use as resource names or references	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method

			control elements, and ultimately used in the in the platform action to access a resource by its name; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.
ASCSM-CWE-120: Buffer Copy without Checking Size of Input	Software that is unaware of buffer bounds incurs the risk of corruption of relevant memory, and perhaps instructions, possibly leading to a crash, the risk of data integrity loss, and the risk of unauthorized access to sensitive data	Avoid buffer operations among buffers with incompatible sizes	Number of instances in which the content of the first buffer is moved into the content of the second buffer while the size of the first buffer is greater than the size of the second buffer.
ASCSM-CWE-129: Array Index Improper Input Neutralization	Software that is unaware of array index bounds incurs the risk of corruption of relevant memory, and perhaps instructions, possibly leading to a crash, the risk of data integrity loss, and the risk of unauthorized access to sensitive data	Avoid failure to check range of user input in use as array index	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the read or write action to access the array; none of the callable or method control element of the transformation sequence being a range check callable and method control element with regards to the array index.
ASCSM-CWE-134: Format String Improper Input Neutralization	Software that is unaware of formatting control incurs the risk of execution of arbitrary code and the risk of	Avoid failure to sanitize user input in use in formatting operations	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the

	information disclosure which can severely simplify exploitation of the software		application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the formating statement; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.
ASCSM-CWE-252-resource: Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Platform Resource	Software unaware of execution status control incurs the risk of bad data being used in operations, possibly leading to a crash or other unintended behaviors	Avoid improper processing of the execution status of resource handling operations	Number of instances where the named callable control element or method control element executes a 'Read', 'Write', or 'Manage Access' action, yet the value of the return parameter from the action is not used by any check control element
ASCSM-CWE-327: Broken or Risky Cryptographic Algorithm Usage	Software using broken or risky cryptographic algorithm incurs the risk of sensitive data being compromised	Avoid failure to use vetted cryptographic libraries	Number of instances where the application uses the cryptographic deployed component which is not part of the list of vetted cryptographic deployed components.
ASCSM-CWE-396: Declaration of Catch for Generic Exception	Software unaware of accurate execution status control incurs the risk of bad data being used in operations, possibly leading to a crash or other unintended behaviors	Avoid failure to use dedicated exception types	Number of instances where the named callable control element or method control element contains a catch unit which declares to catch an exception parameter whose data type is part of a list of overly broad exception data types
ASCSM-CWE-397: Declaration of Throws for Generic Exception	Software unaware of accurate execution status control incurs the risk of bad data being used in operations, possibly leading to a	Avoid failure to use dedicated exception types	Number of instances where the named callable control element or method control element throws an exception parameter whose data type is part of a list of

	crash or other unintended behaviors		overly broad exception data types
ASCSM-CWE-434: File Upload Improper Input Neutralization	Software unaware of file upload control incurs the risk of arbitrary code execution	Avoid failure to sanitize user input in use in file upload operations	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the file file upload action; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.
ASCSM-CWE-456: Storable and Member Data Element Missing Initialization	Software featuring weak initialization practices incurs the risk of logic errors within the program, possibly leading to a security problem	Avoid failure to explicitly initialize software data elements in use	Number of instances where a storable data element or member data element is declared by the 'Create' action, then is evaluated in a 'Read' action without ever being initialized by a 'Write' action prior to the evaluation
ASCSM-CWE-606: Unchecked Input for Loop Condition	Software unaware of iteration control incurs the risk of unexpected consumption of resources, such as CPU cycles or memory, possibly leading to a crash or program exit due to exhaustion of resources	Avoid failure to check range of user input in use in iteration control	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the loop condition statement; none of the callable or method control element of the transformation sequence being a range check control element.

ASCSM-CWE-667: Shared Resource Improper Locking	Software featuring inconsistent locking discipline incurs the risk of deadlock	Avoid data corruption during concurrent access	Number of instances where the shared storable data element or member data element, declared with the Create action, is accessed outside a critical section of the application via the Read or Write action.
ASCSM-CWE-672: Expired or Released Resource Usage	Software unaware of resource lifecycle incurs the risk of unauthorized access to sensitive data that is associated with a different user or entity, and the risk of erroneous later attempts to access the resource, possibly leading to a crash	Avoid access to a released, revoked, or expired resource	Number of instances where the platform resource is deallocated in the Manage action using its unique resource handler value which is transported throughout the application via the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, then used later within the application to try and access the resource in the Read or Write action.
ASCSM-CWE-681: Numeric Types Incorrect Conversion	Software featuring weak numerical conversion practices incurs the risk of using the wrong number and generating incorrect results, possibly introducing new vulnerability when related to resource allocation and security decision	Avoid numerical data corruption during incompatible mutation	Number of instances where a storable element or member element is declared with a numerical data type in the 'Create' action, and then is updated with a value which is cast via a type cast action into a second numerical data type, which is incompatible with the first data type
ASCSM-CWE-772: Missing Release of Resource after Effective Lifetime	Software unaware of resource lifecycle incurs the risk of preventing all other processes from accessing the same type of resource	Avoid resource hoarding and consequently resource depletion	Number of instances where a platform resource is allocated and assigned a unique resource handler value via a manage resource action, and its unique resource handler value is used throughout the application along a transformation sequence composed of action elements with data relations, some of which are part of named callable and method control elements,

			but none of which is a resource release statement
ASCSM-CWE-789: Uncontrolled Memory Allocation	Software that is unaware of buffer bounds incurs the risk of corruption of relevant memory, and perhaps instructions, possibly leading to a crash, the risk of data integrity loss, and the risk of unauthorized access to sensitive data	Avoid failure to check range of user input in use as buffer index	Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the buffer Read or Write access action; none of the callable or method control element of the transformation sequence being a range check control element.
ASCSM-CWE-798: Hard-Coded Credentials Usage for Remote Authentication	Software featuring weak authentication practices incurs the risk of exposing resources and functionality to unintended actors, possibly leading to compromised sensitive information and even the execution of arbitrary code	Avoid the existence of hard-coded credentials elements	Number of instances where a storable data element or member data element is initialized by a 'Write' action, transported throughout the application along the transport sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the remote resource management action ; the transport sequence is composed of assignment operations as updates to the value would not be considered as hard-coded (literal) any more.
ASCSM-CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop')	Software unaware of iteration control incurs the risk of unexpected consumption of resources, such as CPU	Avoid infinite iterations	Number of instances where the named callable control element or method control element features the executioon path whose entry element is found

	cycles or memory, possibly leading to a crash or program exit due to exhaustion of resources		again in the path, while it has no path whatsoever to not return to itself and exit the recursion

7 SPMS Representation of the Security Quality Measure Elements (Normative)

7.1 Introduction

This chapter displays in a human readable format the content of the machine readable XMI format file attached to the current specification. The content of the machine readable XMI format file is the representations of the Quality Measure Elements

- according to the Implementation Patterns Metamodel for Software Systems (SPMS)
- and relating to the Knowledge Discovery Meta-Model (KDM) within their description as frequently as possible, so as to be as generic as possible yet as accurate as possible.

