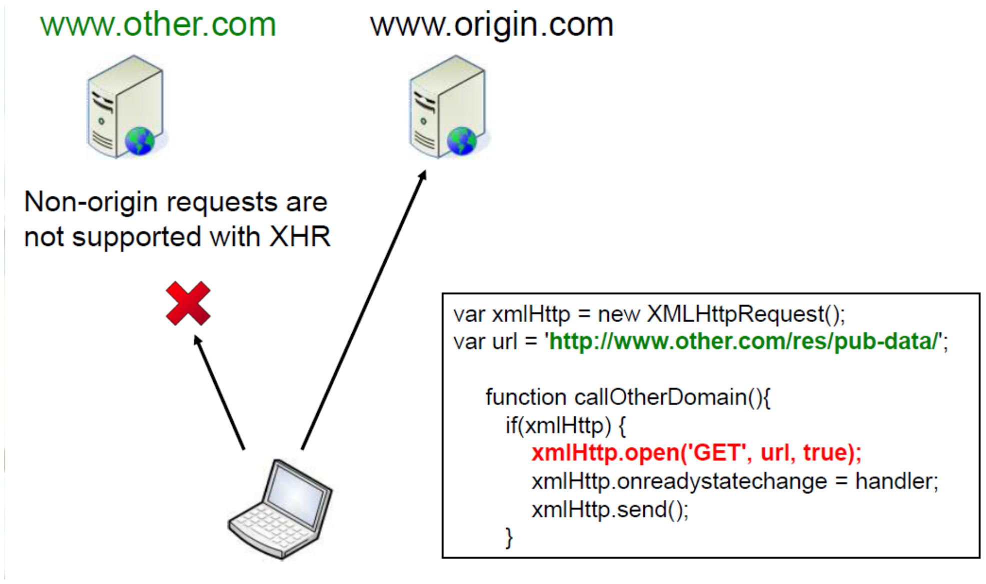
Web Application Security Model: Same-Origin Policy

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.



```
var xmlHttp = new XMLHttpRequest();
var url = 'http://www.other.com/res/pub-data/';

    function callOtherDomain(){
      if(xmlHttp) {
         xmlHttp.open('GET', url, true);
         xmlHttp.onreadystatechange = handler;
         xmlHttp.send();
      }
```

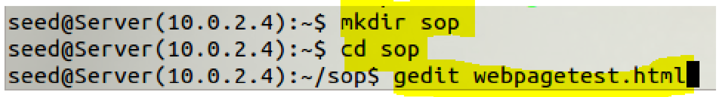
Lab Environment

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

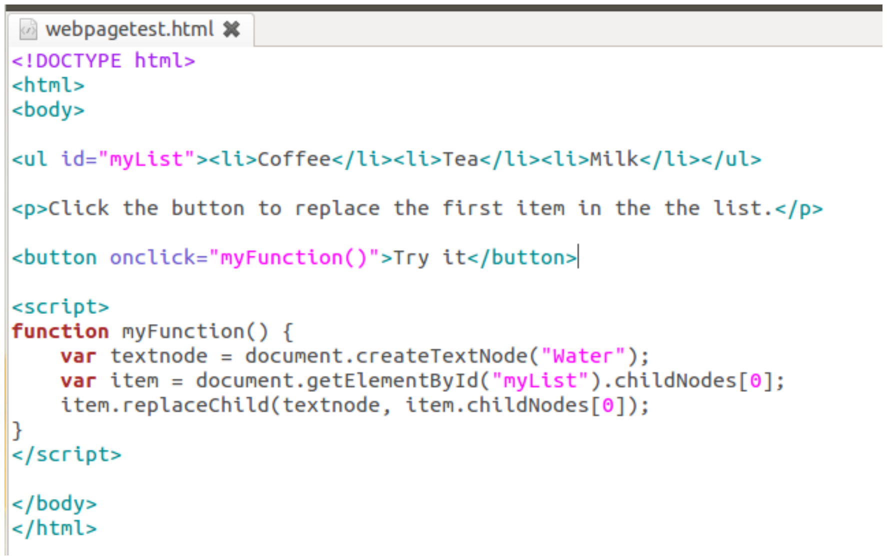
- User ID: root, Password: *seedubuntu*.
    - Note: Ubuntu does not allow root to login directly from the login window. You have to login as a normal user, and then use the command **su** to login to the root account.
- User ID: seed, Password: *dees*

Task 1: Updates a Web Page Dynamically Using JavaScript

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



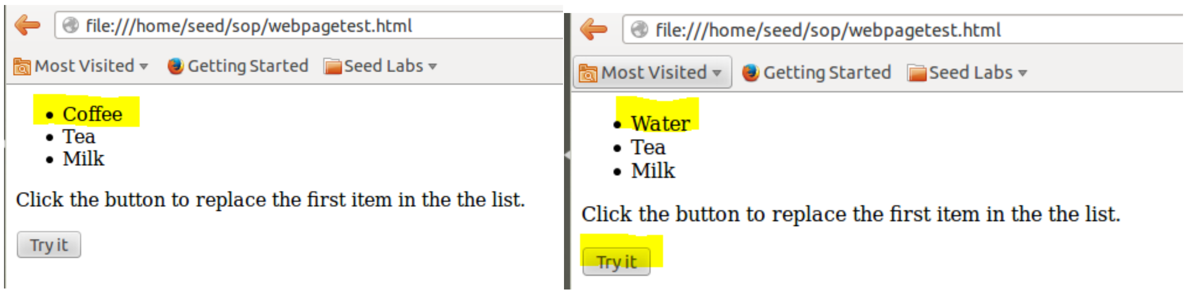
```
seed@Server(10.0.2.4):~$ mkdir sop
seed@Server(10.0.2.4):~$ cd sop
seed@Server(10.0.2.4):~/sop$ gedit webpagetest.html
```



webpagetest.html

```
<!DOCTYPE html>
<html>
<body>

<ul id="myList"><li>Coffee</li><li>Tea</li><li>Milk</li></ul>

<p>Click the button to replace the first item in the the list.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
    var textnode = document.createTextNode("Water");
    var item = document.getElementById("myList").childNodes[0];
    item.replaceChild(textnode, item.childNodes[0]);
}
</script>

</body>
</html>
```

2. Open Firefox and display the HTML page.



file:///home/seed/sop/webpagetest.html

- Coffee
- Tea
- Milk

Click the button to replace the first item in the the list.

