

Bruce Dang
Secure Windows Initiative (SWI)
Microsoft

Methods for Understanding and Analyzing Targeted Attacks with Office Documents

Agenda

- Introduction
- Office binary file format
- Bugs
- Defensive mechanisms
- Exploit structures
- Analysis techniques
- Detection mechanisms

Introduction

- Targeted attacks
 - Very popular in the last couple years
 - Bypasses perimeter security devices/software
 - Most antivirus engines can't detect it
 - No technical information in the public

Office binary file format

- Structured Storage / OLE SS
 - File system inside a binary file
 - Divide data into storage and streams (storage = directory, stream = file)
 - 12-page specification
- Application-specific data stored inside storage/streams.
- Can be frustrating to parse manually

Use Win32 COM API

- StgOpenStorage() on the file. You get back an IStorage object.
- IStorage->EnumElements() enumerates all of the storages and streams.
- IStorage->OpenStream() opens up whatever stream you want. Returns an IStream object.
- IStream->Stat() tells you the stream size.
- IStream->Read() reads n bytes from the stream.

Using Win32 COM API

```
HRESULT hr;
IStorage *is;
IStream *stream;
IEnumSTATSTG *penum;
STATSTG statstg;
StgOpenStorage(L"foo.ppt", NULL, STGM_DIRECT |
    STGM_READ | STGM_SHARE_EXCLUSIVE, NULL, 0, &is);
is->EnumElements(NULL, NULL, NULL, &penum);
hr = penum->Next(1, &statstg, 0);
while( hr == S_OK) {
    wprintf(L"name = %s\tsize = 0x%08x\n",
        statstg.pwcsName, statstg.cbSize);
    ...
    is->OpenStream(statstg.pwcsName, NULL, STGM_READ |
        STGM_SHARE_EXCLUSIVE, 0, &stream);
    stream->Read(data, statstg.cbSize, NULL);
    ... parse data ...
}
```

Doing it with Python

- A bit simpler. Good for experiments.

```
from pythoncom import *
ostore = StgOpenStorage(sys.argv[1], None, 0x10, None, 0)
estat = ostore.EnumElements()
str = estat.Next()
while str != ():
    if str[0][0] == "PowerPoint Document":
        len = str[0][2]
        str = estat.Next()
    ostream = ostore.OpenStream("PowerPoint Document", None,
        0x10, 0)
    data = ostream.Read(len)
    ...parse data...
```

PowerPoint binary file format

- Stores data in the “PowerPoint Document” stream. `OpenStream(L“PowerPoint Document”, ...)`
- Two types of PowerPoint data structures
 - Container – a “directory”
 - Contains other container or atoms.
 - Atom – a “file”
- Records follow TLV style
- MSO objects follow the same format

PowerPoint binary file format

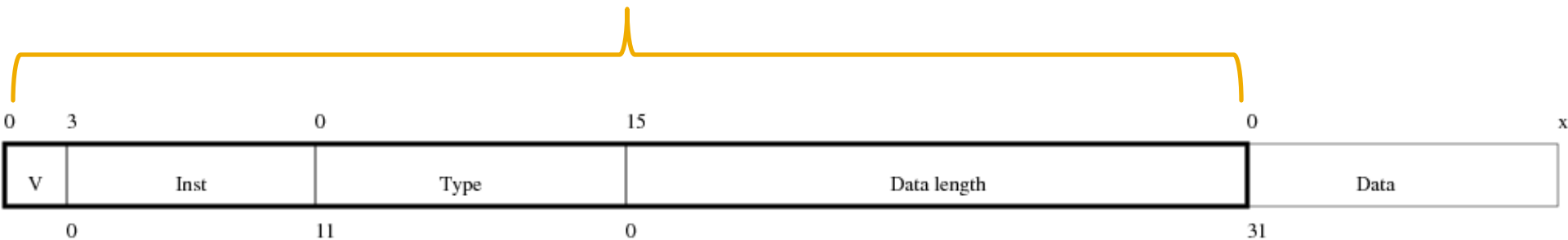
■ 1 struct to rule them all

```
typedef struct
{
    uint2 recVer:4;
    uint2 recInstance:12;
    uint2 recType;
    uint4 recLen;
} PPTRHDR_t;
```

