Technical Overview of the Application Vulnerability Description Language (AVDL) V1.0

Version 1.0, 22 March 2004

Document identifier:

AVDL Technical Overview - 01

Location:

http://TBD

Editor:

Jan Bialkowski, NetContinuum, jan@netcontinuum.com
Kevin Heineman, SPI Dynamics, kheineman@spidynamics.com
Srinivas Mantripragada, NetContinuum, srinivas@netcontinuum.com

Abstract:

This specification describes a standard XML format that allows entities (such as applications, organizations, or institutes) to communicate information regarding web application vulnerabilities. . Simply said, Application Vulnerability Description Language (AVDL) is a security interoperability standard for creating a uniform method of describing application security vulnerabilities using XML.

Status:

This is a non-normative document. This document provides a technical description of AVDL 1.0. It has been produced by the AVDL technical committee. This working draft may be updated, replaced, or obsoleted at any time. Please send comments to the editors.

Committee members should send comments on this specification to avdl@lists.oasis-open.org. Others should subscribe to and send comments to avdl-comment@lists.oasis-open.org. To subscribe, send an email message to avdl-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the AVDL Technical Committee (AVDL TC) web page (http://www.oasis-open.org/committees/avdl/ipr.php).

Table of Contents

1 Introduction

The goal of AVDL is to create a uniform format for describing application security vulnerabilities. The OASIS AVDL Technical Committee was formed to create an XML definition for exchanging information about the security vulnerabilities of applications exposed to networks. For example, the owners of an application use an assessment tool to determine if their application is vulnerable to various types of malicious attacks. The assessment tool records and catalogues detected vulnerabilities in an XML file in AVDL format. An application security gateway then uses the AVDL information to recommend the optimal attack prevention policy for the protected application. In addition, a remediation product uses the same AVDL file to suggest the best course of action for correcting the security issues. Finally a reporting tool uses the AVDL file to correlate event logs with areas of known vulnerability.

A detailed description of the specification draft submitted to OASIS is available at: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=avdl.

2 AVDL Overview

Security managers have grown accustomed to relying on traditional tools, such as network firewalls, IDS, and VPNs to protect corporate networks. The exploding number of application-level security incidents, however certifies that these tools provide few tangible benefits in the area of application security. While next generation application security products now solve many of these problems, these best-of-breed stand-alone systems still require individual and separate user interactions, leaving the overall security management process too manual, time-consuming, and error prone.

Proposed by leading application security vendors and users, the AVDL specification creates a rich and effective set of consistent XML schema definitions to describe application security properties and vulnerabilities. Using AVDL, security tools and products from different vendors will be able to precisely and unambiguously communicate with each other to coordinate their security operations and automate security management.

AVDL integration creates a seamless ecosystem that secures the web application environment in which mundane security operations such as patching and reconfigurations that implement evolving application requirements and security policies become automated freeing security administrators to focus on higher-level security policy analysis. Because all new vulnerability alters can be described consistently in AVDL, automation of security management also vastly reduces the incident response time thus closing critical vulnerability windows and enhancing security posture. AVDL-based security altered bulletins will give users highly efficient access to the collective security expertise of all participants in this dynamic field where even the largest organizations are challenged to keep up with rapid industry revolution.

3 AVDL Architecture

The AVDL technology is rooted in XML. The information passed around between the producers and consumers is mostly in the form of XML, and the format of these XML messages is defined in the AVDL schema.

3.1 AVDL Concepts

AVDL has the following key concepts:

- **Probe:** The basic concept embodied in the AVDL schema is an application-level transaction, called a "probe", which describes HTTP exchanges between browsers and web application servers. The probe defines the basic unit of request-response exchange and its relation to the expected result.
- **Transaction:** AVDL defines markups which allow specifications of the transaction between the browser and server as a series of probes. Such probes may specify valid and expected request-response exchanges between browsers and servers, or may specify application vulnerability exploits. AVDL 1.0 allows specification of the HTTP transaction in full detail at various levels of abstraction (raw byte stream, or parsed to HTTP header constructs).
- **Traversal:** The "traversal" step is a derived extension of a "probe" where the request-response exchange between browser and server is expected (and valid). The traversal step probes supply host of information including target URLs, links, cookies and other headers as well as query for form parameters, their attributes, ranges of legitimate values etc. The traversal probes can be used to automate enforcement of safe usage policies.
- **Vulnerability-probe:** The "vulnerability-probe" is a derived extension of a "probe" where the request-response exchange between browser and server is of illegitimate kind. The probe contains undesirable (or malicious) elements with a primary intent to cause damage to the application.
- **Vulnerability-description:** The "vulnerability-description" highlights specific questionable constructs and supply detailed specifications of vulnerabilities, including human readable description and machine-readable assessment information such as vulnerability severity, applicability, and its historical records. The vulnerability-description supplies enough information necessary to configure protective "deny" rules as well as information about possible hot fixes if any is available, workarounds etc. that can be used to automate management of remediation process.