Try it

file:///home/seed/sop/webpagetest.html

- Water
- Tea
- Milk

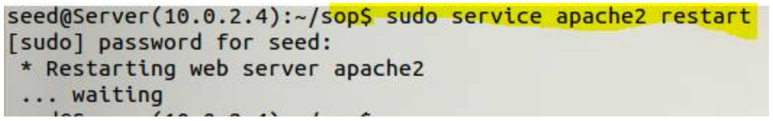
Click the button to replace the first item in the the list.

Try it

3. Questions
   3.1. How to add a textbox?
   3.2. How to add a button?

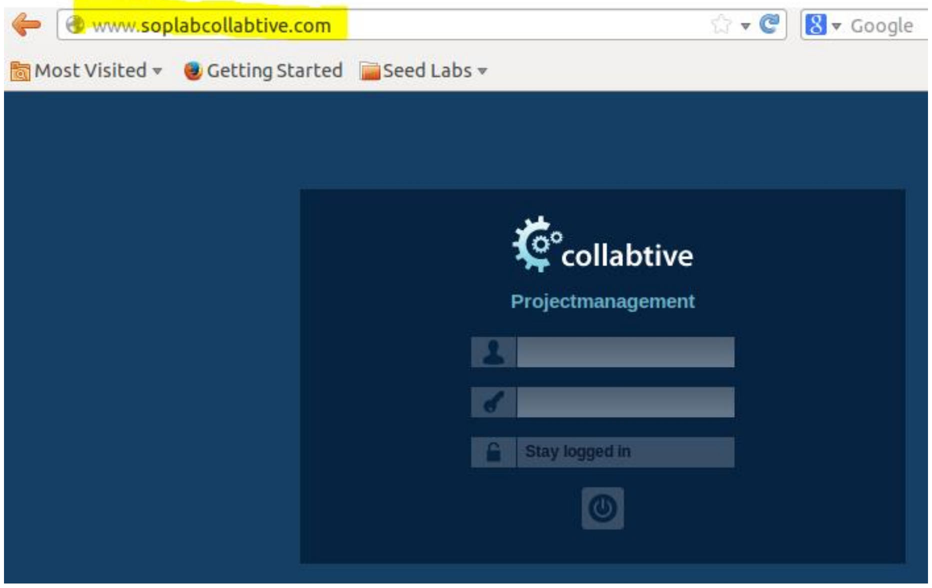
Task 2: Find the Session Cookie of www.soplabcollabtive.com

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

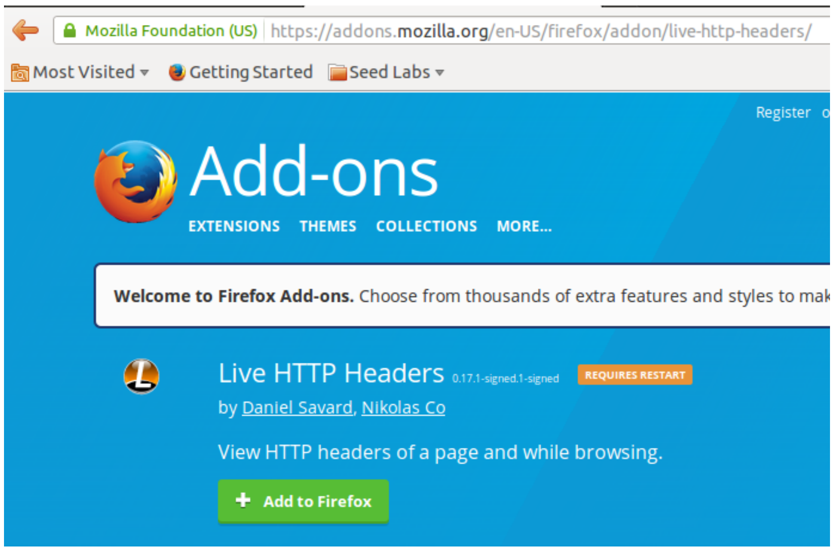


```
seed@Server(10.0.2.4):~/sop$ sudo service apache2 restart
[sudo] password for seed:
 * Restarting web server apache2
 ... waiting
```

