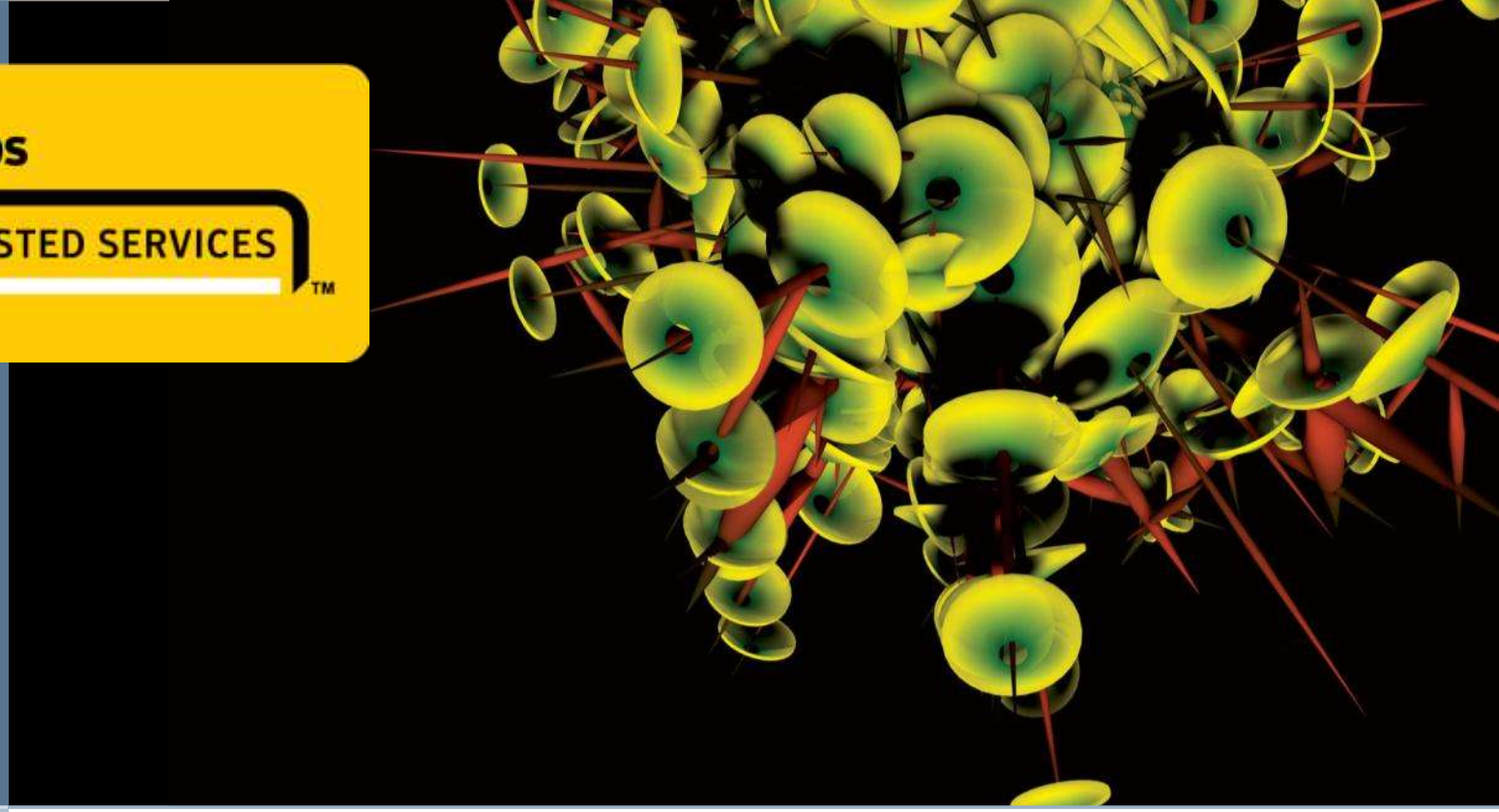


MessageLabs

SYMANTEC HOSTED SERVICES™

Portable Document Malware, the Office, and You

Get owned with it, can't do business without it!

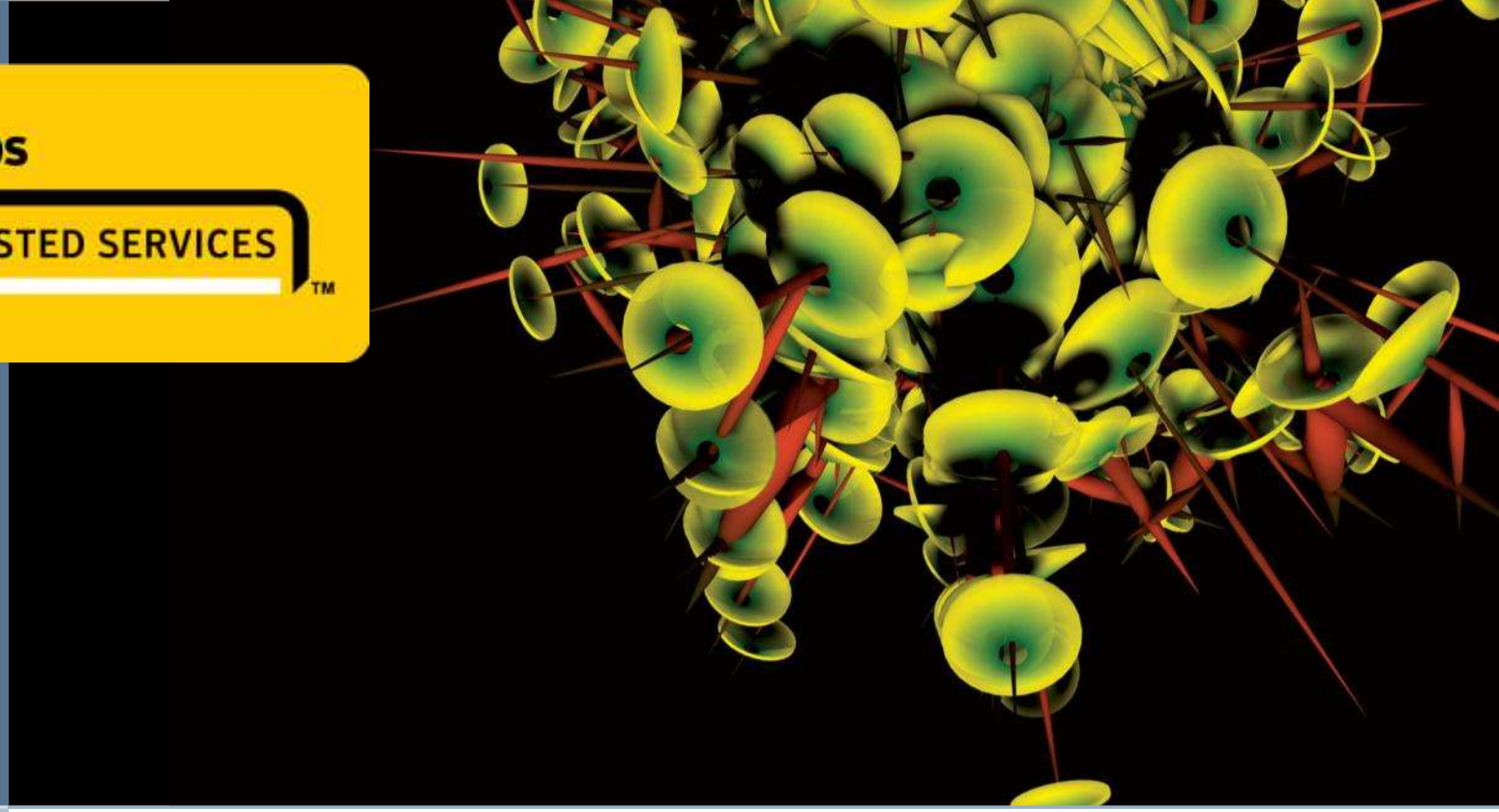
Seth Hardy
Senior Malware Analyst
Threat Research and Response
shardy@messagelabs.com

