Bruce Dang
Secure Windows Initiative (SWI)
Microsoft

Methods for Understanding and Analyzing Targeted Attacks with Office Documents

Agenda

- Introduction
- Office binary file format (<= 2003)
- Bugs
- Defensive mechanisms
- Exploit structures
- Analysis techniques
- Detection mechanisms
- The surprise…
- Patch process (end-to-end)

Introduction

- Targeted attacks
  - Very popular in the last couple years
  - Bypasses perimeter security devices/software
  - Difficult to detect
  - No technical information in the public
- There are things you can do to mitigate and stop most of the attacks.

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

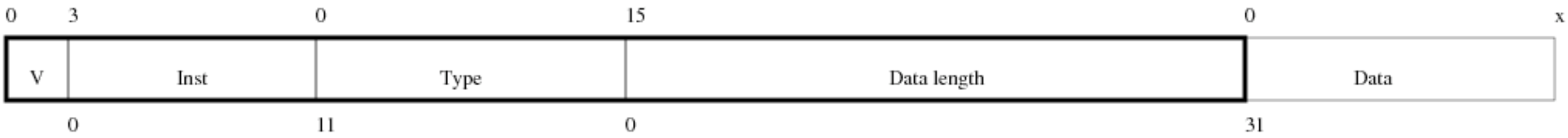
- 1 struct to rule them all

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

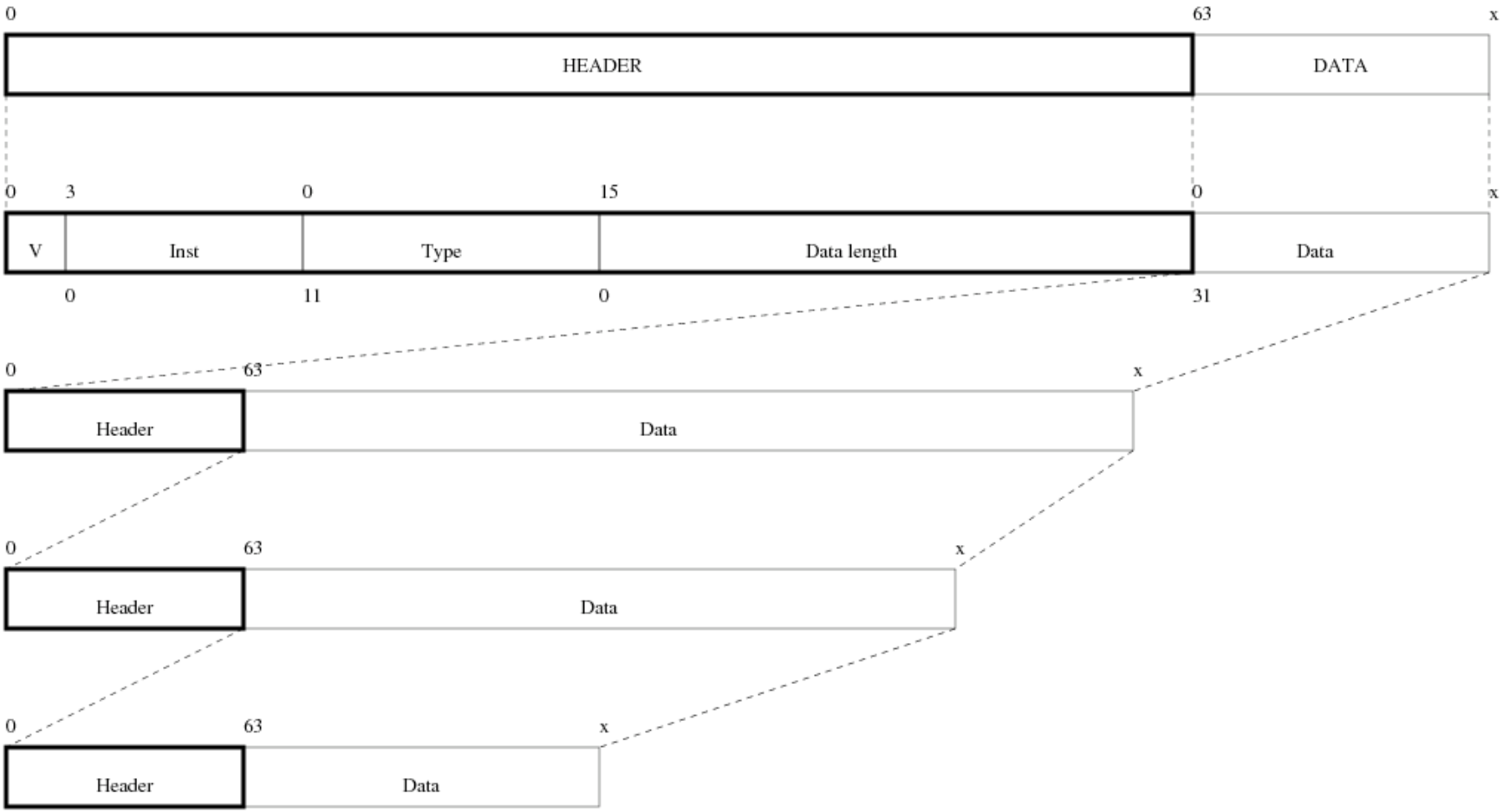
  - If recVer = oxF, then the record is a container; else, it is an atom
  - recType refers to the record type. There are ~ 100 of these in PowerPoint. You can look them up the file format specification.

