Protecting AJAX Code Using Secure Communication

Rami Al-Salman
Department of Computer Engineering,
Jordan University of Science and Technology,
Irbid, 22110, Jordan
Ramialsalman9@gmail.com

Mohammad Fraiwan, Natheer Khasawneh
Department of Computer Engineering,
Jordan University of Science and Technology,
Irbid, 22110, Jordan
{ natheer,mfraiwan}@just.edu.jo

***Abstract*— AJAX is a ubiquitous technology that empowers Web applications by facilitating communication with the server side of the Web transaction. The drawback of this enabling technology is that malicious exploits can use this AJAX "back door" to communicate on behalf the client/server and steal users' information. In this paper, we present a technique that will protect the AJAX-enabled communication. The method works by encrypting the URL and content using the low overhead Tiny Encryption Algorithm. Authentication of the content is done via 3rd party verification of the encrypted URL and content, and the SHA-1 digital signature of the user.**

Keywords ; AJAX, Encryption, Security, Web

I. INTRODUCTION

JavaScript (JS) is a scripting language, which is commonly used in web applications; initially it was mainly used for making the websites more interactive with the clients. However, recently, JavaScript has been injected in Web applications to do more effective tasks such as decreasing the sever-side load (e.g., JavaScript can be used to make format checks and that the input conforms to certain criteria). Moreover, JS has evolved to be more interactive not only with a client side but also with the server side. Asynchronous JavaScript and XML (AJAX) is another JS technology. The main idea of the AJAX is to allow the JS code to communicate with a server using XMLHttpRequest function call. Using XMLHttpRequest, JS can request data from the server, which could be used later in partially-loaded sections of the webpage. Therefore, the JS allows loading parts of the webpage instead of loading the whole webpage. Despite of the big benefits of AJAX, it has critical drawbacks; because most Web crawlers do not execute JavaScript code [1], the data retrieved by AJAX (or XMLHttpRequest) will not be indexed and will not appear in the search results. Another problem is related to usability; if the JavaScript or XMLHttpRequest is disabled, then this means that the clients will not be able to properly browse pages dependent on AJAX. But the most critical drawback is related to the security and confidentiality aspects of the webpage. Anyone is able to view the source code of the AJAX. That means any user could use it to communicate with a server. This weakness allows any user to steal the transferred data from the server to clients and vice versa (i.e., user maybe inject the AJAX code in his Webpage to show data from other server). Thus we propose a complete framework that protects the AJAX code using tiny encryption algorithm, SHA-1 algorithm, and third party authentication.

The contributions of this paper are as follows:

1. We protect the AJAX code (XMLHttpRequest) using complete and simple framework.
2. We conduct performance evaluation studies on various Webpage sizes to show the effectiveness of our proposed scheme.

The remainder of this paper is organized as follows: We discuss the related work in section II. The system architecture is presented in section III. The implementation and comparison experiments are described in section IV. We conclude in section V.

II. RELATED WORK

The protection of scripts which could inject vulnerabilities in web applications is presented in [4], the authors proposed a simple framework which eliminated a wide range of JS (or AJAX) scripts injection vulnerabilities in the web applications. They extended the original JS is Sandboxed by only minor browser modifications, to support other policies. New extended policies were used for preventing AJAX development frameworks such as the Dojo Toolkit, prototype.js, and AJAX.NET from cross-site scripting and RSS injection attacks.

The prevention of cross-site scripting attacks against web applications is presented in [3]. The proposed system is based on the use of X.509 certificates, and XACML for the expression of authorization policies. They gave the web developers and/or administrators the ability to set their requirements and policies at the server side.

Server-based filtering proxies strategy is presented in [2], the authors proposed the analysis and filtering techniques based on a modified PHP interpreter at the server side. Modified interpreter is supported by analysis history table, which can filter the unsafe interpreted PHP statements.

Browser-Enforced Embedded Policies scheme (BEEP) is reported in [7], BEEP gave web application developers and/or administrator the ability to inject policies inside the websites, and these prevented the unsafe JavaScript codes to run beyond the embedded policies. The authors also prove that the

browsers were required only small and localized modifications to support BEEP.

ADSandbox system [6] presented and concerns with analysis malicious websites and focuses on detecting attacks through JavaScript. Since,JavaScript does not have any built-in sandbox concept, the idea is to execute any embedded JavaScript within an isolated environment and log every critical action. Using heuristics on these logs.

