IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 2 Issue 2, February 2015.
www.ijiset.com
ISSN 2348 – 7968

# Web Security: Same Origin Policy and its Exemptions

Saravanan Subramanian
[1] Technical Leader, SPVTG (Service Provider Video Technology Group), Cisco Systems Inc, Chennai, Tamil Nadu, India

Sakthivel Ramachandran Vadivel
Software Engineer, SPVTG (Service Provider Video Technology Group), Cisco Systems Inc, Chennai, Tamil Nadu, India

**Abstract**

The World Wide Web (WWW) is becoming integral part of everyone's life. Starting from browsing the online news magazine to high value bank transactions, the security of the web communication is very critical to everyone's privacy, safety and wealth. The Same Origin Policy (SOP) is a methodology enforced in the WWW to protect the users from most common security attacks. This article studies the Same Origin Policy and how Cross Origin Resource Sharing (CORS) helps to overcome those restrictions without compromising the web security.

***Keywords:*** *WWW, Security, Cloud, HTTP, HTML, REST API, CORS.*

## 1. Introduction

Security Policies enforces set of restrictions on web applications to provide safety and security for the users. But as per the software engineering philosophy, "when you fix something, you break something", this feature comes with its own limitation. These limitations block many of the genuine functions. This article focuses on the limitations enforced by the Same Origin Policy (SOP) and how to overcome them without compromising the web security.

## 2. Web Browser Security Features

### 2.1 Web Security by Restrictions

Today's modern web browsers protects the users by restricting certain functionalities, so that the script which is imported from the web into the browser, running on users personal computing device will not access other resources of the users. The web browsers impose various restrictions. They are:

File Operations: The scripts (ECMA / JavaScript or any other client side script) cannot access the local file system of the clients.

Network Operations: The client side scripts are restricted to open a network socket and establish communication with other servers.

Browser Window Operations: The client side scripts cannot create or delete a browser window. It can close a browser window only on user's confirmation. Also the scripts cannot read the contents of browser elements which are created by other servers.

File Upload: The client side script cannot set the property for "HTML File Upload" element. This restricts the script to steal some information from the client'

The browser depends on the Same Origin Policy implementation to impose few of the above restrictions. But these restrictions may be a blocker for the developers to develop a web application which can run locally on a computing device. Also there could be some genuine reasons for the server to execute the above operations on the client.

Same Origin Policy & CORS will protect the Web Apps from Cross Site Scripting and Cross Site Request Forgery requests.

**Cross Site Scripting (XSS)** is the security threat, the attackers injects executable scripts into the web application through the web application input methods.

**Cross Site Request Forgery (CSRF)**: It's a security threat where the attackers lure a victim to a web page with malicious code while the victim is logged into the target site. Then the User's browser sends HTTP requests to Web Server and performs unwanted action.

**Denial of Service**: Very common security threat, the attacker targets a web server and attacks by sending various requests to connect & retrieve the resources. In this process the attacker will exhausts all the computing, memory and network resources of the target server.

ISSN 2348 – 7968

## 3. Same Origin Policy

### 3.1 Document Origin

The origin of the document is defined by the Universal Resource Locator (URL). The URL contains the protocol, server address and port number of the server. These three properties uniquely identifies the URL, even if a single property is changed then the URL would be considered as a different URL.

i. Documents loaded from different web servers have different origins
ii. Documents loaded through different ports of the same web server have different origins
iii. Documents loaded using different protocols (http, https etc.) have different origins, even if they come from same host.

### 3.2 Script Origin

It is important to note that the origin of the script is not relevant to the same-origin policy, what is targeted here is the origin of the document in which the script is embedded.

For example, a script hosted by Host A is included in a web page served by Host B. The origin of the script is considered as Host B and the script has full access to the content of the document that contains it. If the script opens a third window and loads a document from host C into it, the same-origin policy comes into effect and prevents the scripts from accessing the document.

### 3.3 Impact of Same-Origin Policy

The Same-Origin policy is a security feature enforced both from the web client and web server. It affects the below:

i. Access restrictions on Document Object Model (DOM) HTML elements of the document loaded from other host: This comes into effect when a web page contains <iframe> element or opens other browser windows. In this case, a script attached with the HTML document can read only the properties of windows and documents that have the same origin as the document that contains the script.

ii. Scripted HTTP request to other host: A HTML document which is loaded from host A, should only send a HTTP request to host A. If the document tries to send a request to Host B, it will be restricted by the Same-Origin Policy.

## 4. Overcoming restrictions

The restrictive security features of browser helps to prevent from security attacks; it becomes a limitation in few cases.

### 4.1 Large Web Sites

Think of a large organization have a website with multiple sub web sites. For example: www.company.com might have many sub web sites like http://products.company.com to publish the products on sale. Also it can have http://vendors.company.com for its vendors to communicate them. In this scenario, the client side script in http://vendors.company.com wants to read some properties from http://products.company.com it will be restricted by Same Origin Policy.

So, in this kind of situations the web applications can set the “domain” property of the DOM to a common domain name, and then Same Origin Policy will not restrict the communication between the DOMs. So, in this example, the web applications for http://products.company.com and http://vendors.company.com both set their domain to “company.com”, will solve the restrictions imposed by Same Origin Policy.

