Annotated Example of a CWE Entry in XML Format – Schema version 6.7

Author: Steve Christey Coley (coley@mitre.org)

Schema URL: https://cwe.mitre.org/data/xsd/cwe_schema_v6.7.xsd

Introduction

This document provides an example of how XML is used to represent CWE content. It draws from CWE-89 (SQL injection) in CWE 4.7, although not all content is presented for brevity.

This document ONLY uses a weakness example. There are also elements for Views and Categories, but these structures are generally more simple.

Introduction - General XML format

Loosely, XML documents are formatted as follows. Certain background colors are used to better clarify the kinds of information being used.

<Element Attribute1="ABC" Attribute2="EnumeratedValue">

<SubElement>

<Field1>content</*Field1*>

<Field2>more content</*Field2*>

<Field3>enumerated value</*Field3*>

</*SubElement*>

</*Element*>

- An EnumeratedValue is drawn from an "enumeration" of small set of fixed values. These fixed values are listed in the XSD schema.
- Elements are nested, with a closing </*tag*> that mentions the element's name.

Annotated Example

All collections of weaknesses are presented as a “Catalog” that provides a version and date, with a group of weaknesses, categories, and/or views.

<Weakness_Catalog xmlns="http://cwe.mitre.org/cwe-6" xmlns:xhtml="http://www.w3.org/1999/xhtml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Name="CWE" Version="4.7" Date="2022-04-28" xsi:schemaLocation="http://cwe.mitre.org/cwe-6 http://cwe.mitre.org/data/xsd/cwe_schema_v6.7.xsd">

<Weaknesses>

<!-- all weaknesses must have a unique numeric ID, Name, Abstraction, Structure, and Status -->

<**Weakness** ID="89" Name="Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')"

Abstraction="Base" Structure="Simple" Status="Stable">

<**Description**>The software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.</*Description*>

<**Extended_Description**>

Commented [SC1]: StructuredTextType - contains xhtml

<!-- all xhtml elements have the “xhtml:” prefix -->

<*xhtml:p*>Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.</*xhtml:p*>

<*xhtml:p*>SQL injection has become a common issue with database-driven web sites. The flaw is easily detected, and easily exploited, and as such, any site or software package with even a minimal user base is likely to be subject to an attempted attack of this kind. This flaw depends on the fact that SQL makes no real distinction between the control and data planes.</*xhtml:p*>

</*Extended_Description*>

<!-- Related_Weaknesses is used to refer to other weaknesses that differ only in their level of abstraction. It contains one or more Related_Weakness elements, each of which is used to link to the CWE identifier of the other Weakness. The nature of the relation is captured by the Nature attribute. The optional Ordinal attribute is used to determine if this relationship is the primary ChildOf relationship for this weakness for a given View_ID. This attribute can only have the value "Primary" and should only be included for the primary parent/child relationship. For each unique triple of <Nature, CWE_ID, View_ID>, there should be only one relationship that is given a "Primary" ordinal. The optional Chain_ID attribute specifies the unique ID of a named chain that a CanFollow or CanPrecede relationship pertains to. -->

<Related_Weaknesses>

<Related_Weakness Nature="ChildOf" CWE_ID="943" View_ID="1000" Ordinal="Primary"/>

<Related_Weakness Nature="ChildOf" CWE_ID="74" View_ID="1003" Ordinal="Primary"/>

</Related_Weaknesses>

<!-- Applicable_Platforms specifies the languages, operating systems, architectures, and technologies in which a given weakness could appear. A Technology represents a generally accepted feature of a system and often refers to a high-level functional component within a system. The required Prevalence attribute identifies the regularity with which the weakness is applicable to that platform. When providing an operating system name, an optional Common Platform Enumeration (CPE) identifier can be used to a identify a specific OS. -->

<Applicable_Platforms>

<Language Class="Language-Independent" Prevalence="Undetermined"/>

<Technology Name="Database Server" Prevalence="Undetermined"/>

</Applicable_Platforms>

<Modes_Of_Introduction>

<Introduction>

<Phase>Architecture and Design</Phase>

<Note>This weakness typically appears in data-rich applications that save user inputs in a database.</Note>

</Introduction>

<Introduction>

<Phase>Implementation</Phase>

<Note>REALIZATION: This weakness is caused during implementation of an architectural security tactic.</Note>

</Introduction>

</Modes_Of_Introduction>

<Likelihood_Of_Exploit>High</Likelihood_Of_Exploit>

Commented [SMC2]: Include schema doc strings for this and other elements near the beginning of this doc.

