Web Application Security Model: Same-Origin Policy

|  |
| --- |
| Copyrights 2016-2017 Frank Xu, Bowie State University.  The lab manual is based on the online lab http://www.cis.syr.edu/~wedu/seed/Labs/Web/Web\_SOP\_Collabtive/. Comments and suggestions can be sent to wxu@bowiestate.edu |

**Introduction**

(Wiki) In computing, the same-origin policy (SOP) is an important concept in the web application security model. Under the policy, a web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the same origin. An origin is defined as a combination of [**URI scheme**, **hostname**, and **port number]**. This policy prevents a malicious script on one page from obtaining access to sensitive data on another web page through that page's Document Object Model.



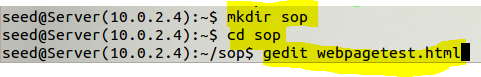
**Lab Environment**

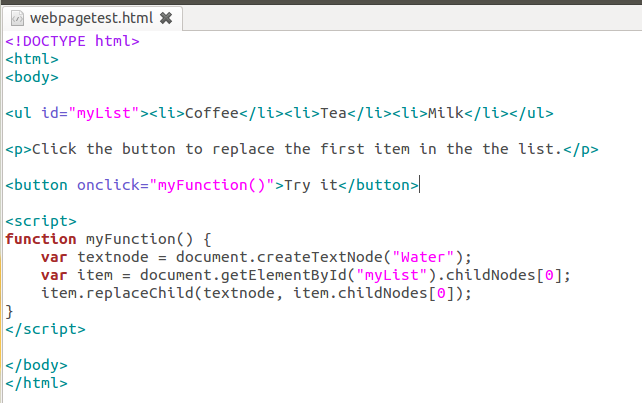
We have created two accounts in the VM. The usernames and passwords are listed in the following:

* + User ID: root, Password: *seedubuntu*.
  + Note: Ubuntu does not allow root to login directly from the login window. You have to login as a normal user, and then use the command **su** to login to the root account.
  + User ID: seed, Password: *dees*

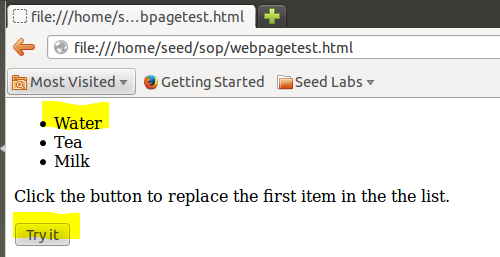
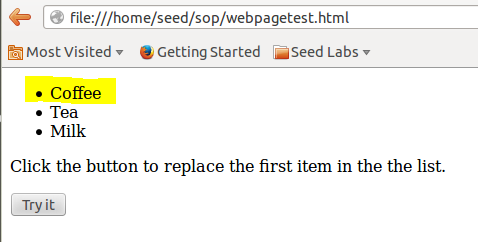
**Task 1: Updates a Web Page Dynamically Using JavaScript**

1. Create a static HTML page and named it webpagetest.html





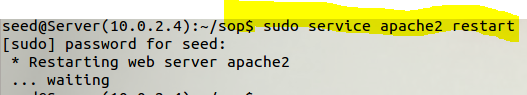
1. Open Firefox and display the HTML page.



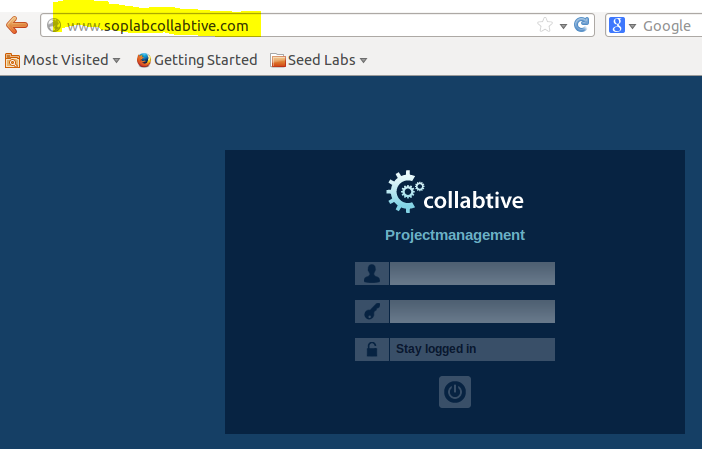
1. Questions
   1. How to add a textbox?
   2. How to add a button?

**Task 2: Find the Session Cookie of www.soplabcollabtive.com**

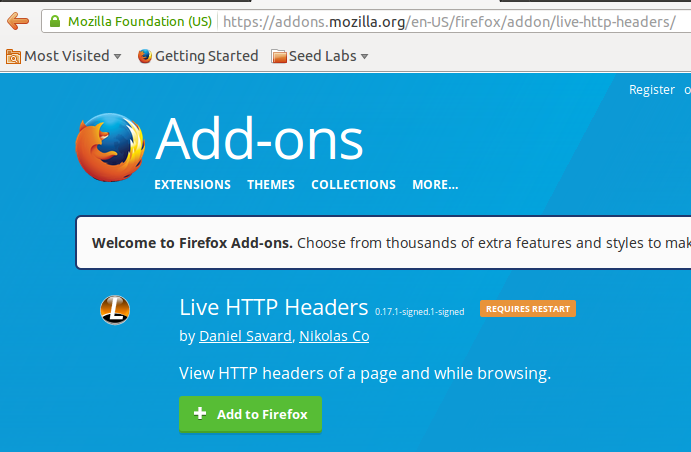
1. Restart the Apache server by running "sudo service apache2 restart".