SPMS

More specifically, the machine readable XMI format file attached to the current specification uses the SPMS Definitions Classes:

- PatternDefinition (SPMS:PatternDefinition): the pattern specification. In the context of this document, each Quality Measure Element is basically the count of occurrences of the described patterns.
- Role (SPMS:Role): “A pattern is informally defined as a set of relationships between a set of entities. Roles describe the set of entities within a pattern, between which those relationships will be described. As such the Role is a required association in a PatternDefinition. […]. Semantically, a Role is a 'slot' that is required to be fulfilled for an instance of its parent PatternDefinition to exist.”
- PatternSection (SPMS:PatternSection): “A PatternSection is a free-form prose textual description of a portion of a PatternDefinition.” In the context of this document, there are 6 different PatternSections in use:
    - “Descriptor” to provide pattern signature, a visible interface of the pattern,
    - “Measure Element” to provide a human readable explanation of the measure,
    - “Description” to provide a human readable explanation of the pattern that is sought after, identifying “Roles” and KDM modeling information,
    - “Objective” to provide a human readable explanation of the intent to get rid of the occurrences of the pattern that is sought after,
    - “Consequence” to provide a human readable explanation of the issue the detection of the pattern is designed to solve,
    - “Input” to provide a human readable of the parameters that are needed to fine-tune the behavior of the pattern detection (e.g.: the target application architectural blueprint to comply with)

    - "Comment" to provide some additional information (until now, used to inform about situations where the same measure element is useful for another one of the categories)

As well as some of the SPMS Relationships Classes:

- MemberOf (SPMS:MemberOf): "An InterpatternRelationship specialized to indicate inclusion in a Category"
- Category (SPMS:Category): "A Category is a simple grouping element for gathering related PatternDefinitions into clusters." In the context of this document, the SPMS Categories are used to represent the 4 Quality Characteristics:
    - "Reliability",
    - "Security",
    - "Performance Efficiency",
    - And "Maintainability".

KDM

More specifically, the machine readable XMI format file attached to the current specification uses KDM entities in the "Description" section of the pattern definitions.   Descriptions try to remain as generic yet accurate as possible so that the pattern can be applicable and applied to as many situations as possible: different technologies, different programming languages, etc. This means:

1. The descriptions include information such as (code:MethodUnit), (action:Reads), (platform:ManagesResource), … to identify the KDM entities the pattern definition involves
2. The descriptions only detail the salient aspects of the pattern as the specifics can be technology- or language-dependant

KDM is helpful for reading this chapter. However, for readers not familiar with KDM, Table 2 presents a primer which translates standard source code element terms into the KDM wording in this specification.

Reading guide

For each numbered sub-clause of this clause

- Sub-clause 7.2 represents the SPMS Category covered by the current specification
- Starting with sub-clause 7.3 represents a new SPMS PatternDefinition member of this SPMS Category

Table 2. **Software elements translated into KDM wording**

Software element	KDM wording
function, method, procedure, stored procedure, sub-routine etc.	named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit)
variable, field, member, etc.	storable data element (code:StorableUnit) or member data element (code:MemberUnit)
class	class element (code:StorableUnit with code:DataType code:ClassUnit)
interface	interface element (code:StorableUnit of code:DataType code:InterfaceUnit)
method	method element (code:MethodUnit)
field, member	member element (code:MemberUnit)
SQL stored procedures	stored callable control elements (code:CallableUnit with code:CallableKind 'stored') in a data manager resource (platform:DataManager)
return code value	value (code:Value) of the return parameter (code:ParameterUnit of code:ParameterKind 'return')
exception	exception parameter (code:ParameterUnit with code:ParameterKind 'exception')
user input data flow	an external value is entered is entered into the application through the 'ReadsUI' user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the 'TransformationSequence' sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used as
execution path	execution path (action:BlockUnit composed of action:ActionElements with action:CallableRelations to code:ControlElements)
Libraries, etc.	deployed component (platform:DeployedComponent)
RDBMS	data manager resource (platform:DataManager)
loop body	loop body block (action:BlockUnit starting as the action:TrueFlow of the loop action:GuardedFlow and ending with an action:Flow back to the loop action:GuardedFlow)
loop condition	loop condition (action:BlockUnit used in the action:GuardedFlow)
singleton	class element (code:StorableUnit with code:DataType code:ClassUnit) that can be used only once in the 'to' assoction of a Create action (action:Creates)
checked	used by a check control element (code:ControlElement containing action:ActionElement with a kind from micro KDM list of comparison actions)

SPMS PatternDefinition sub-clauses are:

- Pattern category: the “SPMS:Category” category the pattern is related to through a “SPMS:MemberOf” relationship.
- Pattern sections: the list of "SPMS:PatternSection" sections from the pattern:
  - “Descriptor”,
  - “Description”,
  - “Objective”,
  - “Consequence”,
  - and, when applicable,
    - “Input”,
    - “Comment”.
- Pattern roles: the list of “SPMS:Role” roles used in the “Descriptor”, and “Description” sub-clauses above.

In the following sub-clauses,

- Data between square brackets (e.g.: [key Reliabity]) identifies “xmi:id” that are unique and used to reference entities. They are machine-generated to ensure unicity.
- Data between paranthesis (e.g.: (code:MethodUnit)) identifies KDM modeling information.
- Data between angle brackets (e.g.: <ControlElement>) identifies SPMS Roles in Description and Input sub-clauses.

7.2 Category definition of Security

[key ASCSM_Security] Security

7.3 Pattern definition of ASCSM-CWE-22: Path Traversal Improper Input Neutralization

Pattern Category

[key ASCSM-CWE-22-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-22-objective]
Avoid failure to sanitize user input in use in path manipulation operations

Consequence

[key ASCSM-CWE-22-consequence]
Software that is unaware of file path control incurs the risk of exposition of sensitive data, the risk of corruption of critical files, such as programs, libraries, or important data used in protection mechanisms

Measure Element

[key ASCSM-CWE-22-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the file path creation statement; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.

Description

[key ASCSM-CWE-22-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <PathCreationStatement> file path creation statement (platform:ManagesResource with platform:FileResource); none of the callable or method control element of the transformation sequence being a vetted sanitization callable and method control element (code:ControlElement) from the <PathTraversalSanitizationControlElementList> list of vetted sanitization control elements.

Descriptor

[key ASCSM-CWE-22-descriptor]
ASCSM-CWE-22(UserInput: userInput,PathCreationStatement: pathCreationStatement, TransformationSequence: transformationSequence, PathTraversalSanitizationControlElementList: pathTraversalSanitizationControlElementList)

Variable input

[key ASCSM-CWE-22-input]
<PathTraversalSanitizationControlElementList> list of control elements vetted to handle path traversal vulnerabilities

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-22-roles-userInput] UserInput
[key ASCSM-CWE-22-roles-pathCreationStatement] PathCreationStatement

[key ASCSM-CWE-22-roles-transformationSequence] TransformationSequence
[key ASCSM-CWE-22-roles-pathTraversalSanitizationControlElementList]
PathTraversalSanitizationControlElementList

7.4 Pattern definition of ASCSM-CWE-78: OS Command Injection Improper Input Neutralization

Pattern Category

[key ASCSM-CWE-78-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-78-objective]
Avoid failure to sanitize user input in use as operating system commands

Consequence

[key ASCSM-CWE-78-consequence]
Software unaware of OS command control incurs the risk of unauthorized command execution, possibly used to disable the software, or possibly leading to unauthorized read and modify data access

Measure Element

[key ASCSM-CWE-78-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the in the platform action to be executed by the execution environment; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.

