A Semantic Web Technology-based Architecture for New Server-side Data Validation in Web Applications

Shadi Aljawarneh and Faisal Alkhateeb
Faculty of Information Technology
Al-isra Private University and Yarmouk University, Jordan
shadi.jawarneh@ipu.edu.jo, alkhateebf@yu.edu.jo

Abstract

Criminals could break the client-side input validation modules. Bypassing input validation is a serious challenge because it might cause failures in the software, and can also break the security upon web applications such as an unauthorized access to data. Even the criminals can not bypass the client and/or server input validation, web application flaws, such as cross-site scripting or SQL injection, now account for more than two thirds of the reported web security vulnerabilities. In this paper, we present a new data validation service which is based upon semantic web technologies to prevent the security vulnerabilities at the application level and to secure the web system even if the input validation modules are bypassed. The architecture of the service consists of the following components: RDFa annotation for elements of web pages, interceptor, RDFa extractor, RDF parser, and data validator.

Keywords

Web application, RDFa, web system, ontology, semantic web technologies, data validation, vulnerabilities, SSL

1. Introduction

Organizations that have a web presence are increasingly worried for their reputations if the web system is subverted. This is because current security tools may not prevent the web system vulnerabilities [9, 11]. For example, with 7,247 vulnerabilities disclosed in 2006, total vulnerability count increased nearly 40% over the previous year. This trend of increase is expected to continue [16].

Web applications[1] is organized into three tiers: a web browser tier, a web server tier, and a backend database tier. The user interaction is proposed in a web browser tier, the program logic (such as ASP and JSP) is run in a web server tier, and the data operations (such as addition, deletion, and updating) are performed in a database server tier [26, 5]. It often have direct access to backend databases and, hence, sensitive data is much more difficult to secure [2]. If there is no direct access to backend databases, attacks can use legitimate application protocols such as HTTP, and Simple Object Access Protocol (SOAP) to capture data and transmissions [2, 4, 7]. The Gartner study found that 75% of Internet assaults are targeted at the web application level [4].

Currently, when initiating user and organization transactions and conducting their business, e-Commerce applications rely on (X)HTML forms including enrolment, authentication, order entry, payment, and profiling, rather than XML forms, because of a lack of security mechanisms supported by SOAP [17]. In addition, the XML form is not supported by the major web browsers because it needs additional installations on the client and server-sides [17, 6].

An input validation scheme is the first defence against web attacks at the application level. Web developers have adopted a number of validation approaches to prevent loss of data integrity.

1. Server-side input validation [6, 27]: this approach can be used to validate sensitive data on a server before processing them by an application server. Depending upon the application and network traffic, the time taken between the submitted form on a web browser and the error message that is returned from a web server can be considerable. In addition, it makes excess network traffic to enter the correct data format. However, inside criminal might bypass the server-side input validation modules through using malicious manipulation software that intercept the user inputs at the server-side.

2. Client-side input validation [6, 10]: this is effective for minimizing the number of necessary communica-

[1] A web application is a collection of integrated static and dynamic web pages on a web system. The web application is run on a web browser, a web server, or both [1, 26].

tion hits between the submitted form and received error message. However, the form validation modules of this approach can be removed. In addition, this approach cannot ensure that the client and server are authentic.

3. The double-checking input validation [6, 10]: this approach duplicates the form validation modules on both client and server sides. This approach adopts alternative validation scheme on a server-side, even though the validation scheme is bypassed at the client-side. However, this approach is expensive and involves high latency.

4. Honkala and Vuorimaa [17, 13] propose extending the XForm form to a digital signature XForm. They adopt the digital signature for XForm forms rather than (X)HTML forms because it is hard to apply a digital signature to an (X)HTML form. They advocate the "what you see is what you sign" approach to secure web form components at the client-side. Therefore, XForms is a new standard for better graphical interfaces and specified to input validation rather than the embedded scripts [27]. However, XForm is only supported by the XSmile browser.

5. Formatta organization defines a Portable Form Files (PFF) form model that is not related to the (X)HTML language. This form allows the user to encrypt and lock its form data before submitting it to a web server. However, the PFF form needs special software to install a web document on a web browser. In addition, the submitted data is sent by E-mail service to an organization web server.