3.2 AVDL Structure and Examples

The AVDL output is divided into two major sections. The first is the Traversal. This output reflects the basic structure of the web site. It describes the requests and responses that were made to the server and the pages that were displayed as a result of the requests. A Traversal is a single transaction containing one or more request/response exchanges, each exchange is enclosed in a separate Traversal Container. The Traversal Container provides a complete topological ordering of the URLs visited in a web site.

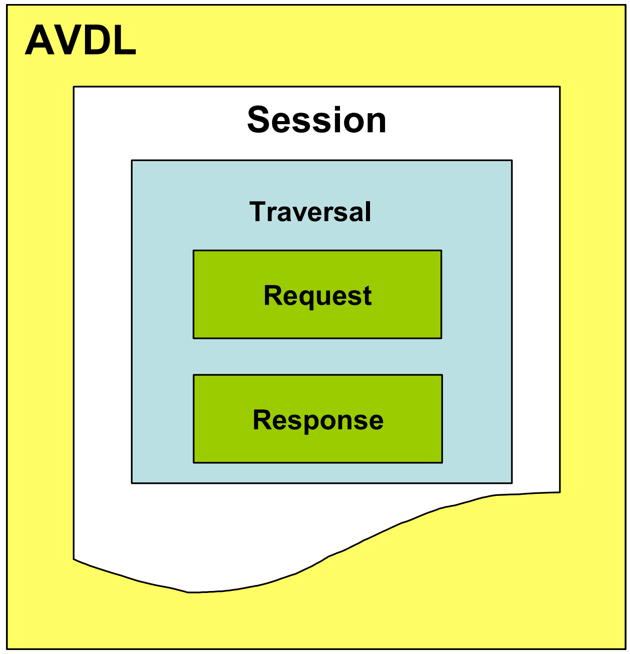


Figure 1: AVDL Traversal Structure

Figure 1 shows the layout of the Traversal structure. The root node <avdl> contains a session header. The session header embodies a user-level transaction activity, e.g. scan for vulnerabilities, a web site crawl etc. The session header contains the ID of the session, the target URI that was crawled and when the activity was started. The session contains a series of traversal headers. The traversal header corresponds to one probe activity describing the request and response details. The traversal header contains a sequence number (a number designating this traversal in the ordered sequence of traversals visited during the crawl). The request and response headers contain detailed information on the HTTP byte stream (both in raw and parsed form) containing detailed information about headers, cookies, URLs, query inputs, POST data, HREF links etc.