SecTor 2009 – October 7, 2009

Full Disclosure

MessageLabs

SYMANTEC HOSTED SERVICES™



The Problem

Malicious Executables

- Common knowledge not to run suspicious executables?
  - Windows will ask if you really want to…
  - IE, Firefox, Chrome will ask if you really want to…
  - AV software doesn't look kindly upon executables.
  - Many AV services just block executables outright.
- Do I really want to double-click on postcard.gif.jpg.pif.bat.exe?

Malicious Documents

- But, what about documents?
- Common knowledge about the dangers of .doc?
  - Don’t trust suspicious macros!
  - Don’t trust suspicious formulas!
  - Different “security zones” in Office make it safe!

- True or false:
  - Turning off macros makes .doc and friends safe?
  - PDF is the safe alternative to .doc?

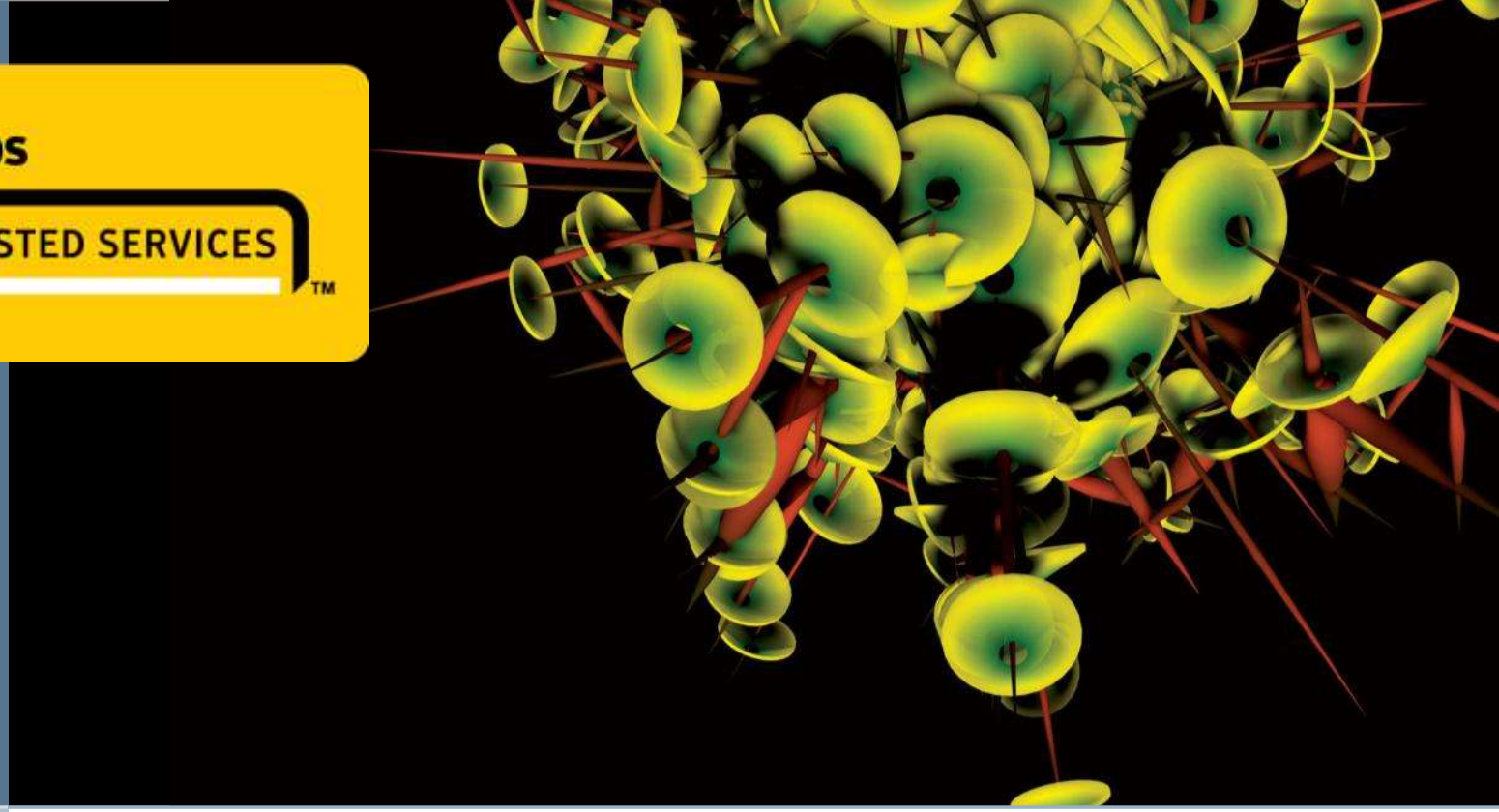
The Problem

- Documents greatly outnumber executables in the workplace.
- We can’t easily give up documents like we can executables.
- We’re convinced that (certain kinds of) documents are safe.

