Design and Implementation of New Data Validation Service (NDVS) Using Semantic Web Technologies in Web Applications

Shadi Aljawarneh* Faisal Alkhateeb†

Abstract—**We have designed a novel server-side data validation service, based upon semantic web technologies to solve the lack of data validation and bypassing validation issues. The NDVS consists of five components: RDFa annotation for elements of web pages, interceptor, RDFa extractor, RDF parser, and data validator. Our solution is implemented as a prototype. In this paper, we have conducted a pilot study to prevent the security vulnerabilities at the application level such as SQL injections. The results of this initial study have shown that the proposed service (NDVS) could provide a high coverage of prevention of security vulnerabilities.**

Keywords: Web application, data integrity, RDFa, web system, ontology, semantic web technologies, data validation, vulnerabilities

1 Introduction

Several security incident reports from emergency response teams such as The Computer Emergency Response Team (CERT) clearly demonstrate that the available security mechanisms have not made system break-ins impossible [21, 3]. Furthermore, the Gartner study found that 75% of Internet assaults are targeted at the web application level [3]. Consequently, the data integrity can be violated on the server even though the communication channel between the server and client is secure.

Web applications 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 [20]. It often have direct access to backend databases and, hence, sensitive data is much more difficult to secure [1]. 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 [1, 3, 5].

*Faculty of Science & Information Technology, P.O. Box 22, Al-Isra Private University,Amman,Jordan 11622,shadi.jawarneh@ipu.edu.jo

†Yarmouk University, Irbid, Jordan, alkhateebf@yu.edu.jo

Data 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 [4]: 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. 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 [4, 8]: this is effective for minimizing the number of necessary communication 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 [4, 8]: 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 [12, 10] 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. However, XForm is only supported by the XSmile browser.

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 [2, 13]. 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 [13]. 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 the prototype of new validation service implementation and the inial testing are described in Section 4. Related work is made in Section 5. Finally, conclusions and future work are offered in Section 6.

2 Case studies for data 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 [9]. However, an input validation scheme is useless if any malicious script or listener is already installed on a server [13, 18, 4].

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 [18].
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 backend database. Many web application security vulnerabilities come from input validation problems including Cross-Site Scripting (XSS) and SQL injection [13, 19, 16]. 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 [13, 16].

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 [11]. 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 [11]:

```
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 Architecture of New Data Validation Service

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.

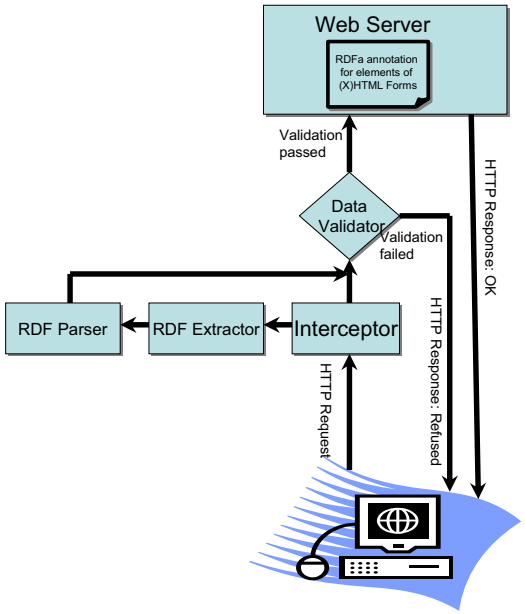

Figure 1: Schematic view of new data validation service architecture

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[1].
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[2].
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.

[1]http://wwww.w3.org/TR/xhtml-rdfa-primer/
[2]Note that there are several RDF extractors available at http://www.w3.org/topic/RDFa

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[3] 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.

3.3 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.

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

```
<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.

```
<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
:. </FORM>
```

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

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.

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:

4 Implementation of New Data Validation Service (NDVS)

The proposed service (NDVS) is implemented in Java using JBuilder (2007) and Java Servlet and filters. The web servers used are Apache 1.3.20 running on MS Windows Server 2003, and Apache Tomcat 5.01 on MS Windows Server 2003. As far as performance is concerned, NDVS is able to prevent infinite number of application attacks. The prototype implementation of NDVS service consists of three major components: HTTP Interceptor Mechanism, RDF parser , and Data validator.