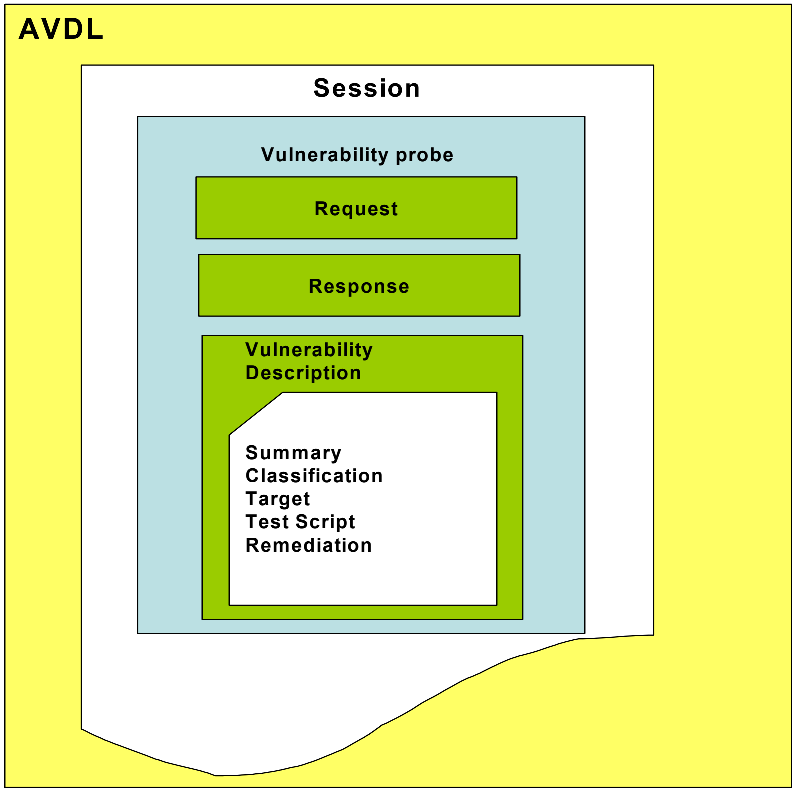


Figure 2: AVDL Vulnerability Probe Structure

Figure 2 shows the layout of the Vulnerability Probe structure. The root node <avdl> contains a session header. The session header contains a vulnerability probe header. While the traversal header maps the web application and describes the requests and responses for each page of a Web application, the Vulnerability Probe header describes the vulnerabilities contained within the Web application. The session structure can contain many vulnerability probes each describing a single vulnerability of the Web application. It is important to note that not all vulnerability probes lead to identifying application vulnerability. Each vulnerability probe header contains the request/response details and a description of the vulnerability if found. The vulnerability description header contains the following 5 items.

- **Summary --** Provides a brief description of the vulnerability
- **Classification --** Provides a logical grouping of the vulnerability, e.g. SQL Injection, Cross-Site Scripting
- **Target --** Provides details on the target, e.g. Host, OS, Architecture, Protocol, Web Server
- **Test Script --** Provides details on how to reproduce the application vulnerability
- **Remediation –** Provides specific recommended actions to close the vulnerability

A detailed description of the above items is discussed in the AVDL draft standard.

3.3 AVDL Design Philosophy

The AVDL schema design is heavily derived from object-oriented (OO) programming concepts. An attempt is made to devise base containers which abstract common properties. The child nodes are derived from the parent base containers to provide effective sharing of common properties. At the root <avdl> node two types of blocks are created, **<root-block>** and **<session-block>**.

The **<root-block>** provides a base to abstract common base properties. The properties include:

- **Summary –** Provides a brief description of the vulnerability
- **Description –** Provides a detailed explanation of the vulnerability
- **Classification –** Provides a logical grouping of the vulnerability, e.g. SQL Injection, Cross-Site Scripting
- **Datum –** A generic tuple entity <name, type, value> of type <xs:QName, validated data type, xs:token>
- **History –** Provides a reference to earlier version(s) of this block, or documents it is based on

The above fields are conditional elements and don't need to be always present. The child nodes of the <root-block> include <session>, <solution> and <vulnerability-description>.

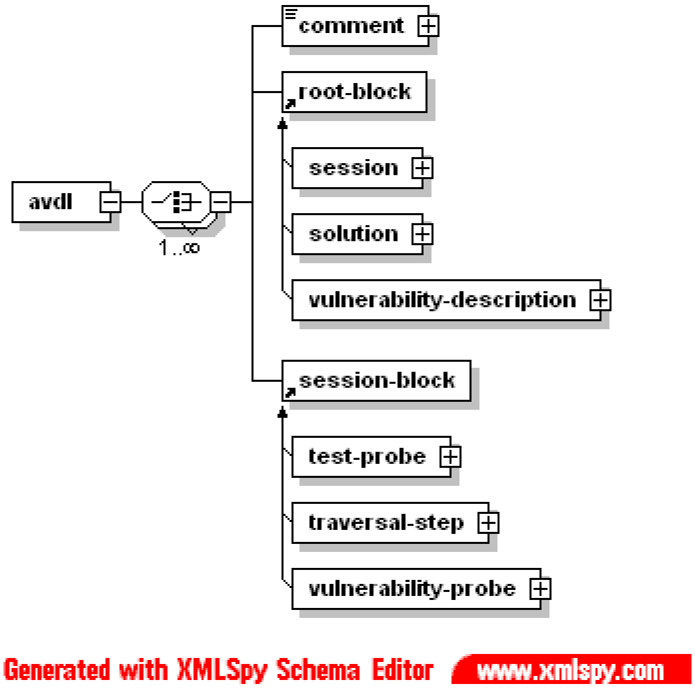


Figure 3: AVDL Top Level Structure

The **<session-block>** in addition to containing the above <root-block> properties also contains information about:

- **Target --** Provides details on the target, e.g. Host, OS, Architecture, Protocol, Web Server
- **Test Script --** Provides details on how to reproduce the application vulnerability
- **Solution --** Provides specific recommended actions to close the vulnerability

The child nodes of the <session-block> include <test-probe>, <traversal-step> and <vulnerability-probe>.

