**The Applicability and Query Abstraction (AQuA) Language Design Document**

September 28, 2017

Contents

[1 Abstract 1](#_Toc7007618)

[2 The Problem 1](#_Toc7007619)

[3 Introduction 2](#_Toc7007620)

[3.1 Design Objectives 2](#_Toc7007621)

[3.2 A Simple Use Case – Security Content Authors 3](#_Toc7007622)

[3.3 How AQuA Works 3](#_Toc7007623)

[4 Security Content Business Case 4](#_Toc7007624)

[4.1 The Security Content Producer 4](#_Toc7007625)

[4.2 The Security Content Consumer 5](#_Toc7007626)

[5 Technical Design Overview 5](#_Toc7007627)

[5.1 Writing a Simple AQuA statement – Revisiting the Security Content Author Use Case 6](#_Toc7007628)

[6 Closing 8](#_Toc7007629)

# Abstract

Traditional query languages are designed to write statements which generate results when run against a given database. These statements are only ever applicable to the targeted database, due to differences in database structure and data formatting. AQuA is a generalized query language intended to be applicable to any database using a high-level, abstract language which is locally mapped to any given database. This approach allows a query author to write a single statement, apply it against many databases, and get back standardized results. AQuA is not intended to replace existing query languages, but rather to serve a very specific set of use cases.

# The Problem

Applicability, in the broadest sense, is the process of determining if a set of conditions are true or false given some set of data. In the computer security context, this often means determining if a given vulnerability is applicable to a given endpoint, set of endpoints, or even to an entire enterprise. Unfortunately, this process requires not only a large amount of data regarding the endpoint in question, but also an expressive and thorough way to express the conditions under which a vulnerability applies.

Imagine a vulnerability that only applies to a certain piece of software, on a certain range of versions, on a certain operating system version, with certain pieces of hardware installed. This quickly becomes a complicated set of inter-connected conditions that must be evaluated.

Today, a combination of specialized queries, standardized formats, and human-driven checks are used to determine applicability; but this approach scales poorly and does not lend itself to automation. As vulnerabilities increase in volume and complexity, the need increases for a standardized, machine-friendly, generalized solution for expressing applicability.

# Introduction

AQuA is a light-weight, domain-independent, read-only query language that can be interpreted in a standardized way to retrieve arbitrary information from a data store. It was developed to provide authors with a simple way to write generic queries against domains of information in a way that is agnostic to the way in which the information is stored. These AQuA statements can be executed against enterprise databases, regardless of its underlying data store technology or schema. This generic approach allows AQuA statements to be written by authors who have no information about how the information they are querying is organized.

This document covers the high-level design of AQuA and provides a semi-technical overview of its implementation.

## Design Objectives

In developing AQuA, there were three core objectives. The first core objective was simplicity.AQuA is designed to support common, basic query expressions using a simple syntax. Many existing query languages include sophisticated analytic capabilities. These capabilities make these languages very powerful tools, but make the query language more complicated, and rely upon the author having a detailed understanding of the underlying data against which the statements are run. Instead, AQuA provides a more minimal language that makes it easy to write basic, common queries.

The second core objective was to be datastore agnostic. AQuA is designed to be agnostic as to how data sources are organized, be they tabular, graph-based, or something else. AQuA does this by employing domain ontologies that identify key concepts for a given domain. These concepts are intended to be universally meaningful within a given domain, and thus broadly understood across many representations. Evaluation of an AQuA statement within a given environment would require creating a mapping between the general concepts in the AQuA statement and the environment’s local representation of the relevant information.

Related to the previous goal, a third core objective was to allow AQuA statements to be widely distributed and consumed by a range of parties. Because AQuA has no dependencies on the structure of the underlying data source against which queries are run, the same statement is usable by many consumers, each of whom might organize the associated data differently. This makes AQuA ideal for use in bulletins, alerts, and other publications that are intended for a broad audience.

