

Implementation of Secure Cross-site Communication on QIIIEP

Dilip Kumar Sharma

GLA University, Mathura, UP, India
todilipsharma@rediffmail.com

A. K. Sharma

YMCA University of Science and Technology, Faridabad, Haryana, India
ashokkale2@rediffmail.com

Abstract

In today's scenario, web is expanding exponentially and internet is evolving towards presentation to social connectivity. The web is in the phase of inter operative extensive data diffusion necessitating proper information dissemination. The query intensive interface information extraction protocol (QIIIEP) is proposed to fulfill this need. QIIIEP is the novel protocol proposed by the same author for efficient retrieval of deep web information. Implementation of QIIIEP is based on cross-site communication, which is a technique to provide communication between different sites by using various client or server side methods. As the functioning of QIIIEP protocol is based on cross-site communication, so purpose of this paper is to find out efficient, safe and secure mechanism for cross-site communication for QIIIEP protocol. In this paper, the different aspects of technological implementations of different available cross-site communication techniques with reference to QIIIEP protocol are analyzed with their relative advantages, limitations and security concerns. The purpose of this analysis is to find out the appropriate solution for query words extraction from web pages so that query words can be efficiently stored on the QIIIEP server [1]. A simple and secure mechanism is proposed in which all the confidential contents are enclosed in <secure> tag. This tag ensures encryption of contents at server side and decryption at client site. It provides a simple and secure cross-site communication in thick client environment for QIIIEP server with minimum security risk.

Keywords: *Deep web, QIIIEP, XMLHttpRequest, JavaScript, AJAX, fXHR, XDomainRequest, JSONRequest, Gear, XDM, iFrame, JSONP.*

I INTRODUCTION

In recent times, the mashups concept has appeared as a core technology of the Web 2.0 and social software movement. In the past few years, web mashups have become a popular technique towards creating a new generation of customizable web applications. Mashups is the technique to create web applications that are used to combine contents from multiple sources. Mashups strengthens the creativity and functionality of web applications by adding some values as per the need of applications. Mashups is implemented through the cross-site communication but the security in the browsers, designed for the cross site communication is compromised against the functionality. Authors have proposed a query intensive interface information extraction protocol (QIIIEP) for retrieval of deep web information [1]. The proposed QIIIEP is designed for efficient deep web crawling without overburdening the requesting server. World Wide Web is a very scary place. The most common problem on World Wide Web is cross-site scripting. Cross-site scripting permits a user to fetch data from another web site, which can result in a data breach. So a secure cross-site communication is very necessary for any web application like QIIIEP.

The main thrust area in this paper is to find out simple and effective secure cross-site communication technique for QIIIEP server. All the techniques available for cross-site communication are dependent on browser's implementation, but none has established a common standard for it.

There are two main types of architectures for implementing cross site communication, namely, server-side and client-side architectures. Server side mechanism illustrated by figure 2, which integrates data from different sites by requesting contents from site at the server-side and sends the composed page on request. For example, mapping API's such as Google maps, Yahoo maps etc. are mainly based on

server-side integration [2]. In client-side architecture, as illustrated in figure 1, the browser works as the medium for direct communication between the consumer and services such as Face book API. Client side cross site communication technique is more useful when the web master of third party's web site does not want to modify server side script or the site is static in nature [3]. The objective of this paper is to analyze the various mechanisms of cross-site communication mashups to find their advantages and limitations with special reference to QIIIEP. On the basis of this analysis, two techniques are identified which are suitable for client side environment on QIIIEP.

The remaining part of this paper is organized as follows: Related work is summarized in section II with comparison of the client-side and server side mashups and different cross-site communication techniques are analyzed and comparison of different techniques for cross-site communication is presented with special reference to QIIIEP server. Based on the analysis and comparison in section II the final proposed solution for cross-site communication for QIIIEP server is presented in section III. Security concerns for cross-site communication are discussed in section IV and the results conducted on the WAMP server equipped with php v. 5.2.6 and mysql v. 5.0.51b. with future directions are discussed in section V. Finally a conclusion is given in section VI.