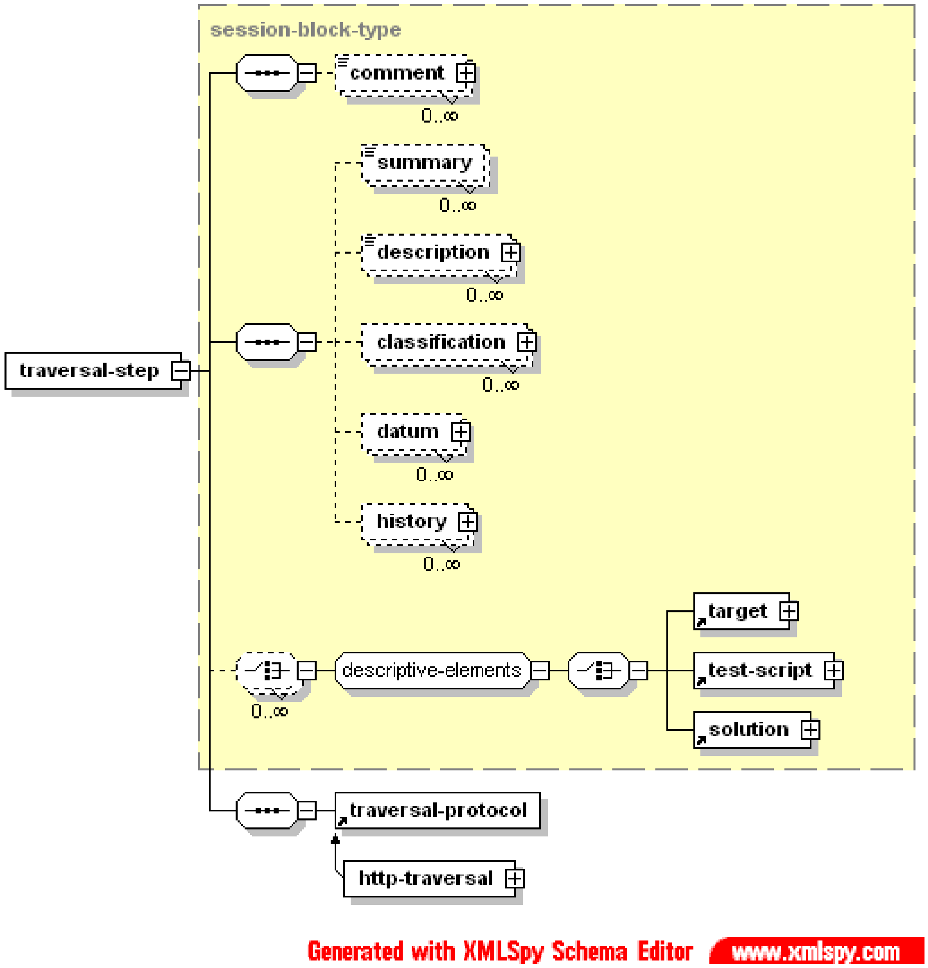


Figure 4: AVDL Traversal Step structure

Figure 4 provides the diagram of the **traversal-step** structure. The traversal-step is the base-class for all stepwise traversals (such as an enumeration of accessible URL's at a given site). The traversal-step results from a probe activity. For example, each URL in a web-spiders crawl would show up as one traversal-step. A single traversal-step can sometimes constitute of multiple internal steps often spanning multiple applications using multiple protocols. The essence of the field item **traversal-protocol** is essentially to capture this effect. It contains a single field item **http-traversal** since the AVDL 1.0 draft addresses only HTTP protocol but allows for extensions in the future.The http-traversal structure details the basic HTTP message type in detail.