1. HTTP Interceptor Mechanism: The HTTP Interceptor takes advantage of the fact that browser requests are directed at both a specific host and a specific port. In this program, the Tomcat server listens on port 8081. The utility listens for browser requests on a default port 80 and redirects to Tomcat. Responses coming to this mechanism are both sent to the client on port 80.

   In this paper, we have developed multi-threaded java application for handling concurrent connections (requests in parallel) using multiple threads that increase the power and flexibility of a web server and client programs significantly.

2. RDF parser: It is written in Java programming language to parse the form inputs and their attributes. Each form input is parsed, the id of input is sent to the Data validator mechanism. It should be noted the attributes of each input also is sent to the the Data validator mechanism.

```
# 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/

3. Data validator: when the description is extracted using RDFa extractor, the validator takes the user inputs for validation process as shown in Section 3. 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.

The testing environment is composed of Apache 1.3.29 with Tomcat container 5.01 on MS Windows Server 2003. The Tomcat web server contains a copy of target web site and shopping cart application. The shopping cart includes five (X)HTML forms. Over 45 attacks have been performed against the generated static and dynamic web content security properties. We exploited different type of security vulnerabilities (such as spoofing attack, and SQL attacks). As a result, all the attacks launched against the suggested server and clients were prevented by the NDVS service.

Note that, the duration of the test was almost exactly 30 minutes where the run-time policy was ramping up (i.e. Generating a number of virtual users that increases throughout the test.) from 2 users by adding 2 users every 2 minutes. The virtual users were connecting at 100Mbps through a local network. The NDVS service tested is a prototype, and thus is not really optimised.

5 Related work

A number of researchers are developing solutions to address this problem. For example, Scott and Sharp [18] 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 [11] 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 [6] 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 [11].

Jovanovic et al. [14] have developed Pixy, which is the first open source tool for statically detecting XSS vulnerabilities in PHP 4 code by means of data follow analysis which is based on a static analysis technique. Although the Pixy prototype is aimed at the detection of XSS vulnerabilities, it can be equally applied to other taint-style vulnerabilities such as SQL injection or command injection.

However, Pixy does not support object-oriented features of PHP. Each use of object member variables and methods is treated in an optimistic way, meaning that malicious data can never arise from such constructs. Further limitation is that they have focused on the problem of identifying vulnerabilities in which external input is used without any prior sanitization (e.g. a particular type of input validation). It should be noted that a sanitization is performed to remove possibly malicious elements from the user input These vulnerability detectors are typically based on data flow analysis that tracks the flow of information from the inputs of application's (called sources) to points in the program that represent security-relevant operations (called sinks). However, the assumption of this approach is that if a sanitization operation is performed on all paths from sources to sinks, then the application is secure.

Cova et al [15] have presented approach to the analysis of the sanitization. This means that they combined static and dynamic analysis techniques to identify faulty sanitization procedures that can be bypassed by the criminal. Therefore, they implemented this approach in a

tool, called Saner, and they applied it to a number of real-world web applications. They have described a static analysis technique that characterizes the sanitization process by modeling the way in which a web application processes input values. This permits us to define the cases where the sanitization is incorrect or incomplete. Furthermore, they introduced a dynamic analysis technique that is able to reconstruct the code that is responsible for the sanitization of application inputs, and then execute this code on malicious inputs to identify faulty sanitization procedures.

6 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 future work, we are intended to optimize the implementation of our solution to increase the effectiveness and performance. Furthermore, we will investigate a number of experiments for security and performance objectives.

7 The References Section

References

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

[2] 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.

[3] 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_TIBCO.pdf.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] Jovanovic,, Nenad and Kruegel,, Christopher and Kirda,, Engin. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 258–263. IEEE Computer Society, Los DC, USA, 2006.

[15] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. PSaner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In *In Proceedings of the 2008 IEEE Symposium on Security and Privacy*, pages 387-401. SP. IEEE Computer Society, Washington, DC, 2008.

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

[17] 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.

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

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

[20] 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.

[21] CERT. CERT Statistics 1988–2006., Jan 2007. http://www.cert.org/stats.