ETW for the lazy reverser

Clément "hakril" Rouault

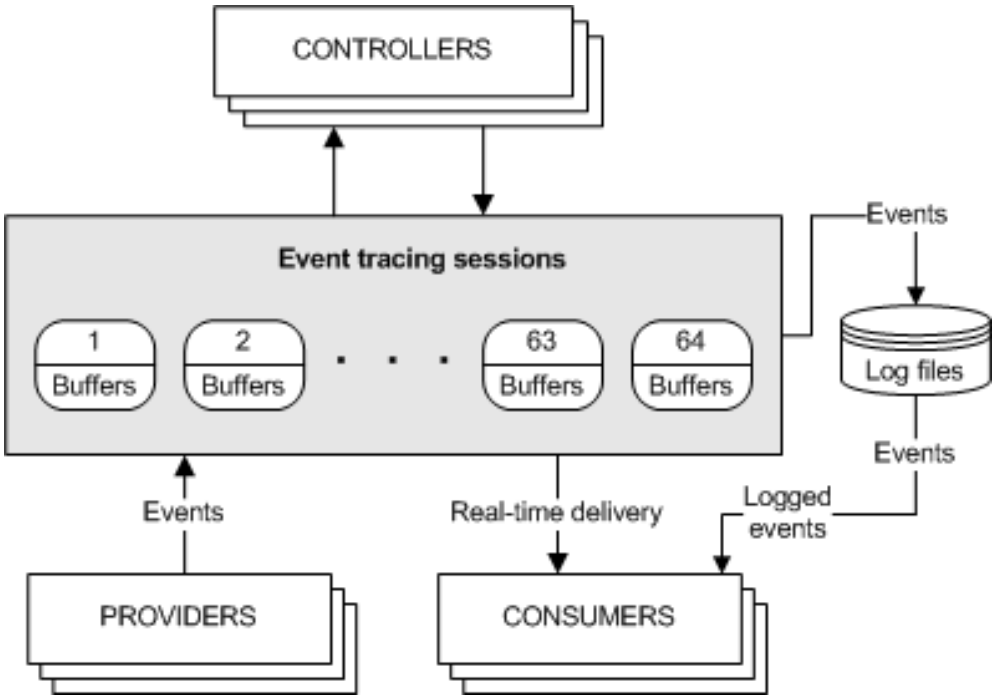
But de la rump

Montrer l'utilité et les possibilités de ETW

- Le titre
  - ETW -> Event Tracing for Windows
  - lazy reverser -> moi
- Contexte
  - Je reverse un processus critique
  - Si j'attache un debugger -> hang + BSOD
  - La flemme de sortir un Kernel DBG (lazy)
  - Ca fait 5 ans que je vois des strings de debug cool et des fonctions "WPP_" que j'ignore
    - Avec des noms de fichiers/fonctions
    - Il est peut-être temps de s'y intéresser !

Event Tracing for Windows

- " ETW is an efficient kernel-level tracing facility that lets you log kernel or application-defined events to a log file. You can consume the events in real time or from a log file […]" MSDN



