International Journal of Soft Computing 9 (5): 280-284, 2014
ISSN: 1816-9503

Automatic Prevention of Cross Site Scripting Attacks Using Intermediary Filter for Web Services

E. Uma and A. Kannan
Anna University, Chennai, Tamil Nadu, India

Abstract: The internet has used widely by the users to handle sensitive data like credit card id, account id and pan number. Many scripting attacks like Cross Site Scripting (XSS), parameter tampering and buffer over flow attacks are targeted to the sensitive data. This XSS attacks are executed by the user's interface. These attacks can be used to run malicious code on or steal personal information in web. The XSS attacks are challenging issue for the internet user because it is easily generated by the attacker but very tough to prevent it from an input filter because of its lacking techniques in the existing systems. In this study, new filtering policy has been proposed for detecting and filtering attacks. The system has been implemented with intermediary services to segregate the untrusted data and trusted data from the input. The proposed XSS filter tested with all possible attacks for verifying the robustness of filtering policy. The results show that the proposed filtering policy is very strong to refine the malicious SOAP message which contains attacks such as XSS. Researchers demonstrated the implementation and accuracy of the approach through extended testing using real-world cross-site scripting exploits.

Key words: Web services, XSS attacks, XSS filters, SOAP, India

INTRODUCTION

OWASP (2010) reports that cross site scripting is one of the common web security flaws. It is a type of code injection vulnerability that enables attackers to send malicious scripts to web clients. It happens when the server side or client side code have not been implemented with input validation. An attacker can embed malicious script through HTML pages and it is stored in server's database. When a client visits the exploited web page unknowingly then the malicious script gets executed from server's page. The malicious script used in an XSS attack can be any kind of client side scripts such as HTML, VBScript, JavaScript and Flash can be interpreted by web browsers. XSS attacks can cause severe security violation such as account hijacking, data theft, cookie theft and poisoning, web content manipulation and denial of services (CWE, 2010).

The W3C defines a web service as a software system designed to support interoperable 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 is available to all internet users. An attacker can read the information like data type, maximum length and sensitive data such as card number and password. Then, he generates Cross Site Scripting (XSS) attacks easily (Negm, 2004). In general, all attackers are using the client side graphical user interface to generate attacks. In Fig. 1, the attacker generates the malicious scripts to server side database by sending request from client side.

Because the parser identifies the malicious script coming from legitimate client side script. The initial state is the data state. When the "<" character is encountered, the state is changed to tag open state (Garsiel and Irish, 2011). Consuming an a-z character causes creation of a start tag token, the state is changed to tag name state. The parser maintains this state until the ">" character is consumed. Each character is appended to the new token name. In the case the created token is an HTML token.