III. THE SYSTEM ARCHITECTURE

We propose a simple but complete framework; it consists of three phases; 1. URL encryption 2. Generating hash value and encrypting the AJAX content URL, 3. Decrypting the AJAX URL and Generating the hash value. The complete view for the framework can be shown in figure 1.

1. URL encryption.

In the phase one, we encrypt the URL for a target server that posses the service. As can be shown in figure 2. Firstly, the target server registers it URL in the mediator (In our case called secure.php).In addition hash value is generated from the URL by the mediator. Then the mediator generates a key to encrypt the URL using tiny algorithm. Then it puts it in the AJAX code. In this way no body could know what is the target of the URL that owns the service. In addition the same key will be sent to server side code in the home page (which includes the AJAX code).

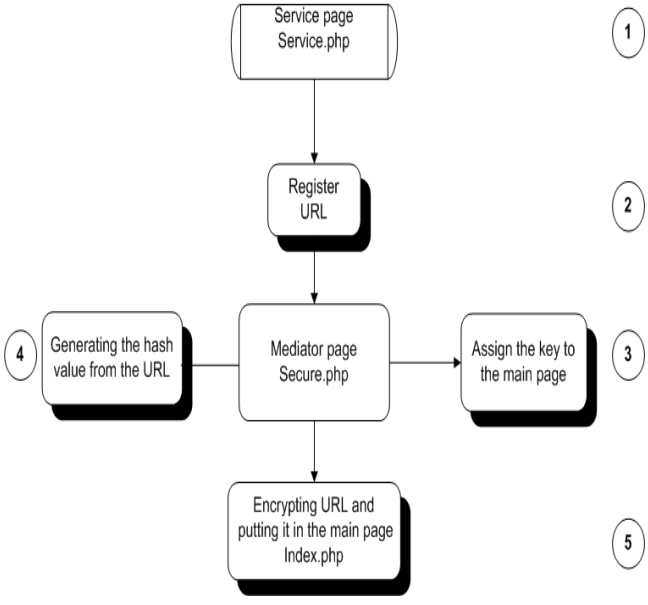
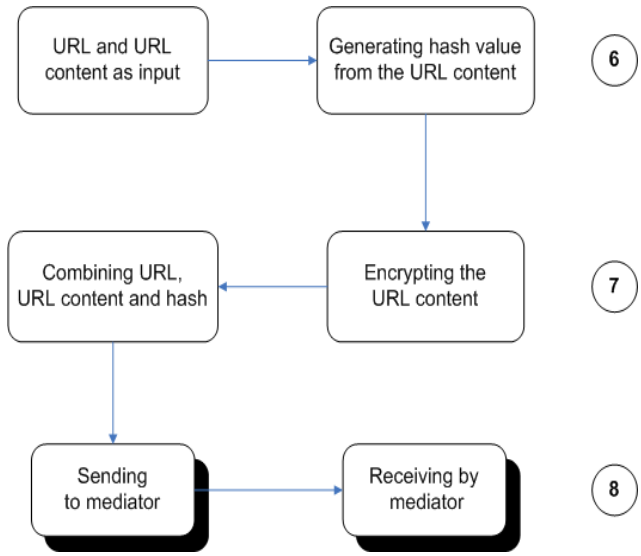


Figure 1, shows how registering, generating hash and encryption for service URL that owns the services for the clients.

2. Generating hash value and encrypting the AJAX content URL.

In the phase two, As can be shown in figure 2**,** we applied an encryption algorithm to encrypt AJAX URL content which presents the variables and values (i.e., service.php?a=100&b=200 a and b are variables, 100 and 200 are the values). After that URL and the content which already are encrypted will be sent via XMLHttpRequest() function to the mediator. The purpose of the encryption is to prevent anyone to know what is the URL content which will be sent to the server from the clients.

For authentication purpose, we generate hash value (message digest) for the AJAX URL using Secure Hash Algorithm (SHA-1) [10]. SHA-1 is a cryptographic hash function designed by the National Security Agency. It is used to produce 160-bit message digest, and commonly used as a part of authentication operation. For simplicity we aliased the message digest output as part 2. In advance step (last step in phase one), the part 1 and part 2 will be combined in one message, which will be sent to mediator (or third party).