https://docs.microsoft.com/en-us/windows/desktop/etw/about-event-tracing

ExaTrack

API

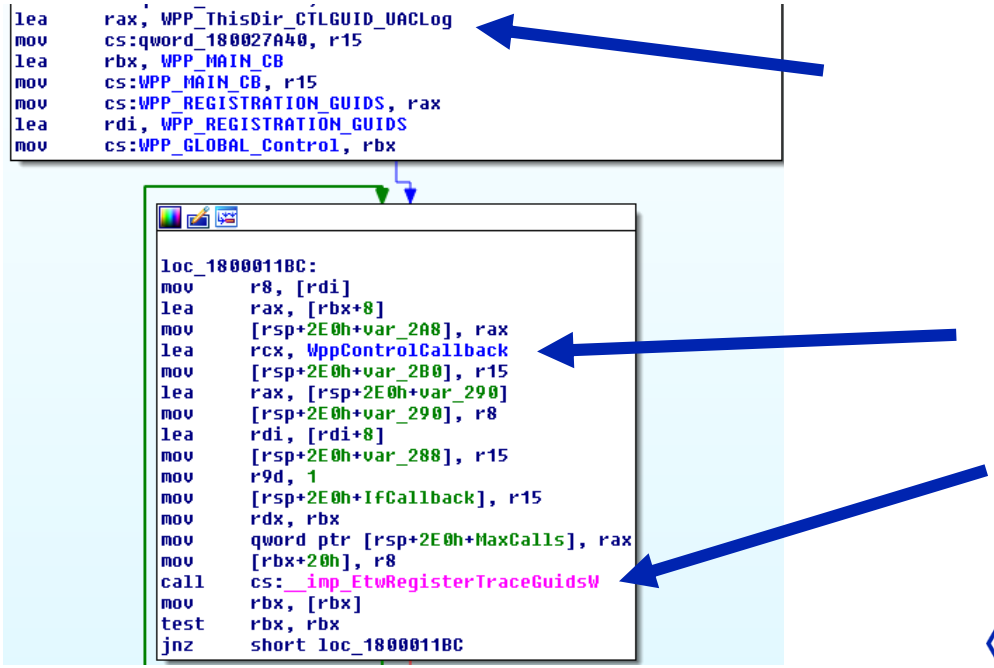
- Coté providers
  - Register le/les providers avec une callback de contrôle
    - `RegisterTraceGuids / EventRegister / NtTraceControl`
    - La callback est appelée quand quelqu'un (dés)active la trace
    - Permet au provider de savoir si quelqu'un consomme les events
  - Génère des events (`TraceEvent / EtwTraceMessage / …`)
- Coté consumers
  - On lance une trace
    - REAL_TIME ou avec un fichier
  - On active les providers que l'on veut recevoir
  - On "process" la trace pour recevoir les évènements
    - Pendant que la trace tourne si REAL_TIME
    - Pendant ou après la fin de trace si on utilise un fichier

Windows software trace preprocessor (WPP)

- Permet de simplifier l'intégration de trace provider dans le code.
  - Gère tout seul la callback de contrôle
    - Qui s'occupe juste de stocker les informations sur la session dans une globale
  - Très fortement utilisé dans le code MS
  - Un problème
    - Normalement il faut connaitre les type d'arguments d'un évènement pour les afficher
    - #YoloPython

Exemple: Appinfo.dll

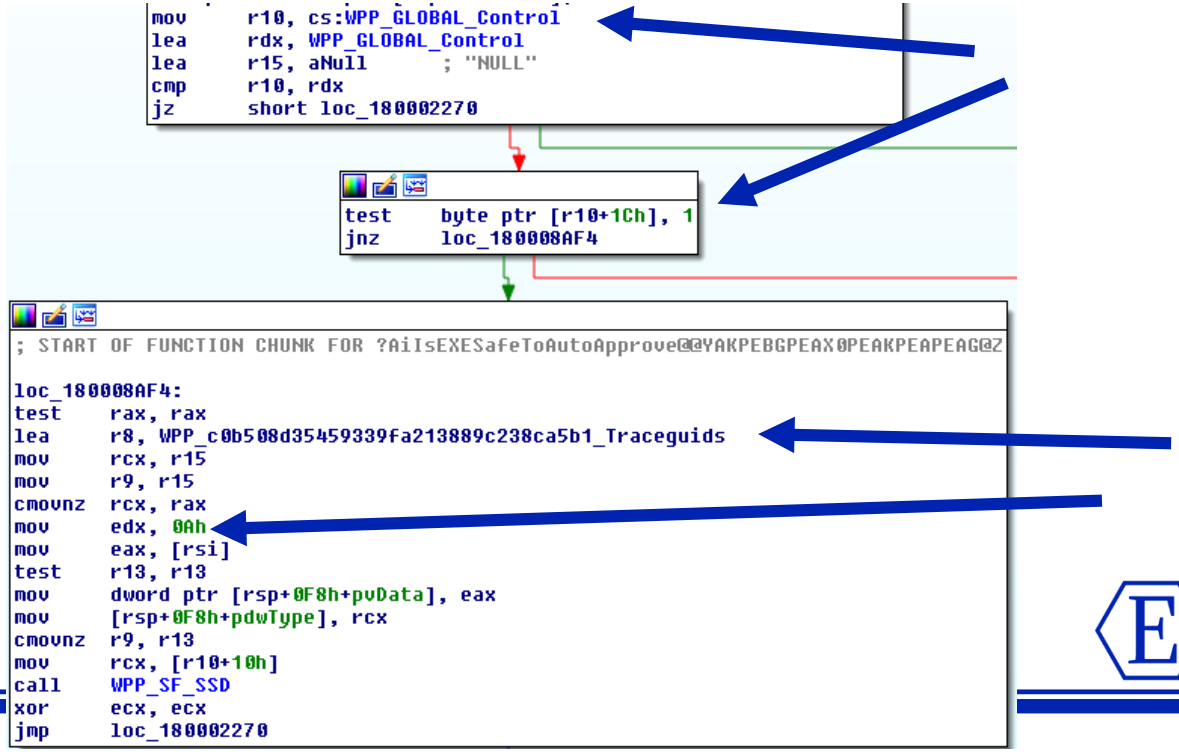
- La dll du service AppInfo
  - Qui s'occupe de l'UAC (User account Control)
    - Toutes les requêtes "RunAs Admin"
- Enregistrement du provider WPP au début de ServiceMain
- `CTLGUID_UACLOG -> CBB61B6D-A2CF-471A-9A58-A4CD5C08FFBA`