## A Simple Use Case – Security Content Authors

Often, producers of computer security content do not know exactly who will consume their content or how consumers will use that content in an operational environment. Therefore, producers need a way to communicate the circumstances under which a particular advisory is applicable (e.g., if a certain product is in use in the enterprise) to consumers using a standardized syntax and semantics that can be interpreted in the context of a target environment. The language used to communicate applicability descriptions should be a simple and efficient to reduce the effort needed to write these statements.

The AQuA Language allows producers of security advisories to write such applicability descriptions. These producers include groups such as: software producers who write advisories on their own products, third parties who write advisories for multiple vendor products, enterprises who wish to enforce their own policies, and inventory system vendors who produce query capabilities on databases they populate. AQuA gives these authors the ability to express certain applicability criteria (e.g., application name, application version, etc.) using a standard language that can be evaluated by enterprises without regard to how the relevant information is collected or stored by those enterprises.

## How AQuA Works

Figure 1 below provides a look at a typical usage of AQuA.



Figure 1. A Potential AQuA Statement Processing Model

There are several components involved in this potential use of AQuA:

* AQuA Statement - An AQuA Statement provides a generic query to be evaluated against an arbitrary data source.
* Enterprise Data Source - AQuA statements ultimately need to be evaluated against data. This can take the form of a database (of any structure) of previously collected information, or it can be dynamically collected based on the query itself.
* AQuA Domain Ontologies - AQuA statements are expressed in a manner that is not dependent on any given data representation. Typically, evaluating an AQuA statement against a data source requires a mapping between the concepts defined in one or more AQuA domain ontologies and the underlying data representation used in the enterprise data source. While it is possible that some party could build their database to perfectly reflect AQuA domain ontologies, in practice mappings between the AQuA language and a data representation will be more complicated. A mapping could take the form of a set of rules, a transform of either the query or the data source, or some other procedure. The mapping for a concept might be relatively straightforward (e.g., the concept is directly captured in some named field in the enterprise data source) or could be more complicated (e.g., various other contextual elements impact where the data appears in the enterprise data source).
* AQuA Language Evaluation Engine - This is the component that take an AQuA statement and evaluates it against one or more enterprise data sources, guided by the mapping information. The evaluation engine is responsible for consistently applying clauses of the query against corresponding data (as guided by the mapping) to determine the truth of the given query clause, and then returning the appropriate result expressed using the AQuA result syntax.

# Security Content Business Case

AQuA originally arose from a need for standardization around applicability in the security content context, and as such, is designed to provide value to both security content producers and consumers.

## The Security Content Producer

Producing applicability statements today is a difficult process, often involving plain text solutions or rapidly aging and specialized standards. As new formats arise for collecting and storing endpoint and enterprise posture information, and the volume of this data increases, these current methods begin the show their weaknesses.

AQuA provides a generalized approach to solving applicability. When new formats and standards arise, only a new Domain Ontology needs to be created and shared, the core AQuA specification, and any tooling built around it, remains stable. Built for automation, AQuA opens new opportunities for tooling and services, while reducing the overhead needed to handle large amounts of data and large numbers of customers.

## The Security Content Consumer

Receiving vulnerability alerts or notifications is often a human driven process, with all of the inefficiencies and overhead that implies. When a new high-criticality vulnerability is discovered, the simple question of “Am I affected?” is often difficult, and expensive, to answer. A standardized solution like AQuA provides the means to automate inquiries into your enterprise’s configuration.

Producers that research vulnerabilities can share AQuA like they share text advisories today, but the CISO, or an automated system, can consume and act on an AQuA statement in minutes, instead of weeks.

# Technical Design Overview

This section will provide a technical overview of the AQuA standard, intended for readers interested in how the specification is designed at a lower level.

An AQuA statement consists of a list of ontology classes to evaluate, a Boolean clause specifying conditions to be met by those classes and their properties, and the classes and properties that are to be returned. Conditions can incorporate comparisons, basic operations, or pre-defined functions. These conditions are used to examine the properties of domain ontology classes. AQuA statement authors use these classes and properties to represent data values, pulled from an enterprise data source, within an AQuA statement.