- The result?
  - Targeted trojans.
  - PDFs now common in exploit packs.
  - MetaPhish for the “social aspect of pentesting.”

MessageLabs

SYMANTEC HOSTED SERVICES™



OLE Office Documents

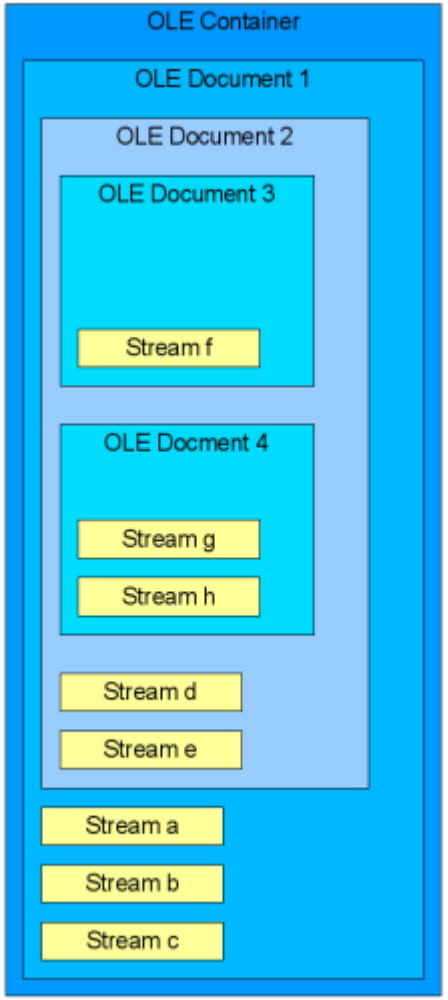
Types of OLE Documents

- The common documents we'll focus on here:
  - Word, Excel, PowerPoint
- But so much more is OLE…
  - Office: Access, Outlook, Publisher, Visio
  - WordPerfect
  - Windows Installer (.msi)
- Office 12 format is *not* OLE (but may have OLE objects).

- File format specs at:
  - http://msdn.microsoft.com/en-us/library/cc313105.aspx

OLE Container

- Similar to a filesystem:
  - Container metadata
  - Storage objects
  - Stream objects
- Storage objects: “directories”
- Stream objects: “files”
- Documents are storage objects with specific streams.
- Documents can be nested easily!



OLE Container Metadata

```
00000000  d0 cf 11 e0 a1 b1 1a e1  00 00 00 00 00 00 00 00  |ÐÏ.à¡±.á........|
00000010  00 00 00 00 00 00 00 00  3e 00 03 00 fe ff 09 00  |........>...þÿ..|
00000020  06 00 00 00 00 00 00 00  00 00 00 00 0a 00 00 00  |................|
00000030  ce 04 00 00 00 00 00 00  00 10 00 00 fe ff ff ff  |Î...........þÿÿÿ|
00000040  00 00 00 00 fe ff ff ff  00 00 00 00 c4 04 00 00  |....þÿÿÿ....Ä...|
00000050  c5 04 00 00 c6 04 00 00  c7 04 00 00 c8 04 00 00  |Å...Æ...Ç...È...|
00000060  c9 04 00 00 ca 04 00 00  cb 04 00 00 cc 04 00 00  |É...Ê...Ë...Ì...|
00000070  cd 04 00 00 ff ff ff ff  ff ff ff ff ff ff ff ff  |Í...ÿÿÿÿÿÿÿÿÿÿ|
00000080  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ|
*
000001f0  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ|
```

- File magic, unique identifier, version numbering.
- Filesystem information: sector size, count, allocation table, start.

Word Documents

- Common streams:
  - WordDocument – FIB, document text, pointers
  - [01]Table – data referenced from elsewhere
  - Data – data referenced from elsewhere
  - \005SummaryInformation – summary information
  - \005DocumentSummaryInformation – summary information
- Common storages:
  - ObjectPool – embedded OLE objects

Excel Documents

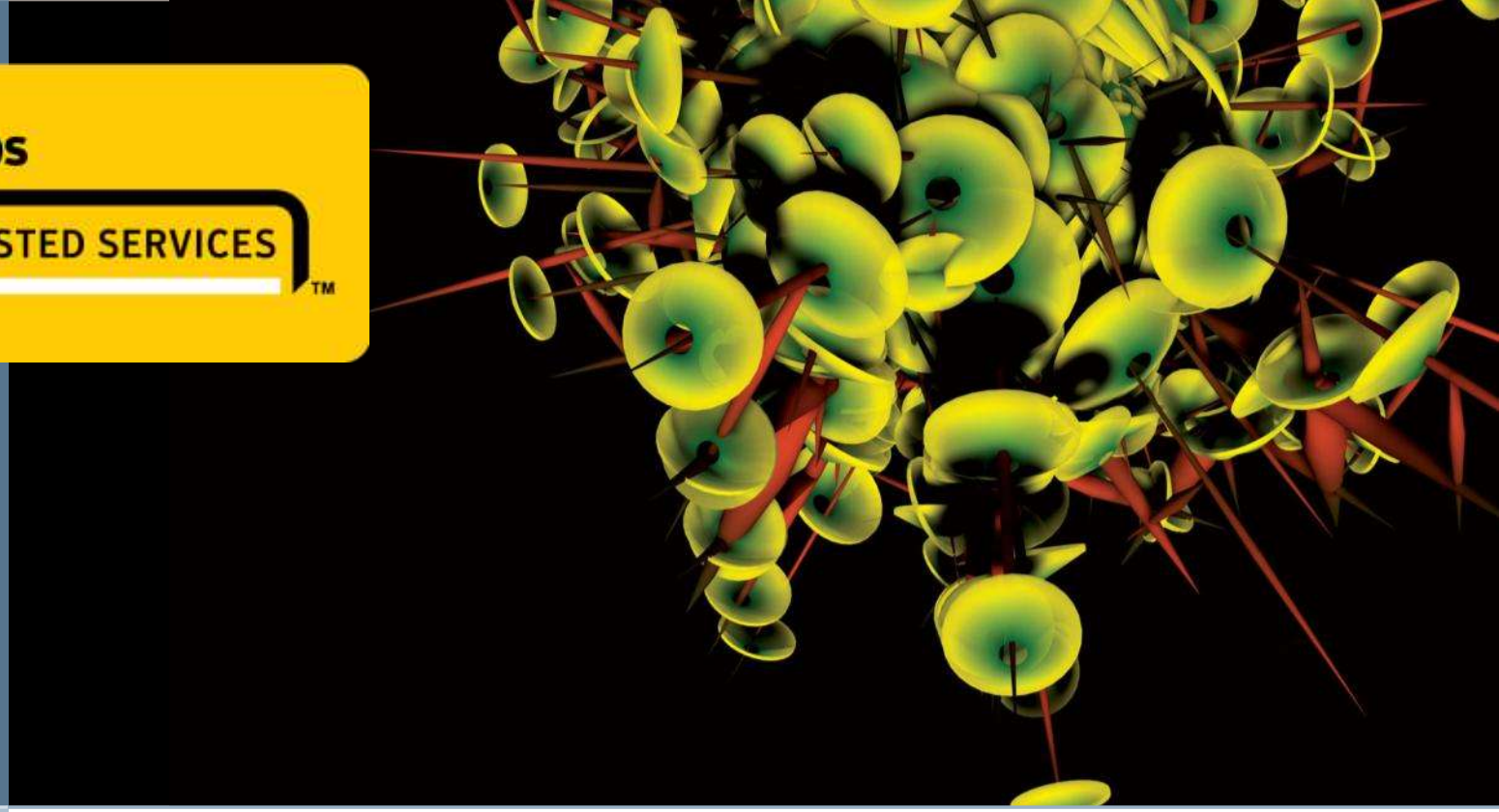
- Common streams:
  - Workbook – BIFF objects, substreams
    - Globals, charts, dialogs, macros, worksheets
  - \005SummaryInformation – summary information
  - \005DocumentSummaryInformation – summary information
- Common storages:
  - _VBA_PROJECT_CUR – VBA storage for macros