II RELATED WORK

Client-Side Mashups

The content or service addition takes place at the client in a client-side mashups. There must be a client-based solution for handling data with respect to size and format that the other web site returns. The client in this case is normally a web browser. The data and code contents from multiple content providers inside the same page are merged and mixed in client side mashups as shown in figure 1. The client side mashup is easy to implement and uses JavaScript library provided by originating site to facilitate the use of its service. It can perform better because a service request and response go and back directly from the browser to the mashup server. It can be implemented without any specific plug-in support. It can reduce processing load on the server because a service request and response is not passed through a server-side proxy. There is no need of modifying server side script. It will not dependent on individual server failure. It allows the use of service specific load balancing techniques. Client side mashups has no buffer to protect it from the problems associated with other sites. Browsers can only send or receive one or two simultaneous XMLHttpRequests. The secure implementation is difficult due to direct script execution on the client side [4] [5].

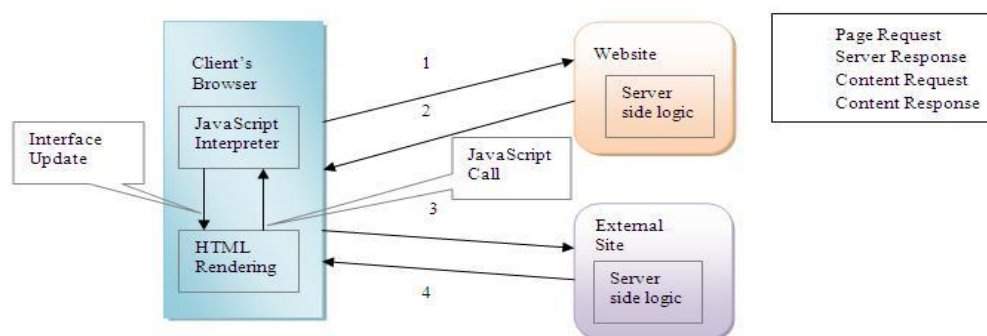


Figure 1 (Client-Side Mashups)

Server-Side Mashups

All the requests from the client go to the server in a server-side mashups. In server-side mashups, server works as a proxy to make calls to the other website. So, the work of web application client is transferred to the server as shown in figure 2. The server-side mashups have some advantages and limitations. In the server side mashups, the small chunk of the data is send to the clients. It needs to transform the data returned by a service into a different format. The proxy works as a buffer in a server-side mashups to take care of problems associated with other sites. In server side, mash up server can

cache data returned by the service. It allows manipulation of the data returned by a service or combines it with other contents before returning it to the client. Security requirements can be handled much more easily from server to server using secure protocols. But pre extraction and the requirement of complete trust on the mashup server by the client are some inherent limitations of server-side mashups. The implicit implementation of the server side proxy is required. There is a delay in receiving response because of two network hops in communication [6] [7].

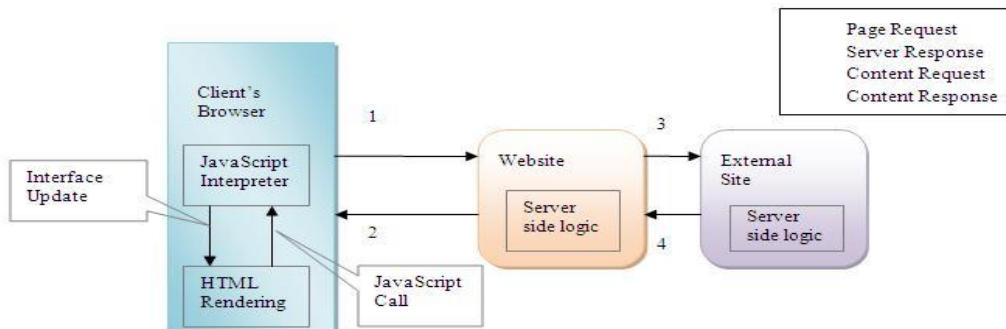


Figure 2 (Server-Side Mashups)

As stated the motive of this study to fetch query words from different sites and store it on the QIIIEP server after processing. The server side solution requires modification in the server side script that is a complex process so client side solutions are preferred. Various techniques for client side cross-site communication are illustrated in next section with reference to QIIIEP server.

Various Existing Technologies for Cross-site Communication

As the main purpose this paper is find out efficient, safe and secure cross-site mechanism for QIIIEP server so different existing technologies for cross-site communication is technically analyzed and implemented to find out their relative strengths and limitations for cross-site communication with reference to QIIIEP server.

