Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

A Semantic Data Validation Service for Web Applications

Shadi Aljawarneh[1], Faisal Alkhateeb[2] and Eslam Al Maghayreh[3]

[1] Isra University, Science and IT Faculty, shadi.jawarneh@ipu.edu.jo
Yarmouk University, Computer Science Department, IT Faculty
[2] alkhateebf@yu.edu.jo, [3] eslam@yu.edu.jo

Received 13 August 2009; received in revised form 9 December 2009; accepted 3 March 2010

Abstract

An Input validation can be a critical issue. Typically, a little attention is paid to it in a web development project, because overenthusiastic validation can tend to cause failures in the software, and can also break the security upon web applications such as an unauthorized access to data. Now, it is estimated the web application vulnerabilities (such as XSS or SQL injection) for more than two thirds of the reported web security vulnerabilities. In this paper, we start with a case study of the bypassing data validation and security vulnerabilities such as SQL injection and then go on to discuss the merits of a number of common data validation techniques. We also review the different solutions to date to provide data validation techniques in e-commerce applications. From this analysis, a new data validation service which is based upon semantic web Technologies, has been designed and implemented to prevent the web security vulnerabilities at the application level and to secure the web system even if the input validation modules are bypassed. Our semantic architecture consists of the following components: RDFa annotation for elements of web pages, interceptor, RDF extractor, RDF parser, and data validator. The experimental results of the pilot study indicate that the proposed data validation service might provide a detection, and prevention of some web application attacks.

Key words: Web Application, Data Validation, Vulnerabilities, e-Commerce, SQL injection, Web system, On the fly, Data Tampering

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

1 Introduction

Several reports have shown a sharp increase in the number of web-based attacks [1]-[2]. For example, The Computer Emergency Response Team (CERT) [1] has reported that there has been a dramatic increase in the number of security vulnerabilities (weaknesses in a computing system) which threaten web content (5990 in 2005 to over 6000 in end of 2008).

A database server tier of a web application often has direct access to backend databases [19] and, hence, sensitive data is much more difficult to secure [3]. 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 [3]-[5]. Access to databases through Web browsers is now common place with some form of web server being provided by all major vendors of database software [5]. E-commerce such as online banking, enterprise collaboration, and e-Purchasing sites have concluded that at least 92% of web applications are vulnerable to some form of attack [6]. The Gartner study found that 75% of Internet assaults are targeted at the web application level [5]. Furthermore, according to the recently released X-Force 2008 Trend and Risk Report, web application vulnerabilities estimated for above 54% of all vulnerabilities disclosed at end of 2007, and 74% of these vulnerabilities still had no vendor-supplied patch as of the end of 2008 [18].

Because an inadequate input validation is a challenge, the OWASP (Site 1) mentioned the top ten security vulnerabilities affecting the web. Several security issues in applications are caused by inadequate input validation including:

- Parameter manipulation, and therefore subversion of logic or security controls
- Code injection, such as Cross Site Scripting, SQL Injection and Operating System command injection attacks (OWASP – 4 and 6)
- Legacy C/C++ vulnerability classes, such as buffer overflows, integer wrap and format string vulnerabilities

In this paper, we have focused the security vulnerabilities that result at the application level. So it is important to distinguish between the vulnerabilities at the network level and at the application level. But some web organizations use firewalls to solve this problem. It should be noted that the firewalls are necessary, but they are not sufficient to ensure data integrity at the application level because fundamentally they are designed to provide protection at the host and network levels. Therefore, they are useless if any malicious script or listener is already installed on a server behind them because they do not prevent the installation of scripts at the application level. Previously, a firewall could suppose that an adversary could only be on the outside. Currently, with mobile environment, an attack might originate from the inside as well, where a firewall can offer no protection [20], [22].

In attempt to solve the poor data validation in web applications, important steps should be taken to ensure that the application processes data in a secure manner. A number of approaches can be adopted when implementing data validation mechanisms within an application, each with its own advantages and disadvantages.

In a web application security, data validation is the process of ensuring that a web application operates on clean, correct and useful data [7]. It uses validation rules that check for correctness, meaningfulness, and security of data that are input to the web-based system. The rules may be implemented through the automated facilities of a data dictionary, or by the inclusion of explicit program validation logic.

This follows two main principles of data validation [8], [12], [23]:

1. Data should be validated in the data model, where the validation rules have maximum scope for interpreting the context; and
2. Escaping of harmful meta-characters should be performed just before the data is processed, typically in the data access components.

A data validation scheme is the first defense against web attacks at the application level. Web developers have adopted a number of standard validation techniques to prevent loss of data integrity. Further details about these approaches are described in our earlier work ([37]). In addition, the following approaches are led to ease to prevent the web application vulnerabilities:

- Server-side input validation [13], [25]: 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.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

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.

- Client-side input validation [13], [24]: 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.

- The double-checking input validation [13], [26]: 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.

- Honkala and Vuorimaa [16] propose extending the XForm [14] 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 [14] is a new standard for better graphical interfaces and specified to input validation rather than the embedded scripts [24]. However, XForm is only supported by the <XSmile> browser.

