A Validation Model of Data Input for Web Services

Rafael Bosse Brinhosa, Carla Merkle Westphall, Carlos Becker Westphall, Daniel Ricardo dos Santos, Fabio Grezele
Post Graduate Program in Computer Science
Federal University of Santa Catarina
Florianópolis, Brazil
{brinhosa,carlamw,westphal,danielrs,fgrezele}@inf.ufsc.br

Abstract— **Web services inherited many well-known security problems of Web applications and brought new ones. Major data breaches today are consequences of bad input validation at the application level. This paper presents a way to implement an input validation model for Web services which can be used to prevent cross-site scripting and SQL injection through the use of predefined models which specify valid inputs. The proposed WSIVM (Web Services Input Validation Model) consists of an XML schema, an XML specification, and a module for performing input validation according to the schema. A case study showing the effectiveness and performance of this mechanism is also presented.**

Keywords—security; Web service; input validation; SOA

I. INTRODUCTION

Different technologies for collaboration and information sharing are emerging and therefore new forms of interaction are evolving and creating new requirements for the development of distributed applications. Enterprises are experiencing increased collaboration and information sharing and a greater need for the use of distributed and computational resources [1].

The paradigm of *Services Oriented Architecture* (SOA) has transformed the Internet from a data repository to a services repository [2]. In SOA style, an application is composed of reusable services that are integrated through standardized interfaces.

Web services technology based on the use of open standards facilitates information exchange, interoperability, and software reuse, and is therefore considered a major component of SOA. Web services are software components that can be discovered and used to implement applications. Web services are suitable to integrate heterogeneous systems because they make extensive use of XML (*Extensible Markup Language*) [3]. The Web service interface, for example, is described using a language based on XML, called WSDL (*Web Services Description Language*). Furthermore, communication among parts of a distributed application is carried out using SOAP (*Simple Object Access Protocol*) messages which are XML-based.

The Internet makes many Web services available for use: it is possible to obtain information about the weather, stock exchange, and postal codes [4] or to provide information to the federal government [5].

While the SOA paradigm provides cost savings by eliminating redundant efforts through software reuse, security is a major concern according to the Gartner Research Institute [6].

To implement security in Web services, various standards and specifications have been created. However, the correct use of standards alone does not guarantee that the right level of security will be achieved [7]–[9].

A report by the SANS Institute (*SysAdmin, Audit, Network, Security*) [10] lists the major risks to cybersecurity, and the OWASP community (*Open Web Application Security Project*) [11] states that validation of data input can be one of the most effective controls for Web applications security.

The validation of data input [12] [13] is a set of controls that an application should carry out on the lexical and syntactic aspects, type checking, integrity, and origin of data. The lack of these controls has become a major problem for software because interfaces exposed to the Internet could be easily exploited by malicious users.

Thus, in the SOA and Web services environment, improving security mechanisms by the use of more robust data validation has become essential [11] [13].

This paper proposes a model for validation of data input in Web services, providing protection against attacks based on malicious input. The proposed model is called WSIVM (*Web Services Input Validation Model*), and is an input validation mechanism composed of an XML schema, an XML specification, and a validation module.

This paper is organized in the following sections: Section 2 presents the related works, Section 3 describes the security problems of Web services, Section 4 presents the proposed model, WSIVM, Section 5 describes the implementation, a case study is shown in Section 6, and Section 7 presents the conclusions of the paper.

II. RELATED WORK

The lack of input validation is a major cause of Web application attacks [11] [13] [14], whether these applications are developed with Web services or with other technologies. This happens because the lack of input validation of data allows multiple attacks listed in [10] [15] to occur. Among

 ISBN: 978-1-61208-245-5

the attacks that can be cited are the injection of malicious code by use of SQL (*Structured Query Language*) and *cross-site scripting* (XSS), which allows code execution (scripts) in the client-side browser to perform malicious actions [15].

Many studies have been undertaken to ensure input validation in Web applications, such as Microsoft Anti-Cross Site Scripting Library [16] and the use of Open Source solutions in PHP. However, there are few specific mechanisms for Web services.

Regarding the implementation of security mechanisms for Web services, the MIT (Massachusetts Institute of Technology) has an implementation called WS-Security Wrapper [17], which is an intermediate between the Web service and the client that carries out validation of certain aspects. However, this work was developed to be compatible only with Web services developed on the platform Microsoft.Net v1.1 and does not include features such as validation of predefined data entries.