Commented [SMC3]: In XML, a "/>" sequence saves space - it closes out the original element. In this case, it's equivalent to "</Related_Weakness>"

<!-- Common_Consequences is used to specify individual consequences associated with a weakness. The required Scope element identifies the security property that is violated. The optional Impact element describes the technical impact that arises if an adversary succeeds in exploiting this weakness. The optional Likelihood element identifies how likely the specific consequence is expected to be seen relative to the other consequences. For example, there may be high likelihood that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact. The optional Note element provides additional commentary about a consequence. -->

<Common_Consequences>

<Consequence>

<Scope>Confidentiality</*Scope*>

<Impact>Read Application Data</*Impact*>

<Note>Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL injection vulnerabilities.</*Note*>

</*Consequence*>

<Consequence>

<Scope>Access Control</*Scope*>

<Impact>Bypass Protection Mechanism</*Impact*>

<Note>If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.</Note>

</Consequence>

</*Common_Consequences*>

<!-- <Detection_Methods> is used to identify methods that may be employed to detect this weakness, including their strengths and limitations. The required Method element identifies the particular detection method being described. The required Description element is intended to provide some context of how this method can be applied to a specific weakness. The optional Effectiveness element says how effective the detection method may be in detecting the associated weakness. This assumes the use of best-of-breed tools, analysts, and methods. There is limited consideration for financial costs, labor, or time. The optional Effectiveness_Notes element provides additional discussion of the strengths and shortcomings of this detection method.

The optional Detection_Method_ID attribute is used by the internal CWE team to uniquely identify methods that are repeated across any number of individual weaknesses. To help make sure that the details of these common methods stay synchronized, the Detection_Method_ID is used to quickly

Commented [SC4]: The identical content of detection methods with the same ID is not enforced - sometimes there is content that is specific to a particular entry.

identify those Detection_Method elements across CWE that should be identical. The identifier is a string and should match the following format: DM-1.-->

<**Detection_Methods**>

<**Detection_Method** Detection_Method_ID="DM-1">

<**Method**>Automated Static Analysis</*Method*>

<**Description**>

<xhtml:p>This weakness can often be detected using automated static analysis tools. Many modern tools use data flow analysis or constraint-based techniques to minimize the number of false positives.</xhtml:p>

<xhtml:p>Automated static analysis might not be able to recognize when proper input validation is being performed, leading to false positives - i.e., warnings that do not have any security consequences or do not require any code changes.</xhtml:p>

<xhtml:p>Automated static analysis might not be able to detect the usage of custom API functions or third-party libraries that indirectly invoke SQL commands, leading to false negatives - especially if the API/library code is not available for analysis.</xhtml:p>

</*Description*>

<**Effectiveness_Notes**>This is not a perfect solution, since 100% accuracy and coverage are not feasible.</*Effectiveness_Notes*>

</*Detection_Method*>

</*Detection_Methods*>

<!-- Potential_Mitigations are optional but can have 1 or more mitigations (also referred to as "controls" in some contexts). Each Mitigation has an optional Mitigation_ID attribute, required Description, optional Phase (one or more), optional Strategy (one or more), and optional Effectiveness/Effectiveness_Notes. -->

<**Potential_Mitigations**>

<**Mitigation** Mitigation_ID="MIT-4">

<**Phase**>Architecture and Design</*Phase*>

<**Strategy**>Libraries or Frameworks</*Strategy*>

<**Description**>

<xhtml:p>Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.</xhtml:p>

<xhtml:p>For example, consider using persistence layers such as Hibernate or Enterprise Java Beans, which can provide significant protection against SQL injection if used properly.</xhtml:p>

Commented [SMC5]: Enumeration - likely change in the future

Commented [SC6]: StructuredTextType - contains xhtml

</Description>

</Mitigation>

<Mitigation Mitigation_ID="MIT-17">

<Phase>Architecture and Design</Phase>

<Phase>Operation</Phase>

<Strategy>Environment Hardening</Strategy>

<Description>

<xhtml:p>Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.</xhtml:p>

<xhtml:p>Specifically, follow the principle of least privilege when creating user accounts to a SQL database. The database users should only have the minimum privileges necessary to use their account. If the requirements of the system indicate that a user can read and modify their own data, then limit their privileges so they cannot read/write others' data. Use the strictest permissions possible on all database objects, such as execute-only for stored procedures.</xhtml:p>

</Description>

</Mitigation>

<Potential_Mitigations>

<-- <Demonstrative_Examples> contains one or more Demonstrative_Example elements, each of which contains an example illustrating how a weakness may look in actual code. The optional Title_Text element provides a title for the example. The Intro_Text element describes the context and setting in which this code should be viewed, summarizing what the code is attempting to do. The Body_Text and Example_Code elements are a mixture of code and explanatory text about the example. The References element provides additional information
The optional Demonstrative_Example_ID attribute is used by the internal CWE team to uniquely identify examples that are repeated across any number of individual weaknesses. To help make sure that the details of these common examples stay synchronized, the Demonstrative_Example_ID is used to quickly identify those examples across CWE that should be identical. The identifier is a string and should match the following format: DX-1. -->

<Demonstrative_Examples>

<Demonstrative_Example>

Commented [SMC7]: Multiple Phases are supported for Mitigations, since the same mitigation might apply in different phases depending on the product and vendor's own processes

Commented [SMC8]: Strategy is a CWE-defined high-level classification of what specific mitigations are trying to achieve. Fixed enumeration of values.