- Brabrand and others [13], [26] introduce a PowerForms form for input validation. PowerForms form is implemented by the XML language to define a rich form that includes input validations without using a scripting language. PowerForms is only supported by the <bigwig> browser. However, unlike Firefox and IE, this browser is not free. In addition, it requires additional installations to interpret PowerForms forms on a web server because the MAWAL language is used in the <bigwig> system for domain of interactive web services. This language is designed to protect the client sessions as extended to SSL technology.

- Formatta organization (Site 2) 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 [11].

The existing validation tools and systems are not sufficient to solve the poor validation issue for the following reasons: (i) some validation tools improve the client validation modules by an encryption approach (such as Client-side encryption approach [10]); however, the data can be bypassed because moving the encryption to a client-side is vulnerable to security risks. The encryption can be exposed through applying penetration strategies such as reverse engineering techniques. (ii) Some validation tools mitigate threats to web application at the client-side for offering a protection in case of suspected XSS attacks that attempt to steal credentials of users. However, the server-side solution (such as Pixy, WAVES, and Saner) have the advantage of being able to discover a larger range of vulnerabilities, and the benefit of a security flaw by the service provider is propagated to all its clients. The server-side techniques can be further classified into dynamic and static approaches. Dynamic tools try to detect attacks while executing the audited program, whereas static analyzers scan the source code of web application for security vulnerabilities. (iii) The detection of web attacks is not sufficient.

Therefore, we will present a Semantic Data Validation (SDV) 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 SDV service is a survivable solution that strategically combines elements of detection and adaptive reaction in the architecture of the system. In addition, The SDV service takes advantages of the client and server - side solutions. Furthermore, this new service could be able to detect the known and unknown attacks.

The rest of the paper is organized as follows: Section 2 discusses SQL injection which is one of the main application-level attacks. The recent existing data validation approaches and tools are made in Section 3. In section 4 we present a comparison of the different solutions based on the criteria such as technology, type of data, and feature and classifications established in Sections 2 and 3. The proposed semantic architecture is described, and a case study is presented in Section 5. The prototype implementation of this service and the system evaluation are discussed in Section 6. Finally, conclusions and future work are offered in Section 7.

2 SQL Injection

An input validation scheme is necessary for both client and server-sides. However, it is not sufficient to protect data integrity, because fundamentally data validation scheme is designed to validate basic properties of the input data: length, range, format, default value, and type. In addition, data validation can be used to enhance resistance to injection attacks such as SQL injection attack because SQL injection vulnerabilities result from insufficient data (input) validation [21]. However, the client validation modules do not make sense if any malicious script is installed on a server [9], [10], [13].

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

One of the approaches can cause loss of content integrity at the (X) HTML form level is a 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 [9], [11]. 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.

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 [17].