Wu and Hisada [18] have proposed a token based metadata to validate semantic notation built on top of ESB (Enterprise Service Bus). This approach uses a different method for input string validation using the ESB for implementing SOA security.

A reusable and independent mechanism for data input is very important in the process of creating a secure Web service. The mechanism proposed in this paper, WSIVM, assists in this task differently from other studies examined. First, because it focuses on the aspect of handling of data input, it differs from IAPF (Integrated Application and Protocol Framework) [19], which seeks to address all the security aspects related to Web services. Moreover, other works [3] [20] have focused on the use of existing technologies such as XML encryption to ensure the security of Web services but do not mention input validation.

With respect to input validation aspects in Web applications, there are some works such as [21] that have developed tools that automatically insert the input validation on the server side by eliminating malicious insertions vulnerabilities. However, this approach has a disadvantage in that it produces a great many false positives; that is, the validator may fail by considering a message invalid when in fact the message is valid.

Besides the works already listed there is another category of work focused on developing firewalls, like Web Service Firewall Nedgty [22], which deals with protection against denial of service and stack overflow attacks. XML firewalls, presented in [23], are concerned with validation of the structure of XML content but not the content itself. Reference [24] mentions protection against SQL injection through an XML schema and a precompiled blacklist of SQL commands, an approach which tends to produce many false positives; however, details about the effectiveness of this work with more extensive tests are not presented.

Among the related works it can be seen that there is a lack of studies specifically addressing input validation for Web services.

III. SECURITY ISSUES IN WEB SERVICES

Web services create new security risks for organizations because old methods of protection such as firewalls and antivirus applications are not able to protect them. Common firewalls that act in the networking layer allow the normal flow of HTTP (Hypertext Transfer Protocol) requests without blocking these flows, because they are designed to make use of HTTP using port 80.

In addition, the Web services functionality is exposed through WSDL files since from the descriptions of the methods and variables of the WSDL file, important information can be obtained in order to accomplish an attack known as WSDL scanning [1].

There are attacks which are directly related to data manipulation: XSS and SQL injection. Reference [25] classifies two types of XSS attacks: first order and second order.

In a first-order XSS attack, the vulnerability results from the application inserting part of the user input on the page itself. The malicious user uses social engineering to convince the victim to click on a URL that contains malicious HTML/JavaScript code. The user's browser displays the HTML page and runs the JavaScript that was part of a malicious URL received, resulting in the theft of session cookies or other sensitive data from the user. This type of attack can hardly be done against Web services.

In the second-order XSS attacks, vulnerability results from the storage of malicious entries by the user in the application database, and then when the HTML page is accessed, the code runs and is shown to the victims (for example, on social network pages). Second-order attacks are more difficult to avoid because the application needs to validate or sanitize inputs, which may contain executable script code. In the context of Web services, by presenting unvalidated data directly to the user, Web services can be attacked. For example, by making use of AJAX (Asynchronous JavaScript and XML), data Web services provided by third parties that may be contaminated can be obtained. Using, for example, the command `document.write(xmlhttp.responseText)`, if the answer to this AJAX call made to a Web service contains HTML and JavaScript data, these data will be interpreted and executed, posing a risk to the user.

Code injection attacks (SQL injection) work through malicious inputs aimed at the execution of SQL commands in the database [15], [25], [26].

In Web services that do not have proper exception handling, the error message may contain valuable data for the attacker to use. Thus, through trial and error, the attacker can find which database technology is being used, tables that can be explored, and all the necessary information to make

 ISBN: 978-1-61208-245-5

an attack. SQL injection attacks can increase the privileges, and thus it is possible to run in administrator mode on the compromised server. It is possible to test whether a Web service is vulnerable by sending SOAP requests with properly handled parameters. For example, by sending "`'"1=1 –`" as a parameter for a particular service, it is possible to obtain in return the outputs of Figs. 1 and 2. These figures show two examples of responses of error outputs from the database that were returned by the server. These responses can be used to discover details of the database and to send new requests to the database, allowing more details to be acquired, and to carry out more complex commands in the database, which can result in elevation of privilege, injection of files, and theft or destruction of the database.