PowerPoint Documents

- Common streams:
  - PowerPoint Document – contains records
    - Container records, atom records
  - Pictures – pictures! MS-ODRAW records.
  - \005SummaryInformation – summary information
  - \005DocumentSummaryInformation – summary information

MessageLabs

SYMANTEC HOSTED SERVICES™



OLE Vulnerabilities

OLE Vulnerabilities

- It's been many years since the old problems…
  - Macro viruses (e.g. W97M/Melissa)
  - Formula viruses (e.g. XF97/Yagnuul)
- … but this isn't what we'll be talking about.
  (Although, Melissa is over 10 years old, and *still in the wild!*)

- The "new" problem: exploitable vulnerabilities in Office programs (Word, Excel, PowerPoint) that lead to code execution.

Types of Vulnerabilities

- Are you expecting groundbreaking new information?
- Sorry, nothing to see here:
  - Buffer overflows
  - Off-by-one errors
  - Lack of parameter validation
  - etc., etc., etc.
- Just your standard C-style errors that come from parsing a very complicated file format that's poorly documented and contains many, many edge cases.

How Common?

MS09-Jan: 0

MS09-Feb: 1

MS09-Mar: 0

MS09-Apr: 1

MS09-May: 1

MS09-Jun: 2

MS09-Jul: 1

MS09-Aug: 1

MS09-Sep: 0

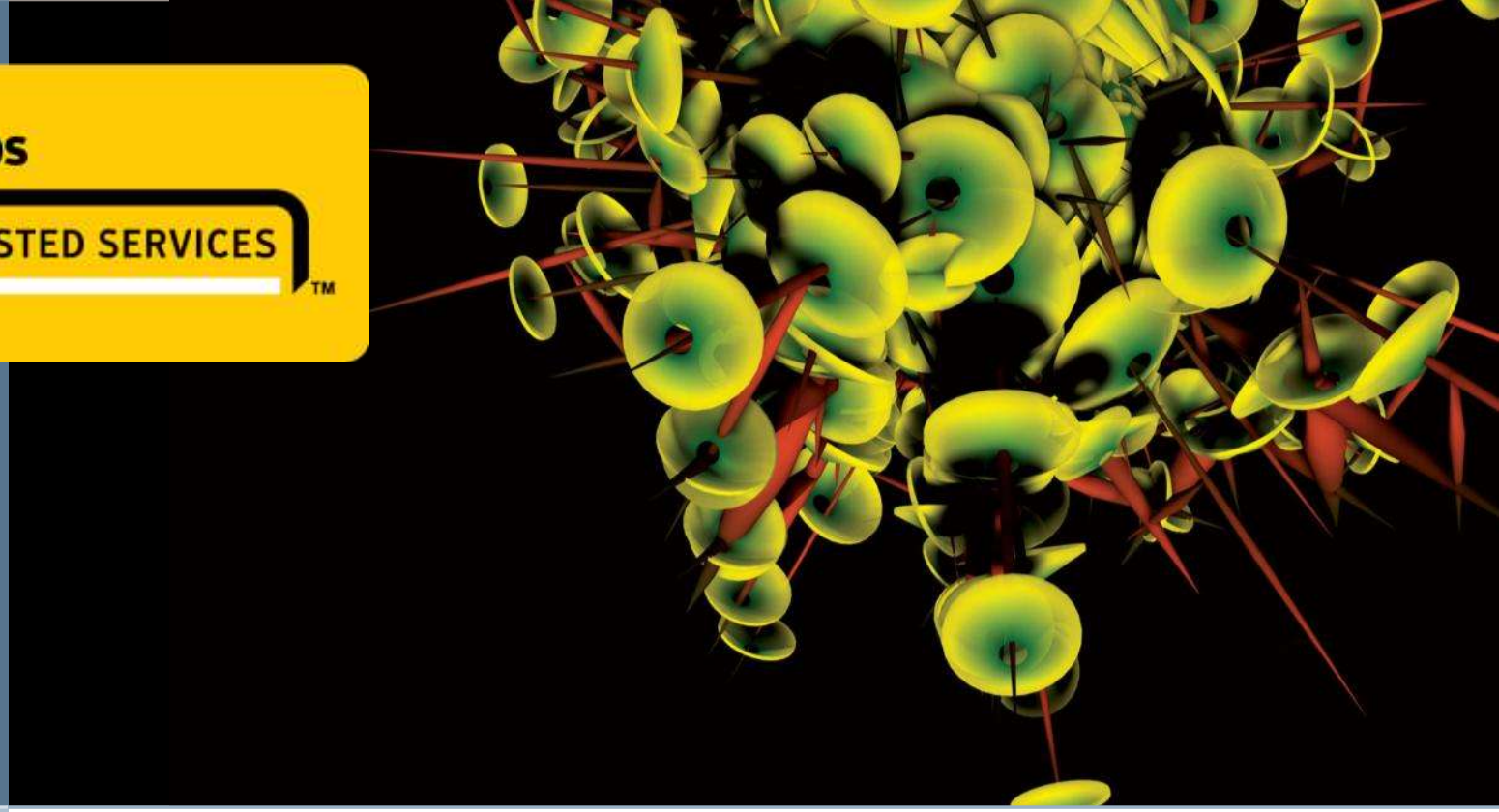
- Total this year so far: **7**
- This counts bulletins, not variations within a bulletin.