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. (X) 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 [17]:

```
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 Related Work

In this section, we now discuss five more recent approaches that attempt to address the lack or bypassing of data validation problems. The current approaches and systems include application-level gateway approach, client-side encryption approach, WAVES approach, Pixy, and Saner.

Scott and Sharp [10] have 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). 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 sent to a security gateway on a server. The gateway analyzed the request and augments it with a Message Authentication Code (MAC1). Mac is used to protect the data integrity. For example, The MAC is used to secure session information in a cookie at the client-side. There are many ways to compute a MAC such as one-way hash algorithms (MD5, SHA-1) to create an unique fingerprint for data within the cookie.

However, this approach has a number of limitations. One of these is that data tampering is still a potential problem because dynamic data that is generated on fly is not validated. Furthermore, the policies of validation constraints and transformation rules are enforced manually and need an engineer to write and check them by hand. Further limitations of this approach are that it is difficult to define all the policies and rules of a legacy web application for every single data entry point, URL entry, and cookie unit because these can consist of complex structures of multiple programming languages and imported binary components with little or no documentation. Therefore, it is not practical for a web administrator or publisher to be familiar with all of the data entry points for the existing web applications.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

Hassinen and Mussalo [15] have proposed a client-side encryption system to protect confidentiality, data integrity, and user trust. They encrypt data inputs using a client encryption key before submitting the content of an (X) HTML Form. The client encryption key is stored on the server and transferred over an HTTP connection. A downloaded web page includes a signed Applet which handles the encryption and decryption values. This applet also reads the encryption key from a local file. In addition, a downloaded web page includes JavaScript methods that invoke the Applet methods for encryption and decryption. This approach uses a one-way hash function. It computes the hash value which is inserted into the main data input before encryption. After a new request, the JavaScript function invokes the Applet decryption method to decrypt the parameter value and places the returned value in the corresponding input field. The message validation includes finding a new hash value from the decrypted message and comparing it to the hash value which is received with the message. If they are the same, the data is accepted, otherwise, the data is deemed to have been altered and the validation will fail. This system has two main requirements [15]:

1. It must be able to run on any major web browser

2. Without the need to install additional hardware or software

However, data integrity could be lost if this approach is adopted because the Java Applets can access the client's local file system. Thus, a criminal can replace the original signed Applet with a faked Applet to access the client's web content. Moreover, if a smart card is used to store the client encryption key, then the client machine requires a card reader and necessary drivers, which fails the second of the above requirements. This encryption system can use the Java Card technology for two aims:

1. storing the generated key on the card so that it can be read only with a PIN to control authorization of the user

2. storing the Applets that executed on the card

Another potential weakness is the potential loss of the client's smart card with its Personal Identification Number (PIN), in which case the whole web-based system would be compromised. In addition, the Applet and JavaScript methods can be bypassed. If this happens, the submitted values will be in plain text. Furthermore, in order to implement this technique, existing web applications will require modification. Finally, it does not include a risk analyzer to protect user information on the web server if the data validation at the client-side has been bypassed.

Another different approach to make self-protection, Huang et al. [17] have used behavior monitoring to detect malicious content before it reaches users. They have developed 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.

Huang et al. [17] were the first who attempt to address data validation issue in the context of PHP (Site 3) applications. They used a lattice-based analysis algorithm derived from type systems and type-state systems. The type of analysis in this algorithm is static analysis. WAVES approach is targeted for mitigating threats to web application at the server-side. Therefore, because WAVES is a server-side solution, it has the advantage of being able to discover a larger range of security vulnerabilities, and the benefit of a security flaw by the service provider is propagated to all its clients.

Jovanovic et al. [28] 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. They adopted PHP as target language since it is commonly used for developing web applications [29] and a substantial number of security advisories refer to PHP programs [30]. 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 [31] have presented an 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.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

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.

4 Comparisons and Discussions

In this section we have compared the data validation solutions from several points of view: based on type of data, technologies implied, and features supported. The most important comparison criterion between different input validation solutions is whether the web application vulnerabilities can be prevented without modification on the structure of the existing web applications. In order to satisfy this criterion, we have to accomplish with other smaller criteria. These are:

1. The solution should be based on dynamic analysis for every HTTP request and HTTP response from/to a web server and a client.

2. The solution should take into account the client and server validation routines for every (X)HTML form hosted on a web server.

3. The solution should protect the dynamic web pages that are generated on the fly.

4. The solution should be based on a semantic technologies to meet the existing web application requirements rather than syntax technologies.

5. The solution should ensure the survivability of a web system in case of any illegal operation on a backend database using provision of a recovery.

Table 1: Type of data-based comparison

Type of Data → / Solutions ↓	Static user data	Dynamic data	Cookies	Session values
Client-side validation [13]	X			
Server-side validation [24]	X			
Double checking validation [25]	X			
XForms [16]	X			
Formmata approach [11]	X			
Client-side Encryption Approach [15]	X		X	X
Application-Level Gateway [10]	X		X	X
PowerForms [13]	X			
WAVES [17]	X			
Pixy [28]	X			
Saner [31]	X			

As shown in Table 1, we have classified the current solutions in relations to the type of data that can be solved. All current solution supports static data validation either on the client side or server side.