Through the output response in Fig. 1 it is possible to identify that it is a MySQL database and that SQL command injection was performed, because of the type of error returned. It can be concluded based on the output response in Fig. 2 that the name of one of the columns of the database is `ItemId`, because of the syntax of the SQL `select` command, and it can be deduced that the database is Microsoft SQL Server by observing the syntax of `stored procedure` "`dbo.`".

```
ERROR: The query was not accomplished.
Description: 1064 - You have an error in
your SQL syntax; check the manual that
corresponds to your MySQL server version
for the right syntax to use near '1=1'' at
line 1
```

Figure 1. Example of output response 1.

```
Line 11: Incorrect syntax near ''')) or
ItemId in (select ItemId from
dbo.GetItemParents('4''. Unclosed
quotation mark before the character
string ')) ) ) > 0 '.
```

Figure 2. Example of output response 2.

The *Blind SQL injection* attack is a type of SQL injection in which the results are not displayed to the attacker. It is very commonly used against Web services, because many servers prevent the error messages generated by the service from reaching the user. In Web services an HTTP 500 error is usually returned when an attempt at *blind SQL injection* is performed; however, there are techniques such as measurement of response times of the server that can be used to determine the parameters necessary to perform the attack successfully.

Although it is difficult to obtain reliable information on security incidents and data breaches as reported in the book of Adam Shostack and Andrew Stewart [27], in *Databreaches* table [28] it is possible to observe that the major attacks in cases of data theft that occurred were related with the injection of malicious data. In the table given in [28], for example, when accessing the URL that exists on the date of the incident of the entity Heartland Payment Systems, in which 130 million data were lost, it is possible to read the reason for the incident: SQL injection.

IV. WSIVM – WEB SERVICES INPUT VALIDATION MODEL

In this section, we describe the WSIVM (Web Services Input Validation Model) which is proposed to validate input data to provide security for Web services. Initially the operation of Web services is described without the use of the model. After that the operation of Web services is explained using the proposed model.

A. Operation of Web Services without WSIVM

In the traditional way, a customer finds the Web service he or she needs through research into repositories of Web services. The repositories store the references in the form of Web services in UDDI (Universal Description, Discovery, and Integration) format. UDDI is a standard protocol that specifies a method of publishing and discovering directories of services in a service-oriented architecture.

After the discovery the client sends a request to the Web service. The Web service returns the response to the client containing the result of his or her request.

In this traditional process, the client request is made directly to the Web service, which processes the inputs and returns the result. The standard used for this exchange of messages is the XML-based SOAP format.

We assume that the Web service executes the user's entry without any kind of validation and that the user input is part of the SQL query described in (1).

SELECT name, age FROM clients WHERE name=**input**; (1)

If the user provides as input the name "Paul", which is a valid entry for the name, the Web service returns as answer the name and age of the customer "Paul". However, if the user provides the following malicious input string: "`Paul' UNION SELECT name, password FROM clients; –`", the SQL query would be represented in (2).

SELECT name FROM clients WHERE name='Paul'
UNION SELECT name, password FROM clients; -- (2)

As the Web service does not perform validation of input represented in (2), it returns the secret value of "account password" to the user that sent the malicious request to the Web service. For the WSDL, the request is valid, because it is a string as specified in the service description. However, confidence in user input and lack of validation of this input resulted in a vulnerability that provided sensitive data.

 ISBN: 978-1-61208-245-5

B. *Operation of Web Services with WSIVM*

The model WSIVM (Web Services Input Validation Model) proposes to validate input data to provide security for Web services.

The proposed model has several advantages compared with the input validation normally done in an application, since: (a) it prevents the waste of server processing with invalid messages, (b) it reduces the possibility of denial of service using content of messages, and (c) it is independent of the technology used for the internal development of services.

With WSIVM, the user makes the request in the usual way, however, that request is validated by WSIVM (Fig. 3).

Figure 3. SOAP request with WSIVM.

If a malicious request is sent, as shown in the example represented in the entry (2), instead of sending the password, the WSIVM validation mechanism validates the request and returns a generic error to the user. Thus the Web service does not receive the malicious request (Fig. 4).

Figure 4. WSIVM blocks malicious request.

In more detail, what happens when a message arrives for validation in WSIVM is that the entry submitted by the user is validated through a module that was developed with the XML specification that is on the server.

This avoids unnecessary consumption of server resources, so, considering the example, the Web service does not execute the SQL query if a malicious request is sent.

The interaction of the components of WSIVM is represented in Fig. 5.

The *WSIVMModule* is a module responsible for calling the other components.

