


# Disabling NTLM Authentication Guide – part 3 – Migrating to Kerberos

 [willssysadmintechblog.wordpress.com/2023/08/29/disabling-ntlm-authentication-guide-part-3-migrating-to-kerberos](https://willssysadmintechblog.wordpress.com/2023/08/29/disabling-ntlm-authentication-guide-part-3-migrating-to-kerberos)

August 29, 2023

Part 2: [Disabling NTLM Authentication Guide – part 2 – Logs](#)

Part 4: [Disabling NTLM Authentication Guide – part 4 – NTLM Restrictions and Testing](#)

Kerberos authentication mitigates one of the biggest pitfalls of NTLM: passwords or their hashes get passed around and stored, even if temporarily. Kerberos relies on tickets to validate identity instead of a password hash. Tickets are encrypted based on your password, sure, but this encryption can be configured to be much stronger than NTLM hashes. Tickets also expire after a defined period of time. Kerberos and its tickets also allows for mutual authentication of client and server, so the server knows you're you, and you know the server is the server it claims to be. Because of this and how heavily Kerberos is baked into Active Directory by default, it is the natural choice to replace NTLM.

## LDAPS – A Questionable Alternative

As you'll find, not all applications support Kerberos. Some do support an LDAPS simple bind, though. Here's how this might work on a vended web service:

1. Client provides their username/password on the website
2. Server connects to an LDAPS server, which will be a domain controller if you have a Windows domain
3. Server attempts a LDAP bind operation, supplying your username/password over the LDAP connections
4. Domain controller validates your credentials internally using NTLM.

The NTLM authentication only happens on the domain controller, so we don't need to worry about NTLM residuals getting left on other computers. Your credentials would get passed over the LDAP connection in plain text, except if you use a LDAPS encrypted

tunnel. That's why it's super important to use LDAPS here instead of unencrypted LDAP. On services that support this LDAP/LDAPS simple bind, the feature will likely be called "LDAP" or "LDAPS". This isn't as good as Kerberos but can be used to get one more server off NTLM.

I'm not a security guy specifically, so I'm not one to make the call on whether LDAPS is preferable to NTLM. Our security team said that in our case, limited use of LDAPS might be preferable than NTLM, partly because it'd reduce the complexity of our project by limiting the number of NTLM exceptions. Passing plain text credentials over a secure channel isn't the most preferable, but is still used by some websites, so they judged the risk to be similar to that of NTLM. We ended up not using this option anywhere to my knowledge, but it could be used.

### Kerberizing a Service

Kerberos is more complicated than NTLM and works differently, so it has different requirements from your environment. Clients need to know what server you're connecting to and what Kerberos realm (or domain) the user account is in. Domain controllers need to know what services are running on each server so they can craft service tickets and give them to users.

### DNS Records

Kerberos has specific DNS requirements to work. A client needs to know where it can go to authenticate via Kerberos. These authentication servers are called Key Distribution Centers (KDCs). In a Windows environment KDCs run on domain controllers. Clients that are Kerberos capable will know to look for certain DNS records to tell them what KDC to go to. They'll usually look for these records:

- 1 `_kerberos._tcp.domain.com`
- 2 `_kerberos._tcp.dc._msdcs.ad.uni.edu`

Here's the output of these commands in an example domain:

```

1  PS> nslookup -type=SRV _kerberos._tcp.domain.com
2  Server:  dns.domain.com
3  Address: 10.0.0.2
4  _kerberos._tcp.domain.com      SRV service location:
5  priority      = 0
6  weight        = 100
7  port          = 88
8  svr_hostname  = dc3.domain.com
9  _kerberos._tcp.domain.com      SRV service location:
10 priority      = 0
11 weight        = 100
12 port          = 88
13 svr_hostname  = dc4.domain.com
14 _kerberos._tcp.domain.com      SRV service location:
15 priority      = 0
16 weight        = 100
17 port          = 88
18 svr_hostname  = dc1.domain.com
19 _kerberos._tcp.domain.com      SRV service location:
20 priority      = 0
21 weight        = 100
22 port          = 88
23 svr_hostname  = dc2.domain.com
24 dc3.domain.com      internet address = 10.0.0.4
25 dc4.domain.com      internet address = 10.1.0.4
26 dc1.domain.com      internet address = 10.2.0.4
27 dc2.domain.com      internet address = 10.3.0.4
28

```

You need to verify that wherever you want to do Kerberos from, these DNS records are available to your clients. If authenticating from another domain, this might mean manually creating DNS records for KDCs, or by forwarding DNS requests to the target domain.