Neither of the solutions is able to address the validation of dynamic data. The dynamic data that generated on the fly is a real-time issue because the generation of dynamic data depends on user interaction [10], [27]. Different user information leads to different generated web content. Therefore, it is very difficult to analyze automatically and even manually the requested page of dynamic data before processing on a web server. The dynamic data of server programming languages needs to be processed on a web server before returning the response to a web browser. As a result, we cannot guarantee that dynamic data is not tampered with even if the static data is validated; therefore the generated data should also be validated.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

In Table 2 we can notice an interesting summary based on the different technologies implied in each solution. On the one hand, there is only one solution that provides cryptography and HMAC technologies that is Application-Level Gateway. On the other hand, there is only one solution that supports smart card, and there are only three solutions that are based on XML technology, that is Application-Level Gateway, PowerForms, and XForms.

Table 2: Technologies-based comparison

Technology → / Solutions ↓	(X) HTML	XML	Smart Card	Encryption	HMAC
Client-side validation [13]	X				
Server-side validation [24]	X				
Double checking validation [25]	X				
XForms [16]		X			
Client-side Encryption Approach [15]	X		X	X	
Application-Level Gateway [10]	X	X		X	X
PowerForms [13]		X			
WAVES [17]	X				

As illustration in Table 3, the solutions are classified according to the features. First, PowerForms, and XForms require special web browser, and the Formmata Form is the only solution that can serve the information offline. This means, the user fills his/her information on Forrmata Form and this form is sent via email. Saner tool only is the solution that uses the static analysis and dynamic analysis. Finally, client-side data validation enables the code transparency. This means that the code can be viewed on a web browser, and hence, this could be vulnerable to security risks on web system.

Table 3: Features-based comparison

Feature → / Solutions ↓	Online	Free Browsers	Static Analysis	Dynamic analysis	Code Transparency
Client-side validation [13]	X	X			X
Server-side validation [24]	X	X			
Double checking validation [25]	X	X			X
XForms [16]	X				
Formmata approach [15]					
Client-side Encryption Approach [15]	X	X			X
Application-Level Gateway [10]	X	X			X
PowerForms [13]	X				
WAVES [17]			X		n/a
Pixy [28]			X		n/a
Saner [31]			X	X	n/a

From our point of view, we recommend that these validation modules be operated on the client and server because if the client validation modules are subverted, the server validation modules can still work. Some developers suppose that if user information has been properly validated, the static data will be secure. However, malicious code might be installed on a server either inside or outside organization. Furthermore, as seen above, the data validation modules can be bypassed.

Another point is that even if the client validation modules are enhanced by an encryption approach (such as Client-side encryption approach), the data can be bypassed because moving the encryption to a client-side is vulnerable to security risks. The encryption can be exposed through applying penetration strategies such as reverse engineering techniques.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

It should be noted the client-side solutions (such as Client-side Encryption Approach) that target for mitigating threats to web application at the client-side could offer a protection in case of suspected XSS attacks that attempt to steal credentials of users. However, the server-side solution (such as Pixy, WAVES, and Saner) have the advantage of being able to discover a larger range of vulnerabilities, and the benefit of a security flaw by the service provider is propagated to all its clients. The server-side techniques can be further classified into dynamic and static approaches. Dynamic tools try to detect attacks while executing the audited program, whereas static analyzers scan the source code of web application for security vulnerabilities. Therefore, our point of view is to design a new approach takes of advantages the client and server sides solutions. This new approach could be able to detect the known and unknown attacks.

Furthermore, the detection itself is not enough, and hence we recommend designing survivable solution. The high level strategy underlying for any possible solution to designing a survivable system is to strategically combine elements of detection and adaptive reaction of in the architecture of the system. Individually, each element is not sufficient to secure the system from any possible damage, such that:

1. Detection provides awareness to the status of the system and allows the system to detect attack.
2. Adaptive reaction enables the system to cope with undesirable modifications caused by adversaries through supporting recovery

5 Solution Methodology: Design and Case Study

After reviewing the existing validation tools including the process, architecture, strengths, weakness and limitations, a novel validation service which is based upon semantic web technologies, called Semantic Data Validation (SDV) service to prevent the security vulnerabilities at the application level and to secure the web system even if the input validation modules are bypassed. Our architecture consists of the following components: RDFa annotation for elements of web pages, interceptor, RDF extractor, RDF parser, and data validator. Therefore, in the next section, we will describe the functional overview of the proposed solution.

As illustrated in Figure 1, 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. It should be noted that the functional overview is mentioned in our work [37]. The SDV service has a number of advantages over other existing solutions as follows:

- It is the first solution that provides a semantic component to process the data validation.
- This solution has intercepting interface to manage all HTTP requests and responses.
- It does not require modifications to existing web application architectures.
- It does not require any additional changes on the client and server.
- It is compatible with all major web browsers.
- It does not rely on a database of known web attacks.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

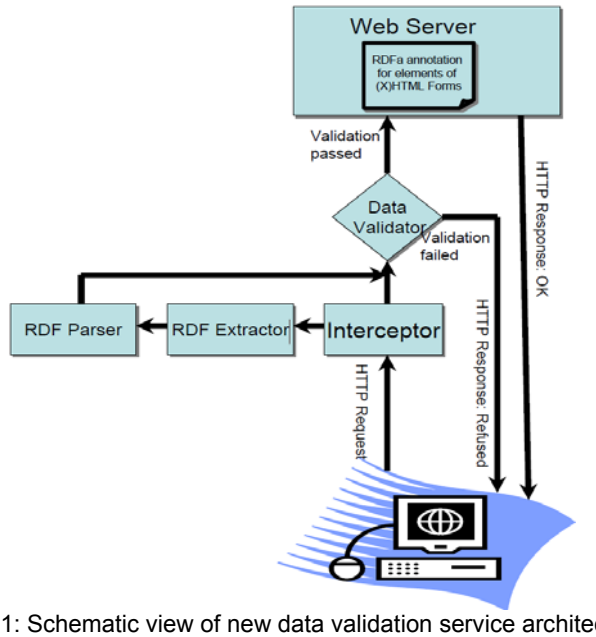

Figure 1: Schematic view of new data validation service architecture.

5.1 Overview of the SDV Framework Architecture

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

1. RDFa annotation for elements of web pages: The World Wide Web (or simply the web) has become the first source of knowledge for all life domains. It can be seen as an extensive information system that allows changing the resources as well as documents. The semantic web is an evolving extension of the Web aiming at giving well defined form and semantics to the web resources (e.g. content of an HTML web page) [36].

   RDF (**R**esource **D**escription **F**ramework) is a knowledge representation language dedicated to the annotation of resources within the Semantic Web [33]. The atomic constructs of RDF are statements, which are triples (subject, predicate, object) consisting of the resource (the subject) being described, a property (the predicate), and a property value (the object). The subject, object and the predicates can be identified by URIs. An URI (Uniform Resource Identifier [32]) generalizes URL (Uniform Resource Locater) for identifying not only web pages but any resource (human, book, or an author property).

   Example 1: The following set of triples represents an RDF graph:

```
{
       <ex:person1 foaf:name "Faisal" >,
       < ex:person1 ex:daughter ex:person2,
       < ex:person2 ex:friend ex:person3>,
       < ex:person3 foaf:knows ex:person1>,
       < ex:person3 foaf:name “Sara”>>
}
```

   Intuitively, this graph means that there exists an entity identified by the URI “ex:person1” named (foaf:name) "Faisal" that has a daughter (ex:daughter) identified by “ex:person2” that has a friend with another entity identified by “ex:person3” whose name is “Sara”, and that knows (foaf:knows) the entity named "Faisal".

   A collection of RDF statements (RDF triples) can be intuitively understood as a directed labeled graph: resources are nodes and statements are arcs connecting the nodes (from the subject node to the object node). For example, the above triples can be represented graphically as the graph shown in Figure 2.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

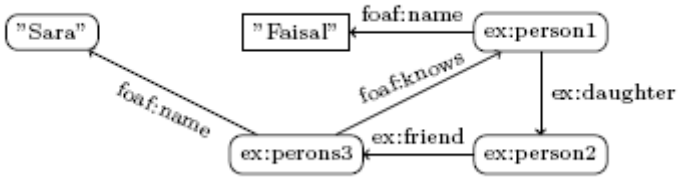

Figure 2: An RDF graph

Currently, many documents are annotated via RDF due to its simple data model and its formal semantics. In addition, RDF can be expressed in a variety of formats including RDF/XML [34], Turtle [35], or n3. It can also be embedded in (X)HTML web pages using the RDF annotations (abbreviated RDFa). Section 3.3 provides an illustration of how to use RDFa to annotate an (X)HTML web page.

2. HTTP 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 (Site 4) is used to extract the RDFa annotations from the (X)HTML web page. For example, it is used to construct the RDF ontology given in Figure 5. In the RDFa Distiller, a mismatched tag or missing quotation mark causes unexplained failure. So that, the user-friendly W3 Validator service is used, and hence, at validator.w3.org, which reports some missing tags and also to save code as .xhtml. Typically, RDFa distiller was designed as a check of the syntax specification, not as a user tool. It should be noted that the Distiller has some caching issues.

4. RDF parser: parses the form inputs and their attributes for validation process. This parser is designed to run in the Interceptor interface, allowing processing RDF on the HTTP response which is sent to a client. Typically, RDF Parser reads RDF files and constructs a graph structure of the RDF ontology extracted in the previous step.

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.

5.2 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 HTML form requesting their name, address, department, and qualification. The (X)HTML form is shown in Figure 3.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

Figure 3: Snapshot of 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>
<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>
```