Commented [SMC9]: [REF-123] style references can be encoded anywhere in free text for most fields. Such REF IDs are defined in <External_Reference> elements in the <External_References> section of a full XML download; within an individual CWE entry, these are in the <References> element (see later in this doc)

<Intro_Text>The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where owner matches the user name of the currently-authenticated user.</Intro_Text>

<Example_Code Nature="bad" Language="C#">

<xhtml:div>...<xhtml:br/>

string userName = ctx.getAuthenticatedUserName();

<xhtml:br/>string query = "SELECT * FROM items WHERE owner = '" + userName + "' AND itemname = '" +

ItemName.Text + "'";

<xhtml:br/>

sda = new SqlDataAdapter(query, conn);

<xhtml:br/>

DataTable dt = new DataTable();<xhtml:br/>

sda.Fill(dt);<xhtml:br/>

...</xhtml:div>

</*Example_Code*>

<Body_Text>The query that this code intends to execute follows:</*Body_Text*>

<Example_Code Nature="informative">

<xhtml:div>SELECT * FROM items WHERE owner = <userName> AND itemname = <itemName>;</xhtml:div>

</*Example_Code*>

<Body_Text>However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string:</*Body_Text*>

<Example_Code Nature="attack">

<xhtml:div>name' OR 'a'='a</xhtml:div>

</*Example_Code*>

<Body_Text>for itemName, then the query becomes the following:</*Body_Text*>

<Example_Code Nature="attack">

<xhtml:div>SELECT * FROM items WHERE owner = 'wiley' AND itemname = 'name' OR 'a'='a';</xhtml:div>

Commented [SMC10]: From a presentation/formatting perspective, Example_Code is probably the most complicated xhtml format that we use. We use a number of xhtml elements, including some "style" information, primarily to help with forcing indentation. We are inconsistent with style information, and I hope to make it more regular in the future so that we can better separate rendering information from "content." David R and Steve CC know the most about this format within CWE.

Commented [SMC11]: Language is an enumeration of a few dozen programming languages

Commented [SMC12]: Various XML entities are used throughout CWE/CAPEC XML. For example, "<" represents the "<" character

Commented [SMC13]: For rendering Example_Code, we have different CSS styles associated with each different "Nature" (attack, bad code, good code, etc.) This CSS is kept external to the XML content itself.

</Example_Code>

<Body_Text>The addition of the:</Body_Text>

<Example_Code Nature="attack">

<xhtml:div>OR 'a'='a</xhtml:div>

</Example_Code>

<Body_Text>condition causes the WHERE clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:</Body_Text>

<Example_Code Nature="attack">

<xhtml:div>SELECT * FROM items;</xhtml:div>

</Example_Code>

<Body_Text>This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.</Body_Text>

</Demonstrative_Example>

</Demonstrative_Examples>

<!-- <Observed_Examples> specifies references to a specific observed instance of a weakness in real-world products. Typically this will be a CVE reference. Each Observed_Example element represents a single example. The optional Reference element should contain the identifier for the example being cited. For example, if a CVE is being cited, it should be of the standard CVE identifier format, such as CVE-2005-1951 or CVE-1999-0046. The required Description element should contain a product-independent description of the example being cited. The description should present an unambiguous correlation between the example being described and the weakness that it is meant to exemplify. It should also be short and easy to understand. The Link element should provide a valid URL where more information regarding this example can be obtained. -->

<Observed_Examples>

<Observed_Example>

<Reference>CVE-2004-0366</Reference>

<Description>chain: SQL injection in library intended

for database authentication allows SQL injection and

authentication bypass.</Description>

<Link>https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-0366</Link>

</Observed_Example>

<Observed_Example>

<Reference>CVE-2008-2790</*Reference*>

<Description>SQL injection through an ID that was supposed to be numeric.</*Description*>

<Link>https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2790</*Link*>

</Observed_Example>

</Observed_Examples>

<!-- The Taxonomy_Mappings element is used to provide a mapping from an entry (Weakness or Category) in CWE to an equivalent entry in a different taxonomy. The required Taxonomy_Name attribute identifies the taxonomy to which the mapping is being made. The Entry_ID and Entry_Name elements identify the ID and name of the entry which is being mapped. The Mapping_Fit element identifies how close the CWE is to the entry in the taxonomy. -->