Figures 2, shows the encryption and generating hash value from URL and URL content, and are fulfilled.

The encryption operation is fulfilled by Tiny Encryption Algorithm (TEA) [11]; The TEA is one of the fastest and most efficient cryptographic algorithms in existence. It is a Feistel cipher which uses XOR, ADD and SHIFT operations; it can encrypt 64 data bit is using a 128-bit key. Thus, we used a 128-bit key. For simplicity we aliased the Encrypted URL output as part 1. See figure 3.

```
var URL="secure.php"+"?"total="+plain+"&"+"ha="+"b";

xmlhttp.open("GET",URL,true);
xmlhttp.send();
```

a

```
var URL=0NoxoNtreWnDAPidWhiyWf/n/EjlqJTCZUhCzxU2INCqwSN/v7BbrOa/
3BY=

xmlhttp.open("GET",URL,true);
xmlhttp.send();
```

b

Fig 3.a shows how the AJAX code sends a data as a plain text; Fig 3.b shows how AJAX code sends a data as a cipher text which is encrypted by TEA.

From the figure 3.a, it is clear that anyone can take the server URL which contains the associated values of variables to communicate with the server, Thus we proposed to encrypt this URL, to prevent attackers to know what the URL that AJAX communicates through is.

3. Decrypting the AJAX URL and the Generating the hash value**.**

Phase 3, As can be shown in figure 4, is started by receiving the mediator to the encrypted output (part1, part 2) which is the final output of phase two. The mediator decrypted the cipher text using the same key (we named decrypted the cipher text as part 3) which used in the phase 1. In the next step message digest (we name it as part 4) is generated from the decrypted message by the same SHA-1 function which is used in the phase 1, 2. Then the mediator compared between part 2 and part 4, if the values are identical, the mediator communicate with target server URL and returns the results to the client(s), in the other case, access denied will be returned.

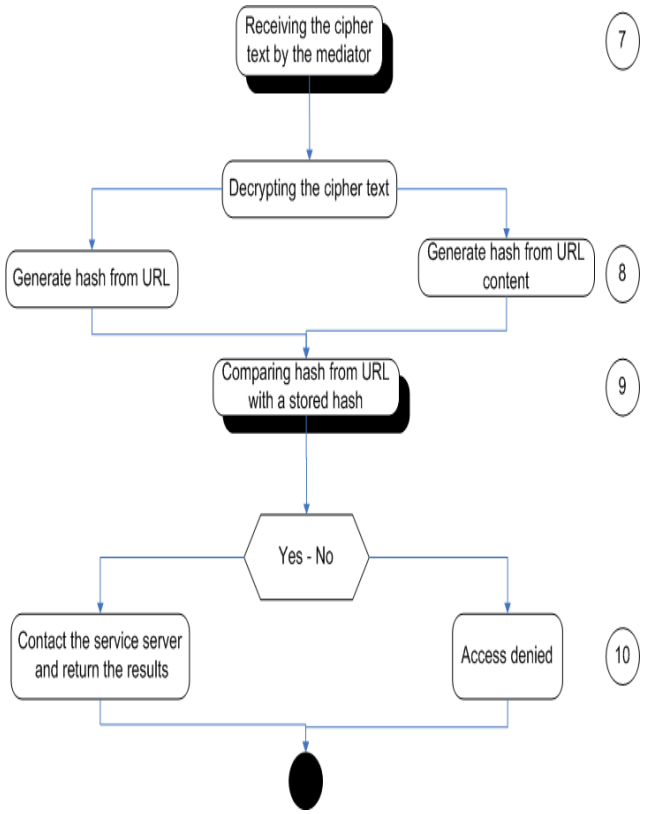


Figure 4, shows how the decryption and authentication are fulfilled.

IV. IMPLEMETATION AND RESULTS

In implementation part, we used XHTML 4.0 as Markup language and JavaScript 0.1 as client side scripting language. On the other side, we used PHP: Hypertext Preprocessor (PHP) as server side scripting language. For TEA and SHA-1, we used open source PHP codes [8] [9].

RESULTS.