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

Figure 4: Snapshot of the modified (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>
```

The modified (X)HTML form together with the ontology description are shown in Figure 4. This ontology means that someone exists 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". If the RDFa distiller is used to extract the RDFa annotations from the (X)HTML form page, then we have the RDF ontology shown in Figure 5:

Figure 5: Snapshot of the employee RDF ontology

```
# 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:member 'WORKDEPT' .
_:someEmployee foaf:title 'EDLEVEL' .
_:someEmployee vcard:ADD 'address' .
```

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

6 Implementation of the SDV Service and Evaluation

The Semantic Data Validation (SDV) service is implemented in Java 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. As far as performance will be concerned, the prototype of SDV service could be able to prevent infinite number of application attacks.

The prototype implementation of this service consists of three major components: HTTP Interceptor, RDF parser, and Data validator. First, 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. The HTTP Interceptor is a 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.

Second, the RDF parser is used 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 Data validator mechanism.

The third component is the Data validator. When the description is extracted using RDFa extractor, the Data validator takes the user inputs for validation process as shown in the previous section. If the integrity check passes, the data is sent to the running process straight away. If it fails, it is refused the user request.

To test our approach, we implement the SDV to verify integrity of inputs of real-world web applications. We conducted a pilot study to test the reliability and performance of SDV and the results obtained appear to demonstrate that the overhead time for the validation process and recovery process are relatively low and that the SDV can efficiently and correctly determine if data input has been compromised. In order to evaluate the security objective, Experiment 1 has been performed to investigate the data validation process on Apache Tomcat web server. To evaluate the effectiveness of the NVDS system, Experiment 2 has been performed to measure end-to-end performance for a web system.

6.1 Case Study - Security Objective

The security objective is difficult to measure. There is no Standard accepted methodology by which to evaluate it. To assess how successful the SDV is able to detect and recover from the web application vulnerabilities, we started to use an ontology to describe all data elements in a web application using RDFa annotation. This annotation takes for every (X) HTML form stored in the designated directories of the suggested web sites/applications hosted on Tomcat and web server. Thus, over 35 attacks (such as buffer overflow, spoofing attack, tampering manipulation, URL parameters, XSS injections, and SQL injections) were performed against the target web application. These attacks exploited different types of vulnerabilities that allowed for the modification of files in the designated directories of a web Server. These types of attacks are occurred in the real-life e-commerce systems because Computer Emergency Report Team (CERT) in Carnage Melon University has reported these types of attacks from several organizations. Most of existing solutions focus only on detection of SQL injection and buffer overflow. In this paper, we have applied the SDV service to prevent different attacks such as XXS and SQL injections.

The HTTP Interceptor intercepts HTTP request, checks the availability of HTTP request on the designated directories of web server, and invokes the RDF extractor and then data validator to compare between the original one and incorrect one during online transactions (when a user request a web resource). Over 35 web application attacks were launched against the designated directories of the suggested web sites hosted on Tomcat web server.

The SDV uses the validation process to detect whether malicious software has replaced the original data input. This means, the validation process checks to see if the data input has been modified since it was used. Based on whether the test passes or fails, the HTTP Interceptor takes the decision about the next step in the process. If the integrity check passes, the (X) HTML form is sent to the running process straight away. If it fails, it is sent to the recovery component.

The SDV validator component offers the following validation controls and features:

- Required Field Control: is used to check that a field contains a value.
- Regular Expression Control: is used to match a value against a regular expression.
- Compare Control: is used to permit two values to be compared to each other.
- String Range Control: is used to check whether a value is within a given range.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

- Length Control: is used to check the length of value input.
- Number Range Control: is used to specify a range of numbers.
- Meta Control: is used to escape meta-characters that have specific meanings in the user inputs.
- Message Authentication Code (MAC) control: is used to secure the hidden inputs in the (X)HTML Form.

6.2 End-to-End Performance Evaluation

The testing environment of Experiment 2 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.

A load test can be used to test an application's robustness and performance, as well as its hardware and bandwidth capacities. In these experiments, we used the Neoload application which is a stress and load testing tool to (i) test a web site's vulnerability to crashing under load and (ii) check response times under the predicted load.

We have measured the end-to-end performance with the (i) implemented SDV service, (ii) and the traditional validation routines. The duration of the test was dedicated by the requirements of the Neoload testing. The run-time policy was ramped-up (i.e. generating a number of virtual users that increases throughout the test until it reaches the specified maximum) from 2 users (initial user number at the same time) adding 2 users every 2 minutes. This is useful for checking the server's behavior under an increasing load. The virtual users were connecting at 100Mbps through a local network. Note that, due to the lack of existing approach's codes, this comparison only includes the SDV service and the traditional validations routines which we implemented by ourselves.

