Kuwait J. Sci. **41** (2) pp. 175-203, 2014

Improved cross site scripting filter for input validation against attacks in web services

ELANGOVAN UMA* AND ARPUTHARAJ KANNAN**

* *Assistant Professor, Department of Information Science and Technology, Anna University, Chennai -600025, India, email: iamumaramesh@gmail.com*
** *Professor and Head, Department of Information Science and Technology, Anna University, Chennai -600025, India email: kannan@annauniv.edu*

ABSTRACT

Nowadays, everybody needs to handle sensitive data like online banking account details and other information related to financial transactions on the Internet. In this scenario, many Web attacks such as injection attacks are targeted on these sensitive data. Such attacks are carried out by running scripts on users computers that utilize vulnerably coded client/server pages. Moreover, these attacks run malicious codes to steal personal information from the server. Though this code can easily be generated by the attacker, it is very difficult to prevent it by the current cross site scripting filters due to their lack in detection accuracy. Therefore, cross site scripting attack is a challenging issue for the Internet users. Hence, it is necessary to detect and prevent the injection attacks through efficient schemes. However, most of the existing schemes lack this capability in terms of accuracy and need further improvement. In this paper, a new self-aware message analysis cum validation algorithm has been proposed for detecting and filtering various types of Web Service attacks. This proposed system receives requests and generates suitable response from the dummy server page to analyze the nature of attack. New policies are created in this work to analyze the response and forward the legitimate request to original Web Service page. The proposed injection filters have been tested with all possible attacks for verifying the robustness of filtering policies. The results obtained from this work show that the proposed filtering policy is highly robust in refining the malicious message. The implementation and accuracy of the proposed approach has been proved through extensive testing using real-world cross site scripting generation and analysis. The results obtained from the work show that the proposed filtering policy is very strong in refining the malicious message, which contains attacks such as cross site scripting, injection, message replay and semantic attacks. We demonstrated the implementation and accuracy of our approach through extended testing using real-world cross site scripting exploits.

Keywords: Cross site scripting attacks; cross site scripting filters; security; semantic attack filter; web services.

INTRODUCTION

W3Cdefines a Web service as a software system designed to suppor tinteroperable machine-to-machine interaction over networks. It contains three major components, namely SOAP, WSDL and UDDI. The service requester sends the request through SOAP protocol to UDDI and then it maps the corresponding URL in WSDL format from service provider. The WSDL of service can easily be viewed by Internet users. The attackers can read the information like data type, maximum length, card number and password. They generate Cross Site Scripting (XSS) attacks to get sensitive data easily (Negm, 2004). In general, they use the graphical user interface to generate attacks from the client side.

XSS is a type of computer security vulnerability, typically found in Web applications. It enables attackers to inject client-side script into Web pages viewed by other users. A cross site scripting vulnerability is used by attackers to bypass access controls. It is categorized into persistent and non-persistent attacks. The persistent (or stored) XSS vulnerability is a more devastating variant of a cross-site scripting flaw. It occurs when the data provided by the attacker is saved by the server, and then permanently displayed on "normal" pages returned to other users. These holes show up when the data provided by a web client, most commonly in HTTP query parameters or in HTML form submissions, are used immediately by server-side scripts to parse and display a page of results for the user, without properly sanitizing the request.

Figure 1 shows how an XSS attack happens and explains how it is allowed inside a Web page. In this example, the attacker generatesa malicious script at the client side and it is stored into server side database through login process as shown in Figure 1. The Website affected by an XSS attacker is shown in Figure 2.

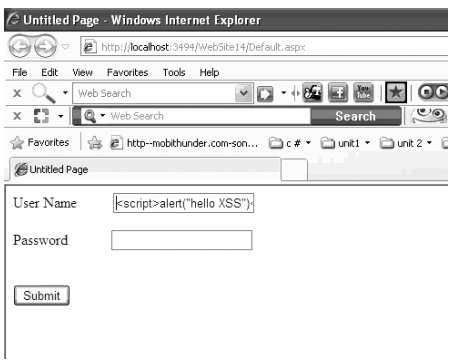

Fig.1. An XSS input from an attacker

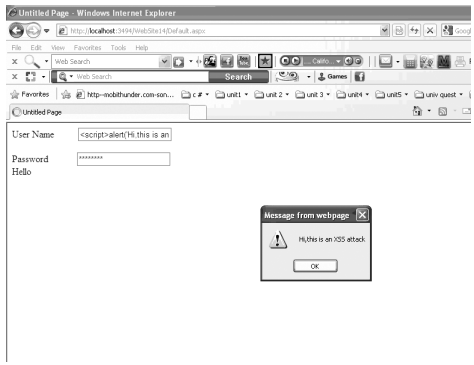

Fig. 2. Website affected by an XSS attacker

This malicious script which is accepted and stored in the database is present

in the server and destroys the vulnerable website (Pete, 2004). The proposed system for XSS filter combines static approach and dynamic approach to filter it.In this work, new filtering policies for specific Web service attacks, namely message replay, coercive parsing, oversized message, parameter tampering and semantic URL attack were proposed and implemented. The main advantage of the proposed system over the existing system is that it provides facilities which dynamically detect Web attacks and prevent them efficiently.

LITERATURE SURVEY

To identify the threats raised by XSS attacks, many researchers have proposed solutions, which are categorized as static and dynamic approaches based on the literature details.

Static approach

A static tainted data flow analysis, using flow-sensitive, inter-procedural, context-sensitive data for PHP checks if user input is used at a target statement without adequate input validation. A string analysis approach for PHP, claimed to be efficient and accurate to approximate the dynamically generated web pages (Jovanovic *et al.*, 2006; Minamide, 2005); Xie & Aiken (2006) carried out an SQL injection vulnerability analysis that gains scalability and efficiency in exchange for soundness by using block and function-summaries. Wasserman & Su (2008) proposed a static analysis for finding XSS vulnerabilities that directly addresses weak or absent input validation. It performs an adapted string analysis to track untrusted string values and checks for untrusted scripts based on formal language techniques using Context Free Grammar (CFG). It is used to delimit the regions of the web page that contains untrusted data.

Kiezun *et al.* (2009) proposed an automatic static analysis technique for creating inputs that expose SQL Injection (SQLI) and XSS vulnerabilities. This technique generates consumed inputs, symbolically tracks taints through execution and changes the inputs to produce concrete exploits. Moreover, their technique creates real attack transmitters, which have few false positives, incur no runtime overhead for the positioned application without requiring modification of application's source code and handles dynamic programming-language constructs. Vogt *et al.* (2007) applied a mechanism to identify sensitive data on the client-end in order to ensure that sensitive information is sent by the JavaScript code only to the processed site. A similar approach was proposed by Gundy & Chen (2009) in which, the content of the website is divided into nodes, represented by classes of confidence marked with a XML namespace prefix, randomly generated. The content of the website is divided into nodes.

Nadji *et al.* (2009) proposed a solution based on the same origin policy, which applies the minimum serialization technique through the use of suffixes and prefixes (markers) belonging to a CFG. This CFG is used to delimit the regions of the web page that contain untrusted data. One of the limitations of their work is that it can give only static detection for all input entries. Moreover, it does not allow important data with restricted non-alphanumeric characters like email address. Hence, it should be implemented only on effective dynamic XSS detection mechanisms.