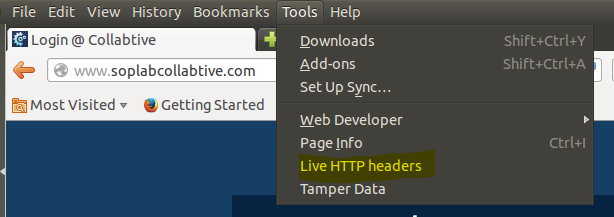
1. Start the website



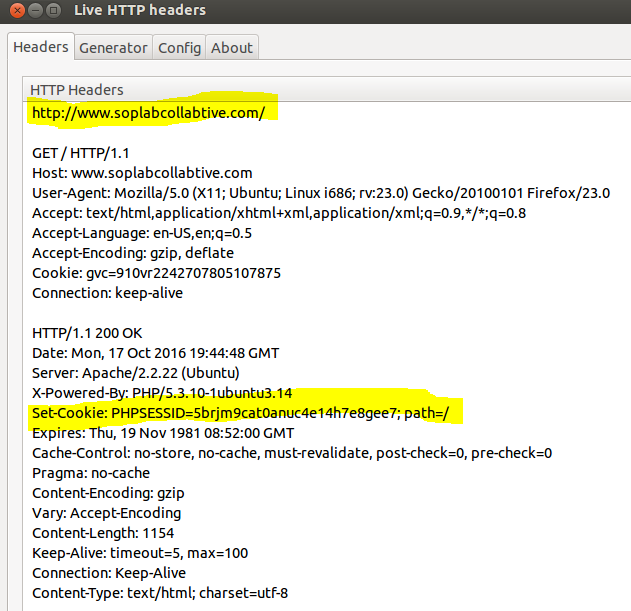
1. Add **Live HTTP header** add-on to Firefox
   1. Search for the add-on



* 1. Open Live HTTP header



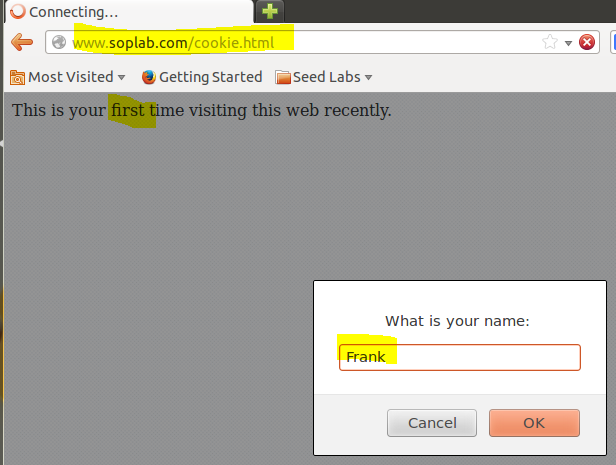
1. Live capture
   1. Close Firefox
   2. Open Firebox
   3. Start Live HTTP header
   4. Type [www.soplabcollabtive.com](http://www.soplabcollabtive.com) at Firefox Address bar­­

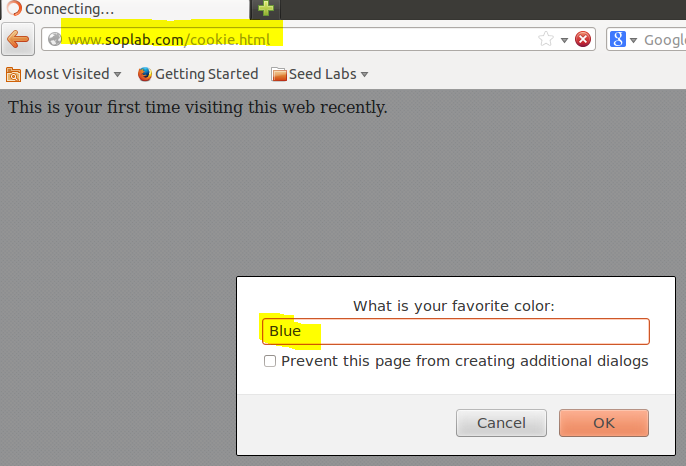


1. Questions:
   1. If you refresh (reload) the page again, will you see the Set-Cookies again? Why?
   2. Why do we need session cookies?