Many online book re-sales (such as Amazon) advocate checking inputs with JavaScript as a mechanism to reduce network traffic. Modern books applications usually advocate doing input validation on the server for security purposes. Nevertheless, major e-Commerce and e-Service sites still use client-side validation and hidden fields [21].

Therefore, Criminals could break the client-side input validation modules. Bypassing input validation is a serious problem because it might cause failures in the software, and can also break the security upon web applications such as an unauthorized access to data [3, 18]. Even the criminals can not bypass the client and/or server input validation, web application flaws, such as cross-site scripting or SQL injection, now account for more than two thirds of the reported web security vulnerabilities [18]. In an attempt to remedy this, we develop a new data validation services, based on semantic web technologies.

This paper is organized as follows: case studies for the bypassing input validation are described in Section 2. The proposed semantic web technology-based architecture is introduced, and a case study is presented in Section 3, and related work is made in Section 4. Conclusions and future work are offered in Section 5.

2. Case studies for input validation bypassing

A validation scheme is necessary for both client and server-sides, but is not sufficient to ensure data integrity of web applications, because fundamentally a client-side input validation scheme is designed to validate basic properties of the input data: length, range, format, default value, and type. In addition, input validation can be used to enhance resistance to injection attacks such as SQL injection attack because SQL injection vulnerabilities result from insufficient input validation [12]. However, an input validation scheme is useless if any malicious script or listener is already installed on a server [18, 22, 6].

As a result of the transparency of code at the web browser level, the following approaches can cause loss of data integrity at the (X)HTML form level:

1. Hidden fields manipulation: an adversary saves the (X)HTML form to a disk, modifies a hidden field value (such as the price of a product), and then reloads this tampered form into a web browser for rendering [22].

2. Script manipulation: an adversary removes the client validation modules from a web browser to submit illegal data to a web server. A web server accepts the tampered form and then the data is saved in a back-end database. Many web application security vulnerabilities come from input validation problems including Cross-Site Scripting (XSS) and SQL injection [18, 23, 24, 19]. This approach is made possible by removing all script modules between the ¡script¿ and ¡/script¿ tags, removing the event-handler that invokes the validation modules, or turning off the script and Java Applet options via web browser settings.

3. Modules of validation analysis manipulation: an adversary applies reverse engineering techniques on the validation modules [18, 23, 19].

4. Session information manipulation [25]: an adversary might access session information, which is saved in cookies. This is possible because a web browser is not fully secured by SSL. In addition, some client-side programming languages provide direct or indirect commands to access user machine resources. However, the cookie-securing mechanism of Park and Sandhu [20] can be adopted to avoid or reduce the danger of tampering the cookie information but it needs to declare the explicit modification to the existing web environment. This mechanism is implemented using CGI and Pretty Good Privacy (PGP) to verify se-

cure cookie cryptography on the client. The encryption scheme is conducted by the server. Therefore, the cookie-securing mechanism is used to secure the session information on the cookies for continuing the Request-Response conversations.

As mentioned above, the SQL injection is one of the common web application vulnerabilities. Normally, web applications use data that read from a user to construct database queries. If the data is not properly processed, malicious code that results in the execution of any SQL can be injected [15].

To explain more about the SQL injection and how to authentication mechanism of a web application can be bypassed, consider the following scenario: a web page includes a (X)HTML form with two edit boxes in its login.html to ask for a username and password. The form declares that the values of the two input fields should be submitted with the variables varUserName and varPassword to login.asp, which includes the following code [15]:

```
SQLQuery = "SELECT * FROM Users_table
WHERE (UserName='" + varUserName + "')
AND (Password='" + varPassword + "');"
```

If a user submits the username "Ali" and the password "2009yosef", the `SQLQuery` variable is interpreted as:

```
"SELECT * FROM Users_table WHERE
(varUserName= 'Ali') AND
(Password='2009yosef');"
```

It should be noted that a user inputs (stored in the `varUserName` and `varPassword` variables) are used directly in SQL command construction without preprocessing, thus making the code vulnerable to SQL injection attacks. If a malicious user enters the following string for both the UserName and Password fields:

```
X' OR 'A' = 'A
```

then the `SQLQuery` variable will be interpreted as:

```
"SELECT * FROM Users_table WHERE
(varUserName='X' OR 'A' = 'A') AND
(Password='X' OR 'A' = 'A');"
```

Because the expression `'A' = 'A'` will always be evaluated as TRUE, the WHERE clause will have no actual effect, and the SQL command will always be the equivalent of `"SELECT * FROM Users_table"`. Therefore, allowing the web application's authentication mechanism to be bypassed.