Scanning Isn't Easy

- Certain things are easy to look for if you can parse OLE files:
  - Embedded executables
  - Bad stream names (\x00ictures, 2Table)
- Vulnerabilities are hard to find:
  - Very few bytes required to trigger.
  - Can look just like a valid value, depends on context.
  - Shellcode can be far, far away from the vulnerability.
- Recent targeted attacks using PowerPoint were MS06-028!

MessageLabs

SYMANTEC HOSTED SERVICES™



OLE Anti-Virus Bypass

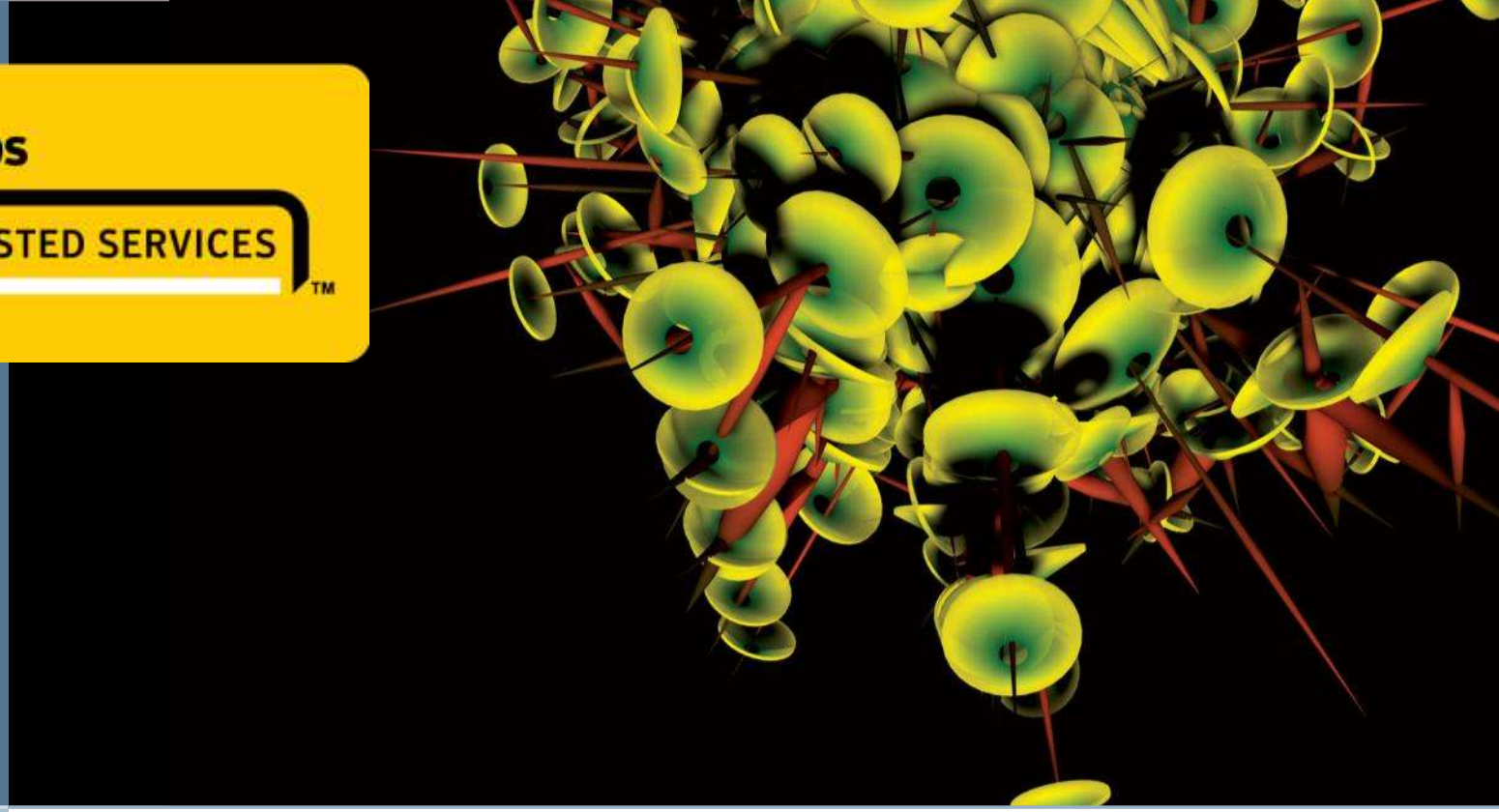
Some Generic Hints

- It’s not that hard… but I’m not going to talk about it in detail.
- Steering very clear of the NDA line.

- Streams that contain free-form data are great to hide things.
- Look for the “alternative” data structures in the spec.
  - Localization, older versions, …
- Look for ways of rearranging things.
  - Office is often too forgiving about non-spec documents.
  - Alternate ordering, missing fields, bad values, …

MessageLabs

SYMANTEC HOSTED SERVICES™



Portable Document Format

PDF File Format

- There's a lot to cover, so here are some interesting points:
  - PDF files are text-based, not binary.
  - Very flexible formatting.
  - Everything outside the PDF structure is ignored.
  - Many different compression filters, image types.
  - Versions are backwards compatible.
  - After v1.7, PDF was adopted as an ISO spec.

- Pre-ISO specs can be found at:

http://www.adobe.com/devnet/pdf/pdf_reference_archive.html

PDF Structure

- Four main parts to the structure:
  - Header
  - Body
  - xref table
  - Trailer
- File is read backwards: trailer, xref, body.
  - Trailer: where is xref? Is the PDF encrypted?
  - xref: where are the objects?
  - Body: what are the objects?
- Incremental updates: just add a new PDF to the end.
  - Since PDF is parsed backwards, appended objects read first.