Dynamic approach

Nguyen-Tuong *et al.* (2005) proposed modifications of the PHP interpreter to dynamically track tainted data in PHP programs. QED is a goal-directed model-checking system that automatically generates attacks exploiting taint-based vulnerabilities in large Java web applications (Martin & Lam, 2008). ARDILLA generates test inputs for SQLI or XSS bug: it symbolically tracks taints through execution (including through database accesses) and mutates the inputs to produce concrete exploits according to the taint information (Kiezun *et al.*, 2009). Saner uses static and dynamic analysis to determine whether the PHP program has been properly sanitized (Balzarotti *et al.*, 2008). Shar & Tan (2012) proposed a technique based on static program analysis and pattern matching techniques. It identifies vulnerabilities in source code and secures with removal mechanism.

Shen *et al.* (2007) proposed a game theoretic high level information fusion based decision and control framework to detect and predict the multi-stage stealthy cyber attacks. It addresses the cyber network security problem from a system control and decision perspective and revises the Markov game model with the knowledge of the cyber attack domain. But this approach does not prevent semantic URL attack. It is designed to prevent URL attack (Liu *et al.*, 2013; Chuang *et al.*, 2013; Adnan *et.al.*, 2011).

Limitations of the Existing Systems:

All existing approaches use the server directly and hence cause data loss due to unexpected malicious script that might affect the server. For example, hex decimal, binary formatted malicious scripts. To overcome the limitation of these existing systems, we propose a dynamic analysis method. Our approach allows the request to dummy Web server page and analyses the request with HTTP response header of the sample database. If the request passes the HTTP response header analysis, then it is retrieved by web service parameters and analyzed for the presence of attacks.

The existing systems with the XSS filter solution which is installed on both sides of the Client/Server reduce the speed of the communication. They also introduce the overload problem to the server. Moreover, most of the existing systems consider URL attack instead of semantic URL attack. In spite of all these techniques, the XSS attacks are not filtered efficiently in current Web application due to the static nature of the filters used in the current systems. Hence, it is necessary to provide a dynamic self defense in addition to the static analysis techniques.

XSS ATTACKS ON WEB SERVICES

The various XSS attacks on Web services are discussed below.

XML Injection Attack

XML Injection occurs when user input is passed onto XML stream. Since user-editable fields can be accessed by Web interface through forms, XML Injection overwrites the "private" role element. For example, consider the customer database defined by the following database schema:

```
<custrecord>
    <id>1234</id>
    <role>user</role>
    <cname>abc</cname>
    <email>abc@yahoo.co.in</email><role>admin</role><email>abc@yahoo.co.in
</email>
<addresss>33-westcross st</address>
<zip>608001</zip>
</custrecord>
```

In this schema, all the fields can be edited by a user. Hence, this can be used by an attacker to destroy the valuable data.

SQL Injection Attack

Simple SQL statements are inserted into the database using the input of a SOAP message by an attacker. If the statements are not sanitized, the attacker may gain access to the databases. This attack uses SQL commands such as SELECT, CREATE, UPDATE, DELETE, DROP, 1 = 1, ALTER and INSERT. These commands are used to embed malicious input in client side interface.

XPath Injection Attack

XPath is used in an "XML-Enabled" Database like SQL Server 2000 and 2005. In SQL, XPath uses delimiters to separate code and data. Since there is no access control in XML or XPath, an attacker can control data in an XPath statement to access arbitrary parts of the XML file or return arbitrary data. The sample code snippet for XPath Injection attack is shown below:

```
Public UserDataLoginUser(string Login, string Password)
{
UserData user = new UserData();
stringxpathQuery = "/Users/User[attribute::Login='" + Login + "' and attribute::
Password='" + Password + "']/*";
XPathNodeIteratorxpathIter = xpathNav.Select(xpathQuery);...
```

Buffer Overflow Attack

The attacker inserts malicious content with well-formed message in SOAP request, which is beyond the allowable size of the buffer and causes Denial of Services attack (DoS). It is called buffer overflow attack.

Message Replay Attack

Amessage replay attackis one in which anattackereavesdrops and obtains a copy of anencryptedmessage and then reuses themessageat a later time in an attempt to reveal the secret messages or to provide a fake identity. For example, when a legitimate client transfers money from his account in a bank to the receiver, the attacker steals the password and uses it multiple times by sending it to the Web service in order to cause money loss.

Parameter Tampering Attack

The WSDL document has parameters to receive inputs from the client. The parameters are visible in a WSDL structure to all users. Here, the attacker tries to send different data types of parameters several times. Then, the Web services may crash.

Coercive Parsing Attack

The attacker sends a SOAP message with an unlimited amount of opening tags in the SOAP Body. It means the attacker sends a very deeply nested XML

document into the targeted Web service. If the parser receives a peculiar format of SOAP messages, it reduces its processing capability and this may result in Distributed Denial-of-Service (DDoS) attack. For example, consider the following code:

```
<soapenv:Envelopexmlns:soapenv="…" xmlns: soapenc:"…">
<soapenv:Body>
<x>
    <x>……
…<x>
```

The nesting provided by tag × leads to an infinite loop resulting in a DDOS attack.

Semantic URL Attack

In asemantic URL attack, aclientmanually adjusts theparametersof its request by maintaining the URL'ssyntax,but altering itsmeaning. This can be avoided by giving token and timestamp for expiration. The existing Web service allows the users to reset their password by answering the security question correctly and allows the users to send the password to the e-mail address of their choice.The receiving page has all the information it needs to send the password to the new e-mail. The hidden variable username contains the value uid001, which is the user identification of the e-mail account as shown in the URL http://urlsemanticdemo.com/resetpwd.aspx?userid = uid001&altemail = alternative%40emailexample.com. When this URL appears in the location bar of the browser, it is possible to identify the user details and the e-mail address through the URL parameters. The malicious user may decide to steal other people's (uid002) e-mail address by visiting the following URL:http://urlsemanticdemo.com/resetpwd.aspx?userid = uid002&altemail = alternative%40emailexample.com. If the resetpwd.php accepts these values, it is vulnerable to a semantic URL attack. The new password of the uid002 e-mail address is generated and sent to alternative@emailexmaple.com which causes uid002's e-mail account to be stolen.

PROPOSED ARCHITECTURE

The architecture of the XSS filter system proposed in this work is shown in Figure 3. This proposed architecture for XSS filter was implemented with three major components in C#. They are server, database and filtering policies. The

filtering policies are XSS filter, message replay filter, coercive parsing filter, oversized message filter, parameter tampering filter and semantic URL filter. The filtering policies are built with SAX parser, which is faster than DOM parser (Chuang *et al.*, 2013). Thus, the system was implemented with SAX parser to improve the filtering speed of the filter.