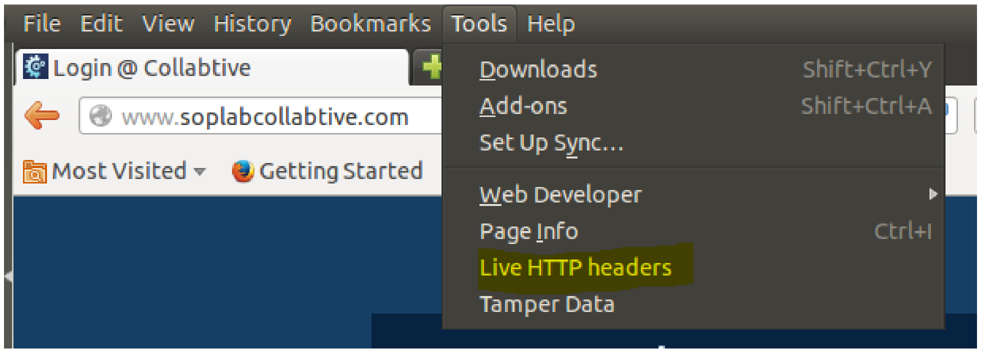
2. Start the website



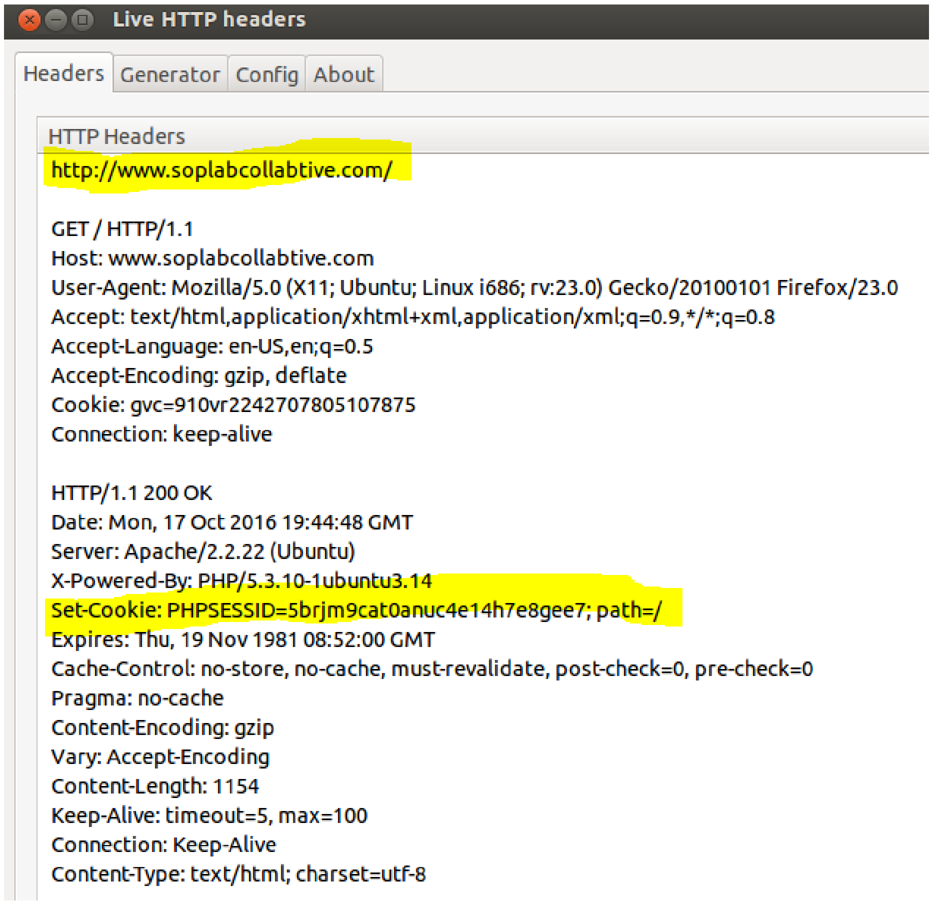
3. Add **Live HTTP header** add-on to Firefox
   3.1. Search for the add-on


Mozilla Foundation (US)
https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/
Most Visited
Getting Started
Seed Labs
Register
Add-ons
EXTENSIONS
THEMES
COLLECTIONS
MORE...
Welcome to Firefox Add-ons. Choose from thousands of extra features and styles to mak
Live HTTP Headers 0.17.1-signed.1-signed
REQUIRES RESTART
by Daniel Savard, Nikolas Co
View HTTP headers of a page and while browsing.
Add to Firefox

3.2. Open Live HTTP header



4. Live capture
   4.1. Close Firefox
   4.2. Open Firebox
   4.3. Start Live HTTP header
   4.4. Type www.soplabcollabtive.com at Firefox Address bar