All these measurements were performed from the client point of view (i.e. all durations show the time between a request and the reception of its answer). Each row in the below tables displays the average response time (request), maximum response time (request), and minimum response time (request) in seconds, of all requests during the test and average page response time for all pages where each page may contain a number of requests. The average response time is the meantime necessary to process a request by each web server when the proxy, browser, and the SDV service are active. It should be noted that the communications (network response time) are parts of the measured times. In this case study, 5421 hits were created, 5397 web pages were served and 1.55MBs (total throughput) were received by client. The number of virtual users launched was between 198 and 204.

Table 4: Comparison test of the response time (in seconds) between the SVD service and the traditional validation routines of all requests conducted on a Tomcat web server.

System	Minimum Response Time (request)	Average Response Time (request)	Maximum Response Time (request)	Average Page Response Time
Traditional Validation routines	0.029	3.14	4.86	3.24
SVD service	0.219	3.95	6.01	3.97

The results indicate that the average response time when using the traditional validation routines is better than when using the SDV. Table 4 illustrates that the average response time (request) through SDV service was 3.95 seconds on Tomcat. In Scenario A (when using traditional validation routines), the average response time (request) was 3.14 seconds on Tomcat. Therefore, the SDV service satisfies the performance objective for validation of data inputs.

The graph in Figure 6 represents the curve of response times when using traditional validation routines on Tomcat. As shown in this graph, the maximum response time was 4.86 seconds with the variation of response times was relatively low. The graph in Figure 7 shows the maximum response time through SDV service was 6.01 seconds on Tomcat. The response times do not vary widely. As SDV service was disconnected, the actual test through the SDV service started from the 4th minute. This kind of error would tend to increase the average response time.

Journal of Theoretical and Applied Electronic Commerce Research
ISSN 0718–1876 Electronic Version
VOL 5 / ISSUE 1 / APRIL 2010 / 39-55

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

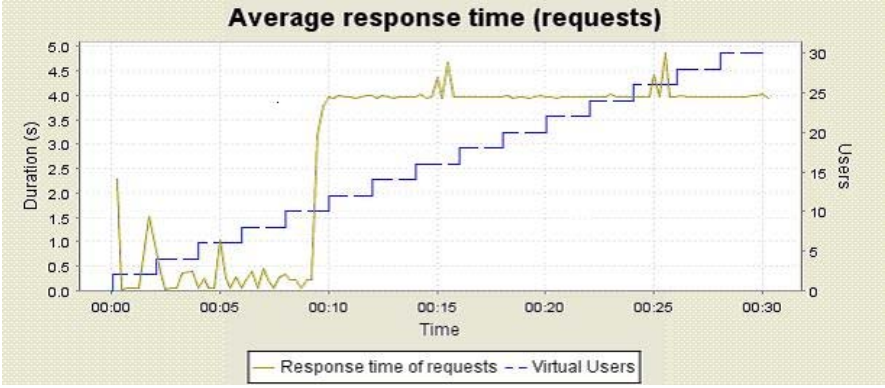

Figure 6: Graph for response time curves when using traditional validation, in seconds, of all requests during the test.

However, some common bottlenecks (such as network bandwidth, CPU, memory and I/O) occurred in our experimental studies, this led to an increase in the overhead of our proposed system. If the CPU utilization is over 80%, it will increase exponentially rather than linearly. In addition, our experimental results indicate that the importance of memory: the memory in our experiments was insufficient and this led to the disconnection of the Tomcat web server.

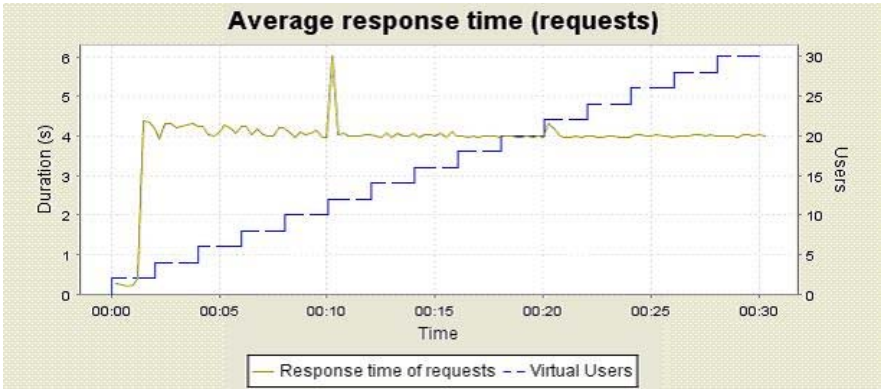

Figure 7: Graph for response time curves when using SDV, in seconds, of all requests during the test.

7 Conclusions and Future Work

In this paper, we have reviewed the existing data validation approaches and considered their strengths, weaknesses, and limitations. In addition, we have compared among the existing approaches, systems and tools. Our findings in the literature review are that the existing validation solutions cannot provide a systemic validation for static and dynamic data. In addition, some tools enhance client validation modules by an encryption approach; however, the data can be bypassed because moving the encryption to a client-side is vulnerable to security risks. The encryption can be exposed through applying penetration strategies such as reverse engineering techniques. Furthermore, most of current tools only offer detection of web attacks.

Therefore, we have developed a Semantic Data Validation (SDV) service 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 SDV is a survivable (i.e. SDV provides a continued reliable and correct services to internal and external users, even though a web data manipulation problem may have occurred) solution that combines elements of detection and adaptive reaction of in the architecture of the system. In addition, The SDV service takes advantages of the client and server - side solutions. Furthermore, this new service could be able to detect the known and unknown attacks.

The SDV 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. Two experimental studies are presented to evaluate the reliability and performance of the developed SDV service: study - security objective (detection and recovery), case study to measure the end-to-end performance through (i) the SDV system, and (ii) with using the standard validation modules.

The results from a series of experimental test studies appear to suggest that the SDV service can satisfy the performance objective and it can be suggested that the SDV provides a detection, and prevention of most web application attacks. We exploited different types of vulnerabilities that allowed for the modification of files in the designated directories of a web server. During testing, all the attacks launched against the web servers were

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

detected and recovered by the SDV service. In future work, we are intended to optimize the implementation of our solution to increase the effectiveness and performance. We believe that the overhead of SDV can be significantly decreased, if the server technical speculation follows the current industry standards. Furthermore, we will also investigate a number of experiments for security and performance objectives.

Websites List

Site 1: The Open Web Application Security Project (OWASP)
http://www.owasp.org/index.php/Category:Vulnerability

Site 2: Formatta organization
www.formatta.com

Site 3: PHP
http://www.php.net

Site 4: RDFa distiller
www.w3.org/2007/08/pyRDFa/

References