Evaluation of any query against a given enterprise data source requires that statement authors and statement consumers have synchronized understandings of certain concepts. Unlike most query languages, AQuA does not require a synchronized understanding of how data is structured within the enterprise data source. However, AQuA authors and consumers still need to be in agreement on the following concepts:

* The AQuA statement language syntax. The AQuA language specification defines syntax and semantics for the various operators that can be leveraged by the AQuA authors and consumers.
* The meaning of each class/property defined in domain ontologies utilized in a given AQuA statement.The ontologies (class definitions) define the schema used in the AQuA statement, and the data type of each class property. The ontologies can be developed by either a commercial vendor or through an open standards process. The AQuA statement interpreter used by an enterprise to evaluate the AQuA statement would need to define a mapping between these class definitions and the structure of the enterprise data source.
* The formatting of values associated with domain ontology properties. This is necessary in order to facilitate accurate comparisons between the values mapped into the domain ontology from the enterprise data source and the values within an AQuA statement. For example, an AQuA statement might use a regular expression comparison (“win(ows)?[789]”). To correctly evaluate, an appropriate string that can be correctly evaluated by this regular expression would need to be mapped to the appropriate domain ontology property during evaluation. Currently, the intended structure of domain ontology values is described in English documentation, mostly generated by software providers and to be read by query writers. The language does not specify how this information is associated with an enterprise data source*.*

## Writing a Simple AQuA statement – Revisiting the Security Content Author Use Case

Recall the described use case where security content authors wish to tag their content with applicability statements that can be used across a variety of enterprises. To support this, they create AQuA statements identifying specific software products and versions to which their advisory is applicable. In order to do this, they use an AQuA domain ontology that captures information about software products. A hypothetical excerpt from such a domain ontology appears in Table 1:

|  |  |
| --- | --- |
| [1] | [ ... |
| [2] | {“COMPUTER” : [ |
| [3] | ... |
| [4] | { “Name” : “Id”, “Type” : “STRING” }, |
| [5] | ... |
| [6] | ] |
| [7] | }, |
| [8] | {“SOFTWARE” : [ |
| [9] | ... |
| [10] | { “Name” : “installedOn”, “Type” : “COMPUTER” }, |
| [11] | { “Name” : “Name”, “Type” : “STRING” }, |
| [12] | { “Name” : “Version”, “Type” : “STRING” }, |
| [13] | ... |
| [14] | ] |
| [15] | } |
| [16] | } |

Table 1: Hypothetic Excerpt from an AQuA Domain Ontology

The ontology defines two classes, ‘COMPUTER’ and ‘SOFTWARE.’ The class COMPUTER (lines 2-7) has an associated data property, ‘Id’, that has a data type of STRING. (Data types are discussed in more detail in section 5.1.) The class SOFTWARE (lines 8-15) has three associated properties - one class property, ‘installedOn’ that can be filled with an entity that is a COMPUTER, and two data properties, ‘Name’ and ‘Version’ that have a data type of STRING.

The enterprise has collected inventory information from the endpoints in their enterprise and stored this in a database. A hypothetical excerpt from such a data source appears in Table 2.

|  |  |  |
| --- | --- | --- |
| **computerID** | **softwareName** | **softwareVersion** |
| c1 | Firefox | 1.2 |
| c2 | Safari | 3.1 |
| c3 | Firefox | 2.3 |
| c3 | Word | 15.1 |

Table 2: Hypothetical Excerpt from an Enterprise Data Source

The AQuA interpreter for a given enterprise includes a mapping between the classes and properties in the software domain ontology and the enterprise’s software data source. The class names and property names in the domain ontology may differ from the data source field names. This mapping is created by each enterprise for its enterprise data source. Table 3 shows equivalencies between the domain ontology and the enterprise data source:

|  |  |
| --- | --- |
| **AQuA Domain Ontology** | **Enterprise Data Source** |
| COMPUTER.Id | computerID |
| SOFTWARE.Name | softwareName |
| SOFTWARE.Version | softwareVersion |

Table 3: Example Mapping between an AQuA Domain Ontology and an Enterprise Data Source

Note that there is not a direct mapping to the SOFTWARE.installedOn property. This mapping would be made by populating COMPUTER instances from the data source, and then pointing to the correct COMPUTER instances from the appropriate SOFTWARE instances.