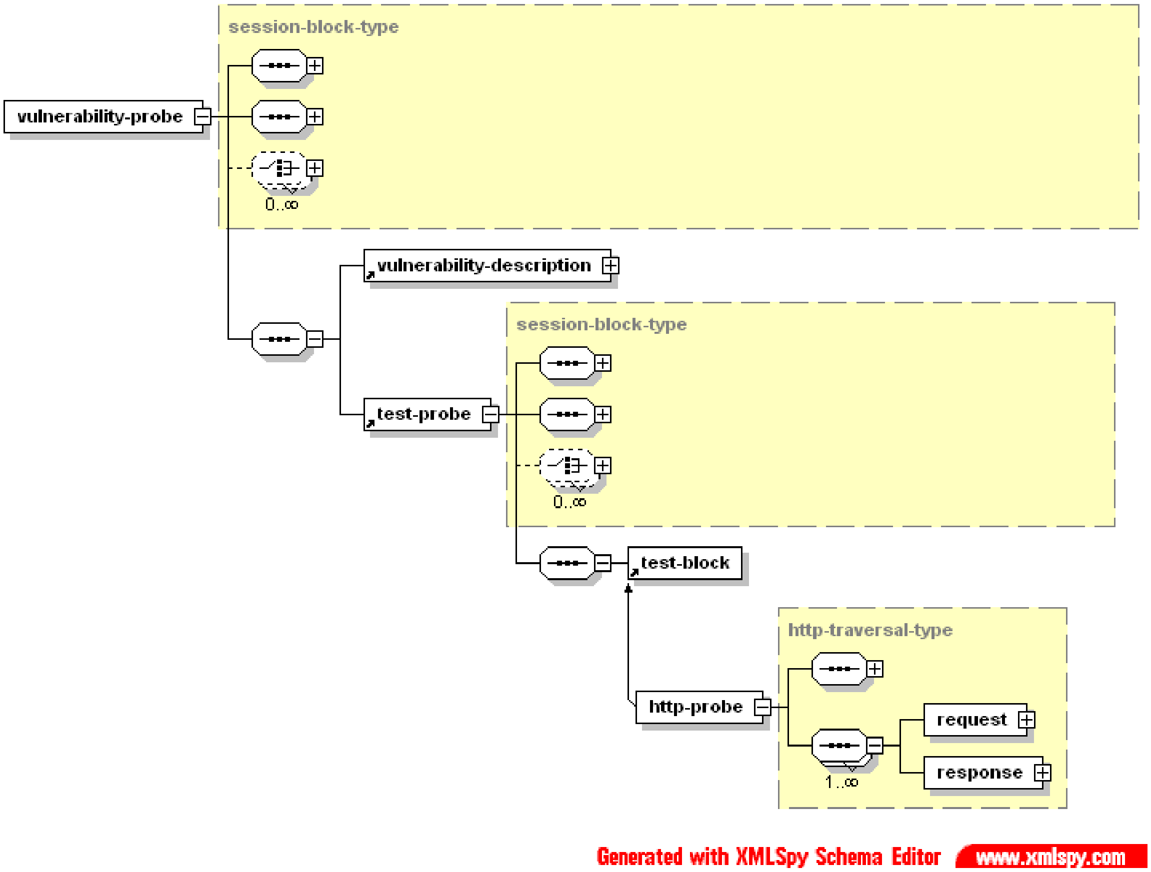


Figure 5: AVDL Vulnerability Probe Structure

Figure 5 explains the Vulnerability-Probe structure. Its children nodes include:

- **Session-block :** Definition provided in earlier section. See also Figure 4
- **Vulnerability-description :** Provides a detailed explanation of the vulnerability
- **Test-probe :** Provides a base class for all Test related stuff
  - **Test-block :** Provides base block for all Test configurations
    - **Http-probe :** A child item of Test-block. Describes how a HTTP probe was performed and the result. Provides detailed script information to automate the detection of the vulnerability, e.g. attack strings, expected return codes.