Cross-document Messaging (XDM)

The integrity and confidentiality of communication of a mashups application with its peer's component needs to be trusted by a particular component. Removal of the need of this trust can be done through the XDM. XDM is a powerful cross-site communication feature that makes a backend request to a remote web service. This permits sites, hosting third-party iFrame-based components to correspondence directly with the parent without unsafely deviating the policy of same site origin. XDM applies for semi-trusted environments by permitting software developers to decide to receive the message from the site and reject it if the contents are unsecure or anticipated. It provides simple implementation for bi-directional cross-document communication. The advantages of XDM are that it has high performance and reliability because developers have to concentrate on task rather than technology. This technique also removes the need for embedding third-party script in the page, lessening the chance of potential information disclosure vulnerabilities like the disclosure of any sensitive data [8] [9].

Window.name Property

The window.name transport is a new technique for secure cross-site browser based data transfer and can be utilized for creating secure mashups with un-trusted sources. Window.name works by loading a cross-site HTML file in an iFrames. The window.name is set to the string content by the HTML file that should be delivered to the requester. The window.name is retrieved as the response by the requester. The requested environment is not accessible by requested response such as cookies,

JavaScript variables and document object model. It supports GET/POST request and error handling like JSONP. The window.name property is secure to similar to other frame based secure transports like subspace and fragment identifier messaging. It requires only one iFrames and one local file. It does not require any plug-ins (like Flash) or alternate technologies (like Java). The only draw back is that iFrames have some security issues because frames can change source location of other frames, but this is not a much serious issue [10] [11].

IFrame

An inline frame [6] [12] [13] [14] places another HTML document in a frame inside a HTML document. An inline frame work as "target" frame for the links defined in other elements. JavaScript in iFrame is created with same site link as host can navigate parent interior document object model structure and their properties. Conversely, the parent frame's JavaScript also navigate iFrame's elements and can see all of its contents. When the source for iFrame is different site than the host of parent frame, then host cannot see the iFrame's contents and the iFrame cannot see the host page's contents. Host can change the source of iFrame but it will not get notified that content is loaded or not. The drawbacks of this technique are that there is no acknowledgment if the iframe successfully receive the data or not and there is no provision for identification that content has loaded completely, so it doesn't know when it's safe to send the new request. The data packet carried by URL can be 40 k in Firefox and 4k in IE. The data from any source can come as contents but this will creates security vulnerability. Global state cannot be managed by iFrame because every time the page is reloaded, iFrame go through every state of loading a page.

Flash Plug-in

The use of Flash plug-in can be highly simple technique to accomplish cross-site communication. Cross-domain access is enabled by placing policy file "crossdomain.xml" on server [6] [13] [15] [16]. Security.loadPolicyFile (path) is utilized to load policies from non-root location. Communication between .swf files with different sites is allowed due to security feature. The flXHR API can be used to replace native XHR with powerful new features like cross-site communication using the features of server policy authorization such as robust error callback handling, timeouts and easy configuration/integration/adaptation. After putting flXHR in code, it replaces the usage of the native XHR and there is no need to alter the existing code because it works with same functionality. This makes it easy to adapt into all types of browsers and JS frameworks [17].

Document.domain Proxy

Domain name property is used to specify server address. It can be set to super-domain in all common browsers. Two documents with same domain and transport protocol can access each other's contents by altering document.domain property [10] [13] [18]. The document.domain proxy can work fairly well in the case of cross-sub-domain intranet mashups. It depends on browser and browser settings so it gets complicated when mashups integrates data from more than one provider. The drawback of this technique is that application require full domain name for loading.

Microsoft Silverlight

Microsoft Silverlight [19] framework is similar to Adobe Flash and used to integrate multimedia, graphics, animations and interactivity into a web application. It can be integrated in browser by using plug-in. It enables creation of sophisticated rich applications for web. The access is controlled by "clientaccesspolicy.xml" or "crossdomain.xml". Microsoft Silverlight implements security measures for cross-site communication. It uses threads for making the request. It also support socket based enabled cross-site communication.