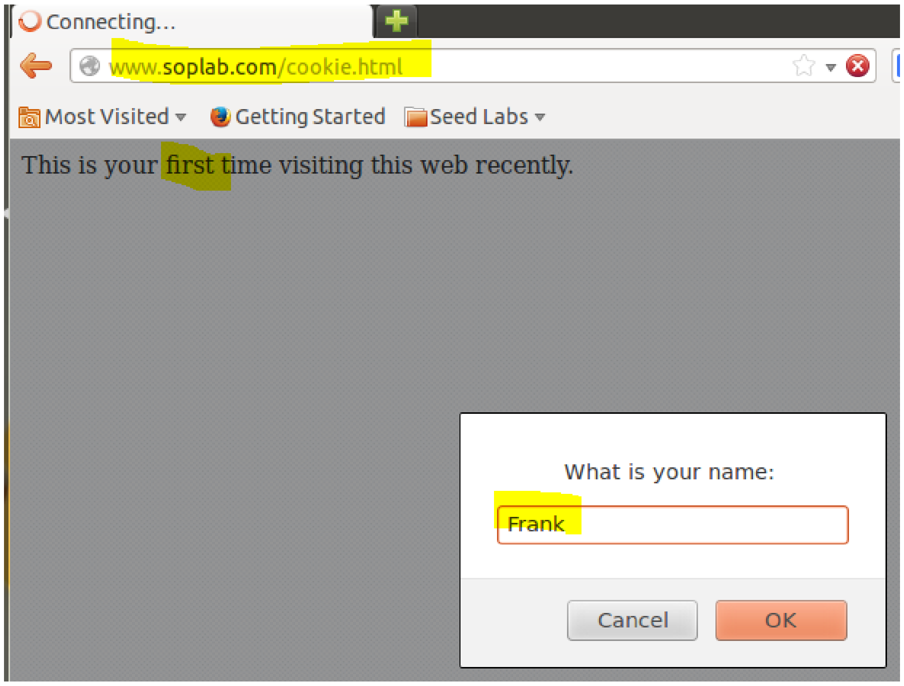


```
http://www.soplabcollabtive.com/

GET / HTTP/1.1
Host: www.soplabcollabtive.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: gvc=910vr2242707805107875
Connection: keep-alive

HTTP/1.1 200 OK
Date: Mon, 17 Oct 2016 19:44:48 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.14
Set-Cookie: PHPSESSID=5brjm9cat0anuc4e14h7e8gee7; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Encoding: gzip
Vary: Accept-Encoding
Content-Length: 1154
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

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

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

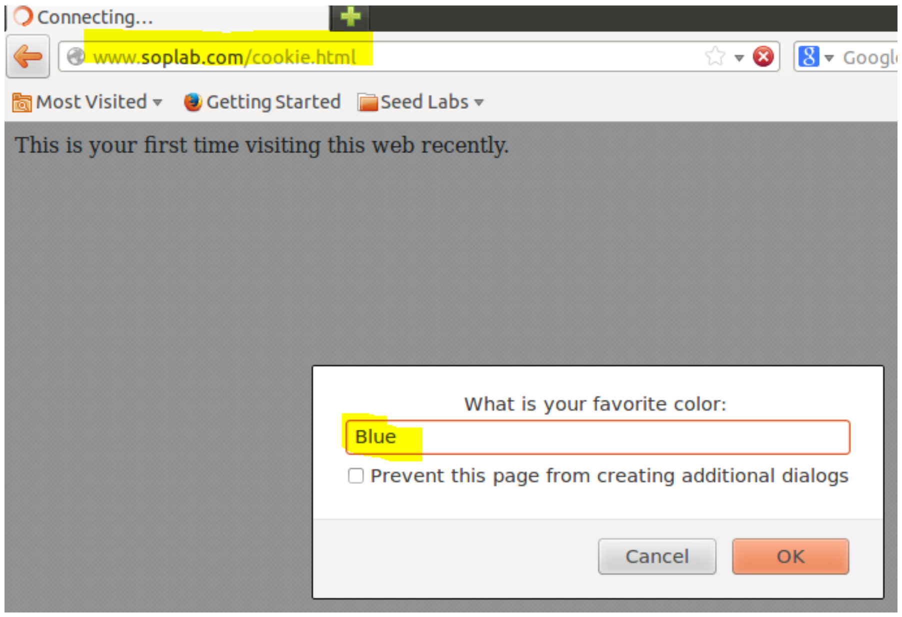
1. Save Name and Favorite Color to a cookie when you access the page the first time.



This is your first time visiting this web recently.

What is your name:

Frank



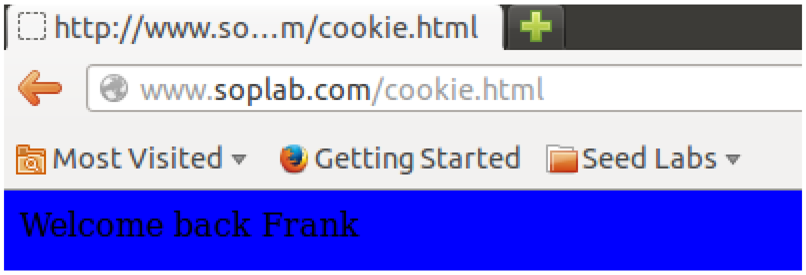
This is your first time visiting this web recently.

What is your favorite color:

Blue

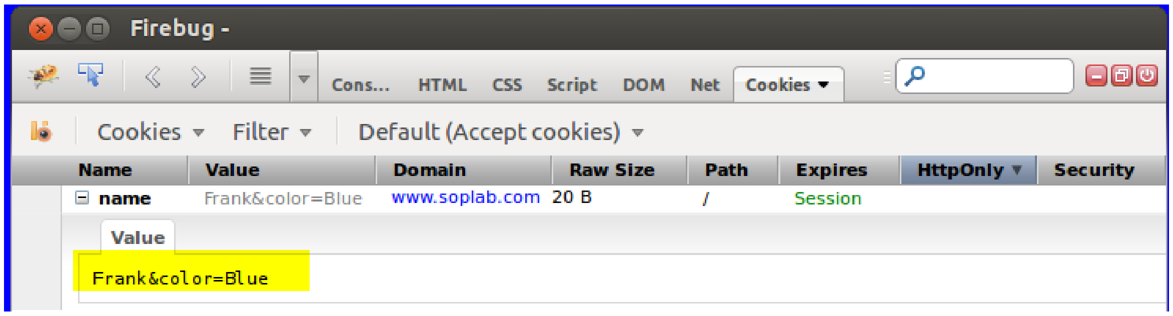
☐ Prevent this page from creating additional dialogs

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



Welcome back Frank

3. View the cookie use the Firebug

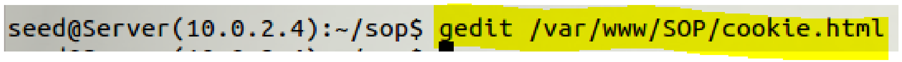

Firebug -
Cons… HTML CSS Script DOM Net Cookies
Cookies Filter Default (Accept cookies)

Name	Value	Domain	Raw Size	Path	Expires	HttpOnly	Security
name	Frank&color=Blue	www.soplab.com	20 B	/	Session		

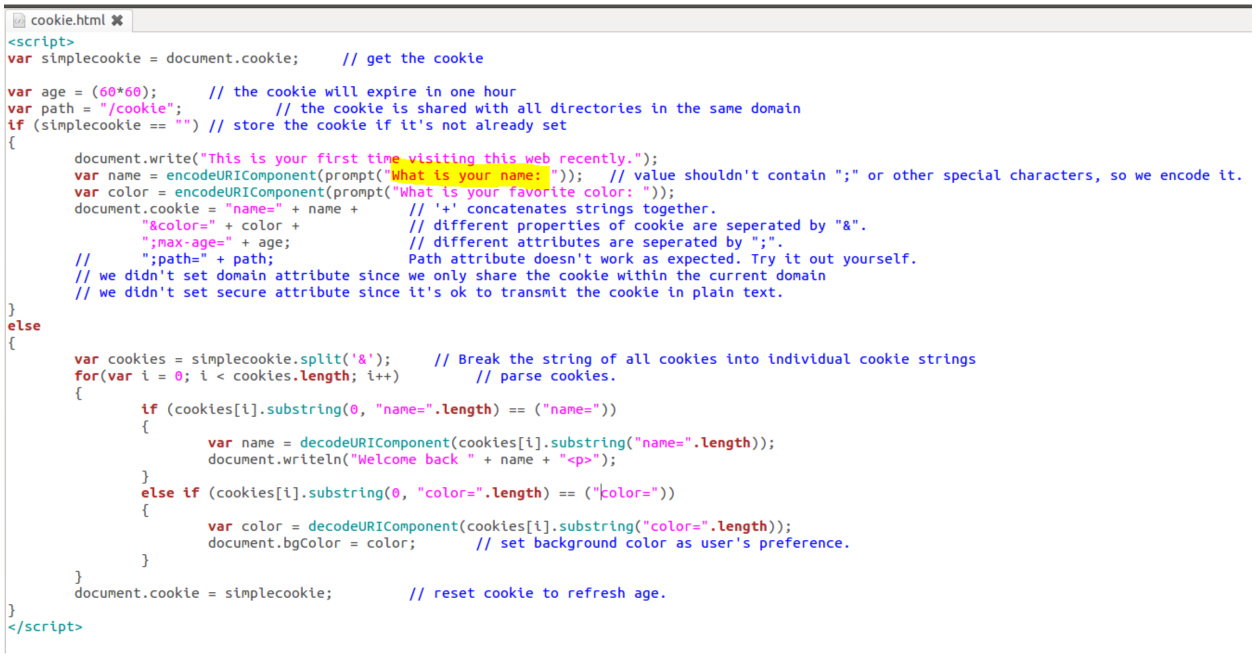
Value

Frank&color=Blue

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