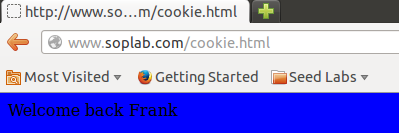
**Task 3: Set and View Persistent Cookie of www.soplab.com**

1. Save Name and Favorite Color to a cookie when you access the page the first time.

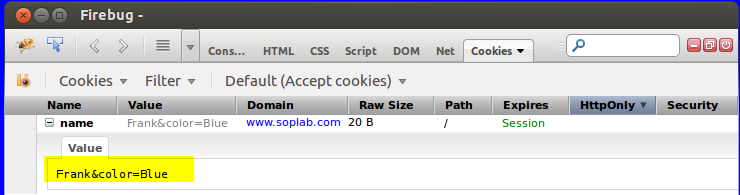




1. Use the cookie to remember you when you visit the page second time. When you refresh the page. You will see a welcome message.

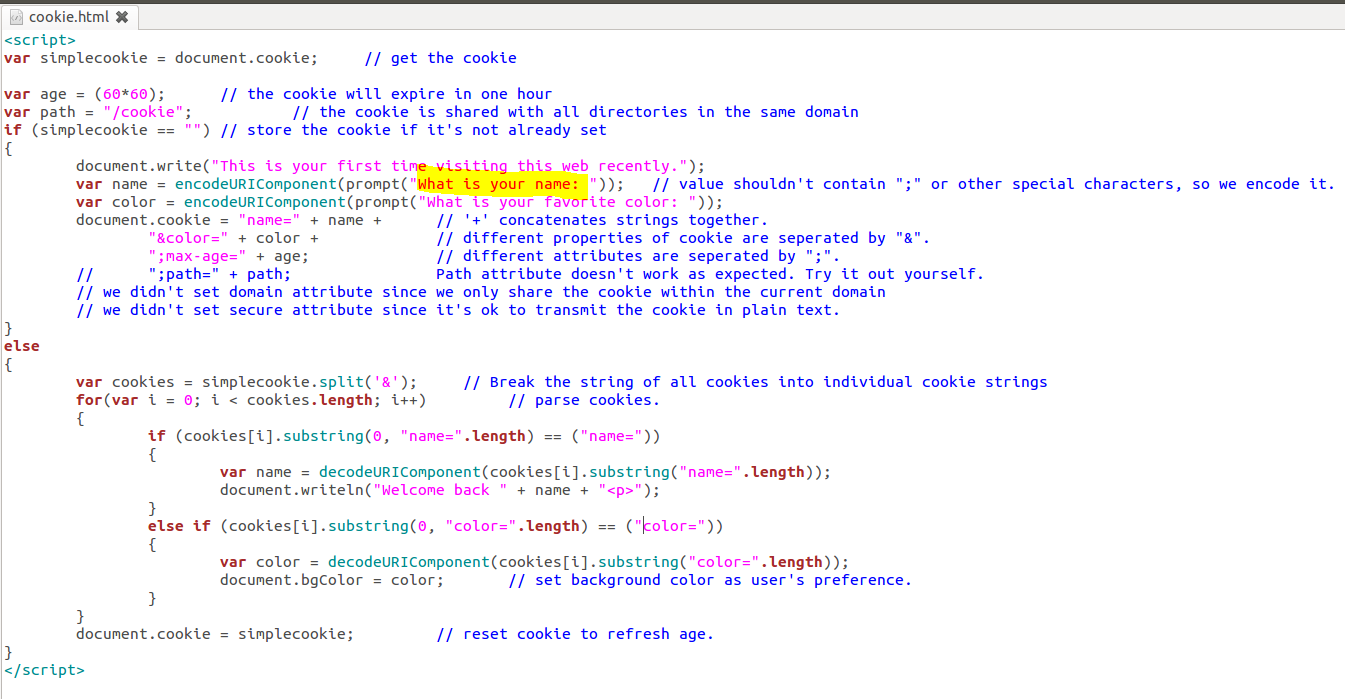


1. View the cookie use the Firebug



1. View the source code to understand how the cookie saves values

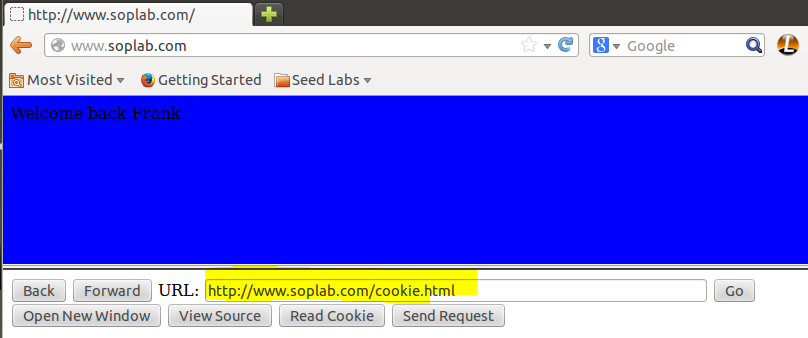




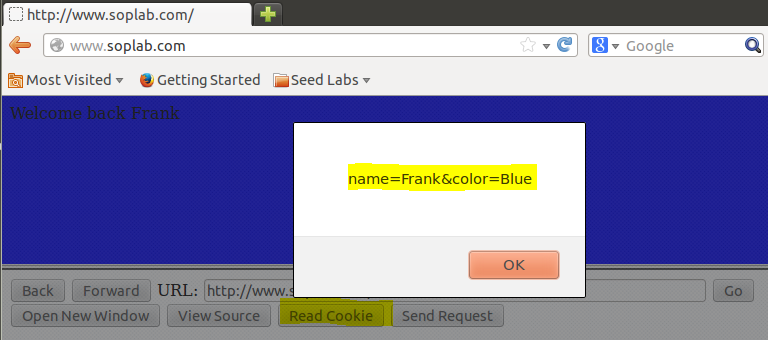
1. Question: Modify the program to trace how many times a user have visited the page.