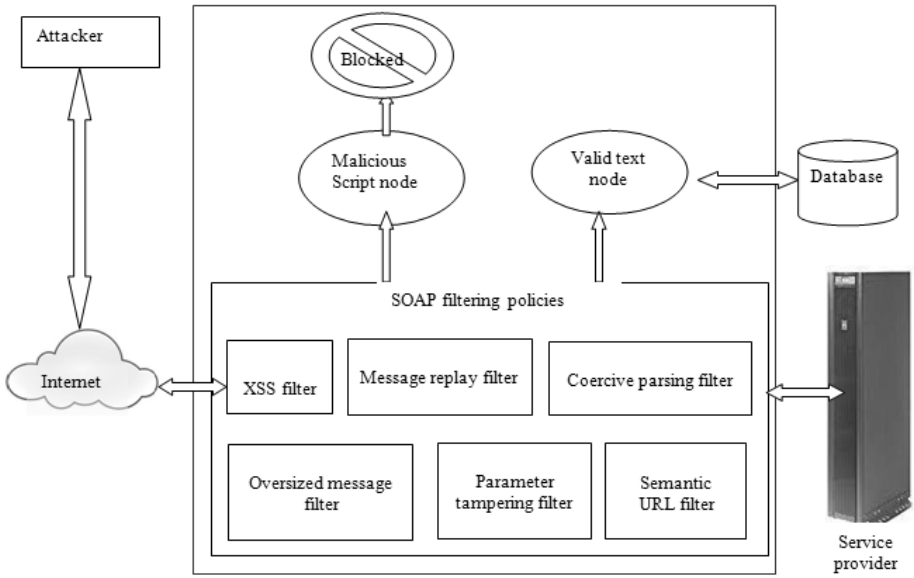

Fig. 3. Architecture of an XSS filter

The attacker tries to send the malicious script to the Web service through the internet. If it is stored in the server directly, then it will cause data theft or redirection problem. Therefore, this vulnerable server should be built with robust security architecture. The malformed input was refined by the filtering policies in the proposed XSS filter architecture. In this system, the script node containing malicious message was blocked by the filter, otherwise the valid SOAP message would be processed and responded by the Web method in the Web services.

Many attacks are targeted at the Web services by the attackers. The proposed technique has to avoid XSS attacks and some XML related attacks like message replay attacks, oversized payload and recursive payloads. Hence, it was decided to block XSS attacks and web service oriented attacks. The filtering policy receives all input from the user. If it receives malicious script node, then it is processed in dummy server page and HTTP response header analysis. Again, it

is passed thorough input validator to find the possibility of XSS attack. Later, it is analyzed for Web service related attacks like message replay, coercive parsing, oversized message, parameter tampering and semantic URL filtering. The system considers some important Web service related attacks. Then the legitimate input is treated as normal text node and allowed to be processed with web service parameters.

Advantages of the Proposed System

In the proposed system, the XSS filter was installed at the server side and it was implemented by SAX parser. It avoids the attacker, by analyzing the servers parameters to get the valid parameters of self validating algorithm. Then the system maintained trust level of the particular IP Address by calculating the number of requests coming with malicious script code. If it finds the attacker then, it rejects the communication with him. In addition, the proposed system used the dynamic approach instead of the static approach followed by the current systems. Also, the proposed system has been designed to protect from non-ASCII attacks like hex decimal, binary attacks.

Filtering Policies

The proposed XSS filter was developed with various filtering policies as mentioned below for preventing different types of Web service attacks. The XSS filter was improved with oversized message filter to identify buffer overflow attack, message reply filter to defend against message replay attack, parameter tampering filter to refine the parameter tampering attack, coercive parsing filter to catch the wrong formatted SOAP format and semantic URL attack filter to deny the semantic attacks.

XSS filter

The proposed system consists of two major phases: Dummy Web server page XSS finder and Input Validator. The first phase identifies unmatched response with expected response using Dummy Web server page and HTTP response header analyzer. If so, it maintains the distrust level for an attacker. The Next phase identifies the malicious script characters in the received input and then increases the distrust level for fixed number of requests. This approach did not maintain black list alone, but it is implemented along with the dummy Web page with HTTP response header analyzer and input validator. Hence, the proposed approach is dynamic when it is compared with existing systems.

Input refining is performed by HTTP response analyzer and regular expression validator. The first module was implemented with attack tester, which held the dummy web server page with dummy database. This dummy web server page received the input from the user and performed with analyzer modules. Then it responded back to user. This module cached the response and then analyzed all fields of the header. At that time, it matched the nature of requested data from Http request header with the nature of expected data from the dummy database. If it does not match with the expected database then, it checks set of alert HTTP response code like server redirection 3xx. Next, it validates input type, length, format and range. Like this, the input was passed to all types of refining stages and the session token was stored to block the communication.

The server-side code was designed to constrain input supplied through client-side input controls or input from other sources such as query strings or cookies for XML injection, SQL injection, XPath Injection and cookie replay detection. The input was checked for non-alphanumeric characters and it is checked with BL (Blacklist) containing scripts for XSS attack. The filter stored user-specific data using session token. It used this data to process requests from the user for which the session state was instantiated. A session state of user was identified by a session ID.

The architecture of the script based injection filter service proposed in this research work is shown in Figure 4. It has been implemented using three major components namely server, database and filtering policy with dynamic and static filter. The attacker tries to send the malicious script to the Web Service through the internet. If it comes to the server directly, it will cause distributed denial of services attack and other attacks.

Through the proposed script injection filter service architecture, the malformed input is refined in the filtering policies. If the script node contains a malicious message then it is blocked by the filter, otherwise the valid SOAP message is processed and responded by the Web method available in the Web Services. The filtering policy receives all input from the user. If it receives malicious script, then it is processed in test Web Service for HTTP response header analysis. Again, it is passed through input validator to find the possibility of XSS attack. Later, it is analyzed for Web Service related attacks like message replay, coercive parsing, oversized message, parameter tampering and semantic URL filtering attacks.

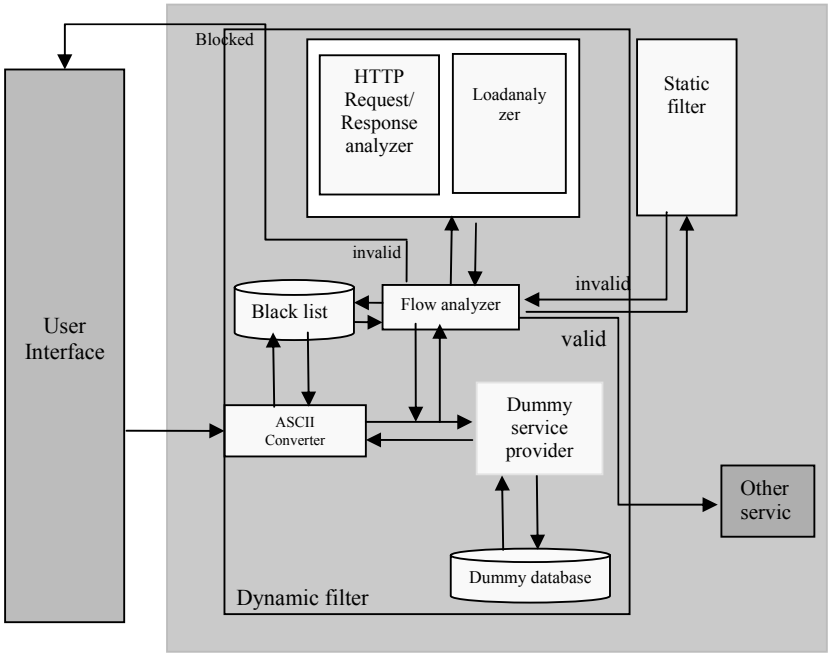

Fig. 4. Proposed architecture of XSS injection filter service

This filter has been built with significant components to achieve dynamic filtering policies using American Standard Code for Information Interchange (ASCII) converter, blacklist, dummy Web Service provider with dummy database, flow analyzer, HTTP request/response analyzer, load analyzer and static filter.

