Kerberos Authentication: A Wrap Up

csandker.io/2017/09/12/KerberosAuthenticationAWrapUp.html

September 12, 2017

12 Sep 2017 (Last Updated: 16 Jul 2021)

This post is intended as a wrap-up to refresh/update our understanding of how Kerberos works in a Windows domain network.

If you want the full picture right from the source check out this post from the Microsoft docs:

> Kerberos Explained

Basics

Kerberos is - since Windows 2000 - the preferred authentication scheme in Windows domain networks (chosen over NTLM).
The Kerberos authentication is based on tickets, which are transmitted over the wire and which need to be presented in order to get access to a certain service/resource.

Prior to getting access to a certain service, a user has to authenticate against the Kerberos Distribution Center (KDC). After successful authentication a TGT is generated for that user by the KDC and returned back to the user. This TGT ticket is stored on the user's client machine and is used from there on to request tickets for individual services (TGS), which have to be provided to each service in order to get access.

For years I thought there were two (completely) different types of tickets, namely TGT tickets and TGS/service tickets, which are presented to the KDC and to the server hosting the service you want to access respectively. From a workflow/architectural view of how Kerberos works this understanding is not totally wrong, however from a technical standpoint there are a few things that should be clarified:

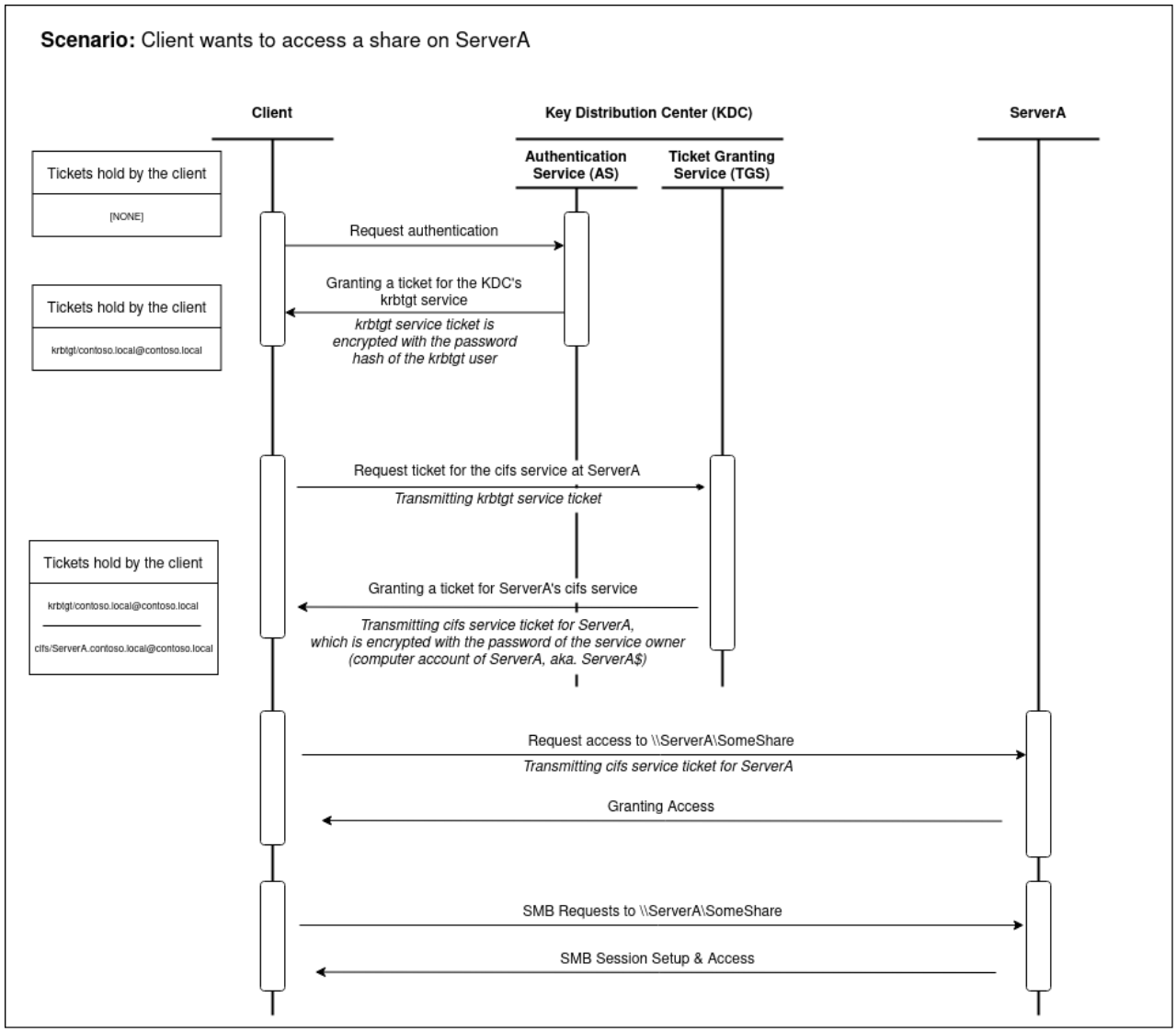
- There is no such thing as a TGS ticket, the Ticket Granting Service (TGS) is - as the name says - a service (provided any hosted by the KDC) and not a special type of ticket.
- When a user wants to access a certain service (and has no ticket for that service yet), the user reaches out to the TGS (providing his TGT to the TGS) to get a ticket for the service he wants to access. This ticket - which I'll call service ticket from now on - however has the same structure and fields as the TGT ticket and there is no field that specifies the type of this ticket.