**Task 4: Same-Origin Policy (SOP) for Cookies and DOM**

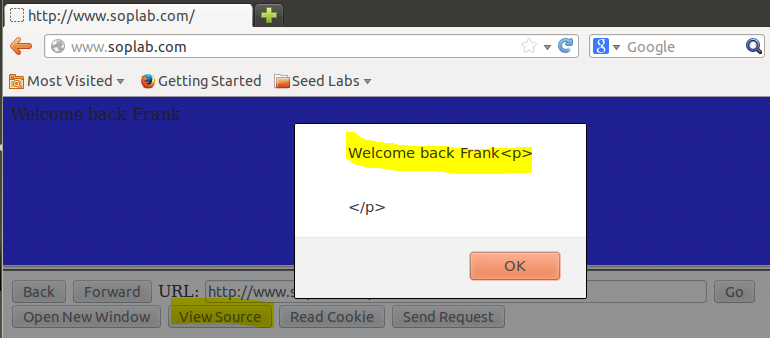
1. SOP **allows** the page www.soplab.com/index.html to access <http://soplab.com/cookie>.html



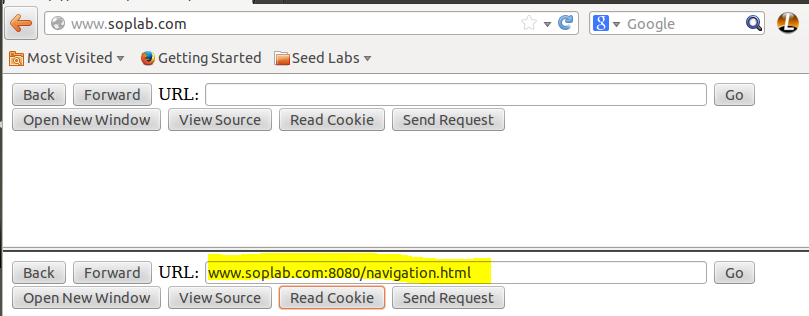
1. SOP **allows** the page www.soplab.com/index.html to access the cookies of <http://soplab.com/cookie>.html



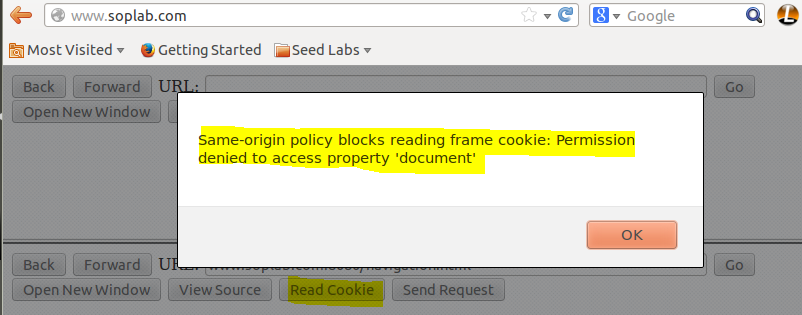
1. SOP **allows** the page www.soplab.com/index.html to access the DOM of <http://soplab.com/cookie>.html



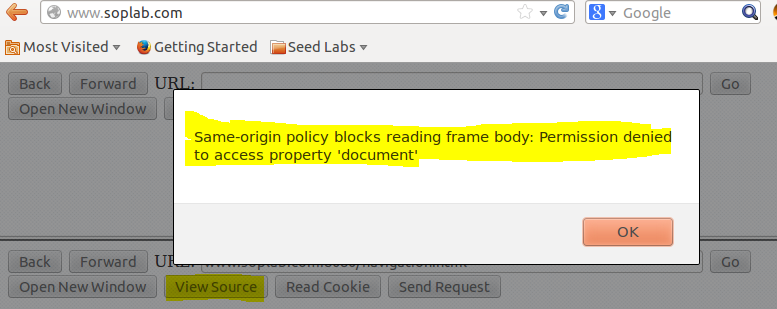
1. SOP **denies** the page www.soplab.com/index.html to access http://soplab.com:8080/navigate.html



1. SOP **denies** the page www.soplab.com/index.html to access the cookies of http://soplab.com:8080/navigation.html



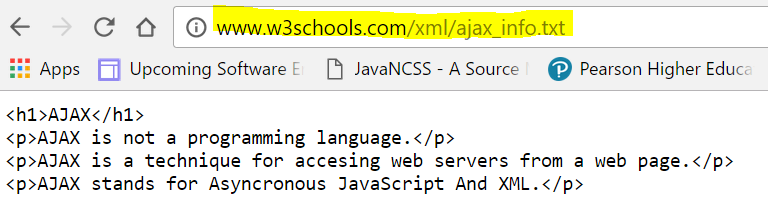
1. SOP **denies** the page www.soplab.com/index.html to access the DOM of www.soplab.com:8080/ navigation.html