Dynamic Filter

ASCII converter

The legitimate/illegitimate user sends an input to the server through the Web user interface provided by the server side. In server side, the converter receives the request and checks its format to find the non-ASCII format like hex, octal, binary form of request. If it presents, the request is converted into ASCII format to compare its value into blacklist. The blacklist contains whole set of updated list of script injection attacks in ASCII format. This is used to compare the request with ASCII format. It identifies the presence of attacks in the message. Then the identified invalid request is forwarded to flow analyzer to block the request.

Blacklist

The non-ASCII input is converted to ASCII to check it with blacklist container. The blacklist container has up-to-date malicious script for the comparison of script based injection attacks. If the input matches with blacklist container then the communication will be cut off from the server side. Otherwise, the input is forwarded to the dummy service provider with dummy database.

Dummy service provider and database

This service models like real web service to identify the anomaly request escaped from above modules. It receives the requests and processes the request even he is not an authenticated user to identify SQL injection. This allowed request gets response from the dummy service provider. If the request contains any SQL injection, then the executed request retrieves dummy user realms from the database. This flow is analyzed using flow analyzer.

Flow analyzer

The incoming and outgoing message from user and dummy Web Service is clearly analyzed by the flow analyzer to make decision for the user. If it identifies and records the illegitimate user, then the flow analyzer blocks the communication. This report is retrieved by three components. Those are blacklist, HTTP request/response analyzer and load analyzer. The escaped request from dynamic filter is forwarded to static filter.

HTTP request/response analyzer

The request is processed and responds back to the client with dummy values. Here, the HTTP request and response are clearly captured by the flow analyzer. Then the flow analyzer gets report from the HTTP Request/Response analyzer about the communication of the user/attacker. It maintains set of policies to identify the illegitimate request. If the input is matched with the policy of flow analyzer, then that users IP address will be added to blacklist of filter. In addition, the request is again forwarded to static filter to verify its nature of the input. It also identifies the redirection of the values from the server to other proprietary servers.

Load analyzer

This is used to find the load value of returned data. The exceeded data value is compared with expected allowable load value of the analyzer.

Static filter

The static filter verifies its format by passing the value to the special character testing policy. This checks the incoming request for the presence of non alphanumeric characters. The encountered request is reported to flow analyzer and then the flow analyzer blocks the communication with that system.

ALGORITHM FOR SELF-AWARE MESSAGE ANALYZING FOR SCRIPT BASED INJECTION ATTACKS

The proposed algorithm consists of two major phases namely response analyzer and malicious script analyzer. In the first phase, the response analyzer identifies unmatched responses with expected responses using Dummy Web Service provider and flow analyzer based on HTTP request/response header analyzer. If they do not match, it maintains blacklist for attacks and such users are black listed. The next phase identifies the malicious script characters in the received input and then it maintains session information of the attacker. This approach not only maintains black lists, but also analyzes the dummy service providers using the flow of attackers input and input validator. Hence, this proposed dynamic approach provides better security, when it is compared with existing systems that use static analysis. In this proposed system, input refining is performed by flow analyzer and static filter.

The first module has been implemented with flow analyzer, which holds the dummy Web page with dummy database. This dummy Web page receives the input from the user and performs the analysis on the dummy page and response are sent to the user. This module caches the response, and then analyzes all fields of the header. At that time, it matches the nature of requested data from HTTP request header with the nature of expected data from the dummy database. If it does not match with the expected database, then it checks set of alert HTTP response codes like server redirection 3xx. Next, it validates the input type, length, format and range. In this way, the input is passed to all the refining stages and the session token is stored to block the communication.

The server-side code has been designed to constrain input supplied through client-side input controls and also the input from other sources such as query strings or cookies for XML injection, SQL injection, XPath Injection and cookie replay detection. The input is checked for non-alphanumeric characters and is checked with BL (Black List) containing scripts for XSS attack. The filter stores the user-specific data using session tokens. It uses this data to process requests from the user for which the session state is instantiated. A session state of user is identified by a session ID.

The detailed steps of the proposed algorithm for XSS filter are given in Figure

5. The malformed requests are calculated by session variables to add particular IP address black list. The received input checks for alphanumeric characters.

Algorithm for script based injection filter service

Input = > Incoming request

Output = > Accept/Deny

```
1.  Call ASCII convertor
2.  If input = = Blacklist
        Send invalid report to flow analyzer
3.  At server side, Send Response with dummyDatabase to the client
4.  Call flow analyzer
5.  Analyze attributes of HTTP Request/Response Header attackers input
6.  Extract Header Information
7.  If (Response code = = Set of Banned Response Code)
        Call LoadAnalyzer (Calculates size of returned data to the clientside)
8.  If (Response! = size of expected data from test database)
9.  Call static filter
    If (Input! = Input Validator [(0-9)&&(A-Z)&&(a-z)])
            Get IP address and send to blacklist.
            Send message (“your input contains malicious data, your
            connection is discarded”).
10. Else
            Call Main Web Service
```

Fig. 5. Algorithm for XSS injection filter service

The users input is refined by validating its characters against symbols used in scripting to deny the XSS attack. The filter maintains the trust level of user by counting the number of malicious requests. If it exceeds a certain level, the Web Service provider drops the communication for such user.

The proposed system has been implemented with service requester, service provider and filter service. The intermediary service receives the request from the client, then it forwards the request to service provider. The intermediary service has been designed with ASCII character converter, HTTP response checker,

load analyzer, flow analyzer and blacklist container. The attacker can send other types of script codes such as hex decimal, binary etc. The tested valid input is forwarded to XML based injection filter service to identify the Web Service based injection attack. The invalid request is rejected, if it is identified by the static filter. The detailed steps of the proposed algorithm for XSS filter are given in Figure 4. The number of requests is calculated by counter variable connectcount used for particular IP address. If counter variable exceeds more than 5 requests, then the communication with that particular IP address is discarded by the proposed XSS filter. The received input checks for alphanumeric character.

The users input is refined by validating its characters against symbols used in scripting to deny XSS attack. The filter maintains the trust level of user, by counting the number of malicious requests. If it exceeds a certain level, the Web service provider drops the communication.