PowerPoint binary file format

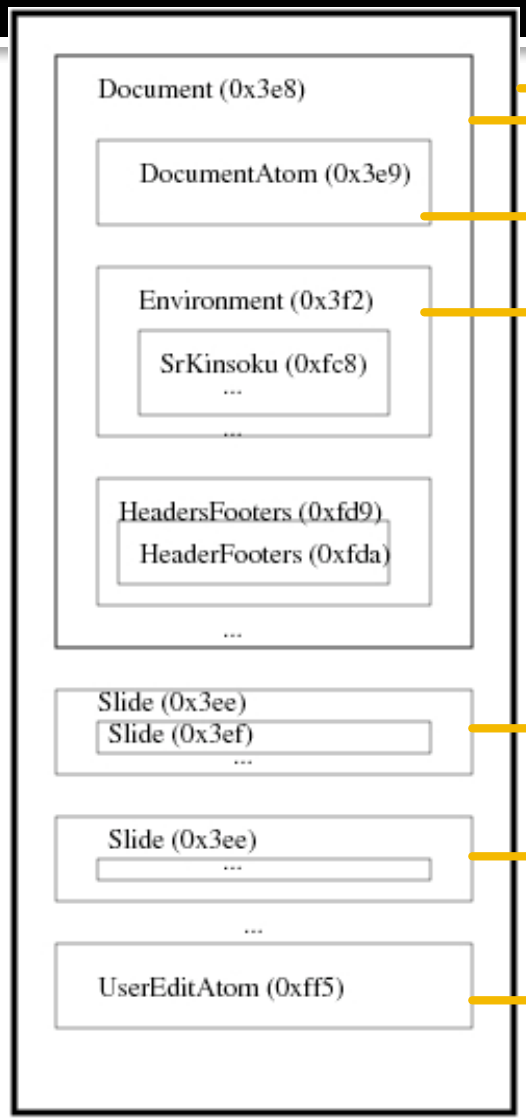
```
typedef struct
{
  uint2 recVer:4;
  uint2 recInstance:12;
  uint2 recType;
  uint4 recLen;
} PPTRHDR_t;
```



PowerPoint binary file format



PowerPoint binary file format



Data from the stream

Document container

Document atom

Environment container

1 slide container per slide

Slide container

Slide container

UserEditAtom atom (usually last)

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.
- Always have the latest patches.

Defensive mechanisms

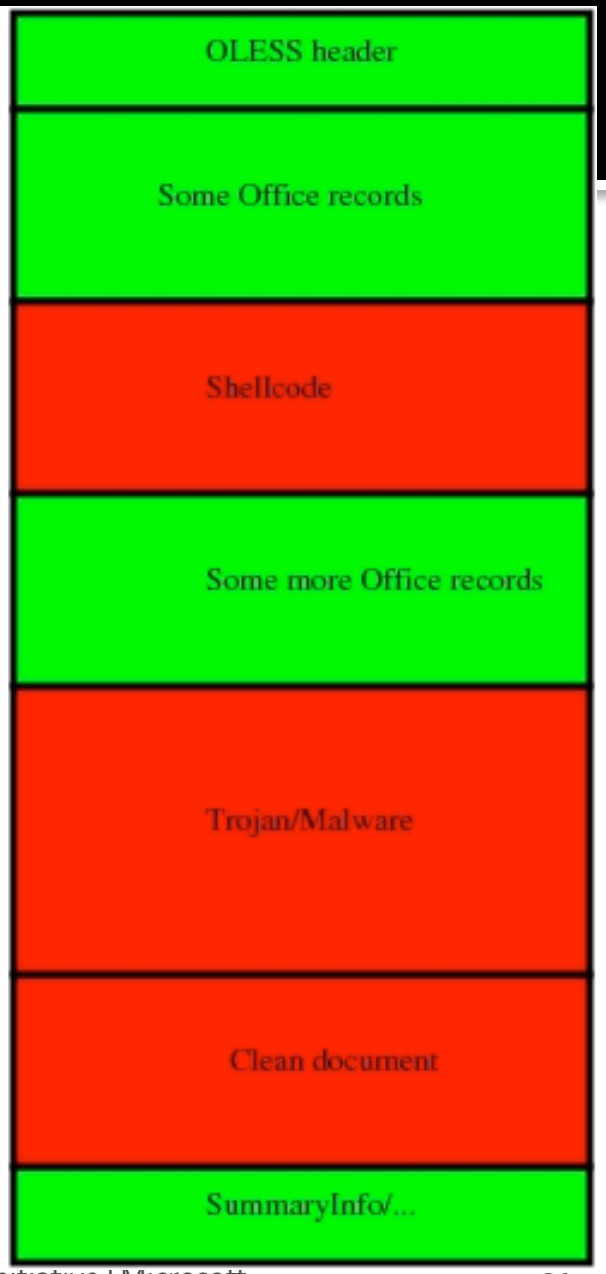
- Use Office 2007

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



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.