```
seed@Server(10.0.2.4):~/sop$ gedit /var/www/SOP/cookie.html
```



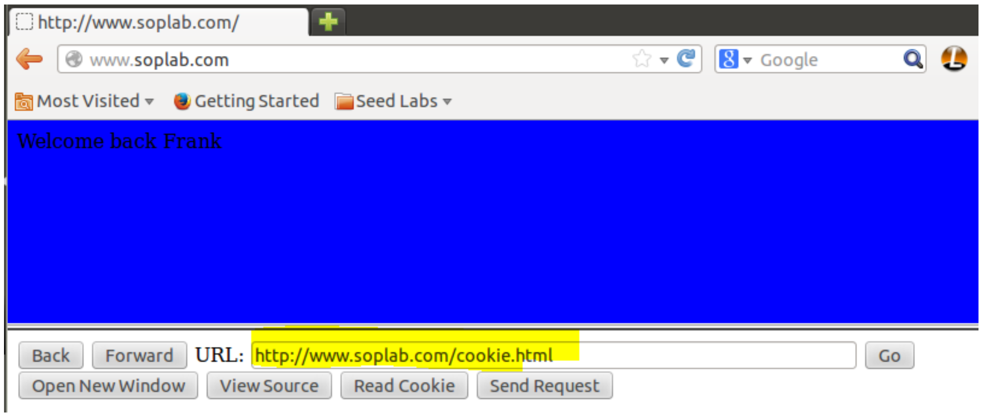
cookie.html

```
<script>
var simplecookie = document.cookie;     // get the cookie

var age = (60*60);      // the cookie will expire in one hour
var path = "/cookie";           // the cookie is shared with all directories in the same domain
if (simplecookie == "") // store the cookie if it's not already set
{
        document.write("This is your first time visiting this web recently.");
        var name = encodeURIComponent(prompt("What is your name: "));   // value shouldn't contain ";" or other special characters, so we encode it.
        var color = encodeURIComponent(prompt("What is your favorite color: "));
        document.cookie = "name=" + name +      // '+' concatenates strings together.
                "&color=" + color +             // different properties of cookie are seperated by "&".
                ";max-age=" + age;              // different attributes are seperated by ";".
//              ";path=" + path;                Path attribute doesn't work as expected. Try it out yourself.
        // we didn't set domain attribute since we only share the cookie within the current domain
        // we didn't set secure attribute since it's ok to transmit the cookie in plain text.
}
else
{
        var cookies = simplecookie.split('&');     // Break the string of all cookies into individual cookie strings
        for(var i = 0; i < cookies.length; i++)         // parse cookies.
        {
                if (cookies[i].substring(0, "name=".length) == ("name="))
                {
                        var name = decodeURIComponent(cookies[i].substring("name=".length));
                        document.writeln("Welcome back " + name + "<p>");
                }
                else if (cookies[i].substring(0, "color=".length) == ("color="))
                {
                        var color = decodeURIComponent(cookies[i].substring("color=".length));
                        document.bgColor = color;       // set background color as user's preference.
                }
        }
        document.cookie = simplecookie;         // reset cookie to refresh age.
}
</script>
```

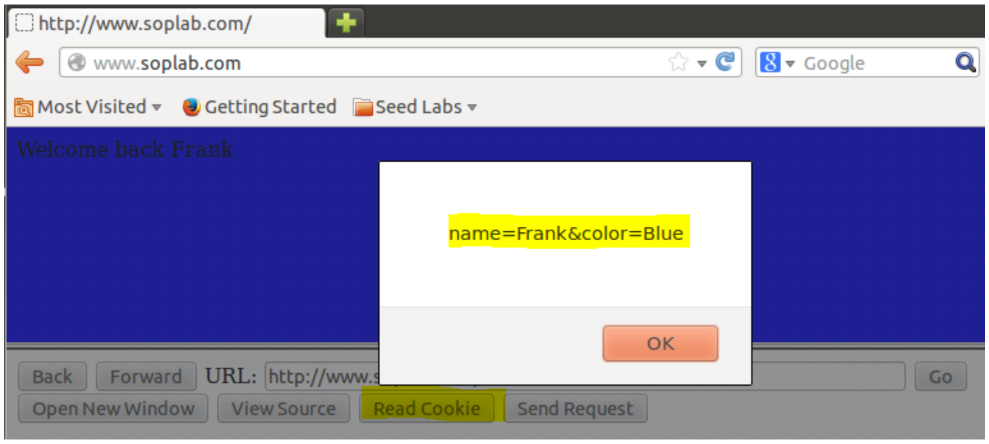
5. Question: Modify the program to trace how many time a user have visited the page.

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

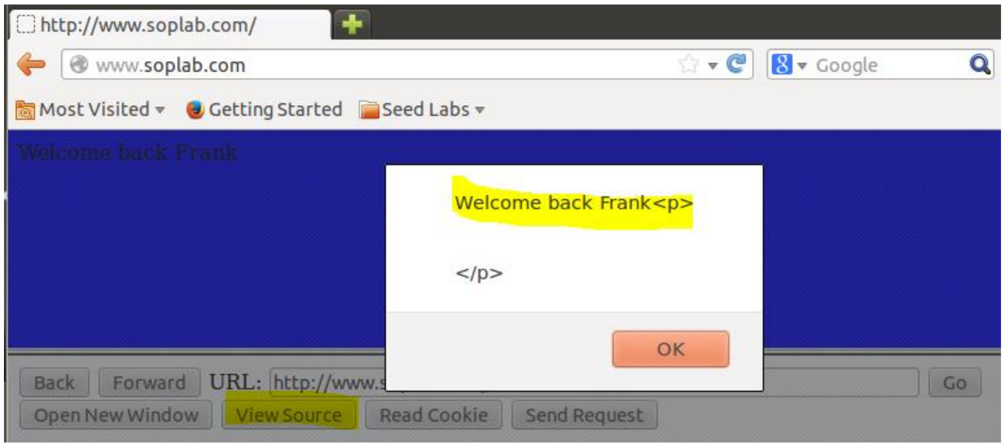


2. SOP **allows** the page www.soplab.com/index.html to access the cookies of http://soplab.com/cookie.html



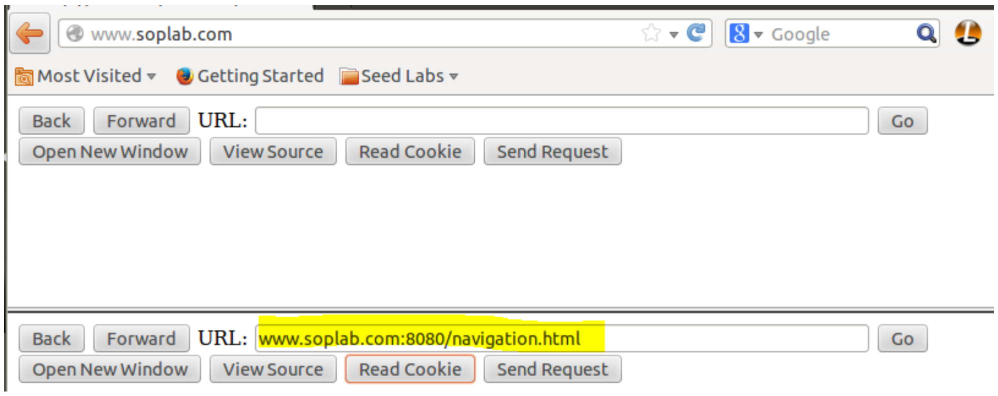
```
name=Frank&color=Blue
```

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

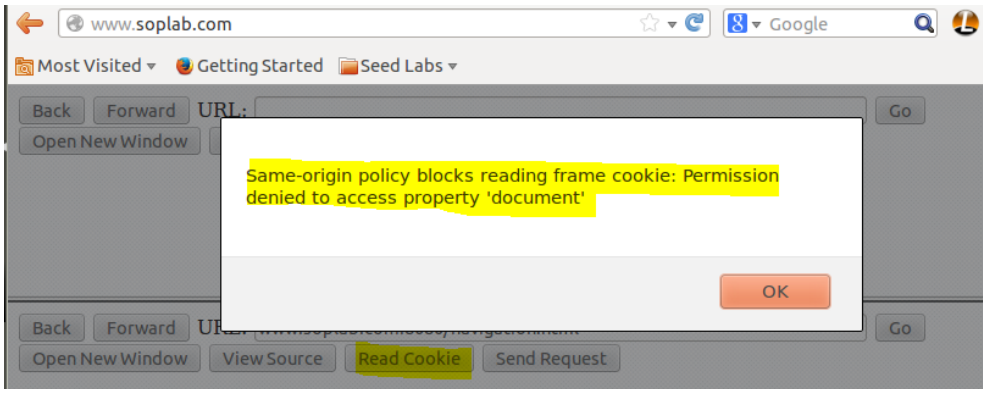