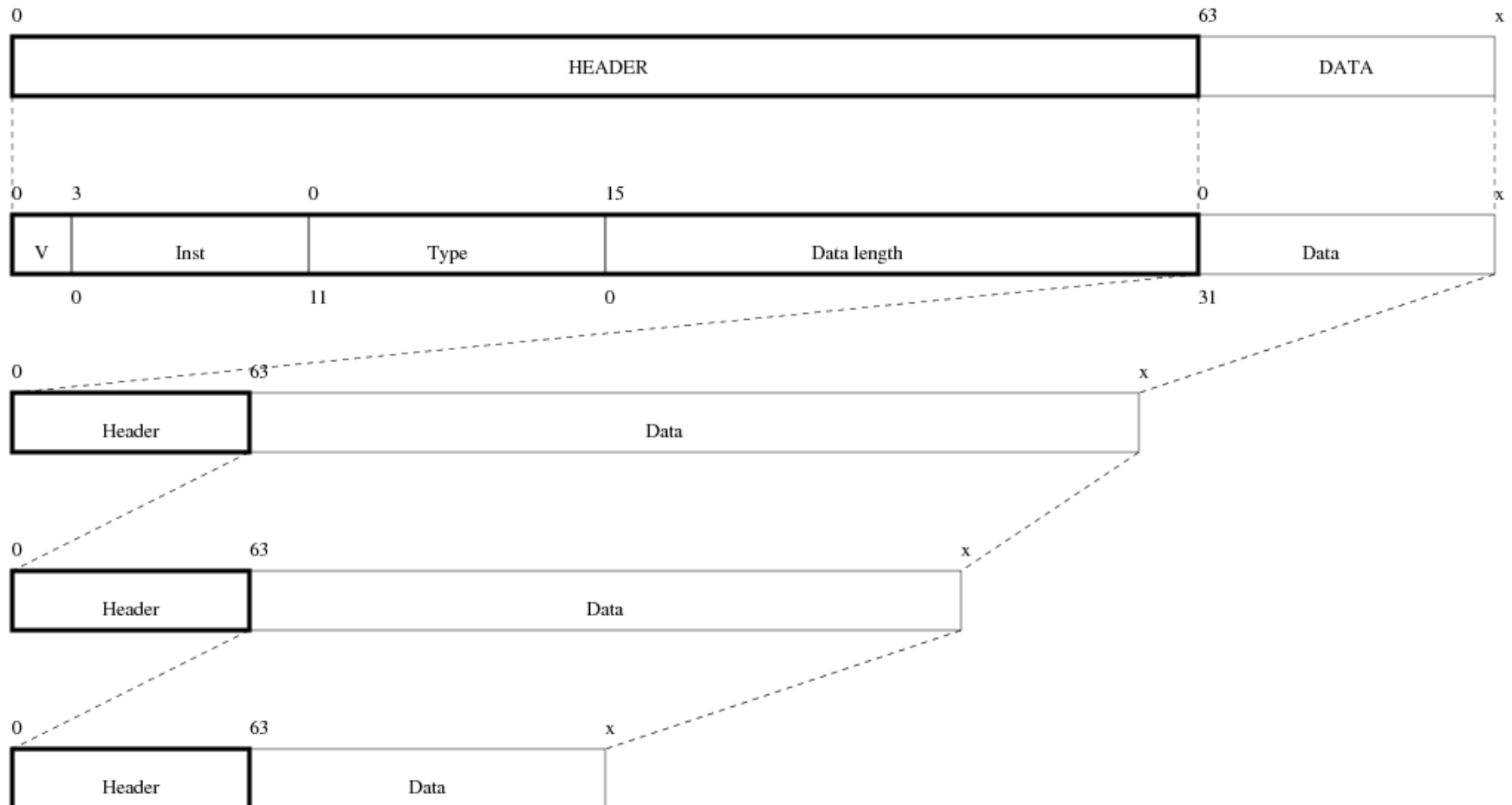
- If recVer = 0xF, then the record is a container; else, it is an atom
- recType refers to the record type. There are ~ 100 of these in PowerPoint. You can look them up the file format specification.