### 4.2 Real-time communication

Consider a scenario, a web application wants to read the real time data from two different web servers and analyze to make intelligent decisions. For example, a stock market online trading services vendor wants to fetch the data from both Bombay stock exchange ( http://www.bseindia.com ) & National Stock Exchange (http://www.nseindia.com) , the CORS and other restrictive behavior of the Web Browser will not allow this type of communication.

There are bunch of new HTML5 features helps us to overcome these scenarios. They are: Server Sent Events & Cross Document Messaging. This mechanism allows the server to send messages to its clients. The clients can listen for this messages and processing of these messages can be done on the event handler for the “onmessage” on the Document Object Model.

### 4.3 CORS

Next section will bring the overview about CORS

ISSN 2348 – 7968

## 5. CORS

### 5.1 What it is

Cross Origin Resource Sharing (CORS) provides exemptions to Same Origin Policy in a controlled manner.

Cross Origin Resource Sharing (CORS) allows the scripted HTTP request to be serviced by a server where the origin of the request is different. For example, a web app in Host A, loaded from Server A sends a request to retrieve some data in Server B will be denied by default, shall be processed by implementing CORS.

### 5.2 How it works

The CORS is a standard that extends HTTP with a new *Origin:* request header and a new *Access-Control-Allow-Origin* response header. It allows the servers to use a header to explicitly list origins that may request a file or to use a wildcard and allow a file to be requested by any site. Browsers use this new header to allow the cross-origin HTTP requests with XMLHTTPRequest that would otherwise have been forbidden by the same-origin policy.

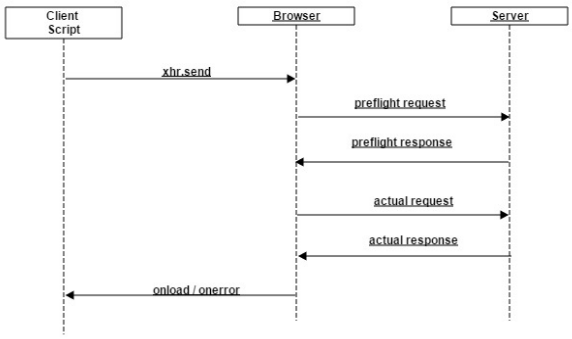


Fig. 1 pre-flight http request and response

### 5.3 Preflight HTTP Messages

For every cross origin resource request by a client, the browser internally generates pre-flight HTTP request to the server.

Below is the sample cross origin request by a client:

*PUT / cors HTTP / 1.1*
*Origin: http://web.cisco.com*
*Host: web.nds.com*
*X-Custom-Header: value*
*Accept-Language: en-US*
*Connection: keep-alive*

For the above HTTP request, the browser internally generates the pre-flight HTTP request to check whether the server is willing to server this request.

*OPTIONS /cors HTTP/1.1*
*Origin: http://web.cisco.com*
*Access-Control-Request-Method: PUT*
*Access-Control-Request-Headers: X-Custom-Header*
*Host: web.nds.com*
*Accept-Language: en-US*
*Connection: keep-alive*

If the server is configured to allow the request, then it replies as below:

*Access-Control-Allow-Origin: http://web.cisco.com*
*Access-Control-Allow-Methods: GET, POST, PUT*
*Access-Control-Allow-Headers: X-Custom-Header*
*Content-Type: text/html; charset=utf-8*

The actual request is sent only after the pre-flight request & response. The below is the sample response from the server for the original request:

*Access-Control-Allow-Origin: http://web.cisco.com*
*Content-Type: text/html; charset=utf-8*

## 6. Conclusions

In this study we have presented the security issues in the web technologies and discussed the role of same origin policy and cross origin resource sharing methodology. Though these methodologies are restrictive in nature, we discussed various ways to navigate around the limitations and develop a secure product. The information security is very critical to web applications. Though security features introduces limitations for the developers, there are proven ways to navigate around those restrictions and develop a secured product.

ISSN 2348 – 7968

## References

[1] http://www.w3.org/TR/cors/
[2] OWASP https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet
[3] Flanagan, David, "The Javascript: Definitive guide"

**Saravanan Subramanian** is with Cisco Systems and has industry experience of 17 years. In Cisco, he is an architect and product owner for Videoscape solutions and he has designed and architected web based applications and enterprise class servers for cloud environment. In his previous assignments he has worked with Ericsson Inc and Alcatel Lucent Inc. He has wide technological expertise starting from embedded OS, networking protocols to designing large scale high available systems. Saravanan has a Master Degree in Computer Applications (MCA) from Bharathidasan University, Tiruchirappalli and Bachelor of Mathematics from the same university.

**Sakthivel Ramachandran Vadivel** is with Cisco Systems for the last 7 years and has total industry experience of 12 years. In Cisco, he is Software Engineer works in Videoscape solutions. In his previous assignments he has worked with NDS. His experience includes embedded OS and Application. He is a M.Tech IT from Sathyabama University, Chennai and B.E Computer Science from Madras University.