Description

[key ASCSM-CWE-78-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <ExecuteRunTimeCommandStatement> platform action (platform:PlatformActions) to be execuded by the execution environment (platform:ExecutionResource); none of the callable or method control element of the transformation sequence being a vetted sanitization callable and method control element from the <OSCommandSanitizationControlElementList> list of vetted sanitization callable and method control elements.

Descriptor

[key ASCSM-CWE-78-descriptor]
ASCSM-CWE-78(UserInput: userInput,ExecuteRunTimeCommandStatement: executeRunTimeCommandStatement, TransformationSequence: transformationSequence, OSCommandSanitizationControlElementList: oSCommandSanitizationControlElementList)

Variable input

[key ASCSM-CWE-78-input]
<OSCommandSanitizationControlElementList> list of control elements vetted to handle Command Injection vulnerabilities

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-78-roles-userInput] UserInput
[key ASCSM-CWE-78-roles-executeRunTimeCommandStatement] ExecuteRunTimeCommandStatement
[key ASCSM-CWE-78-roles-transformationSequence] TransformationSequence
[key ASCSM-CWE-78-roles-oSCommandSanitizationControlElementList] OSCommandSanitizationControlElementList

7.5 Pattern definition of ASCSM-CWE-79: Cross-site Scripting Improper Input Neutralization

Pattern Category

[key ASCSM-CWE-79-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-79-objective]
Avoid failure to sanitize user input in use in output generation operations

Consequence

[key ASCSM-CWE-79-consequence]
Software featuring weak output generation practices incurs the risk of arbitrary code execution, the risk of sensitive data being compromised, and many other nefarious consequences

Measure Element

[key ASCSM-CWE-79-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method

control elements, and ultimately used in the in the user interface WritesUI action; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.

Description

[key ASCSM-CWE-79-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <UserDisplay> user interface WritesUI action (ui:WritesUI); none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the <CrossSiteScriptingSanitizationControlElementList> list of vetted sanitization control elements.

Descriptor

[key ASCSM-CWE-79-descriptor]
ASCSM-CWE-79(UserInput: userInput,CrossSiteScriptingSanitizationControlElementList: crossSiteScriptingSanitizationControlElementList, UserDisplay: userDisplay, TransformationSequence: transformationSequence)

Variable input

[key ASCSM-CWE-79-input]
<CrossSiteScriptingSanitizationControlElementList> list of control elements vetted to deal with cross-site scripting vulnerability

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-79-roles-userInput] UserInput
[key ASCSM-CWE-79-roles-crossSiteScriptingSanitizationControlElementList] CrossSiteScriptingSanitizationControlElementList
[key ASCSM-CWE-79-roles-userDisplay] UserDisplay
[key ASCSM-CWE-79-roles-transformationSequence] TransformationSequence

7.6 Pattern definition of ASCSM-CWE-89: SQL Injection Improper Input Neutralization

Pattern Category

[key ASCSM-CWE-89-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-89-objective]
Avoid failure to sanitize user input in use in SQL compilation operations

Consequence

[key ASCSM-CWE-89-consequence]
Software unaware of SQL command control incurs the risk of unauthorized read, modify, and delete access to sensitive data

Measure Element

[key ASCSM-CWE-89-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the in the SQL compilation statement; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.

Description

[key ASCSM-CWE-89-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <SQLCompilationStatement> SQL compilation statement (data:ReadsColumnSet or data:WritesColumnSet or data:ManagesData or action:Calls to a code:CallableUnit stored in the data:DataResource); none of the callable or method control element of the transformation sequence being a vetted sanitization callable and method control elements from the <SQLInjectionSanitizationControlElementList> list of vetted sanitization control elements.

Descriptor

[key ASCSM-CWE-89-descriptor]
ASCSM-CWE-89(UserInput: userInput,SQLCompilationStatement: sQLCompilationStatement, TransformationSequence: transformationSequence, SQLInjectionSanitizationControlElementList: sQLInjectionSanitizationControlElementList)

Variable input

[key ASCSM-CWE-89-input]
<SQLInjectionSanitizationControlElementList> list of control elements vetted to handle SQL injection vulnerabilities.

Comment
(none applicable)

List of Roles
[key ASCSM-CWE-89-roles-userInput] UserInput
[key ASCSM-CWE-89-roles-sQLCompilationStatement] SQLCompilationStatement
[key ASCSM-CWE-89-roles-transformationSequence] TransformationSequence
[key ASCSM-CWE-89-roles-sQLInjectionSanitizationControlElementList]
SQLInjectionSanitizationControlElementList

7.7 Pattern definition of ASCSM-CWE-99: Name or Reference Resolution Improper Input Neutralization

Pattern Category
[key ASCSM-CWE-99-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective
[key ASCSM-CWE-99-objective]
Avoid failure to sanitize user input in use as resource names or references

Consequence
[key ASCSM-CWE-99-consequence]
Software unaware of resource identification control incurs the risk of unauthorized access to or modification of sensitive data and system resources, including configuration files and files containing sensitive information

Measure Element
[key ASCSM-CWE-799measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the in the platform action to access a resource by its name; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.

Description
[key ASCSM-CWE-99-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of

named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <AccessByNameStatement> platform action (platform:PlatformActions) to access a resource (platform:ResourceType) by its name; none of the callable or method control element of the transformation sequence being a vetted sanitization callable and method control elements from the <NameOrReferenceResolutionSanitizationControlElementList> list of vetted sanitization callable and method control elements.

Descriptor

[key ASCSM-CWE-99-descriptor]
ASCSM-CWE-99(UserInput: userInput,AccessByNameStatement: accessByNameStatement, TransformationSequence: transformationSequence, NameOrReferenceResolutionSanitizationControlElementList: nameOrReferenceResolutionSanitizationControlElementList)

Variable input

[key ASCSM-CWE-99-input]
<NameOrReferenceResolutionSanitizationControlElementList> list of control elements vetted to handle Name or Reference Resolution vulnerabilities

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-99-roles-userInput] UserInput
[key ASCSM-CWE-99-roles-accessByNameStatement] AccessByNameStatement
[key ASCSM-CWE-99-roles-transformationSequence] TransformationSequence
[key ASCSM-CWE-99-roles-nameOrReferenceResolutionSanitizationControlElementList] NameOrReferenceResolutionSanitizationControlElementList

7.8 Pattern definition of ASCSM-CWE-120: Buffer Copy without Checking Size of Input

Pattern Category

[key ASCSM-CWE-120-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-120-objective]
Avoid buffer operations among buffers with incompatible sizes

Consequence

[key ASCSM-CWE-120-consequence]
Software that is unaware of buffer bounds incurs the risk of corruption of relevant memory, and perhaps instructions, possibly leading to a crash, the risk of data integrity loss, and the risk of unauthorized access to sensitive data

Measure Element

[key ASCSM-CWE-120-measure-element]
Number of instances in which the content of the first buffer is moved into the content of the second buffer while the size of the first buffer is greater than the size of the second buffer.