3. Possible Solution

We present a new data validation service which is based upon semantic web technologies to prevent the security vulnerabilities at the application level and to secure the web system even if the input validation modules are bypassed. As illustration in Figure 1, the data validation service architecture consists of the following components: RDFa annotation for elements of web pages, interceptor, RDF extractor, RDF parser, and data validator. The next subsection will describe the functional overview of the proposed solution.

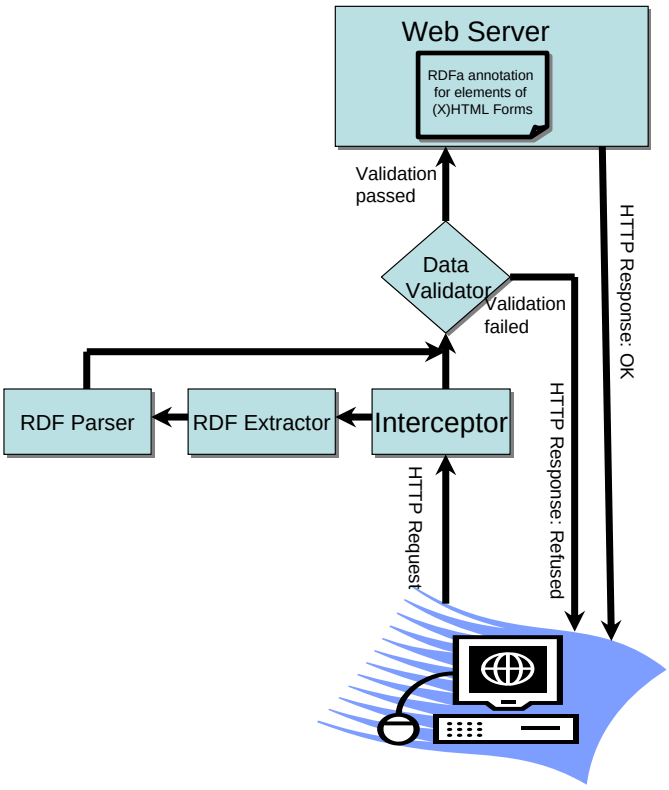

Figure 1. Schematic view of new data validation service architecture

It should be noted that the components of the proposed architecture framework do not need to run on a dedicated machine, they can be run as separate processes on the server.

3.1. Functional Overview

The following steps are performed:

1. Use an ontology to describe all data elements in a web application using RDFa annotation[2].

[2]http://wwww.w3.org/TR/xhtml-rdfa-primer/

2. End user requests (X)HTML form.

3. Interceptor component intercepts each HTTP request at the server-side before the request arrives to web server application for processing.

4. Extracting the RDFa annotations from RDFa ontology vocabulary using the online RDFa extractor[3].

5. Invoking the validator component to validate all user inputs.

6. If the validation is correct then the request sends to web server application for processing, otherwise, the request is refused.

3.2. Overview of the proposed framework architecture

An illustration of RDFa ontology-based architecture is presented in Figure 1. This framework consists of five components:

1. RDFa annotation for elements of web pages: RDF (Resource Description Framework) is a knowledge representation language dedicated to the annotation of resources within the Semantic Web. In its abstract syntax, an RDF document is a set of triples of the form ⟨subject, predicate, object⟩. Currently, many documents are annotated via RDF due to its simple data model and its formal semantics. For example, it is embedded in (X)HTML web pages using the RDFa language, in SMIL documents using RDF/XML, etc. Section 3.3 provides an illustration of how to use RDFa to annotate an (X)HTML web page.

2. Interceptor: mediates between the server and client machines by managing the HTTP requests. It intercepts HTTP request, checks the availability of HTTP request on the designated directories of web server, and invokes the RDF extractor.

3. RDF extractor: The online RDFa distiller[4] is used to extract the RDFa annotation from the (X)HTML web page and construct the RDF ontology given in Figure 2.