PDF (Lack of) Structure

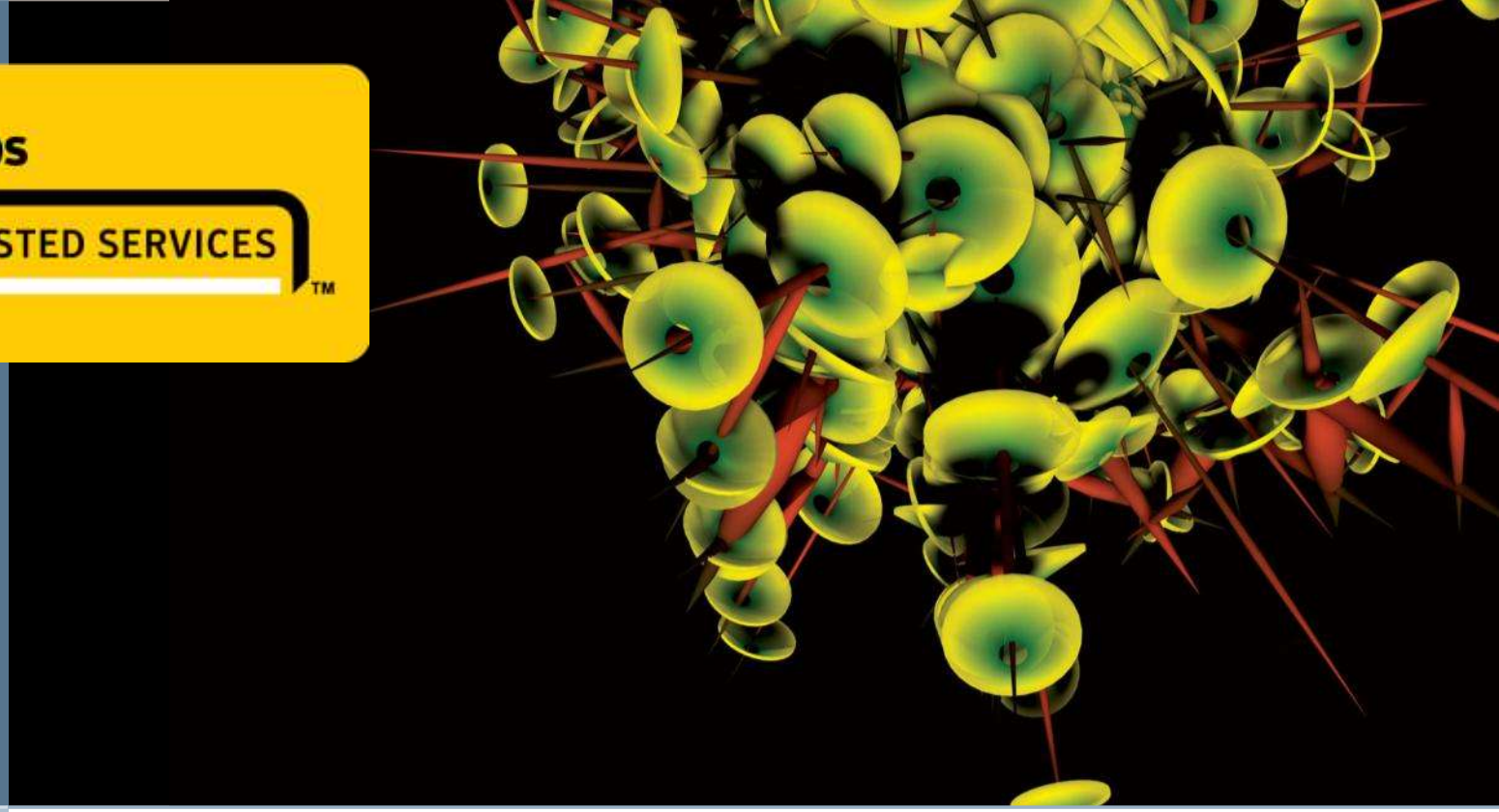
- Since PDFs are text based, there's a lot of flexibility:
  - Whitespace – generally ignored!
  - Name canonicalization – hex codes allowed.
  - Incremental updates (again) – as many as you want.
  - Extra data outside the PDF is ok.
- Adobe Reader is also very forgiving…
  - Broken/missing xref table?
  - Objects not found, or duplicate objects?

PDF (Dangerous) Features

- Some people want features.
- Some people want their documents to be documents.
- Some people want to be able to portscan in PDF, I guess?
  - Many filters, encodings, compression types, …
  - JavaScript
  - Flash
- The attack surface is *huge*.

MessageLabs

SYMANTEC HOSTED SERVICES™



PDF Vulnerabilities

Types of PDF Vulnerabilities

- How are PDFs being used as malware in the wild?
  - Embedded files (not a vulnerability)
  - JavaScript
  - Compressed images (JBIG2 in particular)
  - Flash (not exactly PDF, but…)
- Most PDF vulnerabilities are of the JavaScript variety.
- Exploit packs will have PDFs with multiple JS vulnerabilities.
  - JS gathers version information and uses the appropriate vulnerability!

Embedded Files

- The simplest way? Just put an executable in the PDF…
- MetaPhish does this as a “proof of concept”:
  - Embed executable.
  - JS can open/run the executable.
  - Adobe Reader asks you if you’re sure…
  - …but who reads the dialog box before you click yes?
- The PDF format makes embedding files or arbitrary data easy.
  - Before the PDF header.
  - After the PDF trailer.

PDF JavaScript Vulnerabilities

- Buffer overflows in the JavaScript engine?
  - util.printf() – CVE-2008-2992
  - Collab.getIcon() – CVE-2009-0927
  - getAnnots() – CVE-2009-1492
  - spell.customDictionaryOpen() – CVE-2009-1493

- Often (but not always) require an obvious JS heap spray.

```
for (iCnt=128;iCnt>=0;--iCnt) nop += unescape("%u9090%u9090%u9090%u9090%u9090");
heapblock = nop + payload;
bigblock = unescape("%u9090%u9090");
headersize = 20;
spray = headersize+heapblock.length
while (bigblock.length<spray) bigblock+=bigblock;
fillblock = bigblock.substring(0, spray);
block = bigblock.substring(0, bigblock.length-spray);
while(block.length+spray < 0x40000) block = block+block+fillblock;
mem = new Array(); for (i=0;i<1400;i++) mem[i] = block + heapblock;
```

PDF JavaScript Vulnerabilities

- These are generally easy to identify…
  - If you can find the JavaScript.
  - If you can then extract the JavaScript.
  - If you can then deobfuscate the JavaScript.
- Unfortunately, signature-based solutions don't do any of these.
- Want an example (or five)? Just check milw0rm!