JSONP

JSONP [20][21] is abbreviation of "JSON with padding". It is a JSON extension, where a prefix is used to specify an input argument of the call. It is now used by many web 2.0 applications such as dojo toolkit applications, google toolkit applications and web services. Further extensions of this protocol have been proposed by considering additional input arguments such as the case of JSONP supported by S3DB web services. The server is expected to return JSON data but with a slight twist. It has to prepend the callback function name, which is passed as part of the url. Script tags and calls are used by JSONP that are opened to the world so JSONP is not suitable to handle sensitive data. Some drawbacks are that script tags from remote sites allow the remote sites to inject any content into a website. If the remote sites have vulnerabilities that allow java script injection, the original site can also be affected. It is easy to implement but the SCRIPT tag request is only GET, which limits URL length. There is No error handling capability like XHR and JS Response is executed directly, which can be dangerous.

JSONRequest

JSONRequest. is a client side mechanism that will be implemented as a new browser service that allows for two-way data exchange with any JSON data server. It provides complete security in data exchange. Browser makers have to implement this mechanism into their products in order to enable the next advance in web application development. Three methods i.e. post, get and cancel are provided by JSONRequest which is a global JavaScript object. It can only send/receive JSON-encoded data. Content-type in both directions can be either application or JSONRequest. The JSONRequest is secure because it does not send any authentication information or cookies to third party. The JSON parser provides a safe alternative to the eval() method, which can allow execution of arbitrary script functions to execute [22].

Fragment Identifier

A fragment identifier [23] [24] is a short string which is appended to the URL and used to identify a portion of HTML document. In the HTML applications, <http://www.qiiiep.org/serve.html#words> refers to the element with the id attribute that has the value words (i.e. id="words") in the document identified by the URI <http://www.qiiiep.org/serve.html>, which is typically the location of the document. Fragment identifier provides a secondary resource within the primary resource. The target iFrame polling or onload event handler can be used to detect changes in the fragment. In Internet Explorer 8 a HTML 5 event "hashchanged" implemented to identify changes. The deficiency of this technique is that the URL length limits the request/response length.

Google Gears

Gears is a plug-in that extends browser to create a richer platform for web applications. Gears can be used on any sites. A number of web applications currently make use of Gears including two Google products i.e. Google Docs and Google Reader. In addition to this, Gears are used by rememberthemilk.com and zoho.com since its launching time. It uses workers which act as a thread for information retrieval from a URL. Gears provide a secure way for communication between different sites. The createWorkerFromUrl API can be used for that purpose [6] [25].

XDomainRequest (XDR)

XDR allows page to be connected anonymously with any server and exchange data by using XdomainRequest object that requires an explicit acknowledgement for allowing cross-site calls from the client script and the server. Cross-site requests require mutual approval between the web page and the server. Page can communicate with other domain by initiating a cross-site request by creating an XDomainRequest (XDR) object with the window object and opening a connection to a URL. When the

request is send, Internet Explorer 8 includes an XDomainRequest HTTP request header and in response server sends XDomainRequestAllowed response header. The data strings are transmitted by the send method for further processing after a connection is initiated [21] [26]. There is no need to merge scripting from third parties and no need for frames and proxying. It also support relative path naming and restricted access to HTTP and HTTPS destinations.

CSSHttpRequest

CSSHttpRequest is a mechanism that can be used to allow cross-site requests. It can be implemented by using the CSSHttpRequest.get(url, callback) function. Data is sterilized into the CSS@import rules which is appended after the URI that is encoded in 2KB chunks. The response is decoded and returned to the callback function as a string [27]. This works well because CSSHttpRequest does not obey the same-origin policy similar to JavaScript or JSONP. CSSHttpRequest is limited to making GET requests only but in this, untrusted third-party JavaScript cannot be executed in the context of the calling page like JSONP.

Mozilla Signed Scripts


The creation of signed scripts for Netscape and Mozilla browsers involves acquiring a digital certification from www.thawte.com or www.verisign.com. The signing tool packages the scripts into a .jar file and then this file is signed by the signing tool. The signature on the file guarantees that owner of the certificate is the author of the file. Users trust script that is signed because, when the script does something malicious, they can claim legally on the signing party. When a Netscape or Mozilla browser encounters a signed script by a .jar file, it checks the signature and allows the scripts to execute in extended privileged environment [28]. Signed scripts have a few main restrictions including limited browser accessibility and security warning. The limitation of this technology is that only Firefox or browser based on Mozilla framework can use it. But it can be act as alternative technique after detecting of the browser of client.