4. RDF parser: parses the form inputs and their attributes for validation process.

5. Data validator: when the description is extracted using RDFa extractor, the validator takes the user inputs for validation process. The validation process checks to see if the value of user input is satisfied the conditions of its attributes (such as length, data type, minimum length, and if the value contains code or special characters) the since it was used. Any mismatching causes the content integrity check to fail. Based on whether the test passes or fails, the data validator enforces the policy that makes the decision about the next step in the process. If the integrity check passes, the web content is sent to the running process straight away. If it fails, it is refused the user request.

[3]Note that there are several RDF extractors available at http://www.w3.org/topic/RDFa

[4]www.w3.org/2007/08/pyRDFa/

3.3. Case Study

To illustrate our methodology we consider using our system to secure a simple employee system. Consider the following scenario: As final step in a registration transaction, employees are sent an (X)HTML form requesting their name, address, department, and qualification.

```
<FORM NAME=EmployeeForm ACTION=emp_add.jsp METHOD=post>
<h2>Add Employee Record</h2>
<B><I>Employee Number: <br>(1 to 6 characters)</I></B>
<INPUT TYPE=text NAME=EMPNO>
<BR><B><I>First Name:</I></B>
<INPUT TYPE=text NAME=FNM VALUE=First Name>
<B><I>Middle Initial:</I></B>
<INPUT TYPE=text NAME=MIDINIT VALUE=M>
<BR><B><I>Last Name: </I></B>
<INPUT TYPE=text NAME=LNME VALUE=Last Name>
<BR><B><I>Telephone:</I></B>
<INPUT TYPE=text NAME=telep >
<BR> <B><I>Department:</I></B>
<SELECT NAME=WORKDEPT >
<OPTION VALUE= 1 selected> Sales
<OPTION ALUE= 2 >Marketing
<OPTION VALUE= 3 >Development
</SELECT>
<B><I>Education:</I></B>
<SELECT NAME=EDLEVEL>
<OPTION VALUE=1 SELECTED>BS
<\scriptsize{OPTION VALUE=2 >MS
<OPTION VALUE=3>PhD
</SELECT>
<INPUT TYPE=submit NAME=Submit VALUE=Add>
</FORM>
```

Figure 2. Snapshot of an employee (X)HTML form.

Figure 3 illustrates the modified (X)HTML form as well as the ontology description. The shaded rows denotes to the ontology which describes each field in the (X)HTML form.

```
<FORM NAME=EmployeeForm ACTION=emp_add.jsp METHOD=post>
<h2>Add Employee Record</h2>
<B><I>Employee Number: <br>(1 to 6 characters)</I></B>
<span property=vcard:KEY content=EMPNO/>
<INPUT TYPE=text NAME=EMPNO>
<BR><B><I>First Name:</I></B>
<span property=foaf:firstName content=fnm/>
<INPUT TYPE=text NAME=FNM VALUE=First Name>
<B><I>Middle Initial:</I></B>
<span property=foaf:midname content=mid/>
<INPUT TYPE=text NAME=MIDINIT VALUE=M>
<BR><B><I>Last Name: </I></B>
<span property=foaf:surname content=lnm/>
<INPUT TYPE=text NAME=LNME VALUE=Last Name>
<BR><B><I>Telephone:</I></B>
<span property=foaf:phone content=telephone/>
<INPUT TYPE=text NAME=telep >
<BR><B><I>Department:</I></B>
<span property=foaf:member content=WORKDEPT/>
<SELECT NAME=WORKDEPT >
<OPTION VALUE= 1 selected> Sales
<OPTION ALUE= 2 >Marketing
<OPTION VALUE= 3 >Development
</SELECT>
<B><I>Education:</I></B>
<span property=foaf:title content=EDLEVEL/>
<SELECT NAME=EDLEVEL>
<OPTION VALUE=1 SELECTED>BS
<\scriptsize{OPTION VALUE=2 >MS
<OPTION VALUE=3>PhD
</SELECT>
<INPUT TYPE=submit NAME=Submit VALUE=Add>
</FORM>
```

Figure 3. Snapshot of he modified HTML form with the ontology description.