Image Vulnerabilities

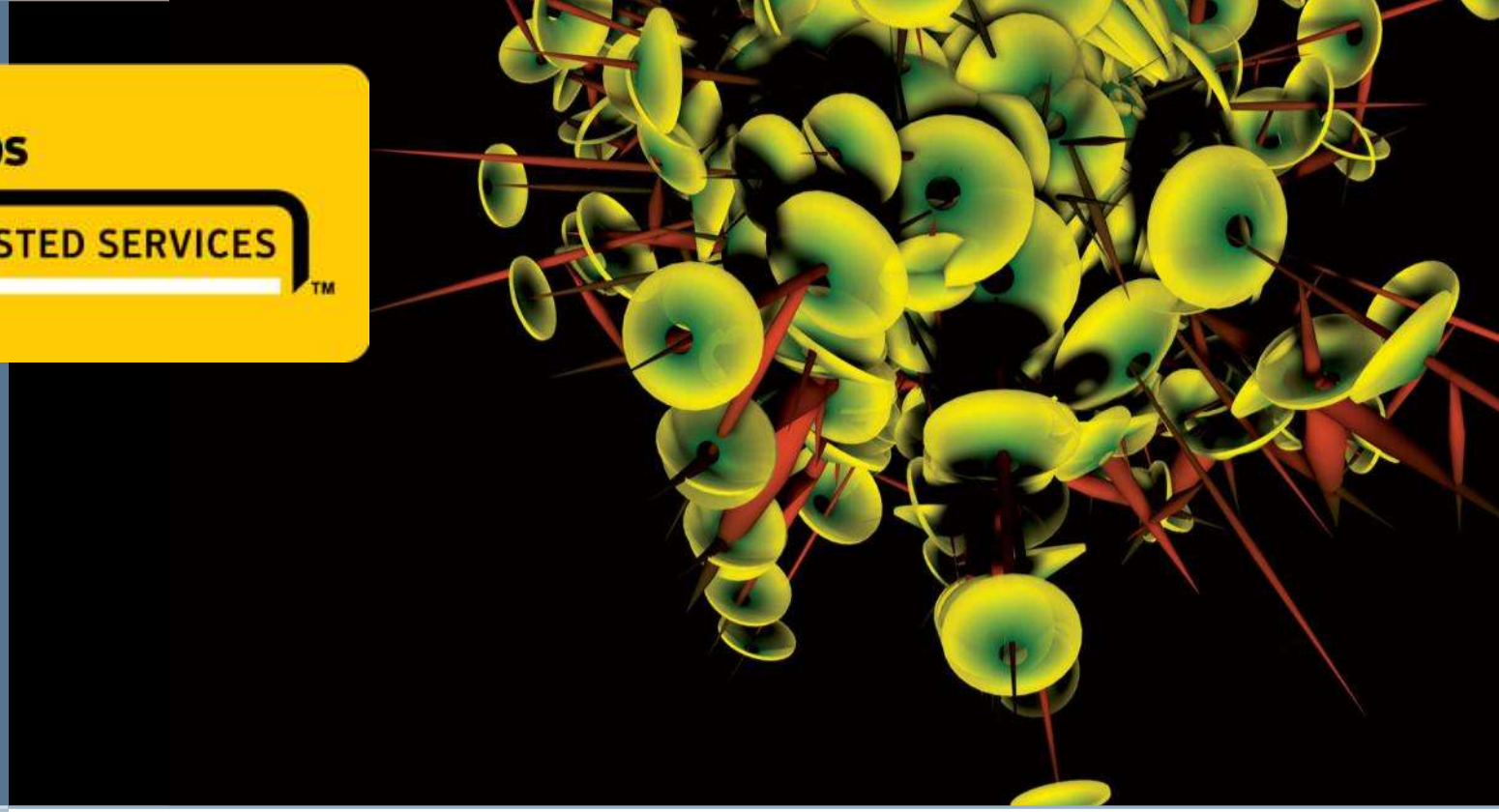
- Other parsers have bugs, too.
- In this case, the JBIG2 image compression:
  - Array indexing errors.
  - Boundary and off-by-one errors.
  - Buffer overflows.
  - Other input validation errors.
- Sound familiar?
- Some of these need an accompanying JS heap spray.
- Some don’t.

Flash Vulnerabilities

- Flash is everywhere, just like PDF.
- Flash and Acrobat share some code.
- Flash has its own problems.
- Rich Media embedding of a SWF in a PDF?
  - All the SWF vulnerabilities…
  - All the PDF obfuscation…
  - ActionScript heap spraying, so we don't need obvious JS!
- What could possibly go wrong?

MessageLabs

SYMANTEC HOSTED SERVICES™



PDF Anti-Virus Bypass

It’s Bad Enough Already

- Detection rates, given the following criteria:
  - PoC directly from milw0rm used.
  - No AV bypass techniques used.
  - Obvious JavaScript.

It’s Bad Enough Already

- Detection rates, given the following criteria:
  - PoC directly from milw0rm used.
  - No AV bypass techniques used.
  - Obvious JavaScript.

- util.printf() – VT **22/41**

It’s Bad Enough Already

- Detection rates, given the following criteria:
  - PoC directly from milw0rm used.
  - No AV bypass techniques used.
  - Obvious JavaScript.

- util.printf() – VT **22/41**
- Collab.getIcon() – VT **16/41**

It’s Bad Enough Already

- Detection rates, given the following criteria:
  - PoC directly from milw0rm used.
  - No AV bypass techniques used.
  - Obvious JavaScript.

- util.printf() – VT **22/41**
- Collab.getIcon() – VT **16/41**
- getAnnots() – VT **15/41** (was VT **7/40** on 4/29/09)

It’s Bad Enough Already

- Detection rates, given the following criteria:
  - PoC directly from milw0rm used.
  - No AV bypass techniques used.
  - Obvious JavaScript.

- util.printf() – VT ***22/41***
- Collab.getIcon() – VT **16/41**
- getAnnots() – VT **15/41** (was VT **7/40** on 4/29/09)
- spell.customDictionaryOpen – VT **15/41** (was VT **6/40** on 4/29/09)

It’s Bad Enough Already

- What about image compression?
- Not much better, unfortunately.

- JBIG2 global stream: VT **11/41** (was VT **4/39** on 2/24/09)

Generic Bypass

- What about using AV bypass techniques?
- There’s no way simple, well-documented methods could work.

Generic Bypass

- What about using AV bypass techniques?
- There's no way simple, well-documented methods could work.

- Put an owner password on the util.printf() PoC:

  no encryption: VT **22/41**

  encryption: VT **9/41**

Generic Bypass

- What about using AV bypass techniques?
- There's no way simple, well-documented methods could work.