```
Welcome back Frank<p>

</p>
```

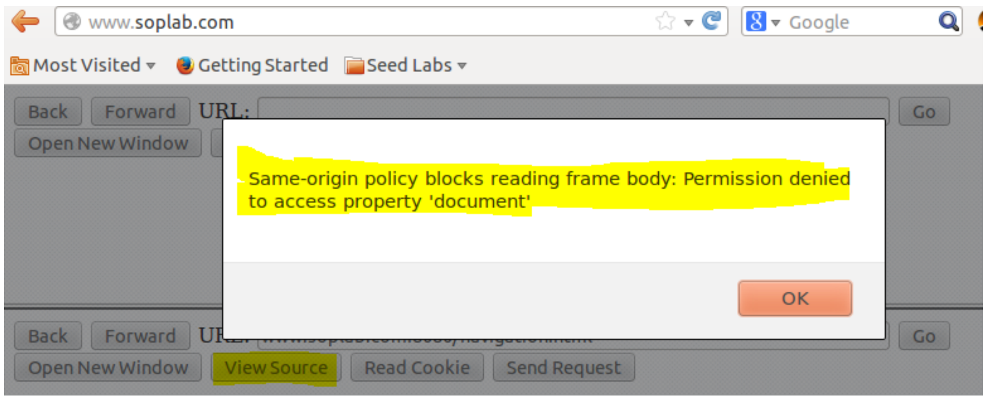
4. SOP **denies** the page www.soplab.com/index.html to access http://soplab.com:8080/navigate.html



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


Same-origin policy blocks reading frame cookie: Permission denied to access property 'document'

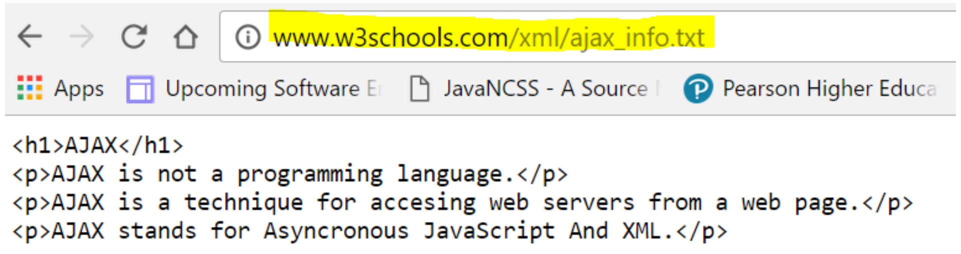
6. SOP **denies** the page www.soplab.com/index.html to access the DOM of www.soplab.com:8080/navigation.html


Same-origin policy blocks reading frame body: Permission denied to access property 'document'

7. 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

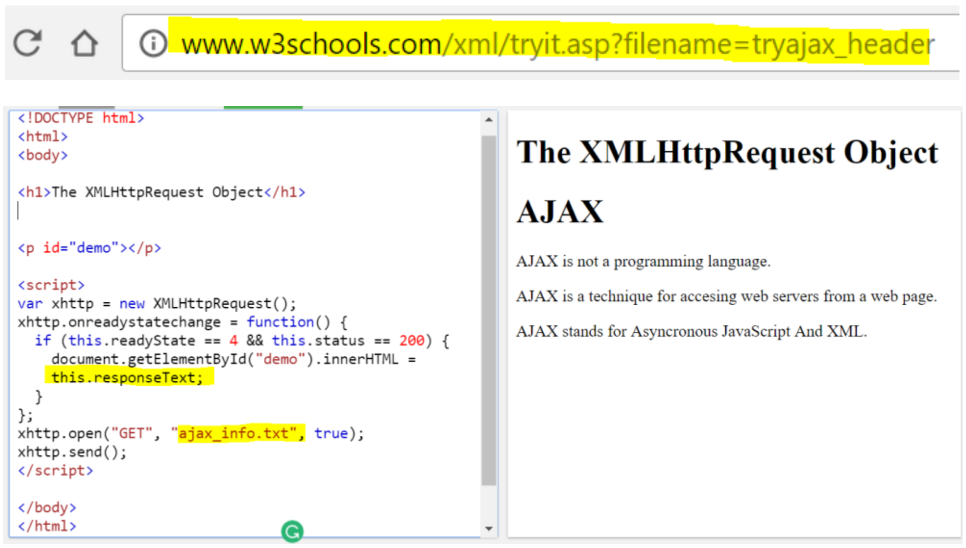
1. Display Ajax_info.txt



www.w3schools.com/xml/ajax_info.txt