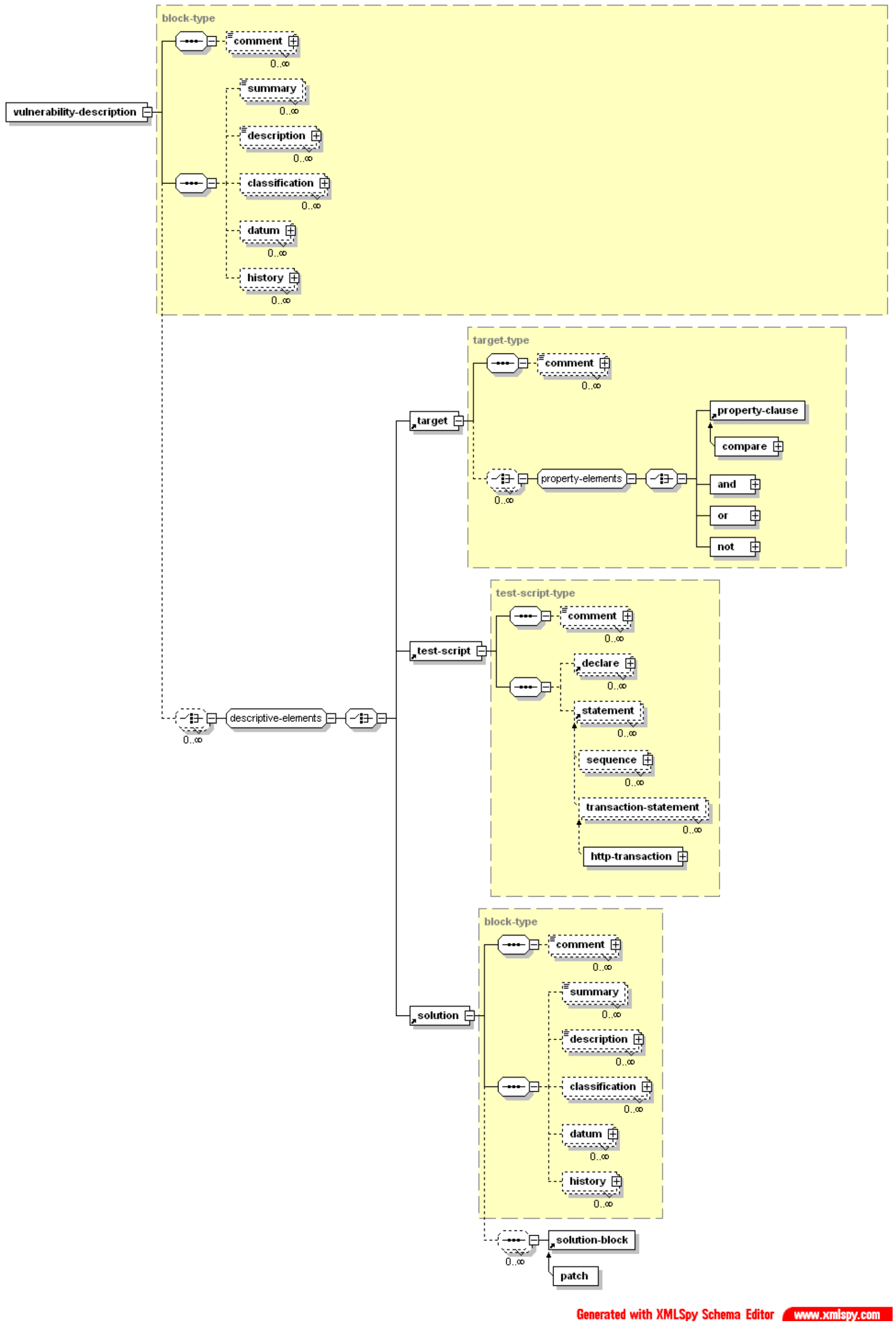


Figure 6: AVDL Vulnerability Description Structure

Figure 6 explains the AVDL Vulnerability Description structure. Its children nodes include:

- **Block-type –** Definition provided in earlier section
- **Target --** Provides details on the target, e.g. Host, OS, Architecture, Protocol, Web Server
- **Test-Script –** Provides details on the test script to reproduce the vulnerability
- **Solution --** Provides specific recommended actions to close the vulnerability

3.2 AVDL Schema Insights

This section provides detailed insights on key structures used in the AVDL schema:

- **Validated data types –** The language provides a rich set of primitive and extended data type definitions to allow tagging of properties to specific datum values. For example, tag a specific parameter in a URL to type **int.** The validated data types are extensions of XML Schema supported base type, "xs:NMTOKEN".
    - **unsigned --** An unsigned integer
    - **int –** A integer quanitity
    - **number –** Any number representation not including NaNs or infinites, e.g. integer, float
    - **date –** A date quantity
    - **time** – A time quantity
    - **date-time** – A date-time quantity
    - **string –** Any possible string quantity
    - **zero-to-unbounded-type –** A non-negative integer or the string "unbounded". Used for max-length
    - **time-stamp-type --** Timestamps are either absolute (xs:dateTime) or relative (xs:decimal as seconds since the sessions session-start)
    - **http-method-type –** Supported HTTP method types as per RFC 2616 section 5.1.1
        - OPTIONS
        - GET
        - HEAD
        - POST
        - PUT
        - DELETE
        - TRACE
        - CONNECT

- **Allowed operators –** The language provides a rich set of operators to test tagged data types with the expected result values. For example, check to see if the return response code of a specific probe matches "200 OK" or specify that the first parameter in a specific URL is always required. The set of operators include:

- **equals –** Equality operator
- **not-equals –** Exact inverse of equals
- **contains –** Matches substring operator
- **not-contains –** Does not contain (exact inverse of Contains)
- **less-than –** Less than operator
- **greater-than –** Greater than operator
- **less-or-equals –** Less than or equal (exact inverse of greater-than)
- **greater-or-equals –** Greater than or equal (exact inverse of less-than)
- **max-length –** Maximum length of data
- **min-length –** Minimum length of data
- **type –** Of type validated-type
- **required –** Specifies if the parameter is required or not
- **regex –** Compare using regular expression
- **not-regex –** Does not match using regular expression (exact inverse of regex)
- **matches –** Match using pattern where '*' matches anything
- **not-matches –** Does not match using pattern where '*' matches anything (exact inverse of match).