Description

[key ASCSM-CWE-120-description]
This pattern identifies situations where two buffer storable elements (code:StorableUnit) or member elements (code:MemberUnit) are allocated with specific sizes in <SourceBufferAllocationStatement> and <TargetBufferAllocationStatement> Create actions (action:Creates), transformed within the application via the <SourceTransformationSequence> and <TargetTransformationSequence> sequences (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), then ultimately used by the application to move the content of the first buffer (action:Reads) onto the content of the second buffer (action:Writes) through the <MoveBufferStatement> statement, while the size of the first buffer is greater than the size of the second buffer.

Descriptor

[key ASCSM-CWE-120-descriptor]
ASCSM-CWE-120(SourceBufferAllocationStatement: sourceBufferAllocationStatement,TargetBufferAllocationStatement: targetBufferAllocationStatement, SourceTransformationSequence: sourceTransformationSequence, TargetTransformationSequence: targetTransformationSequence, MoveBufferStatement: moveBufferStatement)

Variable input

(none applicable)

Comment

[key ASCSM-CWE-120-comment] Measure element contributes to Security and Reliability

List of Roles

[key ASCSM-CWE-120-roles-sourceBufferAllocationStatement] SourceBufferAllocationStatement
[key ASCSM-CWE-120-roles-targetBufferAllocationStatement] TargetBufferAllocationStatement
[key ASCSM-CWE-120-roles-sourceTransformationSequence] SourceTransformationSequence
[key ASCSM-CWE-120-roles-targetTransformationSequence] TargetTransformationSequence
[key ASCSM-CWE-120-roles-moveBufferStatement] MoveBufferStatement

7.9 Pattern definition of ASCSM-CWE-129: Array Index Improper Input Neutralization

Pattern Category

[key ASCSM-CWE-129-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-129-objective]
Avoid failure to check range of user input in use as array index

Consequence

[key ASCSM-CWE-129-consequence]
Software that is unaware of array index bounds incurs the risk of corruption of relevant memory, and perhaps instructions, possibly leading to a crash, the risk of data integrity loss, and the risk of unauthorized access to sensitive data

Measure Element

[key ASCSM-CWE-129-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the read or write action to access the array; none of the callable or method control element of the transformation sequence being a range check callable and method control element with regards to the array index.

Description

[key ASCSM-CWE-129-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <ArrayAccessStatement> read or write action (action:Reads or action:Writes) to access the <Array> array (code:StorableUnit or code:MemberUnit with code:DataType code:ArrayType); none of the callable or method control element of the transformation sequence being a range check callable and method control element with regards to the array index (code:IndexUnit).

Descriptor

[key ASCSM-CWE-129-descriptor]
ASCSM-CWE-129(UserInput: userInput,ArrayAccessStatement: arrayAccessStatement, Array: array, TransformationSequence: transformationSequence)

Variable input
(none applicable)

Comment
(none applicable)

List of Roles
[key ASCSM-CWE-129-roles-userInput] UserInput
[key ASCSM-CWE-129-roles-arrayAccessStatement] ArrayAccessStatement
[key ASCSM-CWE-129-roles-array] Array
[key ASCSM-CWE-129-roles-transformationSequence] TransformationSequence

7.10 Pattern definition of ASCSM-CWE-134: Format String Improper Input Neutralization

Pattern Category
[key ASCSM-CWE-134-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective
[key ASCSM-CWE-134-objective]
Avoid failure to sanitize user input in use in formatting operations

Consequence
[key ASCSM-CWE-134-consequence]
Software that is unaware of formatting control incurs the risk of execution of arbitrary code and the risk of information disclosure which can severely simplify exploitation of the software

Measure Element
[key ASCSM-CWE-134-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the formating statement; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.

Description
[key ASCSM-CWE-134-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with

DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <FormatStatement> formating statement; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the <StringFormatSanitizationControlElementList> list of vetted sanitization control elements.

Descriptor

[key ASCSM-CWE-134-descriptor]
ASCSM-CWE-134(UserInput: userInput,FormatStatement: formatStatement, TransformationSequence: transformationSequence, StringFormatSanitizationControlElementList: stringFormatSanitizationControlElementList)

Variable input

[key ASCSM-CWE-134-input]
<StringFormatSanitizationControlElementList> list of control elements vetted to handle format string vulnerabilities

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-134-roles-userInput] UserInput
[key ASCSM-CWE-134-roles-formatStatement] FormatStatement
[key ASCSM-CWE-134-roles-transformationSequence] TransformationSequence
[key ASCSM-CWE-134-roles-stringFormatSanitizationControlElementList] StringFormatSanitizationControlElementList

7.11 Pattern definition of ASCSM-CWE-252-resource: Unchecked Return Parameter Value of named Callable and Method Control Element with Read, Write, and Manage Access to Platform Resource

Pattern Category

[key ASCSM-CWE-252-resource-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-252-resource-objective]
Avoid improper processing of the execution status of resource handling operations

Consequence

[key ASCSM-CWE-252-resource-consequence]

Software unaware of execution status control incurs the risk of bad data being used in operations, possibly leading to a crash or other unintended behaviors

Measure Element

[key ASCSM-CWE-252-resource-measure-element]
Number of instances where the named callable control element or method control element executes a 'Read', 'Write', or 'Manage Access' action, yet the value of the return parameter from the action is not used by any check control element

Description

[key ASCSM-CWE-252-resource-description]
This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) executes the <ResourceAccessStatement> Read, Write, and Manage Access action (platform:ReadsResource, platform:WritesResource, and platform:ManagesResource) yet the value (code:Value) of the return parameter (code:ParameterUnit of code:ParameterKind 'return') from the action is not used by any check control element (code:ControlElement containing action:ActionElement with a kind from micro KDM list of comparison actions).

Descriptor

[key ASCSM-CWE-252-resource-descriptor]
ASCSM-CWE-252-resource(ControlElement: controlElement,ResourceAccessStatement: resourceAccessStatement)

Variable input

(none applicable)

Comment

[key ASCSM-CWE-252-resource-comment] Measure element contributes to Security and Reliability

List of Roles

[key ASCSM-CWE-252-resource-roles-controlElement] ControlElement
[key ASCSM-CWE-252-resource-roles-resourceAccessStatement] ResourceAccessStatement

7.12 Pattern definition of ASCSM-CWE-327: Broken or Risky Cryptographic Algorithm Usage

Pattern Category

[key ASCSM-CWE-327-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-327-objective]

Avoid failure to use vetted cryptographic libraries

Consequence

[key ASCSM-CWE-327-consequence]
Software using broken or risky cryptographic algorithm incurs the risk of sensitive data being compromised

Measure Element

[key ASCSM-CWE-327-measure-element]
Number of instances where the application uses the cryptographic deployed component which is not part of the list of vetted cryptographic deployed components.