The *WSIVMValidator* maps the SOAP message, obtaining the fields of the body of the message, and sends it to the *WSIVMXMLLoader*.

The *WSIVMXMLLoader* loads the elements and the rules specified in the XML specification and checks the validity or invalidity of the response with the *WSIVMVerifier*.

The *WSIVMVerifier* contains all the pre-defined rules for validation of entries and is responsible for validating these entries.

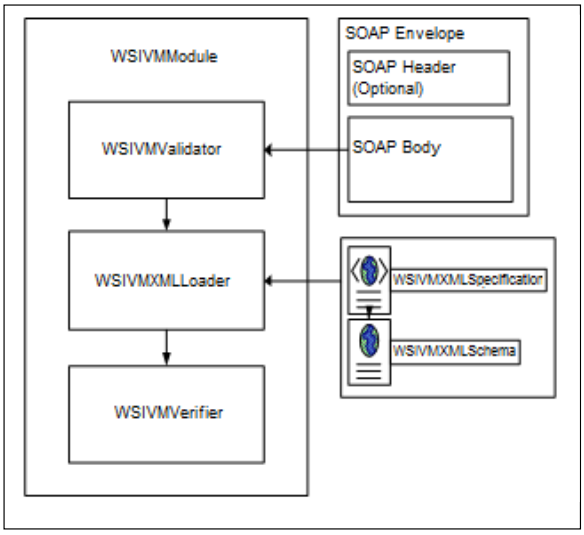

Figure 5. Operation of WSIVM [29].

V. DEVELOPMENT OF THE IMPLEMENTATION

The implementation of the model was developed using the Apache Tomcat Web server and Apache Axis2 framework for SOAP messages [30] (Fig. 6). Apache Axis2 was chosen for the implementation of this work due to its extensibility through modules and the ease of intercepting SOAP messages through the modules.

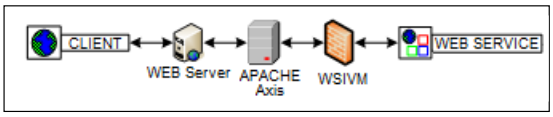

Figure 6. WSIVM.

To implement the validation module for Apache Axis2 the Rampart module was used, which is the mode of extension of Apache Axis2.

The phase of interception can be specified in the file `Module.xml`. It was chosen to intercept the message in the phase `PreDispatch`, which is the phase immediately preceding the sending of the message and its processing by the Web service.

The implementation operation is as follows: the customer, which can be an application, a Web page, or any mechanism capable of communicating with a Web service, sends a message to the Web service. This message passes through the Web server, that is, the Apache Tomcat. The Web server sends this SOAP message to the Apache Axis. The Apache Axis sends the message to be analyzed by WSIVM. After the message is parsed, if it is held to be invalid, an error message is sent to the customer by WSIVM. If it is considered valid, it is usually transmitted for

 ISBN: 978-1-61208-245-5

processing by the Web service, and the result is returned to the client (Fig. 6).

The following is a detailed description of the WSIVM model components: WSIVMXMLSchema, WSIVMXMLSpecification, and WSIVM Rampart module.

WSIVMXMLSchema is the specification of the validation schema of entries. It defines the format of the XML specification and the valid attributes.

WSIVMXMLSpecification is the specification of validation of entries. It specifies valid parameters according to a set of predefined attributes and is used for validating user input. Among the possible parameters are the entries:

- **OperationName:** the name of the operation or function displayed in the Web service referred to in the validation;
- **SanitizeOperation:** defines whether the parameters of this operation or function can be reformulated if necessary for the removal of characters that are not accepted;
- **ParamName:** the name of the parameter or field referred to in the validation;
- **Allowed:** an allowed field type, which is valid (text, html, html+java-script, email, number, and all);
- **Length:** specifies the exact size of the field;
- **Maxsize:** specifies the maximum field size;
- **Minsize:** specifies the minimum field size;
- **Nillable:** determines whether or not it is possible that the field is null (true or false);
- **regEx:** allows a regular expression to be specified for validation.

The *WSIVM Rampart module* is the main component of the mechanism implemented. It is a module for Apache Axis 2, which receives data from the client and validates these data according to XML specification, calling Java classes to perform validations. This module consists of a `wsivm.mar` file that has the following compressed components:

- module.xml: contains a description of the module, the class that will carry out the validation, and the phase in which that validation will occur;
- MANIFEST.MF: a Java manifest file;
- Java classes related to input validation: WSIVMModule, WSIVMValidator, SIVMXMLLoader, and WSIVMVerifier.