```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
<p>AJAX is a technique for accesing web servers from a web page.</p>
<p>AJAX stands for Asyncronous JavaScript And XML.</p>
```

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



www.w3schools.com/xml/tryit.asp?filename=tryajax_header

```
<!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>
```

The XMLHttpRequest Object

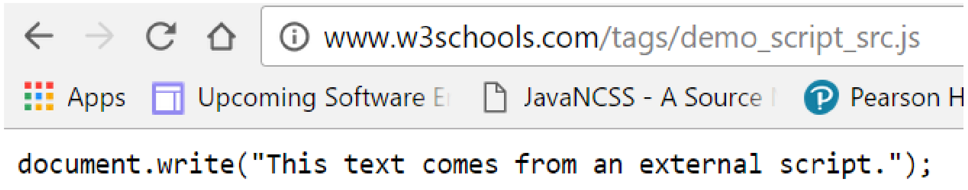
AJAX

AJAX is not a programming language.

AJAX is a technique for accesing web servers from a web page.

AJAX stands for Asyncronous JavaScript And XML.

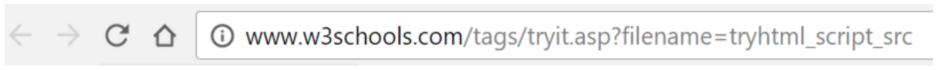
3. Questions:
   3.1. What is XMLHttpRequest?
   3.2. What does the program do?
   3.3. Why can the program on the left access ajax_info.txt file?
4. Display demo_script_src.js



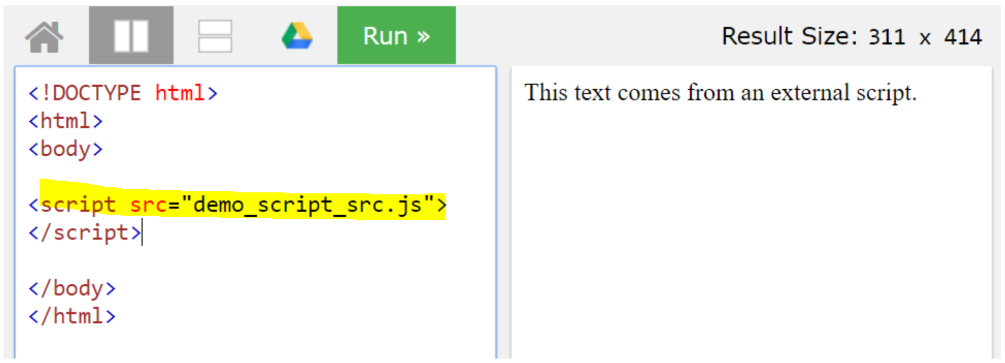
www.w3schools.com/tags/demo_script_src.js

```
document.write("This text comes from an external script.");
```

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



www.w3schools.com/tags/tryit.asp?filename=tryhtml_script_src



Run »

Result Size: 311 x 414

```
<!DOCTYPE html>
<html>
<body>

<script src="demo_script_src.js">
</script>

</body>
</html>
```

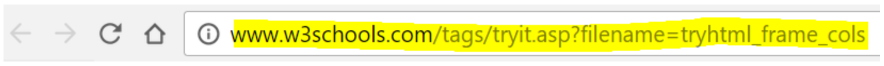
This text comes from an external script.

6. Questions:
   6.1. What does the program do?
   6.2. Does the program violate SOP?

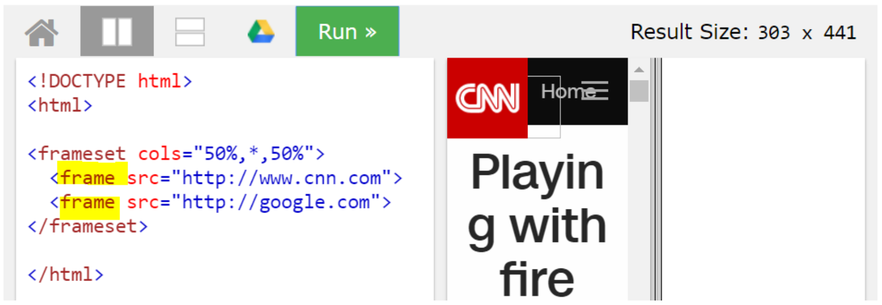
Task 6: Relaxing SOP

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



www.w3schools.com/tags/tryit.asp?filename=tryhtml_frame_cols

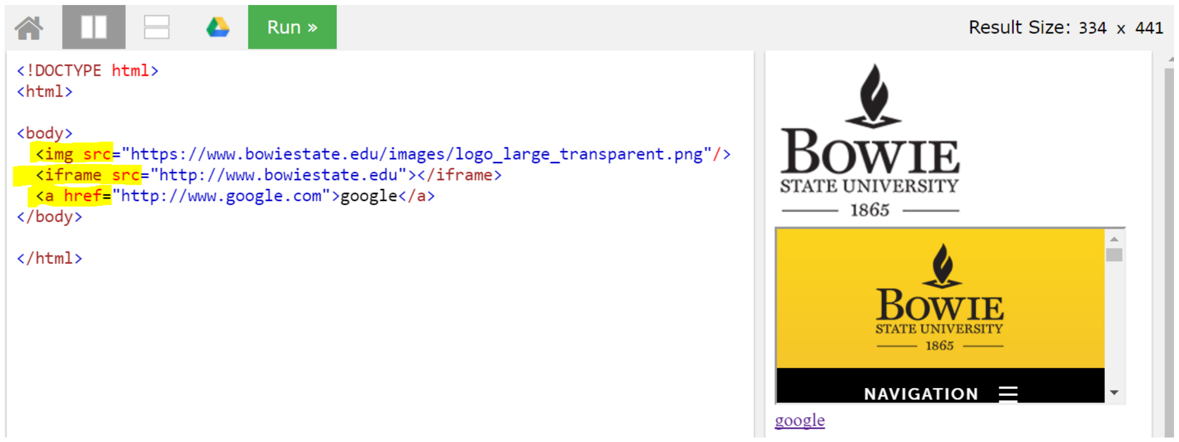


Run »

Result Size: 303 x 441