Description

[key ASCSM-CWE-327-description]
This pattern identifies situations where the <Application> application uses the <CryptographicDeployedComponentInUse> cryptographic deployed component (platform:DeployedComponent) while it is not part of the <VettedCryptographicDeployedComponentList> list of vetted cryptographic deployed components. As an example, FIPS 140-2 features a list of validated implementations.

Descriptor

[key ASCSM-CWE-327-descriptor]
ASCSM-CWE-327(CryptographicDeployedComponentInUse: cryptographicDeployedComponentInUse,VettedCryptographicDeployedComponentList: vettedCryptographicDeployedComponentList, Application: application)

Variable input

[key ASCSM-CWE-327-input]
<VettedCryptographicDeployedComponentList> list of vetted cryptographic deployed components

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-327-roles-cryptographicDeployedComponentInUse] CryptographicDeployedComponentInUse
[key ASCSM-CWE-327-roles-vettedCryptographicDeployedComponentList] VettedCryptographicDeployedComponentList
[key ASCSM-CWE-327-roles-application] Application

7.13 Pattern definition of ASCSM-CWE-396: Declaration of Catch for Generic Exception

Pattern Category

[key ASCSM-CWE-396-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-396-objective]
Avoid failure to use dedicated exception types

Consequence

[key ASCSM-CWE-396-consequence]
Software unaware of accurate execution status control incurs the risk of bad data being used in operations, possibly leading to a crash or other unintended behaviors

Measure Element

[key ASCSM-CWE-396-measure-element]
Number of instances where the named callable control element or method control element contains a catch unit which declares to catch an exception parameter whose data type is part of a list of overly broad exception data types

Description

[key ASCSM-CWE-396-description]
This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) contains the <CatchElement> catch unit (action:CatchUnit) which declares to catch the <CaughtExceptionParameter> exception parameter (code:ParameterUnit with code:ParameterKind 'exception') whose datatype (code:DataType) is part of the <OverlyBroadExceptionTypeList> list of overly broad exception datatypes.
As an example, with JAVA, <OverlyBroadExceptionTypeList> is {'java.lang.Exception'}.

Descriptor

[key ASCSM-CWE-396-descriptor]
ASCSM-CWE-396(ControlElement: controlElement,CatchElement: catchElement, CaughtExceptionParameter: caughtExceptionParameter, OverlyBroadExceptionTypeList: overlyBroadExceptionTypeList)

Variable input

[key ASCSM-CWE-396-input]
<OverlyBroadExceptionTypeList> list of overly broad exception datatypes

Comment

[key ASCSM-CWE-396-comment] Measure element contributes to Security and Reliability

List of Roles

[key ASCSM-CWE-396-roles-controlElement] ControlElement
[key ASCSM-CWE-396-roles-catchElement] CatchElement
[key ASCSM-CWE-396-roles-caughtExceptionParameter] CaughtExceptionParameter
[key ASCSM-CWE-396-roles-overlyBroadExceptionTypeList] OverlyBroadExceptionTypeList

7.14 Pattern definition of ASCSM-CWE-397: Declaration of Throws for Generic Exception

Pattern Category

[key ASCSM-CWE-397-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-397-objective]
Avoid failure to use dedicated exception types

Consequence

[key ASCSM-CWE-397-consequence]
Software unaware of accurate execution status control incurs the risk of bad data being used in operations, possibly leading to a crash or other unintended behaviors

Measure Element

[key ASCSM-CWE-397-measure-element]
Number of instances where the named callable control element or method control element throws an exception parameter whose data type is part of a list of overly broad exception data types

Description

[key ASCSM-CWE-397-description]
This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) throws with the <ThrowsAction> Throws action (action:Throws) the <ThrownExceptionParameter> exception parameter (code:ParameterUnit with code:ParameterKind 'exception') whose datatype (code:Datatype) is part of the <OverlyBroadExceptionTypeList> list of overly broad exception datatypes.
As an example, with JAVA, <OverlyBroadExceptionTypeList> is {'java.lang.Exception'}.

Descriptor

[key ASCSM-CWE-397-descriptor]
ASCSM-CWE-397(ControlElement: controlElement,ThrowsAction: throwsAction, ThrownExceptionParameter: thrownExceptionParameter, OverlyBroadExceptionTypeList: overlyBroadExceptionTypeList)

Variable input

[key ASCSM-CWE-397-input]
<OverlyBroadExceptionTypeList> list of overly broad exception datatypes

Comment

[key ASCSM-CWE-397-comment] Measure element contributes to Security and Reliability

List of Roles

[key ASCSM-CWE-397-roles-controlElement] ControlElement
[key ASCSM-CWE-397-roles-throwsAction] ThrowsAction
[key ASCSM-CWE-397-roles-thrownExceptionParameter] ThrownExceptionParameter
[key ASCSM-CWE-397-roles-overlyBroadExceptionTypeList] OverlyBroadExceptionTypeList

7.15 Pattern definition of ASCSM-CWE-434: File Upload Improper Input Neutralization

Pattern Category

[key ASCSM-CWE-434-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-434-objective]
Avoid failure to sanitize user input in use in file upload operations

Consequence

[key ASCSM-CWE-434-consequence]
Software unaware of file upload control incurs the risk of arbitrary code execution

Measure Element

[key ASCSM-CWE-434-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the file file upload action; none of the callable or method control element of the transformation sequence being a vetted sanitization control element from the list of vetted sanitization control elements.

Description

[key ASCSM-CWE-434-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with

DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <FileUploadStatement> file upload action (platform:ManagesResources with platform:FileResource); none of the callable or method control element of the transformation sequence being a vetted sanitization callable and method control element from the <FileUploadSanitizationControlElementList> list of vetted sanitization callable and method control elements.

Descriptor

[key ASCSM-CWE-434-descriptor]
ASCSM-CWE-434(UserInput: userInput,TransformationSequence: transformationSequence, FileUploadStatement: fileUploadStatement, FileUploadSanitizationControlElementList: fileUploadSanitizationControlElementList)

Variable input

[key ASCSM-CWE-434-input]
<FileUploadSanitizationControlElementList> list of control elements vetted to handle File Upload vulnerabilities

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-434-roles-userInput] UserInput
[key ASCSM-CWE-434-roles-transformationSequence] TransformationSequence
[key ASCSM-CWE-434-roles-fileUploadStatement] FileUploadStatement
[key ASCSM-CWE-434-roles-fileUploadSanitizationControlElementList] FileUploadSanitizationControlElementList

7.16 Pattern definition of ASCSM-CWE-456: Storable and Member Data Element Missing Initialization

Pattern Category

[key ASCSM-CWE-456-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-456-objective]
Avoid failure to explicitly initialize software data elements in use

Consequence

[key ASCSM-CWE-456-consequence]

Software featuring weak initialization practices incurs the risk of logic errors within the program, possibly leading to a security problem

Measure Element

[key ASCSM-CWE-456-measure-element]
Number of instances where a storable data element or member data element is declared by the 'Create' action, then is evaluated in a 'Read' action without ever being initialized by a 'Write' action prior to the evaluation

Description

[key ASCSM-CWE-456-description]
This pattern identifies situations where the <DataElement> storable data element (code:StorableUnit) or member data element (code:MemberUnit) is declared by the <DeclarationStatement> Create action (action:Creates), then evaluated in the <EvaluationStatement> Read action (action:Reads) without ever being initialized by a Write action (action:Writes) prior to the evaluation.