WebSocket

HTML 5 WebSocket represents the next advancement in HTTP communications. The HTML 5 WebSocket specifications define a single-socket bi-directional communication channel for sending and receiving information between the browser and server. Thus, it avoids the connection and portability issues of other technique and provides a more efficient solution than Ajax polling. At present HTML 5 WebSockets is the leading means for facilitating full-duplex, real-time exchange of data on the Web. Web Socket provides simple traverse from firewalls and routers and it is compatible with binary data. It also allows duly approved cross-site data exchange with cookie-based authentication [29].

Comparison of Different Techniques for Cross-Site Communication

The Table 1 given below compares the different parameters for different technologies for cross-site communications implemented on QIIIIEP server. Fourteen technologies are compared against different parameter such that cross-browser compatible, cross-site browser security enforcement, http status codes i.e. error check, support of http get and post, compatibility in html version, plug-ins requirement, cookies access and data origin identification.

Parameters Techniques 	Cross-browser compatible	Cross-site browser security enforced	Capability to receive HTTP status codes. (Error check)	Support for HTTP GET and POST	Compatibility In HTML Version	Plug -ins Requirement	Cookies Access	Capability to identify Data origin
XDM [8,9]	No	Yes	Yes	Yes	HTML 4	No	No	Yes

Window.name property [10,11]	Yes	No	No	Yes	HTML 4	No	Yes	No
iFrame [6,12,13,14]	Yes	No	No	Yes	HTML 4	No	No	No
Flash Plug-in [17]	Yes	Yes	Yes	Yes	HTML 4	Yes	Own Cookies	Yes
Document.domain proxy [10,13,18]	Yes	No	Yes	Yes	HTML 4	No	Yes	Yes
Microsoft Silverlight [19]	Yes	Yes	Yes	Yes	HTML 4	Yes	Yes	Yes
JSONP [20,21]	Yes (Server dependent)	No	No	No	HTML 4	No	No	Yes
JSONRequest [22]	No	Yes	Yes	Yes	HTML 4	No	No	Yes
Fragment Identifier [23,24]	Yes	No	Yes	Yes	HTML 4	No	No	Yes
Google Gears [6,25]	Yes	No	Yes	Yes	HTML 4	No	Yes	Yes
XDR [21,26]	No (i.e. 8,ff 3.5)	Yes	Yes	Yes	HTML 4	No	No	Yes
CSSHttpRequest [27]	Yes	No	No	Get	HTML 4	No	No	No
Mozilla Signed Scripts [28]	No	Yes	Yes	Yes	HTML 4	No	Yes	Yes
WebSocket [29]	No	Yes	Yes	Yes	HTML 5	No	Yes	Yes

Table1: (Comparison of different available technologies for cross site communication)

After critical analysis of some of available techniques to implement client side cross-site communication, it is concluded that there can be two techniques that are suitable for implementing query word submission to QIIIEP server, that are iFrame and JSONP because the basic requirement for QIIIEP query word posting is to send the content and not to receive the response unless QIIIEP site certify it, so there is no risk of vulnerability.

III PROPOSED SOLUTION

The query intensive interface information extraction protocol (QIIIEP) is the novel protocol proposed by the same author for efficient retrieval of deep web information. Implementation of QIIIEP is based on cross-site communication, which is a technique to provide communication between different sites by using various client or server side methods. One of the QIIIEP modules for query word extraction which extract the query word at the time of form submission by user is based on the proper implementation of cross site communication. This cross-site communication must be client-side because server-side cross site communication requires the changes in server-side code deployed on server. Auto query word extraction module will extract the query words supplied by users of site so that QIIIEP [1] server can extend the query word list by exploiting advantage of human curiosity of finding relevant information on that site. It is composed with two sub modules, which are described below.

Script provider:

This module is used to generate script that is embedded in third parties web page where the query interface exists. This script must be embedded at the time of form design or form id generation from QIIIEP server. The input for that module is web page URL of specific form. This module extract the information of form interface from web page and process the information of elements and generate a JavaScript code, so that at the time of user submission, the supplied query word can be received by QIIIEP server.