<Taxonomy_Mappings>

<Taxonomy_Mapping Taxonomy_Name="PLOVER">

<Entry_Name>SQL injection</*Entry_Name*>

</*Taxonomy_Mapping*>

<Taxonomy_Mapping Taxonomy_Name="OWASP Top Ten 2007">

<Entry_ID>A2</*Entry_ID*>

<Entry_Name>Injection Flaws</*Entry_Name*>

<Mapping_Fit>CWE More Specific</*Mapping_Fit*>

</*Taxonomy_Mapping*>

</*Taxonomy_Mappings*>

<!-- The <Related_Attack_Patterns> element contains references to attack patterns associated with this weakness. The association implies those attack patterns may be applicable if an instance of this weakness exists. Each related attack pattern is identified by a CAPEC_ID identifier. -->

<Related_Attack_Patterns>

<Related_Attack_Pattern CAPEC_ID="108"/>

<Related_Attack_Pattern CAPEC_ID="109"/>

</*Related_Attack_Patterns*>

<!-- The <References> element contains one or more Reference elements, each of which is used to link to an external reference defined within the catalog. The required External_Reference_ID attribute

represents the external reference entry being linked to (e.g., REF-1). Text or quotes within the same CWE entity can cite this External_Reference_ID similar to how a footnote is used, and should use the format [REF-1]. The optional Section attribute holds any section title or page number that is specific to this use of the reference. -->

<References>

<Reference External_Reference_ID="REF-44" Section=""Sin 1: SQL Injection." Page 3"/>

<Reference External_Reference_ID="REF-867"/>

<Reference External_Reference_ID="REF-62" Section="Chapter 8, "SQL Queries", Page 431"/>

<Reference External_Reference_ID="REF-62" Section="Chapter 17, "SQL Injection", Page 1061"/>

</References>

<!-- The Content_History element provides elements to keep track of the original author of an entry and any subsequent modifications to the content. The required Submission element is used to identify the submitter and/or their organization, the date, and any optional comments related to an entry. The optional Modification element is used to identify a modifier's name, organization, the date, and any related comments. A new Modification element should exist for each change made to the content. Modifications that change the meaning of the entry, or how it might be interpreted, should be marked with an importance of critical to bring it to the attention of anyone previously dependent on the weakness. The optional Contribution element is used to identify a contributor's name, organization, the date, and any related comments. This element has a single Type attribute, which indicates whether the contribution was part of general feedback given or actual content that was donated. The optional Previous_Entry_Name element is used to describe a previous name that was used for the entry. This should be filled out whenever a substantive name change occurs. The required Date attribute lists the date on which this name change was made. A Previous_Entry_Name element should align with a corresponding Modification element. -->

<Content_History>

<Submission>

<Submission_Name>PLOVER</Submission_Name>

<Submission_Date>2006-07-19</Submission_Date>

</Submission>

<Modification>

<Modification_Name>Eric Dalci</Modification_Name>

<Modification_Organization>Cigital</Modification_Organization>

<Modification_Date>2008-07-01</Modification_Date>

<Modification_Comment>updated Time_of_Introduction</Modification_Comment>

Commented [SMC14]: Note: around CWE 3.0 (or 4.0?) we found so much variability/complexity in representing the variety of references that we used, we tried to simplify the representation - thus overloading the "Section" attribute.

</Modification>

<Modification>

<Modification_Name>CWE Content Team</Modification_Name>

<Modification_Organization>MITRE</Modification_Organization>

<Modification_Date>2021-10-28</Modification_Date>

<Modification_Comment>updated Relationships</Modification_Comment>

</Modification>

<Previous_Entry_Name Date="2008-04-11">SQL Injection</Previous_Entry_Name>

<Previous_Entry_Name Date="2008-09-09">Failure to Sanitize Data into SQL Queries (aka 'SQL Injection')</Previous_Entry_Name>

<Previous_Entry_Name Date="2009-01-12">Failure to Sanitize Data within SQL Queries (aka 'SQL Injection')</Previous_Entry_Name>

<Previous_Entry_Name Date="2009-05-27">Failure to Preserve SQL Query Structure (aka 'SQL Injection')</Previous_Entry_Name>

<Previous_Entry_Name Date="2009-07-27">Failure to Preserve SQL Query Structure ('SQL Injection')</Previous_Entry_Name>

<Previous_Entry_Name Date="2010-06-21">Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')</Previous_Entry_Name>

</Content_History>

</Weakness>

</Weaknesses>

</Weakness_Catalog>

Commented [SMC15]: Many CWEs go through multiple changes to the Name attribute - this is recorded as part of content history but also helps with search. I think that most versions have at least one entry with a Name change.