Descriptor

[key ASCSM-CWE-456-descriptor]
ASCSM-CWE-456(DataElement: dataElement,DeclarationStatement: declarationStatement, EvaluationStatement: evaluationStatement)

Variable input

(none applicable)

Comment

[key ASCSM-CWE-456-comment] Measure element contributes to Security and Reliability

List of Roles

[key ASCSM-CWE-456-roles-dataElement] DataElement
[key ASCSM-CWE-456-roles-declarationStatement] DeclarationStatement
[key ASCSM-CWE-456-roles-evaluationStatement] EvaluationStatement

7.17 Pattern definition of ASCSM-CWE-606: Unchecked Input for Loop Condition

Pattern Category

[key ASCSM-CWE-606-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-606-objective]
Avoid failure to check range of user input in use in iteration control

Consequence

[key ASCSM-CWE-606-consequence]
Software unaware of iteration control incurs the risk of unexpected consumption of resources, such as CPU cycles or memory, possibly leading to a crash or program exit due to exhaustion of resources

Measure Element

[key ASCSM-CWE-606-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the loop condition statement; none of the callable or method control element of the transformation sequence being a range check control element.

Description

[key ASCSM-CWE-606-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <LoopConditionStatement> loop condition statement (action:GuardedFlow with an action:TrueFlow returning to the same action:GuardedFlow); none of the callable or method control element of the transformation sequence being a range check control element (code:ControlElement containing action:ActionElement with a kind from micro KDM list of comparison actions).

Descriptor

[key ASCSM-CWE-606-descriptor]
ASCSM-CWE-606(UserInput: userInput,LoopConditionStatement: loopConditionStatement, TransformationSequence: transformationSequence)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-606-roles-userInput] UserInput
[key ASCSM-CWE-606-roles-loopConditionStatement] LoopConditionStatement
[key ASCSM-CWE-606-roles-transformationSequence] TransformationSequence

7.18 Pattern definition of ASCSM-CWE-667: Shared Resource Improper Locking

Pattern Category

[key ASCSM-CWE-667-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-667-objective]
Avoid data corruption during concurrent access

Consequence

[key ASCSM-CWE-667-consequence]
Software featuring inconsistent locking discipline incurs the risk of deadlock

Measure Element

[key ASCSM-CWE-667-measure-element]
Number of instances where the shared storable data element or member data element, declared with the Create action, is accessed outside a critical section of the application via the Read or Write action.

Description

[key ASCSM-CWE-667-description]
This pattern identifies situations where the <PublicDataElement> shared (code:ExportKind 'public') storable data element (code:StorableUnit) or member data element (code:MemberUnit), declared with the <DataElementDeclarationStatement> Create action (action:Creates), is accessed outside a critical section (action:BlockUnit) of the application via the <DataElementAcessStatement> Read or Write action (action:Reads or action:Writes).
The critical nature of the section is technology and platform dependent. As examples, in C/C++, critical nature comes from the use of 'mtx_lock' and 'mtx_unlock' from the 'threads.h' standard C language API (code:LanguageUnit), or from the use of 'pthread_mutex_lock' and 'pthread_mutex_unlock' from the 'pthreads.h' C/C++ POSIX API, or from the use of 'EnterCriticalSection' and 'LeaveCriticalSection' from the 'windows.h' C/C++ Win32 API. As other examples, in JAVA, critical nature comes from the use of the 'syncrhonized' keyword, and in C#, critical nature comes from the use of the 'lock' keyword.

Descriptor

[key ASCSM-CWE-667-descriptor]
ASCSM-CWE-667(PublicDataElement: publicDataElement,DataElementDeclarationStatement: dataElementDeclarationStatement, DataElementAcessStatement: dataElementAcessStatement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-667-roles-publicDataElement] PublicDataElement
[key ASCSM-CWE-667-roles-dataElementDeclarationStatement] DataElementDeclarationStatement
[key ASCSM-CWE-667-roles-dataElementAcessStatement] DataElementAcessStatement

7.19 Pattern definition of ASCSM-CWE-672: Expired or Released Resource Usage

Pattern Category

[key ASCSM-CWE-672-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-672-objective]
Avoid access to a released, revoked, or expired resource

Consequence

[key ASCSM-CWE-672-consequence]
Software unaware of resource lifecycle incurs the risk of unauthorized access to sensitive data that is associated with a different user or entity, and the risk of erroneous later attempts to access the resource, possibly leading to a crash

Measure Element

[key ASCSM-CWE-672-measure-element]
Number of instances where the platform resource is deallocated in the Manage action using its unique resource handler value which is transported throughout the application via the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, then used later within the application to try and access the resource in the Read or Write action.

Description

[key ASCSM-CWE-672-description]
This pattern identifies situations where the <PlatformResource> platform resource (platform:ResourceType) is deallocated in the <ResourceReleaseStatement> manages action (platform:ManagesResource) using its unique resource handler value which is transported throughout the application via the <TransportSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), then used later within the application to try and access the resource in the <ResourceAccessStatement> read or write action (platform:ReadsResource or platform:WritesResource).

Descriptor

[key ASCSM-CWE-672-descriptor]
ASCSM-CWE-672(PlatformResource: platformResource,ResourceReleaseStatement: resourceReleaseStatement, TransportSequence: transportSequence, ResourceAccessStatement: resourceAccessStatement)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-672-roles-platformResource] PlatformResource
[key ASCSM-CWE-672-roles-resourceReleaseStatement] ResourceReleaseStatement
[key ASCSM-CWE-672-roles-transportSequence] TransportSequence
[key ASCSM-CWE-672-roles-resourceAccessStatement] ResourceAccessStatement

7.20 Pattern definition of ASCSM-CWE-681: Numeric Types Incorrect Conversion

Pattern Category

[key ASCSM-CWE-681-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-681-objective]
Avoid numerical data corruption during incompatible mutation

Consequence

[key ASCSM-CWE-681-consequence]
Software featuring weak numerical conversion practices incurs the risk of using the wrong number and generating incorrect results, possibly introducing new vulnerability when related to resource allocation and security decision

Measure Element

[key ASCSM-CWE-681-measure-element]
Number of instances where a storable element or member element is declared with a numerical data type in the 'Create' action, and then is updated with a value which is cast via a type cast action into a second numerical data type, which is incompatible with the first data type

Description

[key ASCSM-CWE-681-description]
This pattern identifies situations where the <DataElement> storable element (code:StorableElement) or member element (code:MemberUnit) is declared with the <NumericalDataType> numerical datatype (code:IntegerType, code:DecimalType, or coce:FloatType) in the <DataElementDeclarationStatement> Create action (action:Creates), then updated with a value which is cast via the <TypeCastExpression> type cast action (action:ActionElement with micro KDM kind 'TypeCast' or 'DynCast') into the <TargetDataType> second numerical datatype, which is incompatible with the first one.