In this part, we tested our framework using variant data sizes which could be sent via AJAX (XmlHttpRequest). In addition, we compared our results with normal AJAX mode (without Encryption/Decryption and Digital Signature operations). We used a time is taken to send data via AJAX and response from server as a testing measure. For each size test (i.e., 1KB test), we repeated the test 10 times, and then we got the average values for their, the repeating is necessary, the reason that the time which computed, it is affected by our local machine environment (i.e., CPU, RAM, etc...).

Table 1, shows average time which is taken to send and receive data using our framework.

Time is taken (per second) experiments using our framework			
1 KB/sec	3 KB/sec	5 KB/sec	6 KB/sec
0.015	0.029	0.039	0.047
0.012	0.032	0.049	0.058
0.017	0.030	0.045	0.047
0.015	0.031	0.049	0.049
0.019	0.030	0.037	0.047
0.016	0.023	0.034	0.039
0.020	0.022	0.043	0.057
0.011	0.021	0.036	0.062
0.021	0.026	0.035	0.054
0.019	0.028	0.037	0.046
0.0165	**0.0272**	**0.0404**	**0.0506**

Table 2 shows the results for the average time which is taken to send and receive data using normal AJAX mode.

Time is taken (per second) experiments using normal AJAX			
1 KB/sec	3 KB/sec	5 KB/sec	6 KB/sec
0.009	0.020	0.033	0.040
0.011	0.017	0.030	0.045
0.012	0.024	0.029	0.039
0.010	0.026	0.028	0.037
0.007	0.019	0.036	0.042
0.010	0.016	0.029	0.039
0.014	0.018	0.030	0.049
0.013	0.012	0.032	0.048
0.008	0.021	0.031	0.040
0.011	0.025	0.028	0.049
0.0105	**0.0198**	**0.0306**	**0.0428**

Table 3 shows the time penalty when we used our framework comparing to normal AJAX mode.

Size	**1 KB/sec**	**3 KB/sec**	**5KB/sec**	**6 KB/sec**
Avg.Time for normal AJAX	0.0165	0.0272	0.0404	0.0506
Avg.Time for our framework	0.0105	0.0198	0.0306	0.0428
Penalty	***0.0060***	***0.0074***	***0.0098***	***0.0078***

As we can see, there is no a big penalty using our framework, that means we can easily and safety integrate our framework to the current web applications. Additionally, there is no need to add any extra plug-in to the current browser.

V. CONCUSION

We build a complete framework for protecting AJAX code. We build mediator that generates hash value from the URL. In addition it encrypts and decrypts the URL and URL content. Finally we test our work using variant data sizes, to test the usability of our work.

[1] P. Andreas (2007-05-08). "Help Web crawlers efficiently crawl your portal sites and Web sites". IBM. Retrieved 2009-04-22.

[2] C. Reis, J. Dunagan, W. Helen, D. Opher, and E.Saher. BrowserShield: Vulnerability-driven filtering of dynamic HTML. In Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI),

[3] J.Garcia-alfaro and G.Navarro-arribas, Prevention of Cross-Site Scripting Attacks on Current Web Applications Greece ,Proceedings of the 2007 OTM confederated international

[4] B. Livshit is and Ú. Erlingsson. Using web application construction frameworks to protect against code injection attacks. In Proc. Programming Languages and Analysis for Security, June 2007.

[5] T. Pietraszeck. and C. Vanden-Berghe. Defending against injection attacks through contextsensitivestring evaluation. Recent Advances in Intrusion Detection (RAID 2005), pp.124–145, 2005.

[6] D. Andreas, H. Thorsten. and C.Felix. "ADSandbox: Sandboxing JavaScript to fight Malicious Websites",Symposium on Applied Computing (SAC), Sierre, Switzerland,2010.

[7] T. Jim, N. Swamy, and M. Hicks. Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. In Proceedings of the 16th International World Wide Web Conference (WWW'07), pages 601–610, 2007.

[8] http://www.php-einfach.de/sonstiges_generator_xtea.php.

[9] http://php.net/manual/en/function.sha1.php

[10] S. J. Shepherd, "The tiny encryption algorithm," Journal of Cryptologia, Vol. 31, No. 3, pp. 233–245, July 2007.

[11] H. Handschuh , L. Knudsen, and M . Robshaw, Analysis of SHA-1 in encryption mode. In Advances in Cryptology { CT-RSA '01 (2001), D. Naccache, Ed.,Lecture Notes in Computer Science, Springer-Verlag, pp. 70-83}.