- **Basic raw elements –** The language provides extensive support to describe basic raw elements. This has been very useful in implementing precise test configurations to automatically detect vulnerabilities and to support various encoding formats. The set of raw elements include:
    - **eol :** Indicates a protocol appropriate end-of-line
    - **tab:** Indicates a protocol appropriate end-of-tab
    - **space:** Indicates a protocol appropriate space. This is useful to represent sequences of more than one space, since the white space in the raw block is normalized
    - **char:** Indicates one character in the encoding specified by the protocol
    - **code:** A character-type string containing any inline code
    - **base64:** A chunk of base64 data in the encoding specified by the protocol (see RFC2045)
    - **hex:** A chunk of hex data in the encoding specified by the protocol

- **Extended raw elements –** Enumerates extensions to basic raw elements
    - **var :** Substitutes the value of a variable a protocol appropriate end-of-line
    - **name:** Indicates a protocol appropriate end-of-tab
    - **attack:** Marks an attack component. This is useful in configuring specific deny policies e.g. deny a specific header or a URL

4 Enabling Security using AVDL

AVDL enhances the security of web applications by using a multi-tiered approach. The key concepts include:

- Ability to allow good guys using application positive behavior
- Ability to deny bad guys using deep knowledge on specific application vulnerabilities
- Ability to facilitate remediation engines to download patches without human intervention

The examples below show how AVDL can be used to facilitate the above goals.

4.1 Application positive behavioral model

In this specific example, the user has deep knowledge on the application 'plink.asp' hosted on domain 'www.example.com:80' input query requirements. The user then uses the AVDL descriptions to facilitate the goal.

```
<request method="GET" host="www.example.com:80" request-uri="/plink.asp?a=3&c=xyz"
version="HTTP/1.0">
    …
    <query value="a=3&c=xyz" >
        <parameter name="a" value="3" />
            <test type="int" />
            <test greater-or-equals="0" />
            <test less-or-equals="123456" />
        </parameter>
        <parameter name="c" value="xyz" />
            <test max-length="3" />
        </parameter>
    </query>
    …
</request>
```

The query portion takes two input arguments. The specifications describe that the first argument, '**a**' is of type **integer** and expects input values ranging from **0..123456**. The second specification indicates that the second argument, '**c**' expects an input not greater than 3 characters. The two conditions can then get translated to highly tuned URL allow policy by application firewall vendors.

4.2 Application vulnerability

The example below shows a Cross-site scripting vulnerability. The vulnerability occurs when a web application uses client-supplied data in an HTTP response without first filtering out potentially malicious characters. This is one of the most common types of cross-site scripting attacks. The attack strings are embedded within the <attack> … </attack> tokens. This gives specific handles on how the vulnerability was exploited and enough information to application firewall vendors to configure a highly tuned deny policy. As evident, a malicious script has been injected to the parameter **email** that the application expects.

```
<http-transaction>
<request>
    GET/join.asp?name=&email=<attack>>"><script>alert("XSS")</script></attack>&
    surname=&house=&street=&address2=&town=&postcode=&country=&homephone=
    &mobilephone=&msg=Please%2Bfill%2Bin%2Byour%2Bname <var
    name="protocol"/> Referer: http://<var name="host"/>:<var
    name="port"/>/join1.asp Connection: Close Host: <var name="host"/> User-Agent:
    Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0) Pragma: no-cache Cookie:
    ASPSESSIONIDCQADCBSB=NKAAPGKBBAJPBGDPFGEDPANA;
    Keyed=Var2=Second+Value&Var1=First+Value; Second=Oatmal+Chocolate;
    FirstCookie=Chocolate+Chip;
</request>
<response>
    <expect status-code="200" reason-phrase="OK" />
</response>
</http-transaction>
```