1. Question: Why does a web page www.soplab.com/index.html produce different results when it tries to access the pages <http://soplab.com/cookie.html> and <http://soplab.com:8080/navigation.html>, respectively? You can access the complete source code of naviation.html by typing the command: *gedit /var/www/SOP/navigation.html*

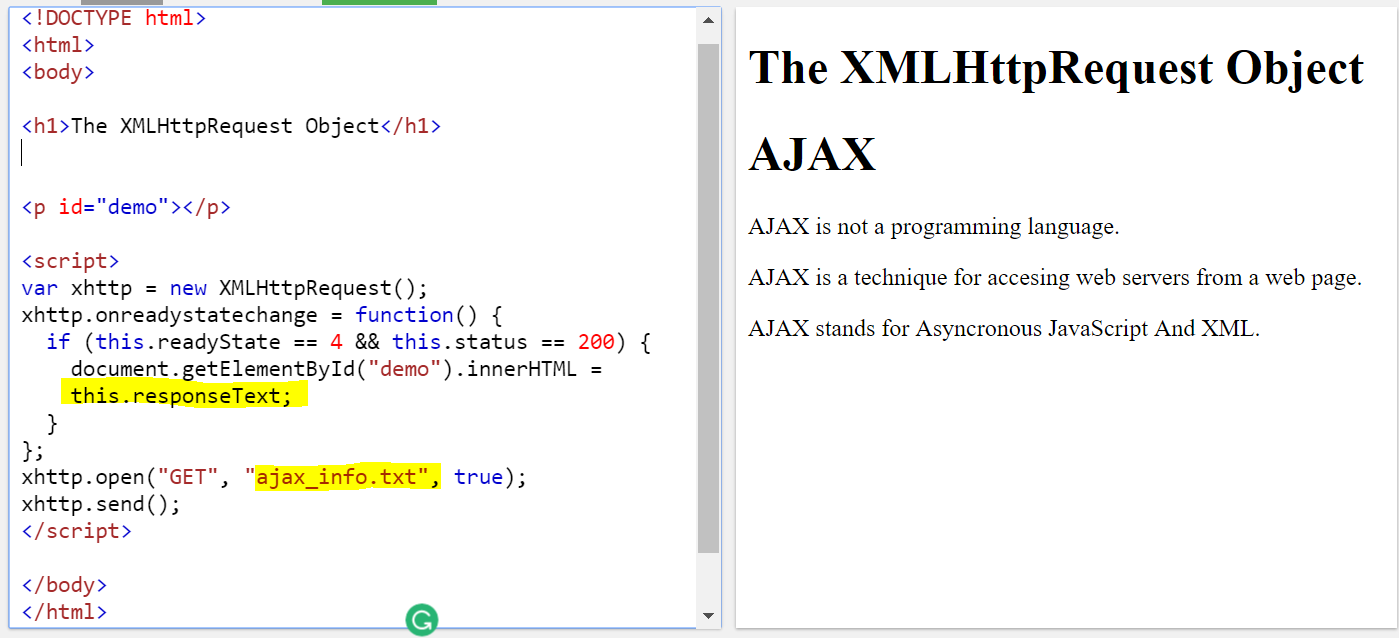
**Task 5: Same-Origin Policy (SOP) for XMLHttpRequest**

1. Display Ajax\_info.txt

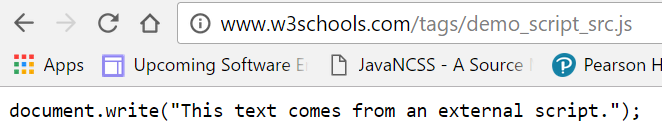


1. Open an online Editor to type the source code on the left screen



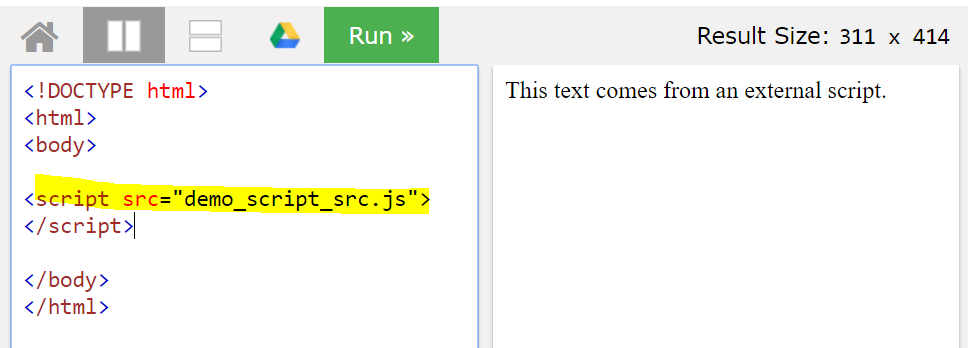


1. Questions:
   1. What is XMLHttpRequest?
   2. What does the program do?
   3. Why can the program on the left access ajax\_info.txt file?
2. Display demo\_script\_src.js