- Put an owner password on the util.printf() PoC:

  no encryption: VT **22/41**

  encryption: VT **9/41**

- Put a whitespace header on it, instead:

  no header: VT **22/41**

  1000 line header*: VT **0/41**

  400 line header: VT **0/41**

Generic Bypass

- What about other ways?
- I won't share the VT results, but…
  - Damage / delete the xref table.
  - Put the nasty stuff in an unreferenced object.
  - Name canonicalization and abbreviation changes.
  - Whitespace.
  - Incremental updates.
  - Change the encoding method.
  - Linearized PDFs.
- Adobe Reader is very forgiving with "damaged" files.

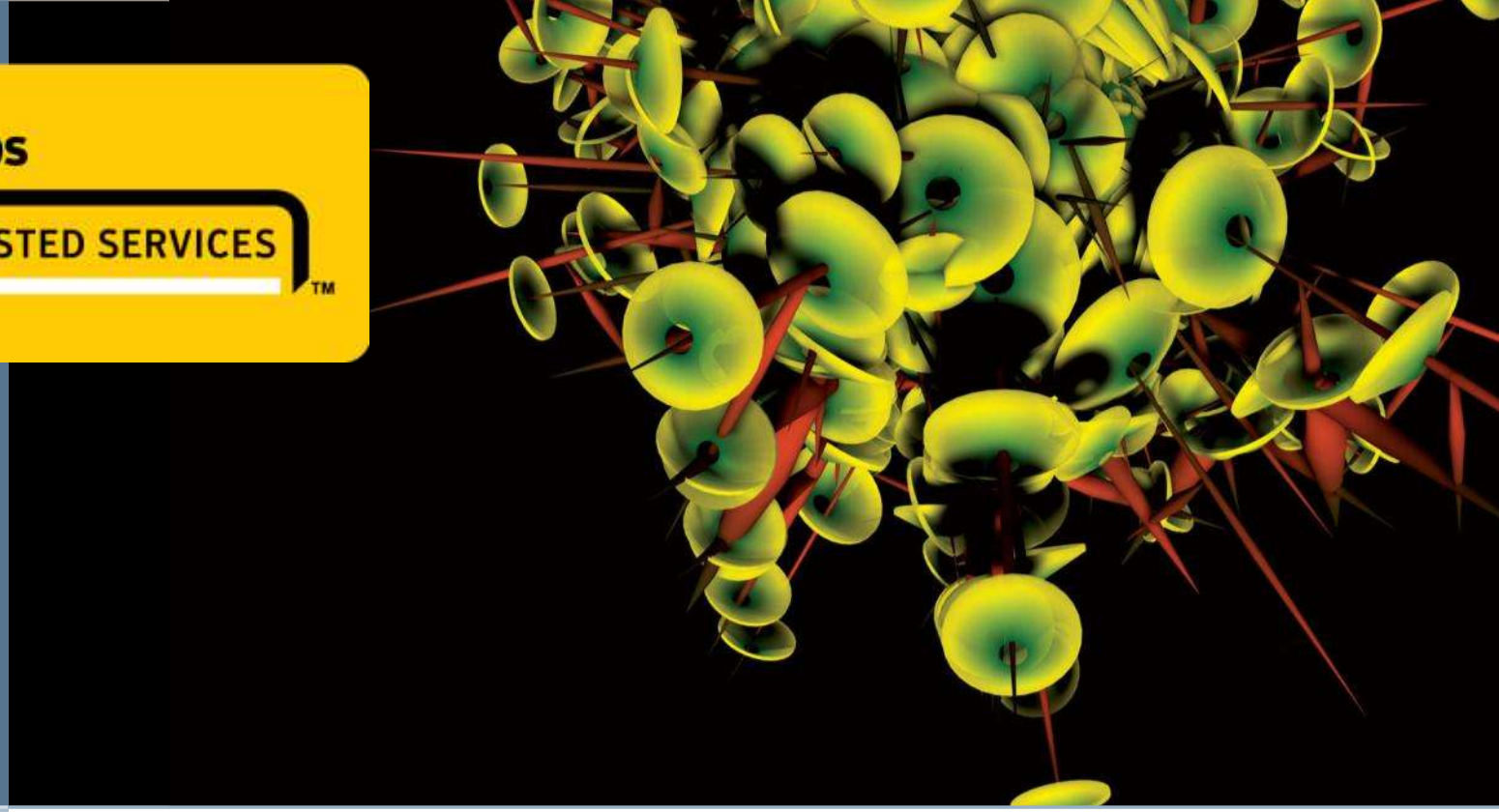
JavaScript Bypass

- JavaScript bypass – what can we do to defeat signatures?
  - Hide the function call that's vulnerable.
  - Hide the heap spray.
  - Hide the shellcode.

- There are enough ways to do this to fill another talk…
- Look for:
  - eval(unescape(…))
  - fromCharCode(…)
  - s//
  - Arrays of numbers or characters

MessageLabs

SYMANTEC HOSTED SERVICES™



A Solution

Two Extremes

- The two different document types here are two extremes.
- OLE documents:
  - Very structured.
  - Very complicated.
  - Many exceptions.
  - Vulnerabilities too small for accurate signaturing.
- PDF documents:
  - Very unstructured.
  - Very complicated.
  - Many exceptions.
  - Vulnerabilities too easily encoded for accurate signaturing.

A Solution

- How do we handle this problem?
- Hopefully this is obvious by now…
  - File format parsing – difficult, but necessary.
  - Validating everything that is there is difficult…
  - Looking for things that shouldn't be there is easier.
  - Either way, still need to find the content.
- Signature-based solutions don't work.

OLE Scanning

- Easier:
  - Look for embedded files.
  - Look for bad streams.
- Harder:
  - Look for shellcode.
  - Look for individual vulnerabilities – need a parser!
- Even the easier suggestions need more than signatures.
- OLE is a difficult problem. Upgrade to Office 12!

PDF Scanning

- Not as bad as OLE, but still some tricky parts.
- Easier:
  - Look for embedded data.
  - Parse all objects (regardless of how they are used).
  - Write simple filter decoders.
- Harder:
  - JavaScript.
  - JavaScript.
  - JavaScript.
- On the plus side, if filter decoders are working, it’s pretty safe to assume any obfuscated JavaScript is bad.

Other Resources

- Understand the file structure first – be familiar with the specs!
- Some tools that may help with analysis:
- OLE:
  - OffVis
  - OfficeCat
- PDF
  - pdf-parser.py
  - pdfid.py

Questions?