The ontology itself extracted using RDFa extractor is shown in Figure 4 This ontology means that there exists someone whose first name ”foaf:firstName” is the ”fnm” (Note this is the name of the label), last name ”foaf:lastname” is ”lnm”, employee key ”vcard:KEY” is ”EMPNO”, phone number ”foaf:phone” is ”telephone”, fax number ”foaf:fax” is ”faxNumber”, mbox ”foafmbox” is ”emailbox”, title ”foaf:title” is ”EDLEVEL”, address ”vcard:ADR” is ”address”. This person is a member ”foaf:member” of ”WORKDEPT”. The employee ontology is stored in the employeeontology.ttl which contains:

```
# this is a comment
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
_:someEmployee rdf:type foaf:Person.
_:someEmployee vcard:KEY 'EMPNO' .
_:someEmployee foaf:firstName 'fnm' .
_:someEmployee foaf:surname 'lnm' .
_:someEmployee foaf:phone 'telephone' .
_:someEmployee foaf:fax 'faxNumber' .
_:someEmployee foaf:mbox 'emailbox' .
_:someEmployee foaf:member 'WORKDEPT' .
_:someEmployee foaf:title 'EDLEVEL' .
_:someEmployee vcard:ADD 'address' .
```

Figure 4. Snapshot of the employee ontology is stored in the employeeontology.ttl file. Note that this RDF ontology is written in the Turtle format, http://www.dajobe.org/2004/01/turtle/

4. Related work

A number of researchers are developing solutions to address this problem. For example, Scott and Sharp [22] proposed a gateway model which is an application-level firewall on a server for checking invalid user inputs and detecting malicious script (e.g. SQL injection attack and cross-site scripting attack). This approach offers protection through the enforcement of a number of defined policies, but fails to assess the code itself or to identify the actual weaknesses. They have developed a security policy description language (SPDL) based on XML to describe a set of validation constraints and transformation rules. This language is translated into code by a policy compiler, which is sent to a security gateway on a server. The gateway analyzes the request and augments it with a Message Authentication Code (MAC).

Another different approach to make self-protection, Huang and others [14] used behavior monitoring to detect malicious content before it reaches users. They develop WAVES (Web application security assessment system) that performs behavior stimulation to induce malicious behavior in the monitored components. However, the testing processes cannot guarantee the identification of all bugs, and they cannot support immediate or direct security for web applications.

MOPS [8] used the static analysis techniques that have been made to identify security vulnerabilities in UNIX programs. Static analysis can also be used to analyze web application code, for instance, ASP or PHP scripts. However, this technique fails to adequately use the runtime behavior of web applications [15]

5. Conclusions and further work

Because of the possibility of bypassing input validation either on client-side or server-side, data integrity of web application can be violated even though the communication channel between the server and client is secure. Therefore, we present the proposed web technology-based architecture for new data validation in the web applications. This architecture includes a real-time framework consisting of five components: RDFa annotation for elements of web pages, interceptor, RDF extractor, RDF parser, and data validator. It might be suggested that the proposed data validation service could provide a detection, and prevention of some web application attacks. In the next stage of this research, we will implement the prototype of our proposed system and will investigate a number of experiments for security and performance objectives.

References

[1] Acunetix. The importance of web application scanning, 2005. http://www.sql-server-performance.com/wpaper_web_app_scanning.asp, Accessed Date: 20/4/2005.

[2] Acunetix. Web applications: What are they? what of them?., 2007. http://www.acunetix.com/websitesecurity/web-applications.htm, Accessed Data: 15/2/2007.

[3] S. Aljawarneh, C. Laing, and P. Vickers. Security policy framework and algorithms for web server content protection. In *ACSF '07*, Liverpool, UK, 12–13 July 2007. Liverpool John Moores University.

[4] T. Bass. CEP and SOA: An open event-driven architecture for risk management. IT Financial Services '07, Portugal, 2007. www.idc.pt/resources/PPTs/2007/Financial_Services/7_TI BCO.pdf.

[5] Boston Consulting Group. Report of the E-Business Opportunities Roundtable. Fast Forward: Accelerating Canada's Leadership in the Internet Economy, Jan 2000. http://www.e-com.ic.gc.ca/epic/site/ecic-ceac.nsf/en/h_gv00222e.html, Canada, Accessed Data: 5/12/2005.