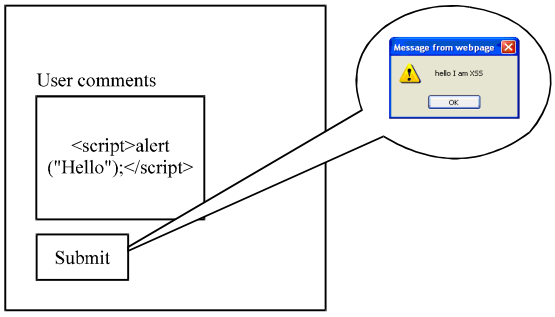

Fig. 1: The web page attacked by an attacker

Corresponding Author: E. Uma, Anna University, Chennai, Tamil Nadu, India

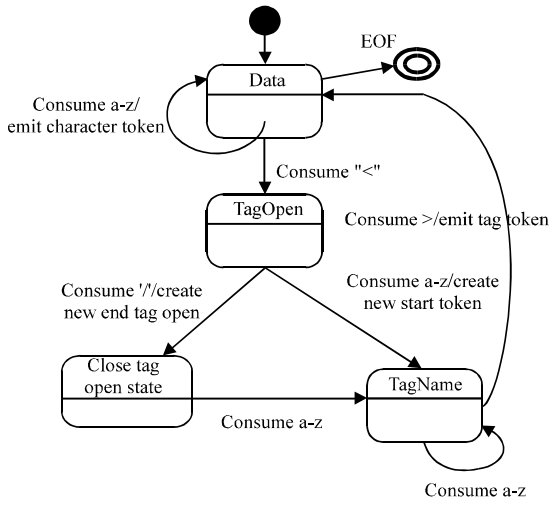

Fig. 2: Tokenizing the attacker's input

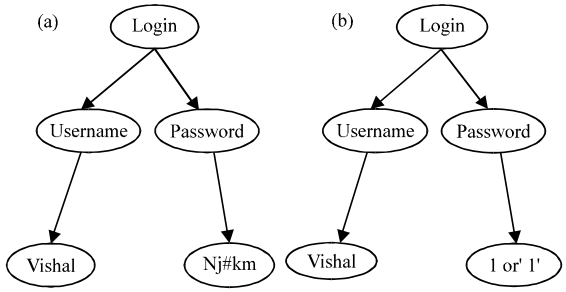

Fig. 3: a, b) Expected input from user

When the > tag is reached, the current token is emitted and the state changes back to the data state. The <body> tag will be treated by the same steps. So, far the html and body tags were emitted. The parser now go to the data state. Consuming the H character of Hello world will cause creation and emitting of a character token, this goes on until the < of </body> is reached. Researchers will emit a character token for each character of Hello world. The parser moves to the Tag open state. Consuming the next input/will cause creation of an end tag token and moves to the Tag name state as shown in Fig. 2.

If the script is accepted and stored in the database present in the server will cause big destruction to the website. Therefore, it is proposed to design XSS filter which can be used to overcome these shortcomings by filtering SOAP messages instead of packets (Lindstrom, 2004). The incoming SOAP message is filtered by parsers in this research, instead of filtering the packets received during transmission. To diminish the threats raised by XSS attackers so many solutions have been proposed.

Another type of attack is injecting sql command through client's user inter face. This command will be accepted and the other table entries are esily retrived by an attacker as shown Fig. 3a and b.

XSS ATTACKS ON WEB SERVICES

Proposed system: The proposed architecture of XSS filter has been implemented with three major components in C#. They are server, database and filtering policies. 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 DDOS attack and other attack mentioned above. This vulnerable system should be built with robust security architecture. Through the proposed XSS filter architecture, the malformed input has been refined by the filtering policies. In the XSS Filter System, the script node containing malicious message has blocked by the filter, otherwise the valid SOAP message has processed and responded by the web method in the web services. The filtering policies are built with SAX parser. This parser is faster than DOM parser. Thus, the system has implemented with SAX parser to improve the filtering speed of the XSS filter. The proposed architecture diagram of an XSS filter is shown in Fig. 4.

The proposed system has been implemented with service requester, service provider and intermediary service. The intermediary service receives the request from the client then it forwards the request to service provider as shown in Fig. 5. The intermediary service has been designed with Message Segregator, ASCII Character Converter, Http Response Checker and Blacklist Container. The Message Segregator separates the trusted and the untrusted data. In this study, the untrusted data are treated as ut data. The data which is not holding any special characters are treated as trusted data and the opposite one has treated as ut data. The ut data are assigned with identification number in to the mapping table. Then, identification number of the ut data will be sent to the service provider for future reference as shown in Fig. 6. The attacker can send other types of script code such as hex decimal, binary etc. The sample of hex decimal attack is shown as unexpected encoding types attacker's input:

The hex value of attacker's input
3c:49:4d:47:20:53:52:43:3d:22:6a:61:76:61:73:63:72:69:70:74:3a:61:6c:65:72:74:28:27:58:53:53:27:29:3b:22:3e
Exact ASCII value of above input
<IMG SRC = "javascript:alert('XSS');">