VI. CASE STUDY

As a case study, a hypothetical system of registration of students for a university named `UniversityManager` was developed. The system comprises a client application called `ClientManager` and a server (a Web service) named `UniversityManager` (Fig. 7).

```
<?xml version="1.0" encoding="UTF-8"?>
<valid_inputs_specification
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
WebServiceID=" UniversityManager "
xsi:noNamespaceSchemaLocation="valid_inputs_specification.xs
d">
<operation name="registerStudent ">
<input name="name" type="String" min="5" max="20"
accept="text" sanitize="true"/>
<input name="age" type="Integer" min="0" max="150"
accept="number" sanitize="true"/>
<input name="email" type="String" min="0" max="200"
accept="email" sanitize="true"/>
<input name="comment" type="String" min="0" max="200"
accept="text" sanitize="true"/>
<input name="site" type="String" min="0" max="300" accept="url"
sanitize="true"/>
<input name="data" type="String" min="0" max="200"
accept="regex" regexpattern= "(\\d{4})-(\\d{2})-(\\d{2})"
sanitize="true"/>
</operation>
<operation name="searchStudent ">
<input name="id" type="Integer" min="0" max="10000"
accept="number" sanitize="true"/>
</operation>
</valid_inputs_specification>
```

Figure 7. WSIVMXMLSpecification – UniversityManager [29].

For the development of the Web service, Java language was used and performance tests were conducted using the program soapUI [30]. For the development of the Web service `UniversityManager`, a class with the operations `searchStudent` and `registerStudent` and a class to handle the operations of the database were created. This Web service was developed without any input validation in the operations of Java classes, purposely leaving the validation to WSIVM.

The `searchStudent` operation receives a registration number (ID) that must be an integer that is greater than zero and no more than 10000 and returns the student record containing a *String* with his or her information. The `registerStudent` operation receives the information on the student, which must not contain HTML or Javascript code, and registers it on the MySQL database. In the database a `student's` table is created with the following fields: ID (auto-incrementing identifier), name, age, email, comment, site, and birthday.

After creating the Web service and the database, a Java class called `managerTest` was created to test them locally.

 ISBN: 978-1-61208-245-5

To operate the WSIVM, WSIVMXMLSpecification – UniversityManager was specified according to the standard model WSIVMXMLSchema and describes the parameters for validation of entries.

This way, the `Services.xml` file required for Apache Axis 2 was created (Fig. 8).

```
<service>
<parameter name="ServiceClass"
locked="false">example.wsivm.university.Manage
r</parameter>
<operation name="registerStudent">
<messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessa
geReceiver"/>
</operation>
<operation name="searchStudent">
<messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessa
geReceiver"/>
</operation>
<module ref="wsivm"/>
<parameter
name="validationXML">file:///C:/WSIVM/valid_in
puts_specification.xml</parameter>
</service>
```

Figure 8. Services.xml – UniversityManager.

A package named `Gerenciador.aar`, containing the class Manager, MySQL, the MANIFEST.MF descriptor, and the services.xml file in the META-INF folder, was created.

In this experiment, two tests were performed: one using the WSIVM input validation model and the other without using it. The following scenario was configured to perform the tests: 150 users are started gradually with a user booting every 2 seconds. The test runs for 300 seconds (5 minutes). The database is clean in order to analyze the number of operations for registration of students that are carried out successfully.