## Privacy Settings

### KDC Port

The KDC listens on UDP/TCP port 88 for granting tickets. This port needs to be open to your clients on your domain controllers. There are projects out there that claim to be Kerberos proxies. I haven't used any of them. Microsoft's Remote Desktop Gateway Server includes a Kerberos proxy server that lets you use RDP via Kerberos from the internet, without needing to expose port 88 on the internet. I will have a special section about this feature in a later post.

### User Domain Specification

Kerberos clients need to know what domain (or Kerberos realm) to look for these DNS records in to authenticate the user account you're requesting tickets for. If coming from within the same domain this is almost always automatically assumed to be your current domain. If coming from a different domain, the client needs to specify. On Windows, this is usually done by specifying usernames in the following formats:

- 1 username@domain.com
- 2 domain.com\username

## Service Principal Names

Clients also need to provide a target server FQDN that the KDC knows about. By default, Windows domain controllers know of 2 FQDNs for every computer:

- 1 HOSTNAME
- 2 HOSTNAME.domain.com

If you pass a DC these server names, the KDC will know what services run on them. It knows this through Service Principal Names (SPNs). SPNs are attributes of AD computer objects, and tell the KDC what Kerberos enabled services are running under a specific user account and on what computer they run on. SPNs have the following format:

ServiceName/ComputerNameItRunsOn

You can view a user account's SPNs with the following command:

- 1 PS> setspn -l USERNAME
- 2 Registered ServicePrincipalNames for
- 3 CN=SERVER,OU=Servers,DC=domain,DC=com:
- 4 TERMSRV/SERVER.domain.com
- 5 TERMSRV/SERVER
- 6 WSMAN/SERVER.domain.com
- 6 WSMAN/SERVER
- 7 RestrictedKrbHost/SERVER
- 8 HOST/SERVER
- 9 RestrictedKrbHost/SERVER.domain.com
- 10 HOST/SERVER.domain.com

These SPNs were created by default and are present on most Windows computer objects. You can create your own for services that don't automatically register SPNs with Active Directory. Note that SPNs can be registered under any account object, so users and computers can both have SPNs attached to them. This is useful to know if your service runs as an external service account, instead of the SYSTEM account on a computer.

When you authenticate via Kerberos and want to connect to a service running somewhere, the KDC creates a Service Ticket (ST) for you. The ST's encryption is based on the password of the service account that runs the service. This makes sure that only the authentic service account can decrypt the ST and validate your authentication. No other account can. When you connect to a service, this ensures the service you're connecting to is authentic and not an imposter. If clients are configured to connect to a different FQDN than what Active Directory knows about in the SPN list, you won't be able to authenticate via Kerberos. The KDC won't be able to find an SPN for the FQDN being

specified, so it won't know what account to create a service ticket for. If you want clients to be able to connect via a non-default FQDN, or the service you run doesn't automatically register an SPN, you'll need to create an SPN.

SPNs can be configured via PowerShell:

```
1 setspn -s  
  "ServiceName/ComputerItRunsOn<:OptionalPortIfNotKnownOrStandard>"  
  "AccountThatRunsService"
```

After you create an SPN it could take 20-30 minutes before you can use it to authenticate via Kerberos. I don't know for sure why this is. I wonder if editing SPNs doesn't trigger an immediate domain controller replication of the object in question, or if something with the KDC needs to refresh.

If I want to create an SPN for a Microsoft SQL Server, running on the computer SQLSERVER.domain.com, running as the user sqlserviceaccount, on port 1433, I would do this:

```
1 PS> setspn -s "MSSQLSvc/SQLSERVER.domain.com:1433" "sqlserviceaccount"
```

The KDC will validate all the supplied information and create an SPN. It might error if a duplicate SPN is found registered for another account. After the SPN is created, clients will be able to say they want to connect to a SQL server at SQLSERVER.domain.com. They wouldn't be able to do this via Kerberos until an SPN existed for that service with the correct service name, FQDN, and account running the service specified.

Part 1: [Disabling NTLM Authentication Guide – part 1 – Prerequisites](#)

Part 2: [Disabling NTLM Authentication Guide – part 2 – Logs](#)

Part 3: [Disabling NTLM Authentication Guide – part 3 – Migrating to Kerberos](#)

Part 4: [Disabling NTLM Authentication Guide – part 4 – NTLM Restrictions and Testing](#)

Part 5: [Disabling NTLM Authentication Guide – part 5 – Printers and Scanners](#)

Part 6: [Disabling NTLM Authentication Guide – part 6 – RDP](#)

Part 7: [Disabling NTLM Authentication Guide – part 7 – Kerberos Logs](#)