The security content author would use the software domain ontology to craft an AQuA statement that describes the conditions that would make their content applicable. An example of such a statement appears below. This statement looks for computers that have Firefox installed and reports each such computer along with the name (always Firefox) and version of that software:

|  |  |
| --- | --- |
| [1] | SELECT (SOFTWARE sw, COMPUTER c) |
| [2] | { |
| [3] | sw.installedOn=c |
| [4] | AND sw.Name=”Firefox” |
| [5] | } |
| [6] | return c.Id, sw.Name, sw.Version |

Table 4: Example AQuA Statement

A brief overview of the important parts of this statement appears below:

* SELECT (line 1) indicates that the statement is to find matching elements in the enterprise data source and returns specific requested information.
* ‘(SOFTWARE sw, COMPUTER c)’ (line 1) ties AQuA domain ontology classes to variables. Here the variable ‘sw’ is defined as an instance of the class SOFTWARE and the variable ‘c’ is defined as an instance of the class COMPUTER. These variables represent instances of the corresponding classes within the body of the AQuA statement.
* The elements specified between the brackets (lines 2-5) define the conditions to be met for a match to occur with elements in the enterprise data source.
  + On line 3, ‘sw.InstalledOn’ examines instances of the SOFTWARE class (represented by the variable ‘sw’). It examines the installedOn property of each of these class instances and determines whether that property value is equal to the instance of the COMPUTER class represented by variable c. Note that the ‘sw.installedOn=c’ clause is effectively performing a join-like operation.
  + On line 4, ‘sw.Name=”Firefox”’ is examining Name property of the instance of the SOFTWARE class. Specifically, it is checking to see if it is equal to the string “Firefox”.
* The ‘return’ portion of the statement identifies the fields in the enterprise data source to return for each combination of instances of the named classes that evaluate to TRUE. The fields to return are the identifier of a computer (‘c.Id’), the name of the software installed on the computer (‘sw.Name’), and the version of the software installed on the computer (‘sw.Version’).

An enterprise data source might (and in most cases probably will) have multiple instances associated with any named class in an AQuA ontology. Each class instance is associated with one instance of the thing the class describes. The result of the mappings between each class declared in an AQuA statement and the enterprise data source is effectively a list of sets, where each set represents all the instances of a named class. Evaluation will take place against every combination of instances among the set of instances. In this example, the enterprise data source identifies 4 distinct pieces of software across 3 different computers. This would result in 4 instances of the SOFTWARE class and 3 instances of the COMPUTER class, resulting in 12 possible combinations in the cross product. The AQuA statement would be evaluated against all 12 of these combinations, and a result entry would be created for each combination for which the statement body evaluated to TRUE.

In our hypothetical data source, there are two combinations of SOFTWARE and COMPUTER instances for which the body would evaluate to TRUE. As a result, the AQuA statement would have the following output:

[

{ “c.Id” : “c1”, “sw.Name” : “Firefox”, “sw.Version” : “1.2” },

{ “c.Id” : “c3”, “sw.Name” : “Firefox”, “sw.Version” : “2.3” }

]

This information could be used by local enterprise administrators to identify specific endpoints to which the security content applies (using their identifiers) as well as any relevant context (in this case, the version of Firefox discovered). While this example is simple and somewhat contrived, it demonstrates the power AQuA can provide in allowing a query author to make a general statement using well defined terms that can then be evaluated across a wide range of environments.

# Closing

This document has provided an overview of the AQuA specification. Further technical details and requirements can be found in the AQuA specification itself. The other documents are as follows:

1. AQuA Processing Requirements
   * Provides core requirements for the execution and handling of AQuA
2. AQuA Statement Language Specification
   * Provides the syntax for the AQuA statement language
3. AQuA Domain Ontology Specification
   * Provides the syntax for the AQuA Domain Ontologies
4. AQuA Results Specification
   * Provides the syntax for AQuA Results