1. Open an online Editor to type the source code on the left screen





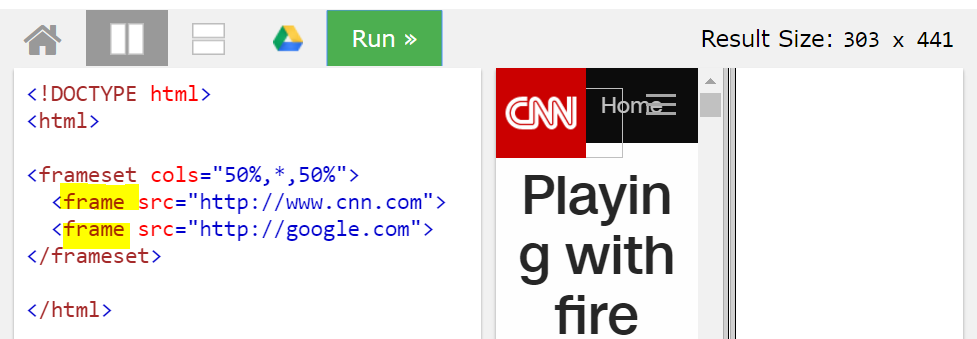
1. Questions:
   1. What does the program do?
   2. Does the program violate SOP?

**Task 6: Relaxing SOP**

In some circumstances the same-origin policy is too restrictive, posing problems for large websites that use multiple subdomains. Some techniques are used for relaxing SOP.

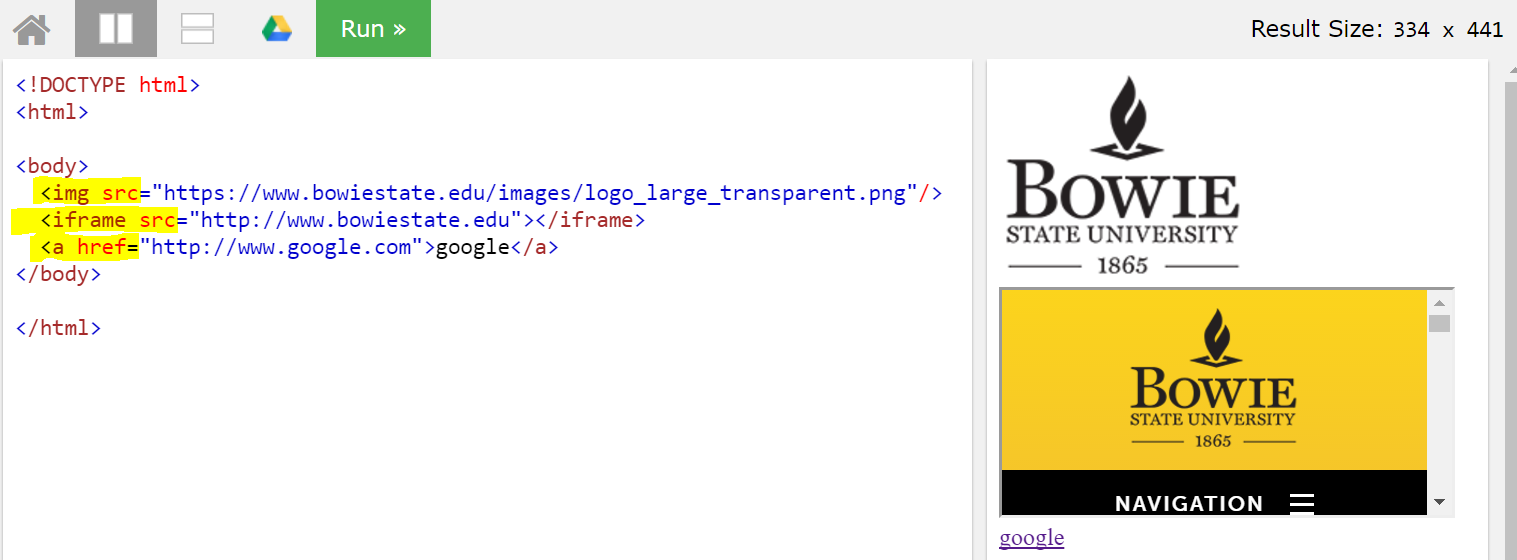
1. Relaxing with the <frame> tag
   1. Open Chrome and type the following address. Note: Some version of Firefox does not support <frame> tag





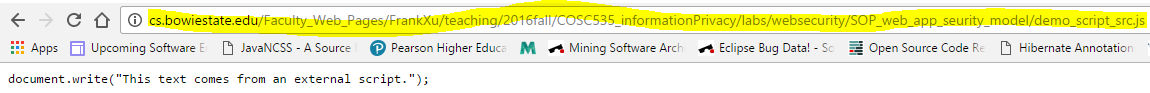
* 1. Change the code same as the left
  2. Questions
     1. What does the source do?
     2. Why can we see CNN website but Google? Hint: study *X-Frame-Options*