[1] CERT. (2009, August) CERT Statistics (Historical). [Online]. Available: http://www.cert.org/stats.
[2] W. B. Glisson and R. Welland, Web development evolution: The assimilation of web engineering security, in Proceedings of the Third Latin American Web Congres, Washington, DC, USA, IEEE Computer Society, 2005, pp. 49-55.
[3] Acunetix. (2007, March) Web applications: What are they? What of them? [Online]. Available: http://www.acunetix.com/websitesecurity/web-applications.htm.
[4] R. Cardone, D. Soroker, and A. Tiwari, Using XForms to simplify web programming, in Proceedings of the 14th international conference on World Wide Web, New York, NY, USA, ACM Press, 2005, pp. 215–224.
[5] T. Bass, CEP and SOA: An open event-driven architecture for risk management, in Proceedings of IT Financial Services, Portugal, 2007.
[6] M. S. Lam, M. Martin, B. Livshits, and J. Whaley, Securing web applications with static and dynamic information flow tracking, in Proceedings of the 2008 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation, San Francisco, California, USA, 2008, pp. 3-12.
[7] L. Mocean, Internet Data Validation, Economy Informatics, vol. 1, no. 1, pp. 96-99, 2007.
[8] Stephen de Vries. (2006, January) A modular approach to data validation in web applications. [Online]. Available: http://www.net-security.org/dl/articles/A_Modular_Approach_to_Data_Validation.pdf.
[9] J. Offutt, Y. Wu, X. Du, and H. Huang, Bypass testing of web applications, in Proceedings of th15th International Symposium on Software Reliability Engineering (ISSRE' 2004), Los Alamitos, CA, IEEE Computer Society, 2004, pp. 187-197.
[10] D. Scott, and R. Sharp, Specifying and enforcing application-level web security policies, IEEE Knowledge Data Engineering, vol. 15, no. 4, pp. 771–783, 2003.
[11] S. Sedaghat, J. Pieprzyk, and E. Vossough, On the web content integrity check boosts users' confidence, ACM Communication, vol. 45, no. 11, pp. 33–37, 2002.
[12] L. D. Stein, Web Security: A Step-by-Step Reference Guide. Addison-Wesley, 1998.
[13] C. Brabrand, A. Moller, M. Ricky, and M. I. Schwartzbach, PowerForms: Declarative client-side form field validation, World Wide Web Journal, vol. 3, no. 4, pp. 205–314, 2000.
[14] R. Cardone, D. Soroker, and A. Tiwari, Using XForms to simplify web programming, in Proceedings of the 14th international conference on World Wide Web, New York, NY, USA, ACM Press, 2005, pp. 215–224.
[15] M. Hassinen, and P. Mussalo, Client controlled security for web applications, in Proceedings of the IEEE Conference on Local Computer Networks 30th Anniversary, Australia, IEEE Computer Society Press, 2005, pp. 810–816.
[16] H. Mikko, and P. Vuorimaa, Secure web forms with client-side signatures, in Proceedings of the 5th International Conference on Web Engineering, Sydney, Australia, Springer, 2005, pp 340–351.
[17] Y. Huang, S. Huang, T. Lin, and C. Tsai, Web application security assessment by fault injection and behavior monitoring, in Proceedings of the 12th international conference on World Wide Web, New York, NY, USA, ACM Press , 2003, pp. 148–159.
[18] IBM. (2008, April) IBM Internet Security Systems X-Force: Threat Insight Quarterly. [Online]. Available: http://www-935.ibm.com/services/us/iss/pdf/xftiq_09q1.pdf.
[19] J. Tzay, J. Huang, F. Wang, and W. Chu, Constructing an Object-Oriented Architecture for Web Application Testing, Journal of Computing and Information Science in Engineering, vol. 18, no. 1, pp. 59-84, 2002.
[20] D. Wallach. (1999) A New Approach to Mobile Code Security, PhD thesis, Princeton University, Department of Computer Science. [Online]. Available: http://citeseer.ist.psu.edu/wallach99new.
[21] W. G. Halfond, and A. Orso, Preventing SQL injection attacks using AMNESIA, in Proceedings of the 28th international Conference on Software Engineering, Shanghai, China, ACM, 2006, pp. 795-798.

This paper is available online at
www.jtaer.com
DOI: 10.4067/S0718-18762010000100005

[22] A. Sengupta, C. Mazumdar, and M. S. Barik, e-Commerce security - a life cycle approach. In Sadhana Journal, vol. 30, number 2-3, pp.119–140, 2005.
[23] J. S. Park and R. S. Sandhu, Secure cookies on the web, IEEE Internet Computing, vol. 4, no. 4, pp. 36–44, 2000.
[24] J. Wusteman, Web forms: the next generation, Library Hi Tech, vol. 21, no. 3, pp. 367–381, 2003.
[25] A. K. Ghosh and T. M. Swaminatha, Software security and privacy risks in mobile e-commerce, ACM Communication, vol. 44, no. 2, pp. 51–57, 2001.
[26] C. Brabrand, A. Moller, and M. Schwartzbach, The bigwig project, ACM Transaction Inter. Tech., vol. 2, no. 2, pp. 79–114, 2002.
[27] F. Ricca, and P. Tonella, Analysis and testing of web applications, in Proceedings of the 23rd International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society, 2001, pp. 25–34.
[28] N. Jovanovic, C. Kruegel, and E. Kirda, Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper), in Proceedings of the 2006 IEEE Symposium on Security and Privacy, Washington, DC, IEEE Computer Society, 2006, pp. 258-263.
[29] S. S. Andreessen. (2005, October) PHP succeeding where Java is not. [Online]. Available:http://www.zdnet.com.au.
[30] BugTraq. (2005, March) BugTraq Mailing List Archive. [Online]. Available: http://www.securityfocus.com/archive/1.
[31] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications, in Proceedings of the 2008 IEEE Symposium on Security and Privacy, Washington, DC, IEEE Computer Society, 2008, pp. 387-401.
[32] T. Berners-Lee, R. Fielding, and L. Masinter. (1998, August) Uniform resource identifiers (URI): Generic syntax. RFC 2396. [Online]. Available: http://www.ietf.org/rfc/rfc2396.txt.
[33] F. Manola and E. Miller. (2004, February) RDF primer. Recommendation, W3C. [Online]. Available: http://www.w3.org/TR/REC-rdf-syntax/.
[34] D. Beckett. (2004, February) RDF/XML syntax specification (revised). Recommendation, W3C. [Online]. Available: http://www.w3.org/.
[35] D. Beckett, Turtle - terse RDF triple language, Hewlett-Packard, Bristol, UK, Technical report, 2006.
[36] T. Berners-Lee, James Hendler, and Ora Lassila. (2001, May) The semantic web. [Online]. Available: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.
[37] S. Aljawarneh, and F. Alkhateeb, A semantic web technology-based architecture for new server-side data validation in web applications, in Proceedings of International Conference on Information Technology, Amman, Jordan, 2009, pp. 155-160.