ExaTrack

Exemple: Appinfo.dll

- Appinfo!AiIsEXESafeToAutoApprove
  - Event WPP contenant Exec/Commandline
    - Trace GUID `c0b508d35459339fa213889c238ca5b1`
    - EventID 0xA
    - Paramètres: `String String Dword (SSD)`



ExaTrack

Exemple: Appinfo.dll

```
import struct
import windows # https://github.com/hakril/PythonForWindows
import windows.generated_def as gdef
makeg = gdef.GUID.from_string

def callback(event):
    if event.guid == makeg("C0B508D3-5459-339F-A213-889C238CA5B1") and event.id == 10:
       windows.utils.sprint(event, "Event")
       exe, cmdline, other = event.user_data.decode("utf-16").split("\x00", 2)
       print("== UAC request ==")
       print(" * Exe:         <{0}>".format(exe))
       print(" * Commandline: <{0}>".format(cmdline))
    return 0

# No filename -> real time trace
my_trace = windows.system.etw.open_trace("REALTIME_UAC_MONITOR")
my_trace.start()
my_trace.enable("CBB61B6D-A2CF-471A-9A58-A4CD5C08FFBA", 0xff, 0xff)
my_trace.process(callback) # Will hang until trace is stopped:
# logman -ets stop REALTIME_UAC_MONITOR
```

ExaTrack

Exemple: Appinfo.dll

```
λ python simple_uac_realtime.py
Event.EventHeader.Size -> 0x78
Event.EventHeader.HeaderType -> 0x0
Event.EventHeader.Flags -> 0x348
Event.EventHeader.EventProperty -> 0xaa
Event.EventHeader.ThreadId -> 0x2354L
Event.EventHeader.ProcessId -> 0xeb4L
Event.EventHeader.TimeStamp -> 0x1d515eeac7e8006L
Event.EventHeader.ProviderId -> <GUID "C0B508D3-5459-339F-A213-889C238CA5B1">
Event.EventHeader.EventDescriptor.Id -> 0xa
Event.EventHeader.EventDescriptor.Version -> 0x0
Event.EventHeader.EventDescriptor.Channel -> 0x0
Event.EventHeader.EventDescriptor.Level -> 0x0
Event.EventHeader.EventDescriptor.Opcode -> 0x0
Event.EventHeader.EventDescriptor.Task -> 0x0
Event.EventHeader.EventDescriptor.Keyword -> 0x0L
Event.EventHeader.DUMMYUNIONNAME.DUMMYSTRUCTNAME.KernelTime -> 0x0L
Event.EventHeader.DUMMYUNIONNAME.DUMMYSTRUCTNAME.UserTime -> 0x0L
Event.EventHeader.DUMMYUNIONNAME.ProcessorTime -> 0x0L
Event.EventHeader.ActivityId -> <GUID "00000000-0000-0000-0000-000000000000">
Event.BufferContext.anon_01.anon_01.ProcessorNumber -> 0x1
Event.BufferContext.anon_01.anon_01.Alignment -> 0x0
Event.BufferContext.anon_01.ProcessorIndex -> 0x1
Event.BufferContext.LoggerId -> 0x23
Event.ExtendedDataCount -> 0x0
Event.UserDataLength -> 0x50
Event.ExtendedData -> NULL
Event.UserData -> 0x662118
Event.UserContext -> None
== UAC request ==
 * Exe:         <cmd.exe>
 * Commandline: <cmd.exe /c echo Hello & pause>
```