[6] C. Brabrand, A. Moller, M. Ricky, and M. I. Schwartzbach. PowerForms: Declarative client-side form field validation. *World Wide Web Journal*, 3(4):205–314, December 2000. Kluwer.

[7] R. Cardone, D. Soroker, and A. Tiwari. Using XForms to simplify web programming. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 215–224, New York, NY, USA, 2005. ACM Press.

[8] H. Chen and D. Wagner. Mops: an infrastructure for examining security properties of software. In *In Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 235–244. ACM Press, 2002.

[9] B. Gehling and D. Stankard. eCommerce security. In *Proceedings of Information Security Curriculum Development (InfoSecCD) Conference 05*, pages 32–37, Kennesaw, GA, USA, Sep 23–24 2005.

[10] A. Ghosh and T. Swaminatha. Software security and privacy risks in mobile e-commerce. *Commun. ACM*, 44(2):51–57, 2001.

[11] W. Glisson and R. Welland. Web development evolution: The assimilation of web engineering security. In *LA-WEB '05: Proceedings of the Third Latin American Web Congress*, page 49, Washington, DC, USA, 2005. IEEE Computer Society.

[12] G. Halfond and A. Orso. Preventing SQL injection attacks using AMNESIA. In *ICSE '06: Proceedings of the 28th international conference on Software engineering, ACM*, pages 795–798, New York, NY, USA, 2006. ACM.

[13] M. Honkala. *Web User Interaction a Declarative Approach Based on XForms*. Technology, Department of Computer Science and Engineering - Helsinki University of Technology, Espoo, Finland, January 2007. ISBN 978-951-22-8566-2.

[14] Y. Huang, S. Huang, T. Lin, and C. Tsai. Web application security assessment by fault injection and behavior monitoring. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 148–159, New York, NY, USA, 2003. ACM Press.

[15] Y.-W. Huang, S.-K. Huang, T.-P. Lin, and C.-H. Tsai. Web application security assessment by fault injection and behavior monitoring. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 148–159, New York, NY, USA, 2003. ACM.

[16] IBM. X-Force 2006 Trend Statistics, Jan 2007. http://www.iss.net/documents/whitepapers/X_Force_Exec_Brief.pdf, Accessed Date: 7/8/2007.

[17] H. Mikko and P. Vuorimaa. Secure Web Forms with Client-Side Signatures. In *ICWE*, pages 340–351, 2005.

[18] J. Offutt, Y. Wu, X. Du, and H. Huang. Bypass testing of web applications. In *ISSRE 2004 15th International Symposium on Software Reliability Engineering*, pages 187–197. IEEE Computer Society, Los Alamitos, CA, 2004.

[19] Open Web Application Security Project. The Ten Most Critical Web Application Security Vulnerabilities. Version 1.1, January 13 2003.

[20] J. Park and R. Sandhu. Secure cookies on the web. *IEEE Internet Computing*, 4(4):36–44, 2000.

[21] F. Ricca and P. Tonella. Analysis and testing of web applications. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 25–34, Washington, DC, USA, 2001. IEEE Computer Society.

[22] D. Scott and R. Sharp. Specifying and Enforcing Application-Level Web Security Policies. *IEEE. Knowl. Data Eng*, 15(4):771–783, 2003.

[23] S. Sedaghat. Web authenticity. Master's thesis, University of Western Sydney, Australia, 2002.

[24] S. Sedaghat, J. Pieprzyk, and E. Vossough. On-the-fly web content integrity check boosts users' confidence. *Commun. ACM*, 45(11):33–37, 2002.

[25] A. Sengupta, C. Mazumdar, and M. S. Barik. e-Commerce security - A life cycle approach. In *Commerce Security; Threats And Vulnerabilities; Security Engineering Life-Cycle; Security Standards; IT Act*, volume vol.30, pages p.119–140. SADHANA, 2005.

[26] J. Tzay, J. Huang, F. Wang, and W. Chu. Constructing an Object-Oriented Architecture for Web Application Testing. *IJ. Information Science and Eng.*, 18(1):59–84, 2002.

[27] J. Wusteman. Web forms: the next generation. *Library Hi Tech*, 21(3):367 – 381, 2003.