Descriptor

[key ASCSM-CWE-681-descriptor]
ASCSM-CWE-681(DataElement: dataElement,DataElementDeclarationStatement: dataElementDeclarationStatement, NumericalDataType: numericalDataType, TypeCastExpression: typeCastExpression, TargetDataType: targetDataType)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-681-roles-dataElement] DataElement
[key ASCSM-CWE-681-roles-dataElementDeclarationStatement] DataElementDeclarationStatement
[key ASCSM-CWE-681-roles-numericalDataType] NumericalDataType
[key ASCSM-CWE-681-roles-typeCastExpression] TypeCastExpression
[key ASCSM-CWE-681-roles-targetDataType] TargetDataType

7.21 Pattern definition of ASCSM-CWE-772: Missing Release of Resource after Effective Lifetime

Pattern Category

[key ASCSM-CWE-772-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-772-objective]
Avoid resource hoarding and consequently resource depletion

Consequence

[key ASCSM-CWE-772-consequence]

Software unaware of resource lifecycle incurs the risk of preventing all other processes from accessing the same type of resource

Measure Element

[key ASCSM-CWE-772-measure-element]
Number of instances where a platform resource is allocated and assigned a unique resource handler value via a manage resource action, and its unique resource handler value is used throughout the application along a transformation sequence composed of action elements with data relations, some of which are part of named callable and method control elements, but none of which is a resource release statement

Description

[key ASCSM-CWE-772-description]
This pattern identifies situations where the <PlatformResource> platform resource (platform:ResourceType) is allocated and assigned a unique resource handler value via the <ResourceAllocationStatement> ManagesResource action (platform:ManagesResources), its unique resource handler value is used throughout the application, along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), none of which being a resource release statement (platform:ManagesResource).

Descriptor

[key ASCSM-CWE-772-descriptor]
ASCSM-CWE-772(PlatformResource: platformResource,ResourceAllocationStatement: resourceAllocationStatement, TransformationSequence: transformationSequence)

Variable input

(none applicable)

Comment

[key ASCSM-CWE-772-comment] Measure element contributes to Security and Reliability

List of Roles

[key ASCSM-CWE-772-roles-platformResource] PlatformResource
[key ASCSM-CWE-772-roles-resourceAllocationStatement] ResourceAllocationStatement
[key ASCSM-CWE-772-roles-transformationSequence] TransformationSequence

7.22 Pattern definition of ASCSM-CWE-789: Uncontrolled Memory Allocation

Pattern Category

[key ASCSM-CWE-789-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-789-objective]
Avoid failure to check range of user input in use as buffer index

Consequence

[key ASCSM-CWE-789-consequence]
Software that is unaware of buffer bounds incurs the risk of corruption of relevant memory, and perhaps instructions, possibly leading to a crash, the risk of data integrity loss, and the risk of unauthorized access to sensitive data

Measure Element

[key ASCSM-CWE-789-measure-element]
Number of instances where an external value is entered into the application through the user interface ReadsUI action, transformed throughout the application along the sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the buffer Read or Write access action; none of the callable or method control element of the transformation sequence being a range check control element.

Description

[key ASCSM-CWE-789-description]
This pattern identifies situations where an external value is entered into the application through the <UserInput> user interface ReadsUI action (ui:ReadsUI), transformed throughout the application along the <TransformationSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored'), and ultimately used as an index element (code:IndexUnit) to access a storable or member data element (code:StorableUnit or code:MemberUnit) in the <BufferAccessStatement> buffer Read or Write access action (action:Reads, action:Writes, action:Addresses); none of the callable or method control element of the transformation sequence being a range check with regards to the 'Buffer' buffer that whose maximum size was defined in the <BufferAllocationStatement> buffer creation action (action:Creates).

Descriptor

[key ASCSM-CWE-789-descriptor]
ASCSM-CWE-789(UserInput: userInput,BufferAccessStatement: bufferAccessStatement, TransformationSequence: transformationSequence, BufferAllocationStatement: bufferAllocationStatement)

Variable input

(none applicable)

Comment
(none applicable)

List of Roles
[key ASCSM-CWE-789-roles-userInput] UserInput
[key ASCSM-CWE-789-roles-bufferAccessStatement] BufferAccessStatement
[key ASCSM-CWE-789-roles-transformationSequence] TransformationSequence
[key ASCSM-CWE-789-roles-bufferAllocationStatement] BufferAllocationStatement

7.23 Pattern definition of ASCSM-CWE-798: Hard-Coded Credentials Usage for Remote Authentication

Pattern Category
[key ASCSM-CWE-798-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective
[key ASCSM-CWE-798-objective]
Avoid the existence of hard-coded credentials elements

Consequence
[key ASCSM-CWE-798-consequence]
Software featuring weak authentication practices incurs the risk of exposing resources and functionality to unintended actors, possibly leading to compromised sensitive information and even the execution of arbitrary code

Measure Element
[key ASCSM-CWE-798-measure-element]
Number of instances where a storable data element or member data element is initialized by a 'Write' action, transported throughout the application along the transport sequence composed of ActionElements with DataRelations relations, some of which being part of named callable and method control elements, and ultimately used in the remote resource management action ; the transport sequence is composed of assignment operations as updates to the value would not be considered as hard-coded (literal) any more.

Description
[key ASCSM-CWE-798-description]
This pattern identifies situations where a literal value (code:Value) is hard-coded in the application via the <InitialisationStatement> Write action (action:Writes), transported throughout the application along the <TransportSequence> sequence (action:BlockUnit) composed of ActionElements with DataRelations relations (action:Reads, action:Writes, action:Addresses), some of which being part of named callable and method control elements (code:MethodUnit or code:CallableUnit with

code:CallableKind 'regular', 'external' or 'stored'), and ultimately used in the <AuthenticationStatement> remote resource management action (platform:ManagesResource with platform:ResourceType); the transport sequence is composed of assignment operations as updates to the value would not be considered as hard-coded (literal) any more.