ExaTrack

UAC Monitor

Une trentaine d'events quand on trigger l'UAC

- Cas d'un "Runas" python64.exe qui trigger l'UAC
  - Et que l'utilisateur refuse

```
λ python uac_trace.py
<EventRecord provider="C0B508D3-5459-339F-A213-889C238CA5B1" id=10>
== UAC request ==
 * Exe:          <C:\Python2764\python64.exe>
 * Commandline: <"C:\Python2764\python64.exe" >
<EventRecord provider="C0B508D3-5459-339F-A213-889C238CA5B1" id=19>
== UAC file  Auto-approve ==
 * False
<EventRecord provider="172FF31C-2D80-31A6-FCA8-EB000D380666" id=25>
== Trusted file ==
'C:\\Python2764\\python64.exe\x14\x02FALSEFALSE\x10'
<EventRecord provider="E9003DED-6036-396C-3B5B-4AC9A7A701D1" id=12>
== Consent.exe result ==
 * REFUSED (0x4c7)
```

ExaTrack

UAC Monitor

Une trentaine d'events quand on trigger l'UAC

- Cas complet du bypass UAC par mauvaise cmdline
  - https://www.rump.beer/2017/slides/from_alpc_to_uac_bypass.pdf

```
λ python uac_trace.py
<EventRecord provider="C0B508D3-5459-339F-A213-889C238CA5B1" id=10>
== UAC request ==
 * Exe:         <mmc.exe>
 * Commandline: <xx,wf.msc MY_BAD_MSC>
<EventRecord provider="C0B508D3-5459-339F-A213-889C238CA5B1" id=19>
== UAC file  Auto-approve ==
 * TRUE
<EventRecord provider="172FF31C-2D80-31A6-FCA8-EB000D380666" id=25>
== Trusted file ==
'mmc.exe\x14\x02TRUEFALSE\x10'
<EventRecord provider="172FF31C-2D80-31A6-FCA8-EB000D380666" id=25>
== Trusted file ==
'C:\\WINDOWS\\system32\\wf.msc\xff\xff\xff\xffTRUEFALSE\x10'
<EventRecord provider="E9003DED-6036-396C-3B5B-4AC9A7A701D1" id=12>
== Consent.exe result ==
 * ACCEPTED
```

ExaTrack

Integration dans du code

Mon cas d'utilisation pour le Reverse

- Grâce au logfile on peut
  - Stocker les event et les processer plus tard
  - Récupérer les actions dans un laps de temps X
- Intégration dans du code de test/dev/exploit
  - Lance notre trace avec un logfile
  - Enable les GUID qui nous intéressent
  - exécute notre code qui interagit avec la cible
  - Stop la trace
  - Process les éléments
- Certains events ont même le nom des fonctions !
  - Très utile pour savoir quand vous avez du code qui trigger des chemins d'erreur dans le processus cible

Integration dans du code

```
def show(event):
    if event.id in (10, 11):
        print(event)
        print(" *" + repr(event.user_data.replace("\x00", "")))

trace = windows.system.etw.open_trace("MY_WMI_TRACE", logfile="mywmi.etl")
trace.start()
trace.enable("047311A9-FA52-4A68-A1E4-4E289FBB8D17", 0xff, 0xff)
# Query Scheduled TASK
task = windows.system.task_scheduler(r"\Microsoft\Windows\Chkdsk")["SyspartRepair"]
print("SDDL:" + task.get_security_descriptor(gdef.DACL_SECURITY_INFORMATION))
# Query Scheduled TASK
trace.stop()
trace.process(show) # process can also take begin/end parameters to filters events
```

```
λ python trace_task_sched.py
SDDL:D:PAI(A;;FR;;;AU)(A;;FA;;;SY)
<EventRecord provider="7F813480-DCE4-329F-376B-E377543993B1" id=11>
 *'\xb8\xe2\xaf\x0f&\\Microsoft\\Windows\\Chkdsk\\SyspartRepairNULL'
<EventRecord provider="7F813480-DCE4-329F-376B-E377543993B1" id=11>
 *'\x90\xe2\xaf\x0f&\\Microsoft\\Windows\\Chkdsk\\SyspartRepairNULL'
<EventRecord provider="8DD8F1D9-DC1F-3CA3-830E-AA8135136539" id=10>
 *'Task'
<EventRecord provider="71919C28-8FF5-3EDC-4D6D-ECEEE6EDCD26" id=10>
 *'Task'
<EventRecord provider="8DD8F1D9-DC1F-3CA3-830E-AA8135136539" id=10>
 *'RegistrationInfo'
<EventRecord provider="71919C28-8FF5-3EDC-4D6D-ECEEE6EDCD26" id=10>
 *'RegistrationInfo'
<EventRecord provider="8DD8F1D9-DC1F-3CA3-830E-AA8135136539" id=11>
 *'SecurityDescriptorD:P(A;;FR;;;AU)(A;;FA;;;SY)'
<EventRecord provider="71919C28-8FF5-3EDC-4D6D-ECEEE6EDCD26" id=11>
 *'SecurityDescriptorD:P(A;;FR;;;AU)(A;;FA;;;SY)'
<EventRecord provider="8DD8F1D9-DC1F-3CA3-830E-AA8135136539" id=11>
 *'URI\\Microsoft\\Windows\\Chkdsk\\SyspartRepair'
```

