

Kerberos, NTLM, and the Ticket That Runs Your Network

Sun, Aug 9, 10:00 AM EDT · <https://scottslab.io/posts/kerberos-ntlm-tgt-how-it-works>



TL;DR

Kerberos is how Active Directory actually authenticates: you prove yourself once to the KDC, get a Ticket Granting Ticket (TGT), and from then on trade that TGT for per-service tickets without your password ever crossing the wire again. NTLM is the older challenge-response scheme it replaced: still lurking for backwards compatibility, and still enabling pass-the-hash, which is why Microsoft's standing advice is to turn it off. Understanding the TGT is also understanding where the attacks (Kerberoasting, golden tickets) live.

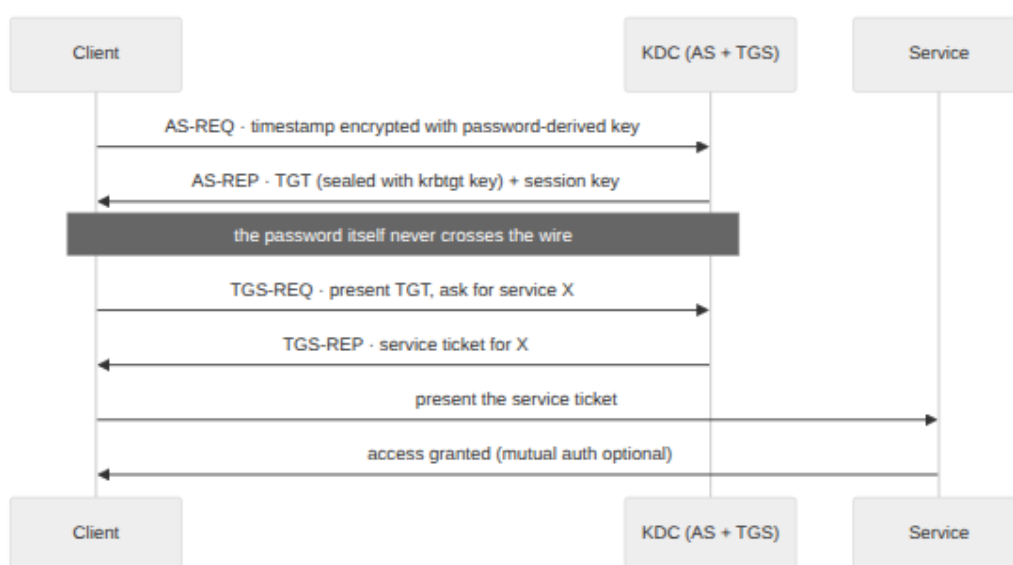
Here's the problem every network authentication protocol is really solving: you need to prove who you are to fifty different servers over the course of a day, and you absolutely do not want to hand your password to fifty different servers to do it. Kerberos answers that with tickets, and the whole system turns on one of them.

How Kerberos works, and where the TGT fits

The center of it is the KDC (the Key Distribution Center), which in Active Directory is just a role the domain controller plays. It has two halves: an Authentication Service (AS) and a Ticket

Granting Service (TGS). You reach it on port 88: UDP by default in Active Directory, with TCP as the fallback when a reply (a ticket stuffed with group memberships, say) is too big for a single UDP datagram.

When you log in, your machine sends the AS a request with a timestamp encrypted using a key derived from your password. The KDC knows your password's key too, so if it can decrypt that timestamp, you've proven you know the password, without the password itself ever being sent. In return you get two things: a **Ticket Granting Ticket**, sealed with a key only the KDC knows (the `krbtgt` account's key), and a session key. That TGT is your "I already proved who I am" token. It's time-stamped and it expires, which is why Kerberos is fussy about clock sync across the domain.



From then on you never re-enter your password. To reach a file share or a database, your machine presents the TGT back to the TGS and says "give me a ticket for this service." The TGS hands back a service ticket, which you present to the service itself. The service can validate it without ever calling back to the KDC. Password sent once, to one place; everything after is tickets. And because the tickets can carry mutual authentication, the service proves itself to you too, so you're not talking to an impostor.

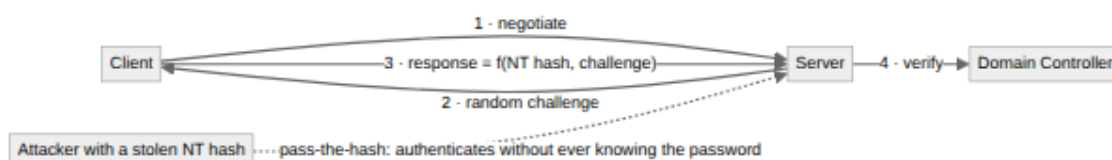
Why the TGT is the crown jewel

Everything convenient about the TGT is also what makes it dangerous. It's a reusable proof of identity, and the key that seals it belongs to a single account: `krbtgt`. Anyone who steals that key can forge a TGT for *any* user, valid for as long as they like. That's a **golden ticket**, and it's game over for the domain because the KDC will trust its own signature without question.

The more common attack is quieter. **Kerberoasting** abuses the fact that any authenticated user can request a service ticket for any service account, and part of that ticket is encrypted with the service account's password hash. Request the ticket, take it offline, and crack it at your leisure: no failed logins, no alarms, just a weak service-account password turning into a foothold. This is why service accounts get long random passwords and why security teams watch for accounts requesting piles of service tickets.

NTLM, and the hash that is the password

NTLM is what came before, and it works differently. There are no tickets and no KDC ticket dance: it's a straight challenge-response. The server sends a random challenge, the client responds with a value computed from the challenge and the NT hash of the user's password, and the server (or the domain controller behind it) checks the math. Like Kerberos, the password itself isn't sent. Unlike Kerberos, there's no mutual authentication, the crypto is weaker, and there's a fatal design property: the hash is the credential.



That last point is **pass-the-hash**. Because the response is computed from the hash and not the plaintext, an attacker who lifts the NT hash out of a machine's memory (the classic Mimikatz move) can authenticate as that user without ever cracking or knowing the actual password. The hash isn't a step toward the credential; it *is* the credential. No amount of password complexity helps once the hash is loose.



What to actually do about it

Microsoft has recommended disabling NTLM for years, and the reason is everything above: it's a weaker protocol whose core mechanic hands attackers a reusable credential. The catch is that it lingers: legacy applications, connections made to a raw IP instead of a hostname, and old appliances

all fall back to it, so ripping it out means finding what still depends on it first. Audit NTLM usage, kill it where you can, and where you can't, isolate it.

For Kerberos, the work is protecting the objects the attacks target: guard the `krbtgt` key (and rotate it if you ever suspect compromise, twice, because of how it caches), give service accounts long random passwords to defeat Kerberoasting, and monitor for the ticket-request patterns that give golden tickets and roasting away. The TGT is a genuinely elegant piece of engineering: prove yourself once, carry a ticket, keep your password off the wire. Just remember that the same ticket that saves you fifty password prompts is the thing an attacker most wants in their pocket.

Written by Scott S. — Contact: scott@scottschlangen.com — scottslab.io — <https://scottslab.io/posts/kerberos-ntlm-tgt-how-it-works>