Oversized message filter

Because of oversized message, the Web service resources such as CPU time, memory usage and database connections keep busy. The filter was implemented to measure the size of the incoming message. Moreover, it verified signature to ensure that the message is not transferred in transit. Finally, it parsed the entire request message for malicious content. This filter avoided buffer overflow attack. The service provider had to set its maximum request length. The filter loaded the XMLdocument and change XSD value maxMessageLength = 1024 or any required buffer size. The algorithm is given below:

1. Get XML document of client
2. Get messaging attribute
3. If maxMessgingLength = 1024 then do Web method
4. Else discard

Message replay filter

This filter catches an identifier for incoming messages, which match an entry in the replay detection cache database. If message signature is valid, the filter compares the message timestamp to its own current clock time value for synchronization. If the message signature is invalid or any time stamp mismatch then, the message is rejected. This can be done by calculating timeTolerance, CacheLifetime and MaxMessageAge. The time tolerance is the acceptable value time difference between the sender and the maximum message age is conFigd as 600 seconds.

* The server calculates the message age by subtracting the created value on the message from the current server time.
* For a message that appears to have been created in the past or, if the server and message creation times are identical, the message will be accepted only, when its message age is less than or equal to the values for the maxMessageAge parameter plus the timeTolerance setting,
* If the Maximum Message Age value is less than or equal to the Time Tolerance setting, the message is accepted.

CacheLifetimeInSeconds = (MMA + TT*2)

CLS = CacheLifetimeinSeconds

MMA = MaximumMessageAge

TT = TimeTolerance

The algorithm of message replay filter is given in Figure 6.

FindReplayedMessage(get envelope of SOAP)

1. Create input/output filter for client/server.
2. Read XML document
3. Get CLS and MMA of incoming message.
4. Calculate the message expiration time based on the cache lifetime configured in the policy assertion.
5. Get the current time.
6. Compute the time difference between the message timestamp and the current time.
   timeDifference = currentDate - timestamp;
7. Assign messageAge = timeDifference
8. Check for messages where sender's clock is slower than the server's clock for first condition.
9. Account for messages where the sender's clock is faster than the server'sclock through the second condition.
   if ((messageAge>maxMessageAge +TimeToleranceInSeconds)
   || (messageAge< 0 &&messageAgeInSeconds>TimeTolerance))
   generates exception SecurityFault Messages for AgeRequirementsNotSatisfied
10. Add the Message, identity value of message and expiration time to message replay cache database.

Fig. 6. The algorithm of message replay filter

This solves message replay attack.

Parameter Tampering Filter

In the Webservice, the received parameters are checked for data type and null values. This filter checks the parameter for valid data; if it fails, then, it throws

an error to the sender once. Even if the sender continues, his misbehaving with parameters leads to the disconnection of communication. To solve this problem, the proposed XSS filter was created with pmcheckerfunction as given in Figure 7. It checks the arguments for null values, start element and end element of the received request.

```
1. Call pmcheckerXml(reader)
2. If (reader == null or empty)
      sendArgumentNullException("reader");
3. Get xsdpath from received XML document
4. GetAttribute("xsdPath" as xsdpath);
5. If (xsdpath != null or xsdpath!=empty){
      assignxsdPath = xsdpath;
else
      send Exception for MissingXsdPath;
6. Get StartElement from received XML document
7. If(StartElement != Empty)
      ReadEndElement
8. If(EndElement != Empty)
      Call Web method
else
      send exception to the attacker and alert administrator
```

Fig.7. Algorithm for proposed XSS filter with pmchecker function

Coercive parsing filter

The filter verifies the received message for wrong format of SOAP message by generating SOAP fault code. This filter blocks the input that has a strange format. This policy is used the values in SOAP fault code. They are:

* Version Mismatch Fault Code

  It finds an invalid namespace for a SOAP envelope and throws exception.

* Must Understand Fault Code

  It indicates whether a header entry is mandatory or optional for the recipient to process.

Many of the things can go wrong with a Web service message. The Web service may encounter a problem, input data may be wrong or a header may come across which the server does not understand. The algorithm for coercive parsing filter is given below.

1. Get XML document
2. If(SOAP version = = valid namespace)

   if(mustUnderstand = = 1)

   allow

   else throw mustunderstand Fault Exception
3. else

   throw Version Mismatch Fault Exception

Semantic URL attack filter

The semantic URL attack is theclientmanually retypes theparametersof its request by keeping the URL'sstructure,but altering itssemanticmeaning. This is protected by giving token and timestamp for expiration. This filter sets the NONCE (Number used ONCE) by generating a random number and assigns the time stamp to use the request URL for limited time. The random number generator needs a seed value to create NONCE. For that, the system takes the process id as seed value, as given below.

$$NONCE = (PID)^{C} \bmod N \,||\, TimeStamp((SD + ST) + D)$$

PID = Process ID

C = Counter value in 100 digits

N = Large Prime number in 200 digits

SD = System Date

ST = System Time

D = 24 Hours

The generated NONCE is very tough to break by brute force attack, because PID is generated by a 32-bit memory operating system. Then the system increases its power with counter variable to createa more complex number. The URL is concatenated with user identity and NONCE.

Client URL = URL || UID || NONCE.

URL = Uniform Resource Locator

UID = User Identity

NONCE = Number used ONCE

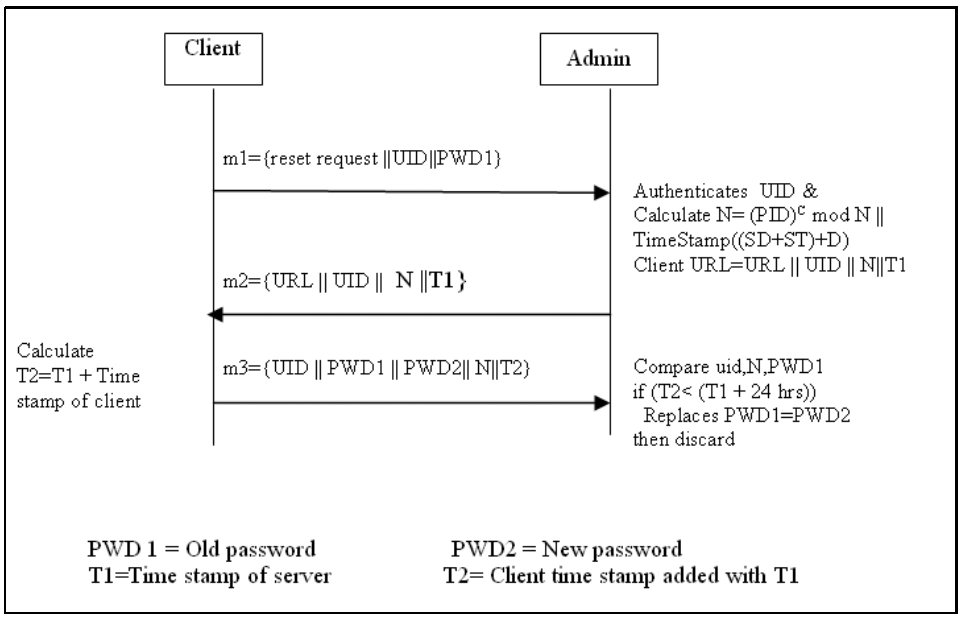

Fig. 8. Password reset policy of the proposed filter

The information flow between client and server is shown in Figure 8. Therefore, the attacker cannot use the URL or imitate the user id embedded in the URL by substituting other user id and random numbers. The attacker cannot predict the seed value of NONCE from the 32-bit operating system or higher configuration. Because each and every time the process id is changed by the operating system.