Query word extractor: The query word submitted on a form interface by user is forwarded to the actual sever but a simultaneous event will send these query words to the QIIIEP server too. After extraction and analysis, these are stored in to knowledge base by auto query word extractor module.

The block diagram of auto query word extraction module is shown in figure 3. It uses cross-site communication for extraction of query word from third parties web form interface.

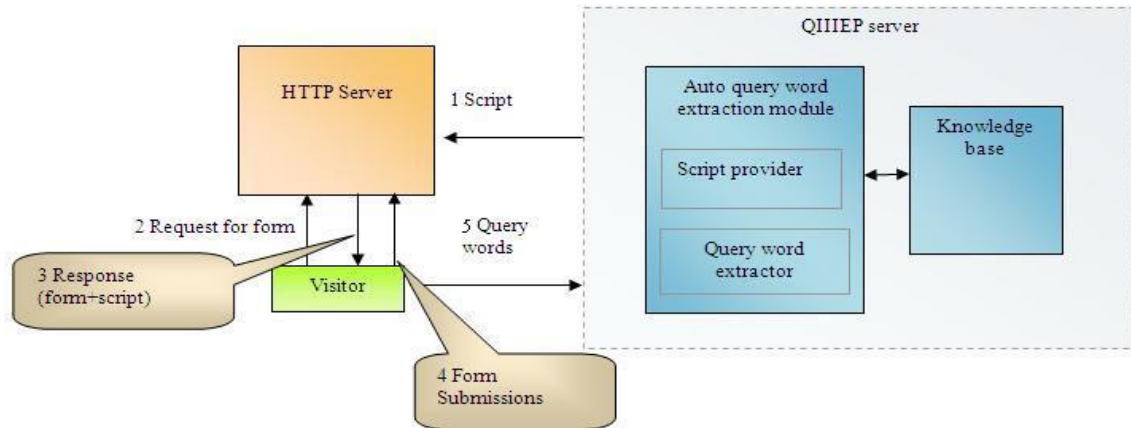


Figure 3 (Auto query word extraction module)

Block wise working of auto query word extractor module is given below.

Query word fetcher: It works for retrieving word and string entered by user at the time of manual form submission. When query words posted on originating server a simultaneous request also post the query words to the QIIIEP server.

Query word analyzer: It analyzes the retrieved query word and stores them on QIIIEP knowledge base. The analyzer adds form id, element id and other properties like time stamp, IP address, and submitted URL.

As the solutions on server side result in considerable degradation of browsing experience as data propagation consume extra time, whereas the client side solutions result in a security threat. So there is need of an efficient client side solution that does not degrade the performance and must be secure enough. The proposed system is designed in order to provide effective way to extract query word by using dynamic script element in the web page. In proposed system, a JavaScript is loaded from QIIIEP server by pointing to the URL of QIIIEP server. When the script is executed, it send back the query words to the QIIIEP server because the same-origin policy doesn't prevent dynamic script insertions and treats the scripts as if they were loaded from the site that provide the web page.

JSONP Based

The document object model (DOM) allows to dynamically creating almost any element that can be on a page, so that it can be used to create a SCRIPT tag. After setting a source, the DOM browsers try to load the script file from the source. At the time of call loadContent, the source javascript file is specified as source and this file is used to transfer the query words to the QIIIEP Server. JSONP is an efficient cross-site communication method that bypasses the same-origin policy boundaries. JSONP gives the remote side arbitrary control over the page content. But the limitation of this solution is that only get request can be used with JSONP. In the scenario of QIIIEP query word submission, all the query words of form must be encoded in get request and posted through JSONP.

iFrame based

In the second solution, forms can use the POST method and JS can submit forms. A form can be dynamically generated with the action attribute set to any URL (including pages on other sites) and encode all query parameters as hidden <input> fields. The frame size is so small that is not visible on page and when user submits the form, a JavaScript function postData() will automatically submit the form to QIIIEP server. This approach will not get any response data from the frame, so it's truly "fire and forget". The first part is the crossDomainPost() JS function. This is the function called from the script to initiate a POST request. The function accepts two arguments:

- writer_url – the URL of the script that will generate the form (given below).
- parameters – query parameters for the POST request, formatted like this : {param1 : 'value1', param2 : 'value2', ... }

As a result, the final script will have three components. The first is a JavaScript function that creates the iFrame. The second script will run inside the frame and generate and submit the form. The third component is the php script that will save the key value on QIIIEP server.