The ASCII character converter identifies and converts the request format int ASCII format then it is

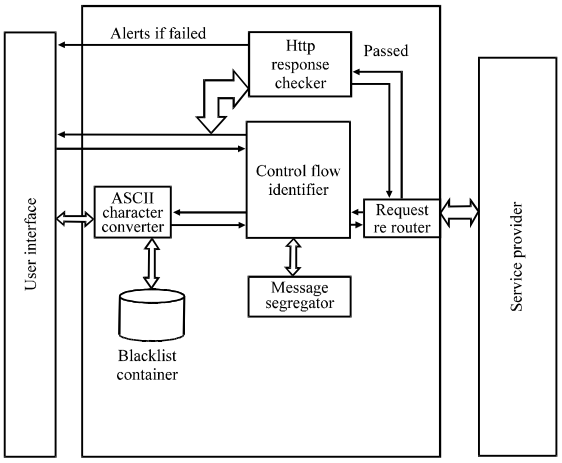

Fig. 4: Architecture of intermediary cross site scripting filter for web services

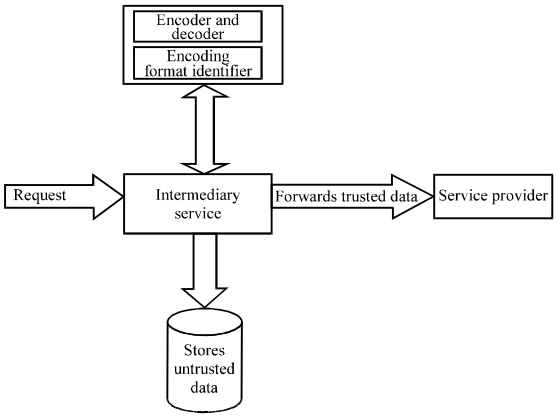

Fig. 5: Segregation of trusted and untrusted data

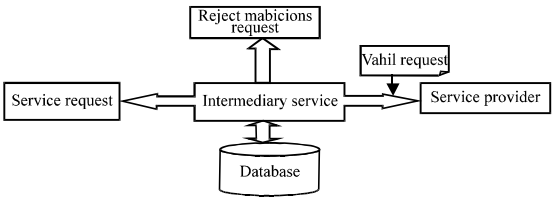

Fig. 6: Architecture of the proposed system

compared with blacklist container. If it finds malicious script then it sends report to the intermediary service for rejection of the malicious request.

It tests the ut data with its sample data in the, database. Here, the http-response header is used to verify the response of the suspected data. Then, the size of data returned to the client is checked by http-content length and cache control of http-response header. Then, the intermediary service compares the previous data with

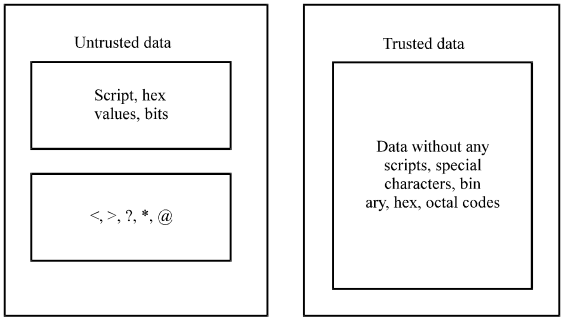

Fig. 7: Classification of data

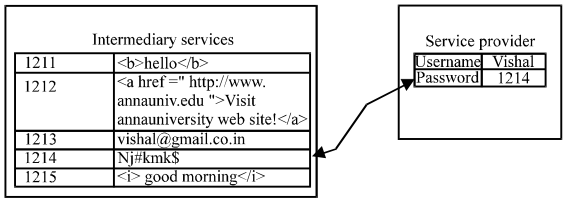

Fig. 8: Untrusted data mapping of the intermediary and service provider

current data for content modification. Then, the intermediary service decides the ut data to be rejected or accepted.