1. Relaxing with the <img><frame> <a> tags

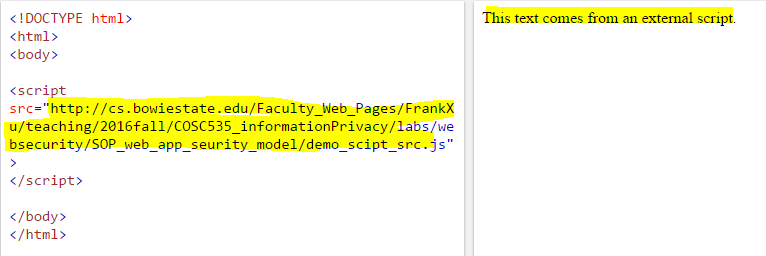


1. Relaxing with the <script src> tag. The most important technique.
   1. I have copied the *demo\_script\_src.js* to my website

|  |
| --- |
| http://cs.bowiestate.edu/Faculty\_Web\_Pages/FrankXu/teaching/2016fall/COSC535\_informationPrivacy/labs/websecurity/SOP\_web\_app\_seurity\_model/demo\_script\_src.js |



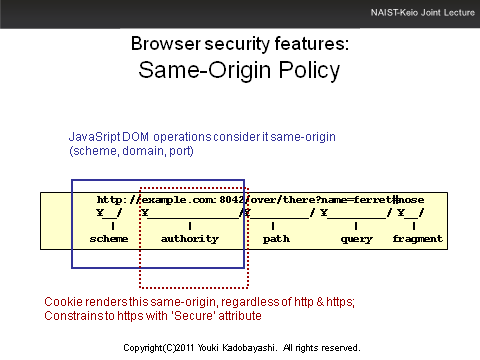
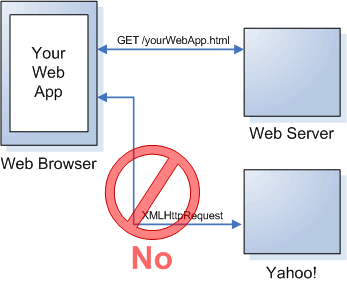
* 1. Create the following page use the new source file.



* 1. Questions
     1. Describe the function of the file *demo\_script\_src.js*?
     2. Can you provide another example of SOP relaxing with the <script src> tag?

How Does It work? (Wiki)

This mechanism bears a particular significance for modern web applications that extensively depend on HTTP cookies to maintain authenticated user sessions, as servers act based on the HTTP cookie information to reveal sensitive information or take state-changing actions. A strict separation between content provided by unrelated sites must be maintained on the client-side to prevent the loss of data confidentiality or integrity.



To illustrate, the following table gives an overview of typical outcomes for checks against the [URL](https://en.wikipedia.org/wiki/Uniform_Resource_Locator) "**http://www.example.com/dir/page.html**".

|  |  |  |
| --- | --- | --- |
| **Compared URL** | **Outcome** | **Reason** |
| **http://www.example.com**/dir/page2.html | Success | Same protocol, host and port |
| **http://www.example.com**/dir2/other.html | Success | Same protocol, host and port |
| **http://**username:password@**www.example.com**/dir2/other.html | Success | Same protocol, host and port |
| http://www.example.com:**81**/dir/other.html | Failure | Same protocol and host but different port |
| **https**://www.example.com/dir/other.html | Failure | Different protocol |
| http://**en.example.com**/dir/other.html | Failure | Different host |
| http://**example.com**/dir/other.html | Failure | Different host (exact match required) |
| http://**v2.www.example.com**/dir/other.html | Failure | Different host (exact match required) |
| http://www.example.com:**80**/dir/other.html | Depends | Port explicit. Depends on implementation in the browser. |

**Security Concerns**

The main reason to have this restriction is that without the same-origin policy there would be a security risk. Assume that a user is visiting a banking website and doesn't log out. Then he goes to any random other site and that site has some malicious JavaScript code running in the background that requests data from the banking site. Because the user is still logged in on the banking site, that malicious code could do anything on the banking site. For example, get a list of your last transactions, create a new transaction, etc. This is because the browser can send and receive session cookies to the banking website based on the domain of the banking website. A user visiting that malicious site would expect that the site he is visiting has no access to the banking session cookie. While this is true, the JavaScript has no direct access to the banking session cookie, but it could still send and receive requests to the banking site with the banking site's session cookie, essentially acting as a normal user of the banking site. Regarding the sending of new transactions, even CSRF protections by the banking site have no effect, because the script can simply do the same as the user would do. So this is a concern for all sites where you use sessions and/or need to be logged in. If the banking site from the example (or any other site of course) only presents public data and you cannot trigger anything, then there is usually no danger which the same-origin policy protects against. Also, if the two sites are under control of the same party or trust each other completely, then there is probably no danger either.

Relaxing the same-origin policy

In some circumstances the same-origin policy is too restrictive, posing problems for large websites that use multiple subdomains.