ExaTrack

NT Kernel Logger

- Une trace des event kernel
  - Le nom est important
  - Les events sont définis par des flags au moment du start
    - ALPC / Image Load / Process / …
- Demande d'être administrateur pour lancer la trace
- EVENT_TRACE_FLAG_IMAGE_LOAD
  - Génère un "snapshot" des images déjà chargées en début de trace

NT Kernel Logger

```
def show(event):
    data = event.user_data.replace("\x00", "")
    name = data[data.find("\\"):]
    print(event.guid, name)

my_trace = windows.system.etw.open_trace("NT Kernel Logger",
logfile="tmp.etl")
my_trace.start(flags=gdef.EVENT_TRACE_FLAG_IMAGE_LOAD)
my_trace.stop()
my_trace.process(show)
```

```
PS C:\Users\hakril\Documents\projets\PythonForWindows\playground\event_tracing> python ktrace_load.py
(<GUID "68FDD900-4A3E-11D1-84F4-0000F80464E3">, '\\Users\\hakril\\Documents\\projets\\PythonForWindows\\playgro
(<GUID "68FDD900-4A3E-11D1-84F4-0000F80464E3">, 'A')
(<GUID "68FDD900-4A3E-11D1-84F4-0000F80464E3">, '')
(<GUID "68FDD900-4A3E-11D1-84F4-0000F80464E3">, 'A')
(<GUID "68FDD900-4A3E-11D1-84F4-0000F80464E3">, 'A')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\Device\\HarddiskVolume2\\Windows\\SysWOW64\\ntdll.dll')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\Device\\HarddiskVolume2\\Windows\\System32\\vertdll.dll')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\Device\\HarddiskVolume2\\Windows\\System32\\ntdll.dll')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\\x01\\SystemRoot\\system32\\ntoskrnl.exe')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\SystemRoot\\system32\\hal.dll')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\SystemRoot\\system32\\kd.dll')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\SystemRoot\\system32\\mcupdate_GenuineIntel.dll')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\SystemRoot\\System32\\drivers\\msrpc.sys')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\SystemRoot\\System32\\drivers\\ksecdd.sys')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\SystemRoot\\System32\\drivers\\werkernel.sys')
(<GUID "2CB15D1D-5FC1-11D2-ABE1-00A0C911F518">, '\\SystemRoot\\System32\\drivers\\CLFS.SYS')
```

Snooping around

Autres trucs funs que l'on peut explorer

- ETW permet deux choses intéressantes
  1. Lister les sessions (traces) existantes
     - `logman -ets` affiche la liste des session avec les noms
     - Mais les API permettent de récupérer plus que ca
       - Le nom de la session
       - Le fichier de log
  2. Lister les providers
     - Avec la liste des processus qui sont providers
     - Et les traces qui consommes leur évènements
- On peut donc savoir pour toutes les traces
  - Quels providers & processus celles-ci consomment