The HTML script tag is the last frontier of unrestricted access for browser-based applications. Depending on viewpoint, it either increases security risk, or a tool to make rich clients even richer. The features of JSONP and iFrame implementation are used with script for making the query word posting to QIIIEP server.

IV SECURITY MEASURES

Every day, hackers are trying to found out the new breakup mechanism. Due to increase in the number of users, there is a requirement of proper security measures, so that any type of web content uploaded by either user or administrator cannot create vulnerabilities in websites. Cross-site scripting (XSS) is a kind of web security deficiency, generally found in web applications that enable malicious attackers to insert client-side script into webpage viewed by other users. Attackers try to bypass access controls such as the same origin policy can use an exploited cross-site scripting vulnerability. The number of tags used for text formatting is very few in web applications. Therefore, simple tag filtering mechanism for web applications to protect against attacks is not sufficient. XSS vulnerabilities arise due to negligence of guidelines in web application developments. The amount of security breach depends upon the structure of the site and single patches cannot fix the XSS vulnerabilities. In order to protect legacy servers, some of the enforcement will have to implement. In fact, even the cross-site request use client side request and processing. The client must look for the header and data access and if it finds something malicious, XDomainRequest denies the header web page to load. QIIIEP server is deployed on premium hardware configuration and it uses SSL for secure transmission so it is authentic and there is no chance of embedding any malicious script on any third party web page [6] [8] [30].

V RESULTS

The experiment has conducted on a machine having Intel Core 2 Duo T5870 @ 2.0 GHz with 1 GB of RAM. This machine was running with Windows 7 OS, Test are performed using WAMP server equipped with php v. 5.2.6 and mysql v. 5.0.51b. All of the tests were performed on Firefox 3.5.4 with JRE (SUN) 1.6 and Shockwave Flash 10. To minimize measurement errors, all tests were performed multiple times. Both the technologies work fine with configuration in which two forms are used, first form have three input elements i.e. PID, Price, and Weight while the second form have Name, Age and


```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<title>My page</title>
</head>
<body>
This is demonstration of secure tag!
<secure>
ÅÇpúÕ-ŽÆÔNVòXCEòØwUéØœÚéiöë$|Y%ø?-.
</secure>
</body>
</html>
```

Figure 8 Source code of encrypted html page by secure tag

The other advantage of this technology is that no one can see the contents directly by visualizing source of the page. In spite, this concept increases the complexity of browser as well as the web server but it can be implemented as browser plug-in and server extension to utilize the model on the sites where cross-site communication is required.

VI CONCLUSION

Authors have proposed query intensive interface information extraction protocol for deep web information retrieval. Cross-site communication is very necessary for any web application like QIIIEP but the security is a very prime concern to avoid the data breach due to unsecured cross-site communication. Authors have analyzed various existing well established cross-site communication technologies for QIIIEP with special reference to security feature. After analyzing the client-side solutions including JSONP, iFrames, usage of the browser window object, and Internet Explorer 8's XdomainRequest etc., it can be concluded that both JSONP and iFrames can be proved a very powerful technique for cross site communication for QIIIEP. But the limitations of JSONP is that it does not have an error handling capability for JSONP calls and it can only support get request and explicit cancel without the possibility of restarting of the request. In iFrame, the error handling is quite complex as well as the response is not traceable. Both of the solutions are satisfactory because for QIIIEP, there is no need of explicit post request and there is no chance of malicious code in reply due to the authenticated process in QIIIEP server. After the simulation on pre composed test queries on QIIIEP server, it is found that the results meet the particular requirement of sending query word to QIIIEP server. As a future directions, a recommendation to implement <secure> tag in HTML specification is advised to diminish the same origin policy requirements, which make the cross-site communication more robust and transparent because all of the information in secure tag can only be used for the same domain.

ACKNOWLEDGMENT

All the companies/products/services names are used for identification purposes and may be the trademarks of their respective owners.

REFERENCES

[1] Dilip Kumar Sharma and A.K.Sharma, "Query Intensive Interface Information Extraction Protocol for Deep Web", In Proceedings of IEEE International Conference on Intelligent Agent & Multi-Agent Systems, PP. 1-5 ,2009. IEEE Explorer.