# Reference:

* http://www.cis.syr.edu/~wedu/seed/lab\_env.html
* <https://en.wikipedia.org/wiki/Same-origin_policy>
* <https://www.hacking-lab.com/misc/downloads/event_2010/simon_egli_same_origin_policy_v1.0.pdf>
* http://www.w3schools.com/

# Source Code

|  |
| --- |
| <!--  This file implements a navigation bar, designed to go in a frame.  Include it in a frameset like the following:  <frameset rows="\*,75">  <frame src="about:blank" name="main">  <frame src="navigation.html">  </frameset>  The code in this file will control the contents of the frame named "main"  -->  <script>  // The function is invoked by the Back button in our navigation bar  function back( )  {  // First, clear the URL entry field in our form  document.navbar.url.value = "";  // Then use the History object of the main frame to go back  // Unless the same-origin policy thwarts us  try { parent.main.history.back( ); }  catch(e) { alert("Same-origin policy blocks History.back( ): " + e.message); }  // Display the URL of the document we just went back to, if we can.  // We have to defer this call to updateURL( ) to allow it to work.  setTimeout(updateURL, 1000);  }  // This function is invoked by the Forward button in the navigation bar.  function forward( )  {  document.navbar.url.value = "";  try { parent.main.history.forward( ); }  catch(e) { alert("Same-origin policy blocks History.forward( ): "+e.message);}  setTimeout(updateURL, 1000);  }  // This private function is used by back( ) and forward( ) to update the URL  // text field in the form.  function updateURL( )  {  try { document.navbar.url.value = parent.main.location.href; }  catch(e) {  document.navbar.url.value = "<Same-origin policy prevents URL access>";  }  }  // Utility function: if the url does not begin with "http://", add it.  function fixup(url)  {  if (url.substring(0,7) != "http://") url = "http://" + url;  return url;  }  // This function is invoked by the Go button in the navigation bar and also  // when the user submits the form  function go( )  {  // And load the specified URL into the main frame.  parent.main.location = fixup(document.navbar.url.value);  }  // Open a new window and display the URL specified by the user in it  function displayInNewWindow( )  {  // We're opening a regular, unnamed, full-featured window, so we just  // need to specify the URL argument. Once this window is opened, our  // navigation bar will not have any control over it.  window.open(fixup(document.navbar.url.value));  }  // Try to read the body code of the main frame  function viewSource()  {  try { alert(parent.main.document.body.innerHTML) }  catch(e) { alert("Same-origin policy blocks reading frame body: "+e.message);}  }  // Try to read the cookie of the main frame  function readCookie()  {  try { alert(parent.main.document.cookie) }  catch(e) { alert("Same-origin policy blocks reading frame cookie: "+e.message);}  }  // Try to use XMLHTTLRequest object to send arbitrary HTTP request  function sendRequest()  {  var request = null;  if (document.navbar.url.value == "")  {  alert("URL empty!");  return;  }  var URL = fixup(document.navbar.url.value);  // Create an XMLHttpRequest object with a constructor call  request = new XMLHttpRequest();    // Register and event handler to receive asynchronous notifications.  // This code says what to do with the response.  request.onreadystatechange = function() {  if(request.readyState == 4)  alert(request.responseText);  }  try  {  // In most cases, submitting a XMLHttpRequest follows the following 3 steps.  // Call the open() method to specify the URL you are requesting and HTTP  // method of the request which is "GET" here. Third argument "true"  // indicates that this is an asynchronous request.  request.open("GET",URL,true);  // Call the setRequestHeader() method to set the request HTTP headers.  // This is not neccessary. Default value will be set if no such calls are made.  request.setRequestHeader("Accept-Language","en");  // Finally, send the request to server.  request.send(null);  }  catch(e) { alert("Same-origin policy blocks XMLHttpRequest: "+e.message);}  }  </script>  <!-- Here's the form, with event handlers that invoke the functions above -->  <form name="navbar" onsubmit="go( ); return false;">  <input type="button" value="Back" onclick="back( );">  <input type="button" value="Forward" onclick="forward( );">  URL: <input type="text" name="url" size="50">  <input type="button" value="Go" onclick="go( );">  <input type="button" value="Open New Window" onclick="displayInNewWindow( );">  <input type="button" value="View Source" onclick="viewSource();">  <input type="button" value="Read Cookie" onclick="readCookie();">  <input type="button" value="Send Request" onclick="sendRequest();">  </form>  <script>  </script> |

|  |
| --- |
| <!DOCTYPE html>  <html>  <frameset cols="50%,\*,50%">  <frame src="http://cnn.com">  <frame src="http://google.com">  </frameset>  </html> |
| <!DOCTYPE html>  <html>  <body>  <h1>The XMLHttpRequest Object</h1>  <p id="demo"></p>  <script>  var xhttp = new XMLHttpRequest();  xhttp.onreadystatechange = function() {  if (this.readyState == 4 && this.status == 200) {  document.getElementById("demo").innerHTML =  this.responseText;  }  };  xhttp.open("GET", "ajax\_info.txt", true);  xhttp.send();  </script>  </body>  </html> |