Algorithm for Password reset policy of the proposed semantic URL filter

Input: reset request (user identity and password)

Output: accept/reject

1. Client sends message M1=reset request || UID || PWD1 to the service provider
2. Service provider authenticates UID and calculates N=(PID)C mod N || TimeStamp((SD+ST)+D)

   Concatenate Client URL=URL||UID||N||T1

   Send M2=URL||UID||N||T1 to the client
3. Calculate T2=T1+Time stamp of client
4. Calculate M3=UID||PWD1||PWD2||N||T2
5. Compare UID,N,PWD1 if (T2<(T1+24hrs)) replace PWD1=PWD2 then discard

Fig. 9. Algorithm for Password reset policy of the proposed semantic URL filter

The algorithm is proposed to send and receive the password resetting request securely is shown in Figure 9. It utilizes the dynamic nonce used for each and every communication.This filter guards password resetting, while the reset request and response is communicated over Internet. Here, the attacker is not visible; instead the legitimate user tries to steal the password of the other legitimate user in that same Web community. First, the user sends the reset request to the server and then the server sends back the response to the client with corresponding URL with required parameters to reset the password. Now, the legitimate user slightly modifies the parameter to locate the other users parameter and receives the password reset access of the other user. In this filter, the mitigation approach is implemented by two significant components of semantic URL filter. Those are random nonce generator, and time stamp calculator. The random nonce generator produces random values from identity of task. Afterwards, the nonce value is concatenated with time stamp values to maintain its freshness.

RESULTS

The proposed XSS filter was tested against a list of known XSS attacks as given in Table1. These attacks were uploaded in custom script of SOAPUI tool. The system was tested by raising attacks shown in Table1. The XSS attack prevention took 1ms or 2ms. It depends on its complexity to find the XSS attributes in XSS attack. Our experiments showed that the proposed XSS filter could detect about 99.99% XSS attacks. Our experiment also showed that our

implementation uses a small amount of CPU time and memory space of the system.

Table 1. Selected test case run with XSS Filter

S.No.	Test Cases	TimeTaken	Results
1.	< script > alert(XSS attack) < script >	1ms	OK
2.	The ending tag of the envelope element < /SOAP-ENV:Envelope > is removed	1ms	OK
3.	Omit the quotes for the attribute in the envelope tag < SOAP-ENV:Envelopexmlns: n1 = http:// www.au.edu/results/ >	1ms	OK
4.	The namespaces starting with xmlns are changed to xs	2ms	OK
5.	Change the name of the service	1ms	OK
6.	A character is inserted in to the element < email > dfg@myWeb.com < role > admin < /rol > < /email >	1ms	OK
7.	Change the original namespace http://www.au.edu/ results to http://www.au.edu/students/	1ms	OK
8.	The message that has the < city > Chennai < /city > tag repeated up to fifty times, i.e., test for replay attacks.	2ms	OK
9.	An additional attribute is inserted into < cardnoxsi:type = xsd:int > 1234 < /cardno >	2ms	OK

The dummy login Webpage is shown to explain how the system prevents the attack. It can be used with any type of Web services with relevant dummy Web service and database. The input received in any sub-link of the page is tested with a relevant set of input fields.

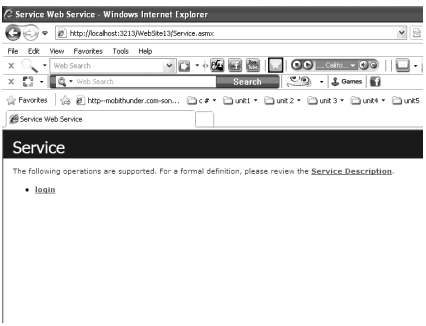

Fig.10. Server side login page of service provider

The proposed filter, embedded in Web service page is shown in Figure 10. The client page of dummy login screen with username and password is displayed in Figure 11. In this page, we have generated the XSS attack like an attacker by typing < script > alert (Hello XSS) < /script > through text box. The service provider validates the users input through the proposed XSS filter. Here the parser validates the input and identifies the input which is a script node. Suddenly, it throws a warning message and blocks the user as shown in Figure 12.

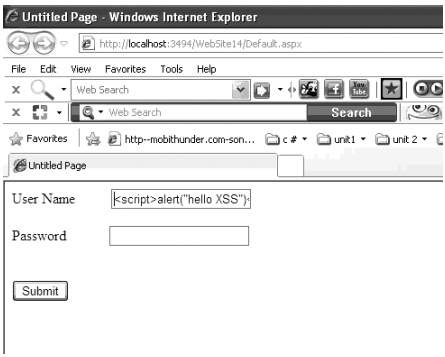

Fig.11. Attacker trying to generate XSS attack

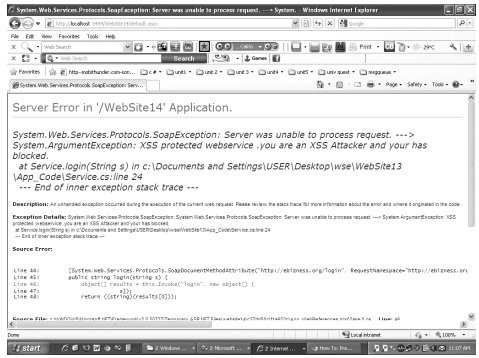

Fig. 12. Attacker has blocked by the XSS filter

PROTECTION EVALUATION

We tested the proposed and existing filters against three sources of XSS attack data that have widely been used in previous research as shown in Table 2.

* xssed: xssed.com includes reports of Websites vulnerable to XSS, along with a URL for a sample attack. Since the dataset is very large, we randomly selected a subset of 500 recent working attacks from among these, in order to estimate the effectiveness of our filter against real-world attacks.

* XSS cheatsheet: The xssed dataset is biased towards very simple attack payloads, since most of them simply inject a script tag. To assess the fillter's protection for more complex attacks, we have created a Web page with multiple XSS vulnerabilities and tried attack vectors from the XSS Cheat Sheet, a well-known and off-cited source for XSS filter circumvention techniques. The proposed XSS filter refined all 350 test cases generated by SOAPUI, also filtered attacks referred by xssed (400 attacks) and cheat sheet (20 attacks).

Table 2. Results for SOAPUI, xssed and cheatsheet

Datasets	Proposed XSS filter	Validate request filter	Anti XSS filter
SOAPUI testing tool	350/350	311/350	324/350
xssed	400/400	348/400	350/400
cheatsheet	20/20	18/20	17/20

A Test Case is a collection of Test Steps that are assembled to test the proposed XSS Filter service. The Implemented XSS Filter is tested successfully with SOAP UI 4.5.1. It passes all test cases of XSS, fuzzing scan, Invalid types, SQL injection, Custom Scripts, XML Bomb and XPath Injection from SOAPUI. We can add any number of Test Cases to a custom script. It generates text file report.txt to provide details of security test.