```
<!DOCTYPE html>
<html>

<frameset cols="50%,*,50%">
  <frame src="http://www.cnn.com">
  <frame src="http://google.com">
</frameset>

</html>
```

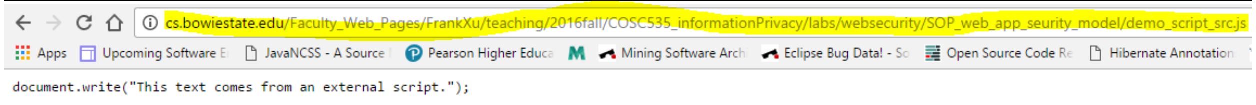
   1.2. Change the code same as the left
   1.3. Questions
      1.3.1. What does the source do?
      1.3.2. Why can we see CNN website but Google? Hint: study `X-Frame-Options`
2. Relaxing with the <img><frame> <a> tags



```
<!DOCTYPE html>
<html>

<body>
  <img src="https://www.bowiestate.edu/images/logo_large_transparent.png"/>
  <iframe src="http://www.bowiestate.edu"></iframe>
  <a href="http://www.google.com">google</a>
</body>

</html>
```

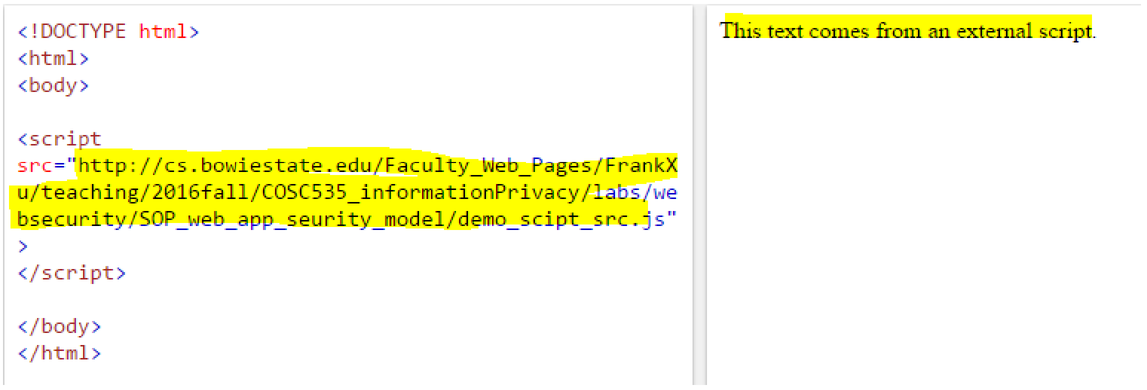
3. Relaxing with the <script src> tag. The most important technique.
   3.1. I have copied the *demo_script_src.js* to my website

http://cs.bowiestate.edu/Faculty_Web_Pages/FrankXu/teaching/2016fall/COSC535_informationPrivacy/labs/websecurity/SOP_web_app_seurity_model/demo_script_src.js


cs.bowiestate.edu/Faculty_Web_Pages/FrankXu/teaching/2016fall/COSC535_informationPrivacy/labs/websecurity/SOP_web_app_seurity_model/demo_script_src.js
Apps
Upcoming Software E
JavaNCSS - A Source
Pearson Higher Educa
Mining Software Arch
Eclipse Bug Data! - So
Open Source Code Re
Hibernate Annotation

```
document.write("This text comes from an external script.");
```

   3.2. Create the following page use the new source file.



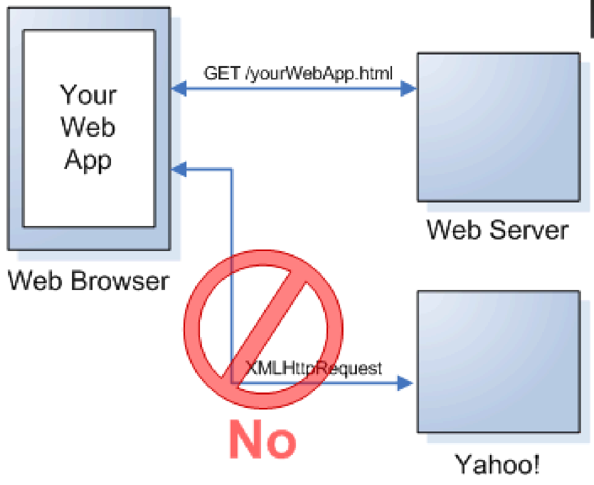
```
<!DOCTYPE html>
<html>
<body>

<script
src="http://cs.bowiestate.edu/Faculty_Web_Pages/FrankXu/teaching/2016fall/COSC535_informationPrivacy/labs/websecurity/SOP_web_app_seurity_model/demo_scipt_src.js"
>
</script>

</body>
</html>
```

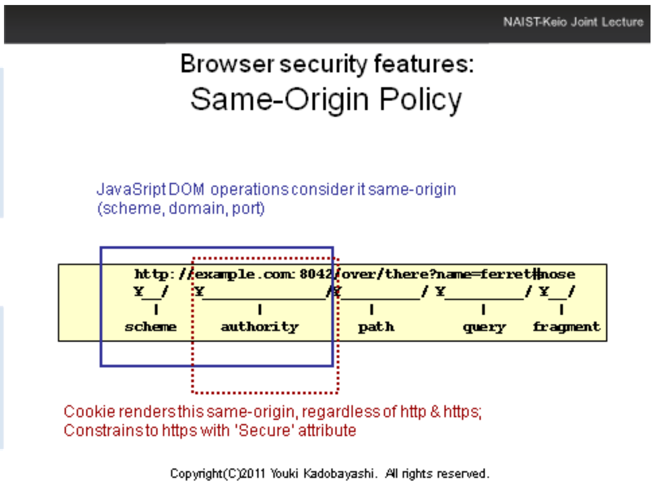
This text comes from an external script.

   3.3. Questions
      3.3.1. Describe the function of the file *demo_script_src.js*?
      3.3.2. 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.




NAIST-Keio Joint Lecture
Browser security features:
Same-Origin Policy
JavaScript DOM operations consider it same-origin (scheme, domain, port)
http://example.com:8042/over/there?name=ferret#nose
scheme
authority
path
query
fragment
Cookie renders this same-origin, regardless of http & https; Constrains to https with 'Secure' attribute
Copyright(C)2011 Youki Kadobayashi. All rights reserved.

To illustrate, the following table gives an overview of typical outcomes for checks against the URL "**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>
```