The intermediary service sends the reference id of the data to be stored in service provider. It does not allow the original ut data to be stored in a service provider. The service provider gets the ut data from the intermediary through the reference id of the ut data.

The ut data are stored in the intermediary service as shown in Fig. 7 and 8. The service provider links the ut data with its reference when ever it needs. So, the ut data gets indirect access after the validation of intermediary service. Here, the response communication will be validated through intermediary service and then it forwards the response to the client.

Advantages of proposed system: In the proposed system, the XSS filter has installed in intermediary service and it has implemented by SAX parser. The system will not allow ut data to server's database. There is no need to install any application on client side page. The intermediary service decides what to be done for the ut data. So, it is a highly automated system. Then, the system has been developed with encoding type identifier to prevent binary, hex decimal and octal attacks.

LITERATURE REVIEW

Shar and Tan (2012) developed Safer XSS tool to identify XSS attacks. This system has been developed

with static analysis and pattern matching techniques. They prevent scripting execution through escaping mechanism. It prevents all types of attacks except DOM based XSS attacks. But it fails to prevent hex, binary types of attacks.

Kirda *et al*. (2009) proposed Noxes tool in Client-side cross-site scripting protection to prevent XSS attacks. Noxes acts as a web proxy and uses both manual and automatically generated rules to mitigate possible cross-site scripting attempts. It does not prevent intruder attack bet the server and legitimate user.

Mike Ter Louw implemented BLUEPRINT: robust prevention of cross-site scripting attacks for existing browsers on IEEE symposium on security and privacy 2009. This system is vulnerable to hex decimal, binary types of attacks.

RESULTS

The proposed XSS filter is tested against with list of known XSS attacks as given in Table 1. The XSS attack prevention takes 1 or 2 msec, it depends its complexity to find the XSS attributes in XSS attack. The experiments showed that proposed XSS filter can detect about 100% XSS attacks. The experiment also showed that the implementation uses small amount of CPU time and memory space of the system.

The filter has compared with existing mechanism by the factor of non ASCII format attacks, intermediary protection and trusted data segregation. Many other systems were failed to refine non ASCII attacks like hex, binary attacks of cross site scripting. There is no segregation of important data has tabulated in Table 1.

The proposed system has been tested by SOAP UI tool. The service has located and several attacks were sent from testing tool to verify the robustness of the service. It is checked by 1000 various types of cross site scripting attacks and took 811 msec to filter the attacks. The report generated by the testing tool is displayed in Fig. 9.

Table 1: Comparison of proposed system with existing schemes

Systems	Non ASCII format attacks	Intermediary protection	Trusted data segregation
Lwin khin shar	No	No	No
Engin Kirda	No	No	No
Mike Ter Louw	No	No	No
Proposed system	Yes	Yes	Yes

Table 2: Results for SOAPUI, xssed and cheatsheet

Datasets	Proposed XSS filter	Validate request filter	Anti XSS filter
SOAPUI testing tool	1000/1000	756/1000	800/1000
xssed	600/600	547/600	500/600
Cheatsheet	100/100	85/100	90/100

PROTECTION EVALUATION

Researchers tested the proposed and existing filters against three sources of XSS attack data that have been widely 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.