It contains test starting time, date and total hours to finish the test and status of script. The status of the script means whether the scripts are cached or not by the filters as shown in Figure 13.

```
SecurityTest started at 2012-08-30 12:48:55.328
Step 1 [login] No Alerts: took 720 ms
SecurityScan 1 [Cross Site Scripting] No Alerts, took = 187
[Cross Site Scripting] Request 1 - OK -
[sss=<SCRIPT>document.write("<SCRI");</SCRIPT>PT
SRC="http://soapui.org/xss.js"></SCRIPT>]: took 2 ms
[Cross Site Scripting] Request 2 - OK - [sss=<SCRIPT a=">'>"
SRC="http://soapui.org/xss.js"></SCRIPT>]: took 2 ms
[Cross Site Scripting] Request 3 - OK - [sss=<SCRIPT a=`>`
SRC="http://soapui.org/xss.js"></SCRIPT>]: took 1 ms
[Cross Site Scripting] Request 4 - OK - [sss=<SCRIPT "a='>'"
SRC="http://soapui.org/xss.js"></SCRIPT>]: took 1 ms  …… …. …….
```

Fig.13. Report generated by SOAPUI4.5.1 Web service testing tool

At last, SOAPUI returns all test case reports with done status as shown in Figure 14. For this system, it raises all malicious script and custom script for testing. This system received the done and no alert status while it checks the filters in the dummy service web page.

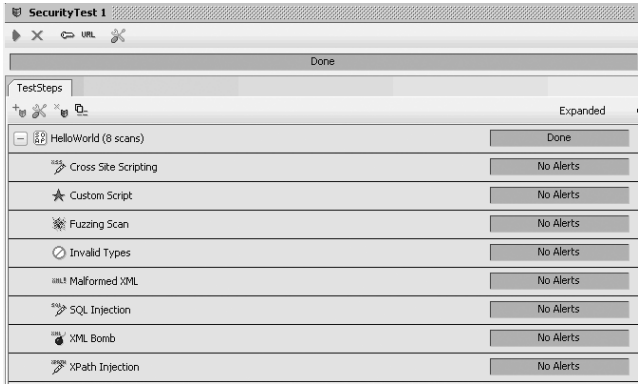

Fig.14. Tested with SOAPUI4.5.1 Test cases

The result shows that the proposed technique is more robust, when we compare this to existing methods. The proposed system took 720ms to filter the test cases of SOAPUI testing tool as shown in Table3 and Figure 15.This is comparatively lesser than the validate request filter of ASP.net Webservice and anti XSS filter of Java.

Table 3. Comparison of proposed XSS filter with the inbuilt XSS filter of Asp. net Web services and jdk1.4SDK with respect to speed

s.no	Language	Time taken to finish
1	Proposed XSS Filter	720ms
2	Validate request filter	950ms
3	AntiXSS filter	1201ms

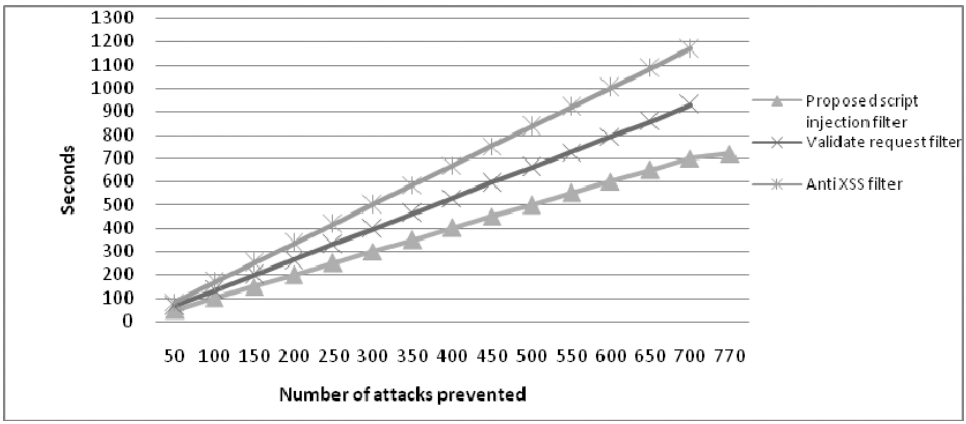

Fig.15. Comparison on number of attacks prevented in seconds

CONCLUSION

In this paper, we explained the design and implementation methods for the filtering policies. The secured system was implemented with XSS filter, oversized message filter, message replay filter, parameter tampering filter, coercive parsing filter and semantic URL attack filter. We presented a dynamic approach together with a static approach, by self-aware message analyzing algorithm against script based injection attacks on web services. It does not allow the inputs to service provider directly. Instead, it sends the request to dummy server page and analyzes the nature of the request and response. Then, input is passed on to input validator to find the malformed input. Further, attacks are analyzed with the other filters to prevent Web service based attacks. The filters were tested with valid and invalid SOAP messages. The results show that the proposed filters are capable of identifying the malicious elements in the SOAP messages and then block the messages. This can prevent various attacks such as message replay attacks, oversized payload, recursive payloads and XSS attacks. The system has to be implemented with its extended modules like authentication, digital signature and certificates for providing more security features.

ACKNOWLEDGEMENT

The authors wish to acknowledge the collaborative funding support from the UGC, Govt. of India, New Delhi. (UGC F.No. 41-1363/2012(SR)

REFERENCES

Adnan, Gutub., Abdul-Rahman, El-Shafe. & Mohammed, Aabed. 2011. Implementation of a pipelined modular multiplier architecture for GF(p) elliptic curve cryptography computation. Kuwait Journal of Science and Engineering, **38**(2B): 125-153.

Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C. & Vigna, G. 2008. Saner: Composing static and dynamic analysis to validate sanitization in web applications. Proceedings of the 2008 IEEE Symposium on Security and Privacy: 387-401.

Chuang, M. C., Lee, J. F & Chen, M. C. 2013. SPAM: A secure password authentication mechanism for seamless handover in proxy mobile IPv6 networks. IEEE Systems Journal, **7**(1) :102-113

Gundy, M. & Chen, H. 2009. Noncespaces: Using randomization to enforce information flow tracking and thwart cross-site scripting attacks. Proceedings of16th Annual Network & Distributed System Security Symposium. NDSS Symposium.

Jovanovic, N., Kruegel, C. & Kirda, E. 2006. Pixy: A static analysis tool for detecting web application vulnerabilities. Proceedings of IEEE Symposium on Security and Privacy: 258-263.

Kieyzun, A., Guo, P. J., Jayaraman, K. & Ernst, M. D. 2009. Automatic creation of SQL injection and cross-site scripting attacks. Proceedings of 30th International Conference on Software Engineering (ICSE):199-209.

Liu, H., Ning, H., Zhang, Y., He, D., Xiong, Q. & Yang, L.T. 2013. Grouping-proofs based authentication protocol for distributed RFID systems. IEEE Transactions on Parallel and Distributed Systems, **24**(7):1321-1330.

Martin, M. & Lam, M. S. 2008. Automatic generation of XSS and SQL injection attacks with goal-directed model checking.Proceedings of 17th USENIX Security Symposium: 31-43.

Minamide, Y. 2005. Static approximation of dynamically generated Web pages. Proceedings of the 14th International Conference on World Wide Web, Chiba, Japan : 432-441.