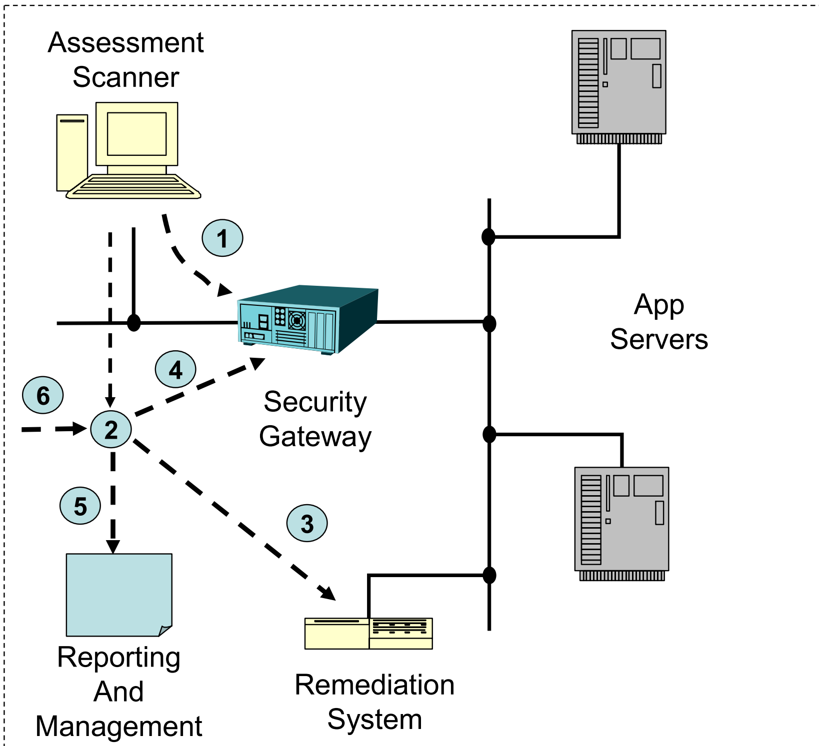
4.3 Remediation

Remediation is the recommended action to close the vulnerability. It includes an identifier for the remedy, a description, and the vendor responsible for creating the remedy. The action code is vendor specific to the vendor specified by the Vendor field. In addition, it includes an open block that allows for machine-readable code. This may include code for the remediation software to download the patch to fix the vulnerability.

```
- <recommendation>
- <patch name="Microsoft patch Q256888_W2K_SP1_x86_en" lang="english" test-ref="test-
1">
  <description>Microsoft has released a patch which eliminates this
vulnerability.</description>
  <vendor name="Microsoft" />
  <patch-source
href="http://download.microsoft.com/download/win2000platform/Patch/Q256888/NT5/EN-
US/Q256888_W2K_SP1_x86_en.EXE" patch-ref="Q256888_W2K_SP1_x86_en" />
  <remediation vulnID="02134" language="VBScript" modDate="030911131212" vendor="Citadel"
actionhref="http://vendor.remediation.com/library/q25688.vb" actionCode="REM Copyright 2003,
Citadel Security Software, Inc. All Rights Reserved. All product names are trademarks or registered
trademarks of their respective owners. Specifications subject to change without notice. REM Script
Generated Automatically by skey at 9/10/2003 2:04:30 PM Option Explicit
HercClient.SetScriptReturnCode( 5 ) REM Failure Dim sVersion, sFull, sSP, bPassed bPassed = true If
bPassed = true Then If HercClient.IsWindowsXP() = True then If HercClient.WindowsCSDVersion >
Service Pack 1 Then bPassed = True Else bPassed = False End If End If End If" />
  </patch>
  </recommendation>
  </vulnerability-description>
  </vulnerability-probe>
```

```
</session>
</avdl>
```

5 AVDL Life Cycle Example



The illustration shows an example of how AVDL enabled world would look like. The example shows how AVDL automation creates an integrated security environment for a typical web application deployment.

1. First, a scan maps the application structure and detects any security holes recording AVDL probes.
2. The AVDL probes can be placed in a file for user-initiated off-line batch processing (or not shown) can be used for real-time communications.
3. A remediation system reads this AVDL file and uses information from its “vulnerability probes” to automate the application of appropriate patches and hot fixes.
4. An application security gateway uses this vulnerability probes to configure secure perimeter that protects from attacks and uses traversal probes to automate the enforcement of site’s legitimate access policies.

5. AVDL data is presented to the user in the form of security audit reports and through other management tools.
6. AVDL data can also be supplied from other sources (not pictured) such as security policy managers, security alerts and bulletins, application development platforms or host-based security analyzers.

407 # Revision History

Rev	Date	By Whom	What
wd-01	2004-03-22	Srinivas Mantripragada	Version 1.0
wd-02	2004-03-22	Srinivas Mantripragada	Review comments added

408

Appendix A. Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself does not be modified in any way, such as by removing the copyright notice or references to OASIS, except as needed for the purpose of developing OASIS specifications, in which case the procedures for copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an “AS IS” basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.