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, researchers created a web page

```
SecurityTest started at 2013-01-10 17:54:36.312
Step 1 [HelloWorld] No Alerts: took 811 msec
SecurityScan 1 [Boundary Scan] Skipped, took = 0
[Cross Site Scripting] Request 1 - OK - [uma=<META HTTP-EQUIV="refresh" CONTENT="0;
URL=http://;URL=javascript:alert('XSS');">]: took 3 ms
[Cross Site Scripting] Request 2 - OK - [uma=<META HTTP-EQUIV="refresh"
CONTENT="0;url=data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3Njcmlw]: took 4 msec
[Cross Site Scripting] Request 3 - OK - [uma=<META HTTP-EQUIV="refresh"
CONTENT="0;url=javascript:alert('XSS');">]: took 3 msec
[Cross Site Scripting] Request 4 - OK - [uma=¼script¾alert(¢XSS¢)¼/script¾]: took 4 ms
[Cross Site Scripting] Request 5 - OK - [uma=<IMG SRC="livescript:[code]">]: took 4 ms
[Cross Site Scripting] Request 6 - OK - [uma=<IMG SRC="mocha:[code]">]: took 3 ms
[Cross Site Scripting] Request 7 - OK - [uma=<IMG SRC='vbscript:msgbox("XSS")'>]: took 4 msec
[Cross Site Scripting] Request 8 - OK - [uma=<STYLE>li {list-style-image:
url("javascript:alert('XSS')");}</STYLE><UL><LI>XSS]: took 8 ms
[Cross Site Scripting] Request 9 - OK - [uma=<XSS STYLE="behavior: url(xss.htc);">]: took 3 msec
[Cross Site Scripting] Request 10 - OK - [uma=<STYLE>BODY{-moz-
binding:url("http://soapui.org/xssmoz.xml#xss")}</STYLE>]: took 4 msec
[Cross Site Scripting] Request 11 - OK - [uma=<META HTTP-EQUIV="Link"
Content="<http://soapui.org/xss.css>; REL=stylesheet">]: took 3 msec
…… ……
```

Fig. 9: Sample test report from soap ui testing tool

with multiple XSS vulnerabilities and tried attack vectors from the XSS Cheat Sheet (Hansen, 2008), a well-known and off-cited source for XSS filter circumvention techniques. The proposed XSS filter refines all 350 test cases generated by SOAPUI also filters attacks referred by xssed (400 attacks) and cheatsheet (20 attacks).

The result shows that the proposed technique is more robust when researchers compare this to existing methods. The proposed system has taken 811 msec to filter the test cases of SOAPUI testing tool is shown in Table 2. This is comparatively lesser than the validate request filter and anti XSS filter.

CONCLUSION

The proposed input filters for protecting the web services overcome all scripting attacks. The proposed input filter mainly concentrates on attacks generated specifically for web services. In this study, researchers explained the implementation methods for the filtering policies. The secured system has implemented with XSS filter. The filter has been tested with valid and invalid SOAP messages of XSS attacks. The results show that the proposed filter is capable to identify the malicious elements in the SOAP messages and blocks the messages.

ACKNOWLEDGEMENTS

Researchers would like to acknowledge the financial support provided by University Grant Commission, Govt. of India, New Delhi (UGC F.No. 41-1363/2012(SR).

REFERENCES

CWE, 2010. CWE-79: Improper neutralization of input during web page generation (Cross Site Scripting). http://cwe.mitre.org.data/definitions/79.html.

Garsiel, T. and P. Irish, 2011. How browsers work: Behind the scenes of modern web browsers. Slate, Slate, August 5, 2011. http://taligarsiel.com/Projects/howbrowserswork1.htm.

Hansen, R., 2008. XSS (cross-site scripting) cheat sheet-Esp: For filter evasion. http://ha.ckers.org/xss.html.

Kirda, E., N. Jovanovic, C. Kruegel and G. Vigna, 2009. Client-side cross-site scripting protection. Comput. Secur., 28: 592-604.

Lindstrom, P., 2004. Attacking and defending web services. A Spire Research Report, January 2004, www.spiresecurity.com.

Negm, W., 2004. Anatomy of a web services attack: A guide to threats and preventative countermeasures. White Paper. Forum Systems Inc., Boston, MA. http://csis.org/files/media/csis/insights/anatomy_of_attack_wp.pdf.

OWASP, 2010. OWASP top ten project: The ten most critical web application security risks. The Open Web Application Security Project. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

Shar, L.K. and H.B.K. Tan, 2012. Automated removal of cross site scripting vulnerabilities in web applications. Inform. Software Technol., 54: 467-478.