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-
envelope"
xmlns:univ="http://university.wsivm.example"
>
<soap:Header/><soap:Body>
   <univ:registerStudent>
   <univ:name>John</univ: name >
   <univ:age>12</univ: age >
   <univ:email>john@hsj.com</univ:email>
   <univ:comment>Passed</univ: comment >
<univ:site>http://www.gol.com</univ:site>
<univ:birthday>1980-09-12</univ: birthday >
</univ: registerStudent >
</soap:Body></soap:Envelope>
```

Figure 9. Example of SOAP message sent by soapUI.

SoapUI offers a friendly interface for testing. The tests are performed by making direct calls to the Web service. The SOAP message is sent as shown in the example in Fig. 9.

Fig. 10 shows the graph of results of response times of the tests with and without input validation. The X-axis shows the elapsed time of the test and the Y-axis shows the value of the response time in milliseconds.

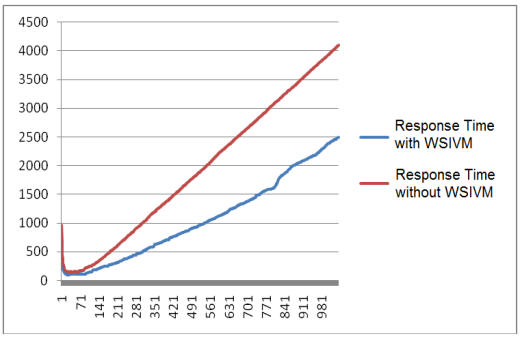

Figure 10. Response times with and without the use of WSIVM.

Fig. 11 shows the graph of the results of throughput tests with and without the WSIVM input validation. The X-axis shows the elapsed time of the test and the Y-axis shows the number of bytes per second (B/s).

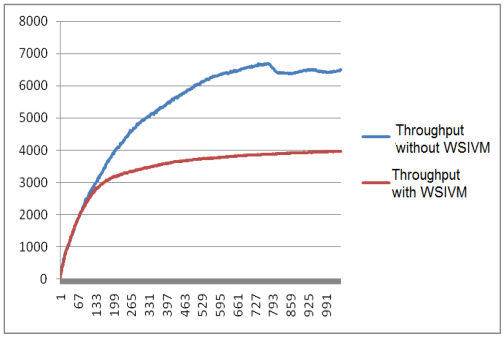

Figure 11. Throughput with and without the WSIVM validation.

It can be observed that the rate of transfer of bytes per second (B/s) or throughput falls considerably with the use of WSIVM.

In Table 1, the calculations that appear in the “Total” row in each of the columns were carried out as follows: the entry in the "Without WSIVM" column is subtracted from that in the "With WSIVM" column and this value is divided by the value of the column "-Without WSIVM".

 ISBN: 978-1-61208-245-5

TABLE I. CONSOLIDATED RESULTS FOR THE CASE STUDY [29].

Comparison	Min. Time	Max. Time	Average Time	Transferred Bytes	Bytes per second (*throughput*)	Insertions in the Database
Without WSIVM	35 ms	27848 ms	2494,85 ms	1974195 B	6506 B/s	10078
With WSIVM	64 ms	13346 ms	4541,24 ms	1236330 B	4012 B/s	5134
Total	83%	-52%	82%	-37%	-38%	-49%

The test results show that when WSIVM is used, a significant increase (82%) in the average response times can be observed, the total throughput decreases by 38%, and the number of students registered in the database decreases by 49% from 10,078 to 5134.

The time spent in the processing of XML messages had an impact on the performance of transactions, which are validated one by one and compared with the rules specified for valid entries. The interpretation of the messages is a costly task in terms of processing and memory requirements, so the validation is done before the processing by the Web service and the return of the response.

There was also a decrease in the total number of bytes transferred because messages took longer to process and therefore a smaller number of messages were processed and the number of responses was lower.

In this case, due to the time required for validation of each message, the application was able to process fewer messages in the same period of time, resulting in fewer insertions of students in the database.

A decrease in performance was expected due to the time required to go through the validation of XML trees in order to validate fields, which is often costly in terms of processing. However, preventing the insertion of invalid data by validating fields can compensate for the loss of performance. This performance loss can be addressed in future work: tests using other mechanisms for interpreting XML files may be carried out as well as tests of the use of a Web service that requires more processing, demonstrating the gain with less waste due to processing of invalid messages. Even so, the protection of services obtained through the use of the model is an advantage that should be considered.

In the tests that were performed no improper entry has been processed since the environment was properly configured to filter invalid entries.

VII. CONCLUSION AND FUTURE WORK

Because unevaluated data entry is the biggest challenge for any application development team in the Web environment and is the source of security problems in many applications [11] [15], a reusable and independent mechanism for data entry validation such as the WSIMV proposed in this paper is an important contribution to the security of Web services.

The WSIVM focuses on validation of data entry, allowing only valid entries to be accepted, since it is based on the white list approach, in which only predefined values are accepted and others are considered invalid.