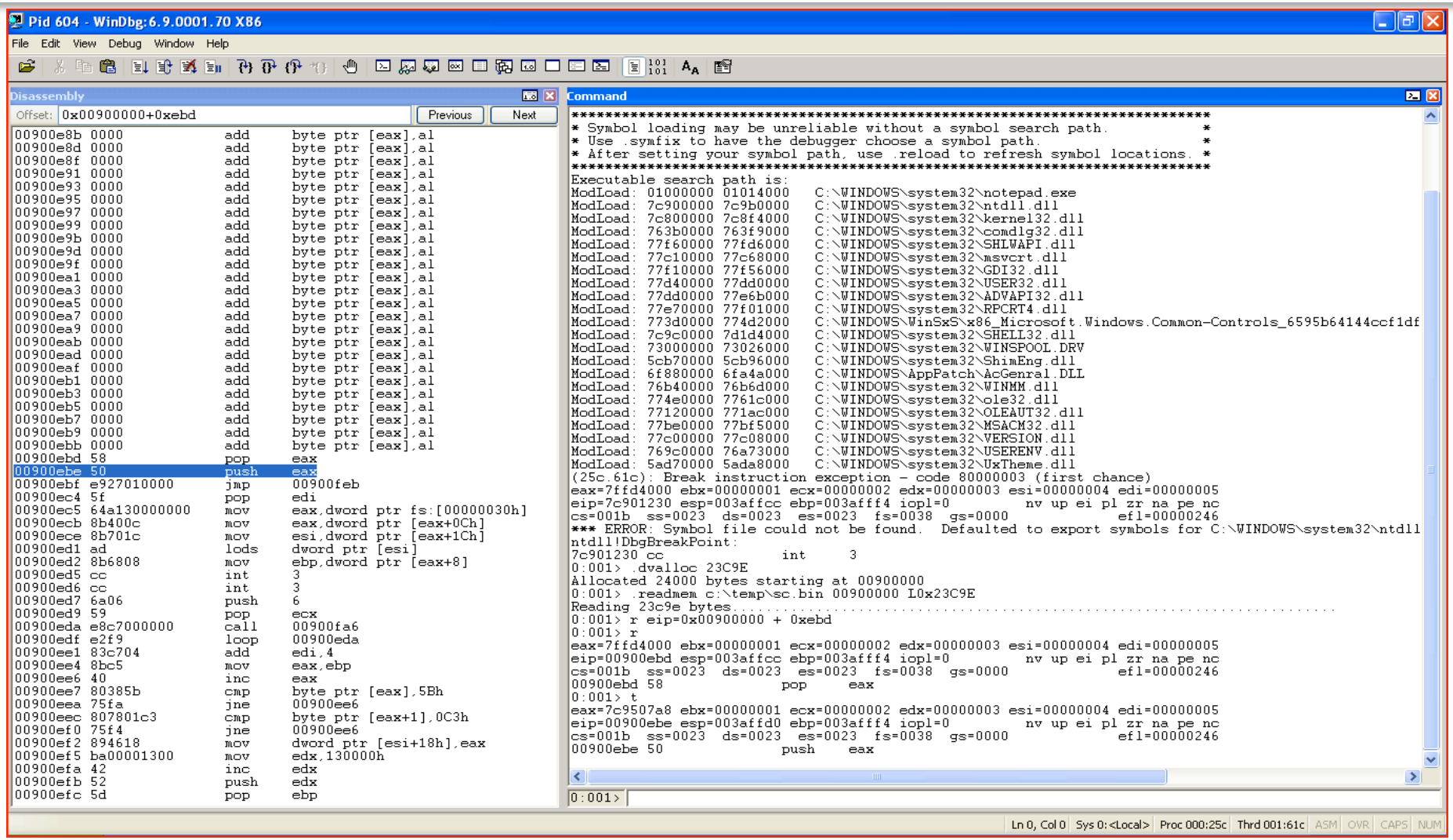
Analysis techniques

- 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



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.mspx
- Given the information here and those specifications, you can actually write code to parse and check the validity of the records.

MAPP

- To write good IDS/AV signatures, you usually have to understand the vulnerabilities.
  - Reverse engineer the patches
  - We will give you the vulnerability details
- MAPP (Microsoft Active Protection Program)

MAPP

- Information available at 5pm PST Monday
- What is included?
  - A technical description of the vulnerability
  - A repro file / packet trace
  - Crash dump / stack trace / disassembly
  - Detection logic
- How to get the data?

FIN

- My team's blog (SWI blog): http://blogs.technet.com/swi
  - We talk about vulnerability details there
  - It is written by people who triage the vulnerability
- Got a bug to report? **secure@microsoft.com**
- Got interesting Office samples and want some help in triaging them? Send them to us.
- My email: **bda@microsoft.com**
- **0x3f…**