Nadji, Y., Saxena, P. & Song, D. 2009. Document structure integrity: a robust basis for cross-site scripting defense. Proceedings of 16th Annual Network & Distributed System Security Symposium, NDSS Symposium.

Negm, W. 2004. Anatomy of a Web services attack: A Guide to Threats and Preventative Countermeasures, http://www.bitpipe.com/detail/RES / 1084293354294.html

Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J. & Evans, D. 2005. Automatically hardening web applications using precise tainting. Proceedings of 20th IFIP International Information Security Conference, Makuhari-Messe, Chiba, Japan:295-307

Pete, Lindstrom, A. 2004. Attacking and defending Web services. Technical Report, Spire Security, LLC.

Shar, L. K. & Tan, H. B. K. 2012. Automated removal of cross site scripting vulnerabilities in web applications. Information & Software Technology **54**(5): 467-478.

Shen, D., Chen, G. & Jose, B. Cruz. 2007. Theoretic solutions to cyber attack and network defense problems. Proceedings of 12th International Command and Control Research and Technology Symposium.

Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C. & Vigna, G. 2007. Cross-site scripting prevention with dynamic data tainting and static analysis. Proceedings of Network and Distributed System Security Symposium, NDSS,San Diego.

Wasserman, G. & Su, Z. 2008. Static Detection of cross-site scripting vulnerabilities. Proceedings of 30th International Conference on Software Engineering. :171-180.

Xie, Y. & Aiken, N. 2006. Static detection of security vulnerabilities in scripting languages. Proceedings of the 15th USENIX Security Symposium: 179-192.

Submitted : *18/07/2013*
Revised : *12/01/2014*
Accepted : *21/01/2014*

مرشح محسن لتدقيق المدخلات عبر المواقع ردءاً لهجمات خدمات الوب

*إيلانقوفان أوما، ** أربوثاراج كنّان

*قسم علوم المعلومات والتكنولوجيا - جامعة آنا - شيناي - الهند
**استاذ دكتور ورئيس قسم علوم المعلومات والتكنولوجيا - جامعة آنا - شيناي - الهند

خلاصة

يحتاج الجميع في هذه الايام الى معالجة البيانات الحساسة مثل تفاصيل الحسابات المصرفية الشبكية وبيانات اخرى تابعة للمعاملات المالية على شبكة الانترنت. في هذا السيناريو، تستهدف العديد من الهجمات مثل هجمات الحقن هذه البيانات الحساسة، وتنفذ هذه الهجمات بواسطة وصلات برمجية يتم تشغلها على حواسيب المستخدمين، وتسمح الاخيرة للثغرات الامنية في صفحات العميل / الخادم. وتشغل هذه الهجمات برمجيات خبيثة هدفها سرقة المعلومات الشخصية المخزنة على الخادم. بينما يستطيع المهاجم تطوير مثل هذه البرمجيات بسهولة، الا انه يصعب جدا صدها باستخدام مرشحات الوصلات البرمجية عبر المواقع الحالية وذلك بسبب محدودية قدرة اكتشافها بسهولة. لهذا تعد هجمات الوصلات البرمجية عبر المواقعتحدِ صعب امام مستخدمي الشبكة. وعليه يجب اكتشاف وصد هجمات الحقن بواسطة طرق فعالة. ولكن معظم الطرق الحالية تنقصها هذه القدرة وتحتاج الى المزيد من التحسين. في هذا البحث، تم اقتراح لوغارثم جديد على دراية بذاته لتحليل وتدقيق الرسائل بهدف الكشف عن وفلترة انواع مختلفة من الهجمات على خدمات الوب. يستلم هذا النظام المقترح الطلبات ويقوم بإعداد الرد المناسب من صفحة خادم وهمية وذلك للتعرف على طبيعة الهجمة. هذا وسوف يتم استحداث منهجيات جديدة في هذا البحث لتحليل الرد وتوجيه الطلب المشروع (أي الخالي من الهجمات) لصفحات الوب الاصلية. لقد تم اختبار المرشحات المقترحة على مختلف انواع الهجمات، وذلك لتأكيد متانة منهجيات الترشيح. تبشر نتائج الاختبار أن منهج الترشيح المقترح ذو قدرة عالية على ترشيح الرسالة الملغومة. كما تم اثبات دقة وقدرة الاسلوب المقترح من خلال فحص مكثف لتطوير وتحليلوصلات برمجية تابعة لمواقع حقيقية. تدل النتائج المستوحاة من هذه

الدراسة ان منهج الترشيح المقترح قوي جدا وقادر على تصفيةالرسالة الملغومة والتي تحتوي على هجمات مثل الوصلات البرمجية عبر المواقعو الحقن و الرد على الرسالة و الهجمات المعنوية. لقد استعرضنا دقة وبرمجة اسلوبنا في اختبارات مكثفة على مواقع من العالم الحقيقي متضمنة على وصلات برمجية مكشوفة.

كلمات مفتاحية: الحماية، مرشح الهجمات المعنوية، خدمات الوب، هجمات وصلات برمجية عبر المواقع، مرشحات وصلات برمجية عبر المواقع.

مجلة العلوم الاجتماعية

فصلية – أكاديمية – محكّمة

تصدر عن مجلس النشر العلمي – جامعة الكويت

تعنى بنشر الأبحاث والدراسات في تخصصات السياسة والاقتصاد والاجتماع والخدمة الاجتماعية وعلم النفس والأنثروبولوجيا الاجتماعية والجغرافيا وعلوم المكتبات والمعلومات

رئيس التحرير: هادي مختار أشكناني

تفتح أبوابها أمام

- أوسع مشاركة للباحثين العرب في مجال العلوم الاجتماعية لنشر البحوث الأصيلة والاسهام في معالجة قضايا مجتمعاتهم
- التفاعل الحي مع القارئ المثقف والمهتم بالقضايا المطروحة.
- المقابلات والمناقشات الجادة ومراجعات الكتب والتقارير.
- تؤكد المجلة إلتزامها بالوفاء والانتظام بوصولها في مواعيدها المحددة إلى جميع قرائها ومشتركيها.

توجه جميع المراسلات إلى:

رئيس تحرير مجلة العلوم الاجتماعية

جامعة الكويت

ص.ب 27780 الصفاة، 13055- الكويت

تليفون: 00965-4810436

فاكس 4836026

E-mail:JSS@kuc01.kuniv.edu.kw

الاشتراكات

الكويت والدول العربية		الدول الأجنبية	
أفراد	3 دنانير سنوياً ويضاف إليها دينار واحد في الدول العربية	أفراد	15 دولاراً
مؤسسات	15 ديناراً في السنة 25 ديناراً لمدة سنتين	مؤسسات	60 دولاراً في السنة 110 دولارات لسنتين

تدفع اشتراكات الأفراد مقدماً نقدا أو بشيك باسم المجلة مسحوباً على أحد المصارف الكويتية ويرسل على عنوان المجلة، أو بتحويل مصرفي لحساب مجلة العلوم الاجتماعية رقم 07101685 لدى بنك الخليج في الكويت (فرع العديلية).

Visit our web site: http://pubcouncil.kuniv.edu.kw/jss