PowerPoint binary file format

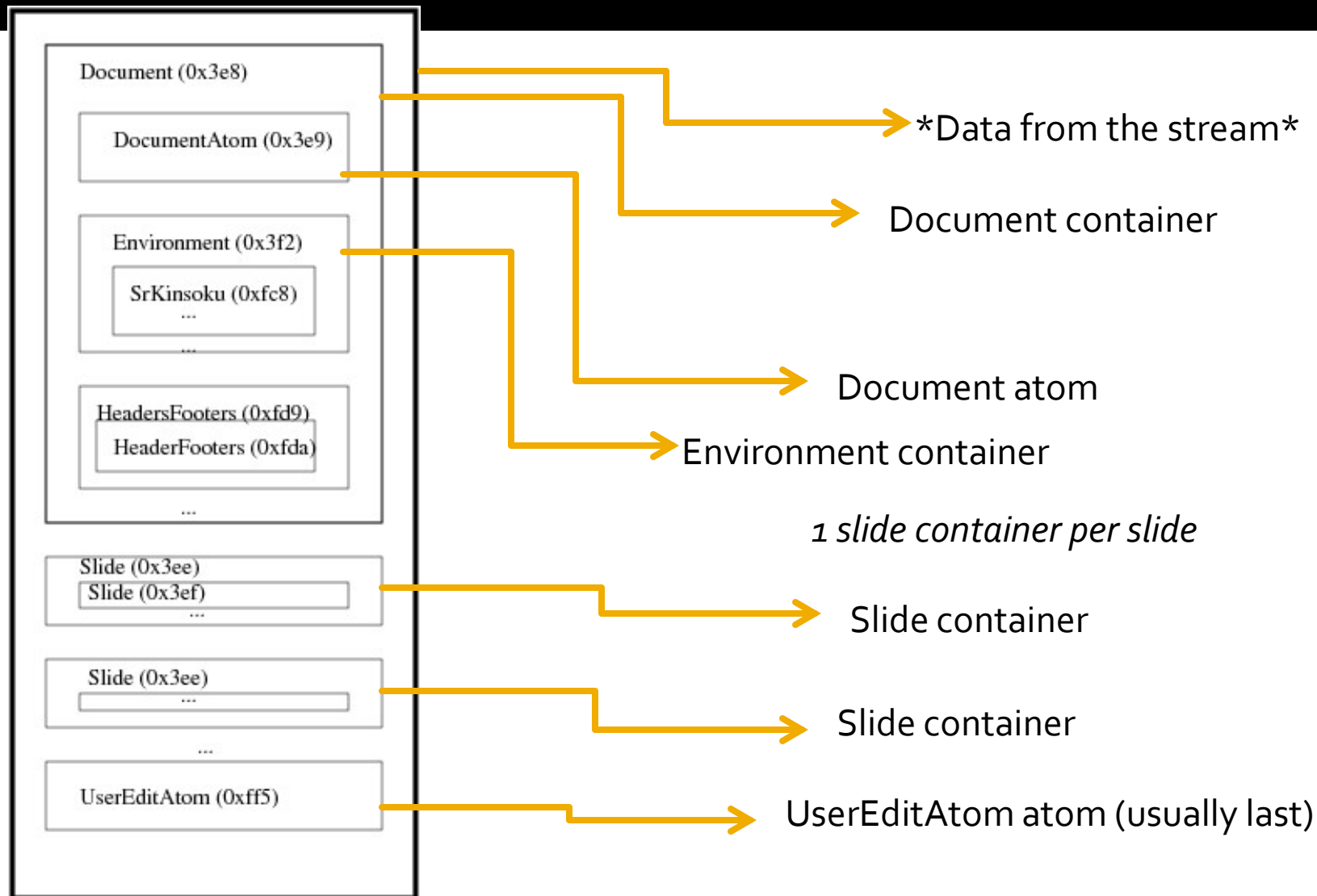
```
typedef struct
{
    uint2 recVer:4;
    uint2 recInstance:12;
    uint2 recType;
    uint4 recLen;
} PPTRHDR_t;
```



PowerPoint binary file format



PowerPoint binary file format



Excel binary file format

- Data is stored inside the “Workbook” stream
- No containers/atoms. Just plain BIFF records.
- BIFF records also follow TLV format
- Record data has an upper bound of ~ 2000-8000 bytes (BIFF version dependent). If longer, use a CONTINUE record
- Data inside the “Workbook” stream is organized like: BOF <data> EOF BOF <data> EOF BOF <data> EOF ...

Bugs

- Nothing out of the ordinary: integer over/underflow, off-by-one, double free, uninitialized variables, bad pointer reuse, stack/heap overflow, ...

DEMO

Defensive mechanisms

- Use MOICE
 - Free
 - It requires Office 2003
 - It only works on OLE structured storage files
 - It uses the Office 2007 compatibility pack
 - It converts your binary file format to the new XML format and opens it up

Defensive mechanisms

- Office 2003 SP3
 - Free
 - Result of a major security / SDL push
 - If you had Office 2003 SP3, then you would not be affected by any of the Office zero-days acknowledged in the public since.
- If you have Office 2003, install SP3

Defensive mechanisms

- Office 2003 SP3 and MOICE actually eliminates / mitigates most of the Office vulns (probably 99% of them).
- Always have the latest patches.