Snooping around

```
import sys
import windows
import windows.generated_def as gdef

target = sys.argv[1]
# Search session by enumeration
for session in windows.system.etw.sessions:
    if session.name == target:
        print("Session: <{0}>".format(session.name))
        print("  * logfile {0}".format(session.logfile))
        print("  * guid {0}".format(session.logfile))
        target_id = session.Wnode.HistoricalContext
        print("  * id {0}".format(target_id))
        break
# Enum providers and search for our trace in session
for provider in windows.system.etw.providers:
    for instance in provider.instances:
        for session in instance.sessions:
            if session.LoggerId == target_id and instance.Pid:
                proc = [p for p in windows.system.processes if p.pid ==
instance.Pid][0]
                print(provider.guid, proc.name)
                break
```

ExaTrack

Snooping around

```
λ python provider_enum.py MpWppTracing-20190524-102954-00000003-ffffffff
Session: <MpWppTracing-20190524-102954-00000003-ffffffff>
  * logfile C:\ProgramData\Microsoft\Windows Defender\Support\MpWppTracing-20190524-102954-00000003-ffffffff.bin
  * guid C:\ProgramData\Microsoft\Windows Defender\Support\MpWppTracing-20190524-102954-00000003-ffffffff.bin
  * id 31
(<GUID "AC45FEF1-612B-4066-85A7-DD0A5E8A7F30">, u'MsMpEng.exe')
(<GUID "FBF2C915-2279-4D24-AAC9-201CC3F84658">, u'MsMpEng.exe')
(<GUID "B702D31C-F586-4FC0-BCF5-F929745199A4">, u'NisSrv.exe')
(<GUID "4DBA7E44-DA4B-496A-9777-BE4ECD6D60B4">, u'svchost.exe')
(<GUID "1B0AC240-CBB8-4D55-8539-9230A44081A5">, u'explorer.exe')
(<GUID "1B0AC240-CBB8-4D55-8539-9230A44081A5">, u'explorer.exe')
(<GUID "1B0AC240-CBB8-4D55-8539-9230A44081A5">, u'svchost.exe')
(<GUID "1B0AC240-CBB8-4D55-8539-9230A44081A5">, u'SecurityHealthService.exe')
(<GUID "1B0AC240-CBB8-4D55-8539-9230A44081A5">, u'MsMpEng.exe')
(<GUID "0C62E881-558C-44E7-BE07-56B991B9401A">, u'MsMpEng.exe')
(<GUID "C0011577-86D0-41DB-BB2A-5108F58B0712">, u'MSASCuiL.exe')
(<GUID "DB30E9DC-354D-48B5-9DC0-AEAEBC5C6B54">, u'powershell.exe')
(<GUID "DB30E9DC-354D-48B5-9DC0-AEAEBC5C6B54">, u'explorer.exe')
(<GUID "DB30E9DC-354D-48B5-9DC0-AEAEBC5C6B54">, u'MSASCuiL.exe')
(<GUID "DB30E9DC-354D-48B5-9DC0-AEAEBC5C6B54">, u'NisSrv.exe')
(<GUID "DB30E9DC-354D-48B5-9DC0-AEAEBC5C6B54">, u'SecurityHealthService.exe')
(<GUID "DB30E9DC-354D-48B5-9DC0-AEAEBC5C6B54">, u'MsMpEng.exe')
(<GUID "ED403794-AF80-40DF-87FC-69DF262388DC">, u'chrome.exe')
(<GUID "ED403794-AF80-40DF-87FC-69DF262388DC">, u'splwow64.exe')
(<GUID "ED403794-AF80-40DF-87FC-69DF262388DC">, u'powershell.exe')
(<GUID "ED403794-AF80-40DF-87FC-69DF262388DC">, u'spoolsv.exe')
(<GUID "B1FD0E2C-6B17-4B34-8DB5-FB08C1AF2D65">, u'SecurityHealthService.exe')
(<GUID "5638CD78-BC82-608A-5B69-C9C7999B411C">, u'MsMpEng.exe')
(<GUID "81A06CDF-FB7F-437E-90B2-07214A2B527D">, u'SecurityHealthService.exe')
```

ExaTrack

Questions ?

- Clément Rouault (@hakril)
- https://github.com/hakril/pythonforwindows
  - C'est sur la branche dev pour le moment
- https://exatrack.com

Source

Les deux articles sur le sujet qui m'ont marqué

- Data Source Analysis and Dynamic Windows RE using WPP and TraceLogging
  - SpecterOps
  - M'a motivé à regarder les API / Faire l'implem Python
- THE WORST API EVER MADE
  - Lu APRES avoir fait l'implémentation
  - M'a bien fait rire :')