This model is particularly interesting for the case of Web services that require processing of large amounts of data entries, because by ensuring that only valid entries are accepted it avoids the waste of processing by the application.

Carrying out input validation using the presented model is a solution for legacy applications that were not designed with validation of input data, since carrying out validation at entry points to the Web service decreases the need for a greater number of changes in the existing application, reducing development costs.

Moreover, according to Tsipenyuk et al. [31], the white list approach is more reliable than the blacklist. In the blacklist approach all values are considered valid unless explicitly specified. This approach has some problems; for example, if the validation of a field that does not contain HTML code is desired and a blacklist is created based on the current version of HTML, in the case of new versions, this list may no longer be considered valid.

The white list approach used in WSIVM results in a reduction in false positives and is a more reliable means of validating data entries. In contrast, the work reported in [21] has the disadvantage of obtaining large false positives; that is, the validator may fail by considering a message invalid when in fact the message is valid.

The number of false positives and true positives or false negatives will depend on the WSIVM XML Specification defined. More restrict regular expression specifications could have a negative impact on false positive numbers. The framework provides the specification to be customized according to the Web Service requirements and needs.

This study found a solution for the prevention of data injection attacks in Web services, providing a reusable protection mechanism which prevents the processing of malicious calls and is able to provide validation of input data regardless of the implementation of the Web service that uses this solution.

In the case study, it was observed that improved security had a negative impact on the performance of the developed Web service, which is quite common in security research. However, the validation of inputs reduces the possibility of

 ISBN: 978-1-61208-245-5

inserting invalid data and thus prevents attacks that would stop the correct execution of the Web service, offsetting the decrease in performance.

Using SQLmap, SQLninja or Acunetix or majority of available dynamic black-box security tools to test was not considered because most of these tools do not support web services testing.

Tests would be limited to the kind of web service or to the specification, the contribution of the framework with its inherited flexibility is supposed to be more valuable than tests on specific situations, however as the model advances new tests and comparisons will be proposed.

Our previous work published in Brinhosa et al. [29] is a reduced version of these research results. Here, in this paper, we presented in a detailed way: security issues in web services, the WSIVM model as well as the case study development and results obtained with tests.

There are different aspects that can be addressed in future work: (a) optimization of the implementation to improve the performance of the proposed model; (b) development of a semi-automatic generator of security specifications from WSDL; (c) verification of SOAP messages and paths in XPath format; (d) use of artificial intelligence or an anomaly detection system; and (e) making a feedback loop filter validation of invalid entries.

REFERENCES

[1] A. Belapurkar, A. Chakrabarti, H. Ponnapalli, N. Varadarajan, S. Padmanabhuni, and S. Sundarrajan, *Distributed Systems Security Issues, Processes and Solutions*. Hoboken, NJ: John Wiley and Sons, 2009.

[2] M. Q. Saleem, J. Jaafar, and M. F. Hassan, "Model driven security frameworks for addressing security problems of service oriented architecture," in *Proc. 2010 Int. Symp. Information Technology (ITSim)*, June 15–17, vol. 3, pp. 1341–1346.

[3] N. A. Nordbotten, "XML and Web services security standards," *Communications Surveys & Tutorials, IEEE*, vol. 11, no. 3, pp. 4–21, 2009.

[4] CEP. (2011). *CEPWebService*. Available: http://www.i-stream.com.br/webservices/cep.asmx. [retrieved: November, 2012]

[5] SIORG. (2009, Oct.). "Sistema de informações organizacionais do governo federal – SIORG – descrição do Web service SIORG versão 2.0". Available: http://catalogo.governoeletronico.gov.br/arquivos/Documentos/SIORG-DocumentacaoWebServicev.2-091006.pdf. [retrieved: November, 2012]