Defensive mechanisms

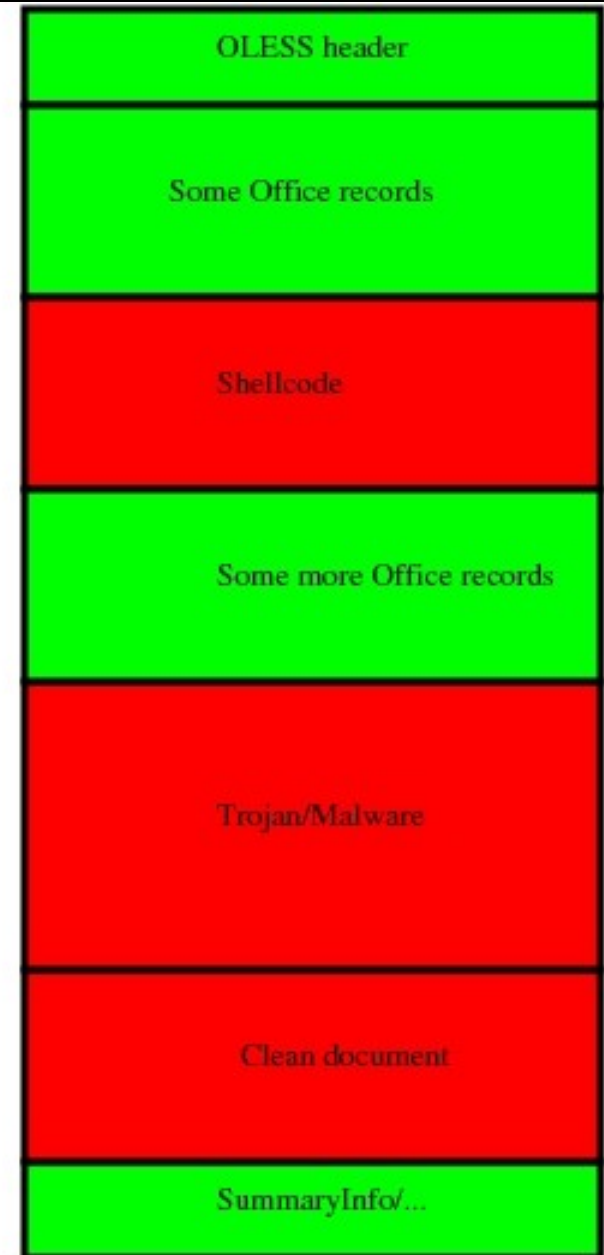
- Use Office 2007

Exploit structure

- Not out of the ordinary
- Basic structure
 - Shellcode
 - Malware
 - Clean document
- Techniques

Exploit structure

- Everything is included in the document
- There can be variations
 - Multiple shellcode stages
 - Multiple trojans
 - Obfuscation of trojans/doc
- Very rarely uses URLMON routines



Exploit structure

- Techniques
 - Standard GetEIP / PIC
 - Custom encoders
 - PEB retrieval
 - File handle bruteforce
 - Application relaunch

Exploit structure

- Why file handle bruteforce?
 - Exploit must find itself in memory (this is not the same as GetEIP)
 - Exploit cannot simply scan the entire process address space looking for itself (speed)
 - Very easy/short implementation in assembly. It's literally:

```
int fh;
for (fh=0; fh < 65536; fh += 4)
{
    if (GetFileSize(fh, NULL) == mysize) return fh;
}
```

Exploit structure

- What it does (there can be variations)
 - Shellcode decodes itself and runs
 - Builds up a list of function pointers
 - Finds itself in memory (file handle)
 - Read data from specific locations in the file
 - Extract the trojan and the clean document
 - Run the trojan and relaunch the app to open the new file
 - Exit the current process
- SetFilePointer, ReadFile, WriteFile, CloseFile, WinExec, ExitProcess.

Analysis techniques

- Tools
 - Hexeditor
 - Disassembler
 - Optional: Debugger (WinDBG is sufficient)
- Objectives
 - Identify the shellcode
 - Understand it
 - Extract the malicious components
 - [Identify the exact vuln]

Analysis techniques

- How many do you recognize?