Descriptor

[key ASCSM-CWE-798-descriptor]
ASCSM-CWE-798(InitialisationStatement: initialisationStatement,AuthenticationStatement: authenticationStatement, TransportSequence: transportSequence)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-798-roles-initialisationStatement] InitialisationStatement
[key ASCSM-CWE-798-roles-authenticationStatement] AuthenticationStatement
[key ASCSM-CWE-798-roles-transportSequence] TransportSequence

7.24 Pattern definition of ASCSM-CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop')

Pattern Category

[key ASCSM-CWE-835-relatedPatts-security] ASCSM_Security

Pattern Sections

Objective

[key ASCSM-CWE-835-objective]
Avoid infinite iterations

Consequence

[key ASCSM-CWE-835-consequence]
Software unaware of iteration control incurs the risk of unexpected consumption of resources, such as CPU cycles or memory, possibly leading to a crash or program exit due to exhaustion of resources

Measure Element

[key ASCSM-CWE-835-measure-element]
Number of instances where the named callable control element or method control element features the executioon path whose entry element is found again in the path, while it has no path whatsoever to not return to itself and exit the recursion

Description

[key ASCSM-CWE-835-description]
This pattern identifies situations where the <ControlElement> named callable control element (code:CallableUnit with code:CallableKind 'regular', 'external' or 'stored') or method control element (code:MethodUnit) features the <RecursiveExecutionPath> execution path (action:BlockUnit composed of action:ActionElements with action:CallableRelations to code:ControlElements) whose entry element (action:EntryFlow) is found again in the path, while it has no path whatsoever to not return to itself and exit the recursion.

Descriptor

[key ASCSM-CWE-835-descriptor]
ASCSM-CWE-835(ControlElement: controlElement,RecursiveExecutionPath: recursiveExecutionPath)

Variable input

(none applicable)

Comment

(none applicable)

List of Roles

[key ASCSM-CWE-835-roles-controlElement] ControlElement
[key ASCSM-CWE-835-roles-recursiveExecutionPath] RecursiveExecutionPath

8. Calculation of Security and Functional Density Measures (Normative)

8.1 Calculation of the Base Measure

A count of total violations of quality rules was selected as the best alternative for measurement. Software quality measures have frequently been scored at the component level and then aggregated to develop an overall score for the application. However, scoring at the component level was rejected because many critical violations of security quality rules cannot be isolated to a single component, but rather involve interactions among several components. Therefore, the Automated Source Code Security Measure is computed as the sum of its 22 quality measure elements computed across the entire application.

The calculation of the Automated Source Code Security Measure begins with determining the value of each of the 22 security measure elements. Each security measure element is measured as the total number of violations of its associated quality rule that are detected through automated analysis. Thus the value of each of the 22 security measure elements is represented as CISQ-SecME_i where the range for i runs from 1 to 22.

$$\textbf{CISQ-SecME}_i = \Sigma \text{ (all violations of type CISQ-SecME}_i \text{ detected through automated analysis)}$$

The value of the un-weighted and un-normalized Automated Source Code Security Measure (CISQ-Sec) is the sum of the values of the 22 security measure elements.

$$\textbf{CISQ-Sec} = \sum_{i=1}^{22} \textbf{CISQ-SecME}_i$$

Higher values of CISQ-Sec indicate a larger number of security-related defects in the application.

8.2 Functional Density of Security Violations

In order to better compare security results among different applications, the Automated Source Code Security Measure can be normalized by size to create a density measure. There are several size measures with which the density of security violations can be normalized, such as lines of code and function points. These size measures, if properly standardized, can be used for creating a density measure for use in benchmarking applications. However, the OMG Automated Function Points measure offers an automatable size measure that, as an OMG Supported Specification, is standardized, adapted from the International Function Point User Group's (IFPUG) counting guidelines, and commercially supported. Although other size measures can be legitimately used to evaluate the density of security violations, the following density measure for security violations is derived from OMG supported specifications for Automated Function Points and the Automated Source Code Security Measure. Thus, the functional density of Security violations is a simple division expressed as follows.

$$\textbf{CISQ-Sec-density} = \frac{\textbf{CISQ-Sec}}{\textbf{AFP}}$$

9. Alternative Weighted Measures and Uses (Informative)

9.1 Additional Derived Measures

There are many additional weighting schemes that can be applied to the Automated Source Code Security Measure or to the security measure elements that compose it. Table 3 presents several candidate weighted measures and their potential uses. However, these weighting schemes are not derived from any existing standards and are therefore not normative.

Table 3. Informative Weighting Schemes for Security Measurement

Weighting scheme	Potential uses
Weight each Security measure by its severity	Measuring risk of security problems such as data theft and malicious internal damage
Weight each Security measure element by its effort to fix	Measuring cost of ownership, estimating future corrective maintenance effort and costs
Weight each module or application component by its density of Security violations	Prioritizing modules or application components for corrective maintenance or replacement

10. References (Informative)

Common Weakness Enumeration. http://cwe.mitre.org

Consortium for IT Software Quality (2010). http://www.it-cisq.org

Curtis, B. (1980). Measurement and experimentation in software engineering. *Proceedings of the IEEE*, 68 (9), 1103-1119.

International Organization for Standards. ISO/IEC 25010 Systems and software engineering – System and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models

International Organization for Standards (2012). *ISO/IEC 25023 (in development) Systems and software engineering: Systems and software Quality Requirements and Evaluation (SQuaRE) — Measurement of system and software product quality*.

International Organization for Standards (2012). *ISO/IEC TR 9126-3:2003, Software engineering — Product quality — Part 3: Internal metrics.*

Object Management Group (2014). Automated Function Points. formal 2014-01-03 http://www.omg.org/spec/AFP/, .

Appendix A: CISQ

The purpose of the Consortium for IT Software Quality (CISQ) is to develop specifications for automated measures of software quality characteristics taken on source code. These measures were designed to provide international standards for measuring software structural quality that can be used by IT organizations, IT service providers, and software vendors in contracting, developing, testing, accepting, and deploying IT software applications. Executives from the member companies that joined CISQ prioritized the quality characteristics of Reliability, Security, Performance Efficiency, and Maintainability to be developed as measurement specifications.

CISQ strives to maintain consistency with ISO/IEC standards to the extent possible, and in particular with the ISO/IEC 25000 series that replaces ISO/IEC 9126 and defines quality measures for software systems. In order to maintain consistency with the quality model presented in ISO/IEC 25010, software quality characteristics are defined for the purpose of this specification as attributes that can be measured from the static properties of software, and can be related to the dynamic properties of a computer system as affected by its software. However, the 25000 series, and in particular ISO/IEC 25023 which elaborates quality characteristic measures, does not define these measures at the source code level. Thus, this and other CISQ quality characteristic specifications supplement ISO/IEC 25023 by providing a deeper level of software measurement, one that is rooted in measuring software attributes in the source code.

Companies interested in joining CISQ held executive forums in Frankfurt, Germany; Arlington, VA; and Bangalore, India to set strategy and direction for the consortium. In these forums four quality characteristics were selected as the most important targets for automation—reliability, security, performance efficiency, and maintainability. These attributes cover four of the eight quality characteristics described in ISO/IEC 25010. Figure 1 displays the ISO/IEC 25010 software product quality model with the four software quality characteristics selected for automation by CISQ highlighted in orange. Each software quality characteristic is shown with the sub-characteristics that compose it.