[6] J. Feiman, "Security in the SOA world: methodologies and practices," *Enterprise Integration Summit*, Sao Paulo, Brazil, Apr. 13–14, 2010. Available: http://www.gartner.com.br/tecnologias_empresariais/pdfs/brl37l_a4.pdf. [retrieved: November, 2012]

[7] J. Viega and J. Epstein, "Why applying standards to Web services is not enough," *IEEE Security & Privacy*, New York, NY, vol. 4, no. 4, pp. 25–31, 2006.

[8] M. Jensen, N. Gruschka, and R. Herkenhöner, "A survey of attacks on Web services," *Computer Science – Research and Development*, vol. 24, no. 4, pp. 185–197, 2009.

[9] S. Lakshminarayanan, "Interoperable security standards for Web services," *IT Professional*, vol. 12, no. 5, pp. 42–47, Sept./Oct. 2010.

[10] SANS. (2011). *The Top Cyber Security Risks*. Available: http://www.sans.org/top-cyber-security-risks/. [retrieved: November, 2012]

[11] OWASP. (2011). "OWASP code review guide. Codereview-Input validation." Available: http://www.owasp.org/index.php/Codereview-Input_Validation. [retrieved: November, 2012]

[12] E. Bertino, L. Martino, F. Paci, and A. Squicciarini, *Security for Web Services and Service-Oriented Architectures*. New York: Springer-Verlag, 2009.

[13] T. Scholte, D. Balzarotti, and E. Kirda, "Quo vadis? A study of the evolution of input validation vulnerabilities in Web applications," in *Proc. Int. Conference on Financial Cryptography and Data Security '11*, St. Lucia, 2011.

[14] CENZIC. (2009). *Web Application Security Trends Report*. Available: http://www.cenzic.com/downloads/Cenzic_AppSecTrends_Q1-Q2-2009.pdf. [retrieved: November, 2012]

[15] OWASP. (2010). "OWASP top 10 Web application security risks". Available: http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. [retrieved: November, 2012]

[16] Microsoft. (2012). "Microsoft Anti-Cross Site Scripting Library V4.2" Available: http://www.microsoft.com/en-us/download/details.aspx?id=28589. [retrieved: November, 2012]

[17] SSA (Sosnoski Software Associates Ltd). (2007). *WS-Security Wrapper*. Available: http://wsswrapper.sourceforge.net/. [retrieved: November, 2012]

[18] R. Wu and M. Hisada, "SOA Web Security and Applications", *Technology*, vol. 9, nº. 2, p. 163-177, 2010.

[19] N. Sidharth and J. Liu, "A framework for enhancing Web services security," in *Proc. 31st Ann. Int. Computer Software and Applications Conf., 2007, COMPSAC 2007*, Jul. 24–27, vol. 1, pp. 23–30.

[20] L. Sun and Y. Li, "XML and Web services security," in *Proc. 12th Int. Conf. Computer Supported Cooperative Work in Design*, *CSCWD 2008*, April 16–18, pp. 765–770.

[21] J. Lin and J. Chen, "An automated mechanism for secure input handling," *Journal of Computers*, vol. 4, no. 9, pp. 837–844, 2009.

[22] R. Bebawy, H. Sabry, S. El-Kassas, Y. Hanna, and Y. Youssef, "Nedgty: Web services firewall," in *Proc. IEEE Int. Conf. Web Services – ICWS*, Orlando, pp. 597–601, 2005.

[23] A. Blyth, "An architecture for an XML enabled firewall," *International Journal of Network Security*, vol. 8, no. 1, pp. 31–36, 2009, ISSN 1816–3548.

[24] Y. Loh, W. Yau, C. Wong, and W. Ho, "Design and implementation of an XML firewall," in *Proc. 2006 Int. Conf. Computational Intelligence and Security*, Nov. 3–6, vol. 2, pp. 1147–1150.

[25] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst, "Automatic creation of SQL injection and cross-site scripting attacks," in *Proc. 31st Int. Conf. Software Engineering (ICSE '09)*, IEEE Computer Society, Washington, DC, USA, pp. 199–209.

[26] J. Clarke, *SQL Injection Attacks and Defense*. Syngress Media Inc., 2009.

[27] A. Shostack and A. Stewart, *The New School of Information Security*, Boston: Addison-Wesley, 2008.

[28] Databreaches. (2009). "Top 10 worst data losses or breaches, updated." Available: http://www.databreaches.net/?p=7691. [retrieved: November, 2012]

[29] R. B. Brinhosa, C. M. Westphall, and C. B. Westphall, "Proposal and Development of the Web Services Input Validation Model, " *in Proc. IEEE Network Operations and Management Symposium (NOMS 2012), Maui, Hi, USA, pp. 643-646.*

[30] Apache. (2012). "Welcome to Apache Axis2/Java," Available: http://axis.apache.org/axis2/java/core/. [retrieved: November, 2012]

[31] K. Tsipenyuk, B. Chess, and G. McGraw, "Seven pernicious kingdoms: a taxonomy of software security errors," *Security & Privacy, IEEE*, vol. 3, no. 6, pp. 81–84, 2005.

 ISBN: 978-1-61208-245-5