- EB 10 5B 4B 33 C9 66 B9 96 03 80
34 0B FD E2 FA EB 05 E8 EB FF FF
FF
- 64 A1 30 00 00 00
- 64 8B 1D 30 00 00 00
- D9 74 24 F4

Analysis techniques

- Debug DOC/XLS/PPT?
- Static method
 - Decode the shellcode and read it
- Dynamic method
 - A bit more interesting

Analysis techniques

- Method 1
 - Identify the shellcode, patch the first few bytes to `0xCC`
 - Start up Office, attach WinDBG to it and 'g'
 - Open up the document
 - If you did it right, you should hit the `int 3` and then single step as needed. If not then you probably got infected.

Analysis techniques

- Method 2
 - Pick an any executable
 - Copy the shellcode and put it in the binary and set eip there.
 - Single step just like any an executable.

Analysis techniques

- Method 3
 - Save the file, i.e., "c:\temp\sc.bin"
 - Open up notepad.exe, calc.exe, whatever.
 - Attach WinDBG to it.
 - .dvalloc <size of sc.bin>
 - .readmem c:\temp\sc.bin addr <size of sc.bin>
 - Real shellcode address = <addr + sc offset>
 - r eip=<addr + sc offset>;t (not 'g')

Analysis techniques

The screenshot displays the WinDbg 6.9.0001.70 X86 interface. The title bar indicates the process being debugged is 'Pid 604 - WinDbg: 6.9.0001.70 X86'. The menu bar includes File, Edit, View, Debug, Window, and Help. The toolbar contains various icons for file operations, debugging actions, and window management. The 'Disassembly' window is active, showing the instruction stream starting at offset 0x00900000+0xebd. The instructions are as follows:

```
00900e8b 0000 add byte ptr [eax],al
00900e8d 0000 add byte ptr [eax],al
00900e8f 0000 add byte ptr [eax],al
00900e91 0000 add byte ptr [eax],al
00900e93 0000 add byte ptr [eax],al
00900e95 0000 add byte ptr [eax],al
00900e97 0000 add byte ptr [eax],al
00900e99 0000 add byte ptr [eax],al
00900e9b 0000 add byte ptr [eax],al
00900e9d 0000 add byte ptr [eax],al
00900e9f 0000 add byte ptr [eax],al
00900ea1 0000 add byte ptr [eax],al
00900ea3 0000 add byte ptr [eax],al
00900ea5 0000 add byte ptr [eax],al
00900ea7 0000 add byte ptr [eax],al
00900ea9 0000 add byte ptr [eax],al
00900eab 0000 add byte ptr [eax],al
00900ead 0000 add byte ptr [eax],al
00900eaf 0000 add byte ptr [eax],al
00900eb1 0000 add byte ptr [eax],al
00900eb3 0000 add byte ptr [eax],al
00900eb5 0000 add byte ptr [eax],al
00900eb7 0000 add byte ptr [eax],al
00900eb9 0000 add byte ptr [eax],al
00900ebb 0000 add byte ptr [eax],al
00900ebd 58 pop eax
00900ebe 50 push eax
00900ebf e927010000 jmp 00900feb
00900ec4 5f pop edi
00900ec5 64a130000000 mov eax,dword ptr fs:[00000030h]
00900ec8 8b400c mov eax,dword ptr [eax+0Ch]
00900ece 8b701c mov esi,dword ptr [eax+1Ch]
00900ed1 ad lodsd dword ptr [esi]
00900ed2 8b6808 mov ebx,dword ptr [eax+8]
00900ed5 cc int 3
00900ed6 cc int 3
00900ed7 6a06 push 6
00900ed9 59 pop ecx
00900eda e8c7000000 call 00900fa6
00900edf e2f9 loop 00900eda
00900ee1 83c704 add edi,4
00900ee4 8bc5 mov eax,ebp
00900ee6 40 inc eax
00900ee7 80385b cmp byte ptr [eax].5Bh
00900eea 75fa jne 00900ee6
00900eec 807801c3 cmp byte ptr [eax+1].0C3h
00900ef0 75f4 jne 00900ee6
00900ef2 894618 mov dword ptr [esi+18h],eax
00900ef5 ba00001300 mov edx,130000h
00900efa 42 inc edx
00900efb 52 push edx
00900efc 5d pop ebp
```

The 'Command' window shows the following text:

```
*****
* Symbol loading may be unreliable without a symbol search path. *
* Use .symfix to have the debugger choose a symbol path. *
* After setting your symbol path, use .reload to refresh symbol locations. *
*****
Executable search path is:
ModLoad: 01000000 01014000 C:\WINDOWS\system32\notepad.exe
ModLoad: 7c900000 7c9b0000 C:\WINDOWS\system32\ntdll.dll
ModLoad: 7c800000 7c8f4000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 763b0000 763f9000 C:\WINDOWS\system32\comdlg32.dll
ModLoad: 77f60000 77fd6000 C:\WINDOWS\system32\SHLWAPI.dll
ModLoad: 77c10000 77c68000 C:\WINDOWS\system32\msvcrt.dll
ModLoad: 77f10000 77f56000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d40000 77dd0000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dd0000 77de6000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e70000 77f01000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 773d0000 774d2000 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df
ModLoad: 7c9c0000 7d1d4000 C:\WINDOWS\system32\SHELL32.dll
ModLoad: 73000000 73026000 C:\WINDOWS\system32\WINSPOOL.DRV
ModLoad: 5cb70000 5cb96000 C:\WINDOWS\system32\ShimEng.dll
ModLoad: 6f880000 6fa4a000 C:\WINDOWS\AppPatch\AcGenral.DLL
ModLoad: 76b40000 76b6d000 C:\WINDOWS\system32\WINMM.dll
ModLoad: 774e0000 7761c000 C:\WINDOWS\system32\ole32.dll
ModLoad: 77120000 771ac000 C:\WINDOWS\system32\OLEAUT32.dll
ModLoad: 77be0000 77bf5000 C:\WINDOWS\system32\MSACM32.dll
ModLoad: 77c00000 77c08000 C:\WINDOWS\system32\VERSION.dll
ModLoad: 769c0000 76a73000 C:\WINDOWS\system32\USERENV.dll
ModLoad: 5ad70000 5ada8000 C:\WINDOWS\system32\UxTheme.dll
(25c.61c): Break instruction exception - code 80000003 (first chance)
eax=7ffd4000 ebx=00000001 ecx=00000002 edx=00000003 esi=00000004 edi=00000005
eip=7c901230 esp=003affcc ebp=003afff4 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=0038 gs=0000 efl=00000246
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\WINDOWS\system32\ntdll
ntdll!DbgBreakPoint:
7c901230 cc int 3
0:001> .dvalloc 23C9E
Allocated 24000 bytes starting at 00900000
0:001> .readmem c:\temp\sc.bin 00900000 10x23C9E
Reading 23c9e bytes.....
0:001> r eip=0x00900000 + 0xebd
0:001> r
eax=7ffd4000 ebx=00000001 ecx=00000002 edx=00000003 esi=00000004 edi=00000005
eip=00900ebd esp=003affcc ebp=003afff4 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=0038 gs=0000 efl=00000246
00900ebd 58 pop eax
0:001> t
eax=7c9507a8 ebx=00000001 ecx=00000002 edx=00000003 esi=00000004 edi=00000005
eip=00900ebe esp=003affd0 ebp=003afff4 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=0038 gs=0000 efl=00000246
00900ebe 50 push eax
0:001>
```

The status bar at the bottom shows 'Ln 0, Col 0 | Sys 0: <Local> | Proc 000:25c | Thrd 001:61c | ASM | OVR | CAPS | NUM'.

Analysis techniques

- Method 4 (best one)
 - Save the file as "c:\temp\sc.bin"
 - Open the file in an editor (notepad, vim, ...); anything that opens the file.
 - Attach WinDBG to it.
 - .dvalloc <size of sc.bin>
 - .readmem c:\temp\sc.bin addr <size of sc.bin>
 - Real shellcode address = <addr + sc offset>
 - r eip=<addr + sc offset>;t (not 'g')

Detection mechanisms

- You need to be able to fingerprint an OLE structure storage file
 - D0 CF 11 E0 A1 B1 1A E1
- Get the stream name to determine the file type (DOC, PPT, XLS)
- Read the stream content and parse it as we showed earlier
- Determine what records are affected and detect them

Detection mechanisms

- We released the file format specifications for DOC, XLS, PPT, and MSO
 - <http://www.microsoft.com/interop/docs/OfficeBinaryFormats.msp>
- Given the information here and those specifications, you can actually write code to parse and check the validity of the records
 - More information will come soon at BlackHat Vegas...

Miscellaneous

- My team's SWI blog:
<http://blogs.technet.com/swi>
 - We talk about vulnerability details there
 - It is written by people who triage the vulnerability
 - More info? Suggestions?
- Got a bug to report? secure@microsoft.com
- My email: **bda@microsoft.com**