- Framed in other words: The RFC4120 describing the Kerberos authentication scheme in version 5 (which is the current version used in Windows) describes only one form of ticket - a Kerberos ticket - with a unified structure in section 5.3 of the RFC. There is no TGT ticket structure and a service ticket structure or even a 'type' field within the ticket structure that would mark a ticket as TGT or service ticket.

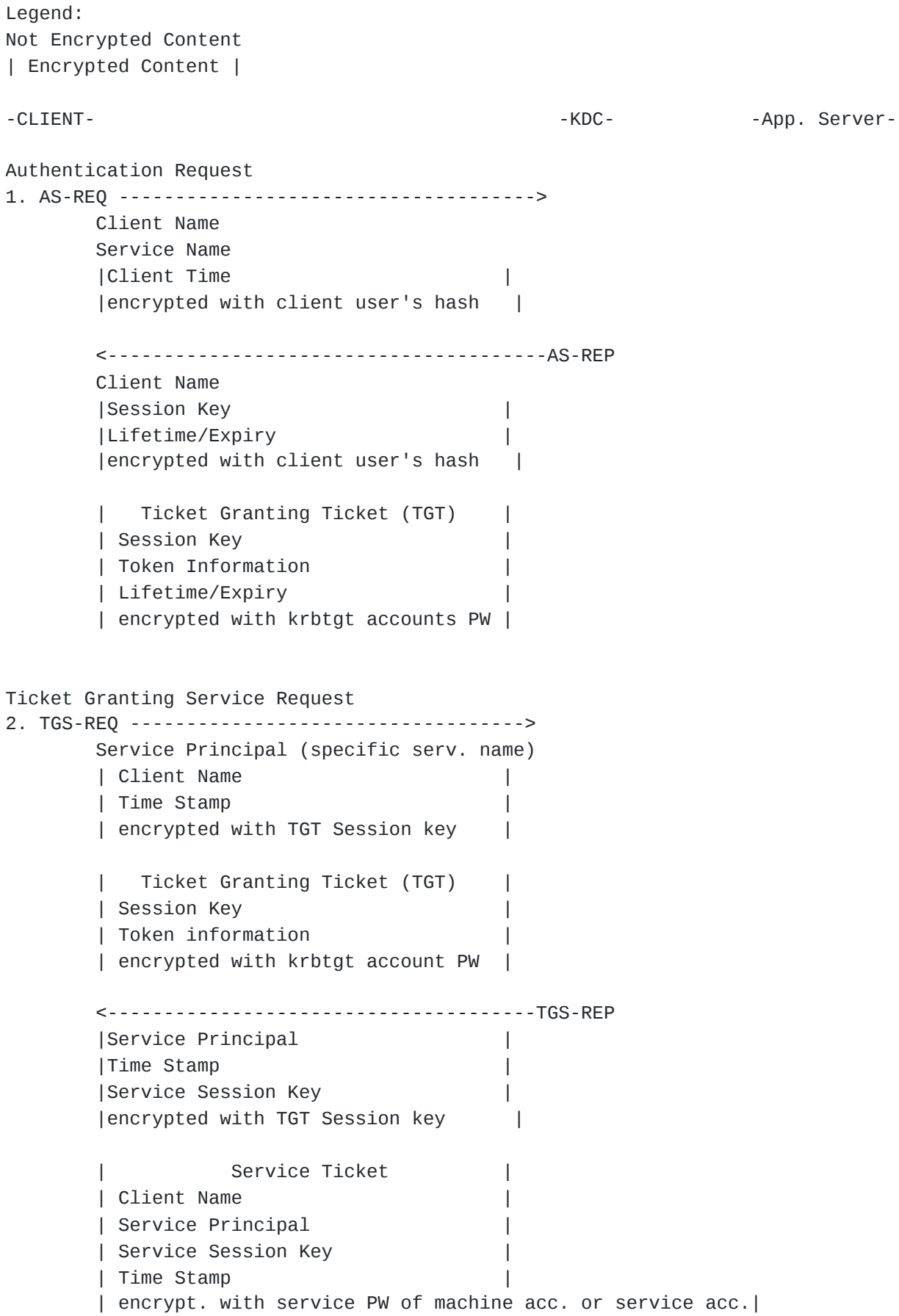
Framed in other words (again): A "TGT Ticket" is just a service ticket for krbtgt service hosted on the KDC. All Kerberos tickets are service tickets.

Workflow

An abstraction of the Kerberos authentication scheme workflow is shown below:



As it can be seen here, Kerberos is solely based on tickets as a basis to grant or deny access. Additionally to get some more insights on what information is exchanged during the workflow steps shown above the following scheme is provided (Note that all encrypted data is enclosed in pipes):


Legend:
Not Encrypted Content
| Encrypted Content |
-CLIENT-
-KDC-
-App. Server-
Authentication Request
1. AS-REQ
Client Name
Service Name
|Client Time |
|encrypted with client user's hash |
AS-REP
Client Name
|Session Key |
|Lifetime/Expiry |
|encrypted with client user's hash |
| Ticket Granting Ticket (TGT) |
| Session Key |
| Token Information |
| Lifetime/Expiry |
| encrypted with krbtgt accounts PW |
Ticket Granting Service Request
2. TGS-REQ
Service Principal (specific serv. name)
| Client Name |
| Time Stamp |
| encrypted with TGT Session key |
| Ticket Granting Ticket (TGT) |
| Session Key |
| Token information |
| encrypted with krbtgt account PW |
TGS-REP
|Service Principal |
|Time Stamp |
|Service Session Key |
|encrypted with TGT Session key |
| Service Ticket |
| Client Name |
| Service Principal |
| Service Session Key |
| Time Stamp |
| encrypt. with service PW of machine acc. or service acc.|
Application Request
3. AP-REQ
| Client Name |
| Time Stamp |
| encrypted with TGT Session key |

```
|           Service Ticket          |
| Client Name                       |
| Service Principal                 |
| Service Session Key               |
| Time Stamp                        |
| Token information                 |
| encrypted with service's PW (e.g. of machine acc. or service acc.)|
```

Attack Surface

Disclaimer: In the following i will only cover Kerberoasting and AS-REP Roasting as these are the most relevant attacks (from a RedTeamers perspective). Other attack may be added later on…

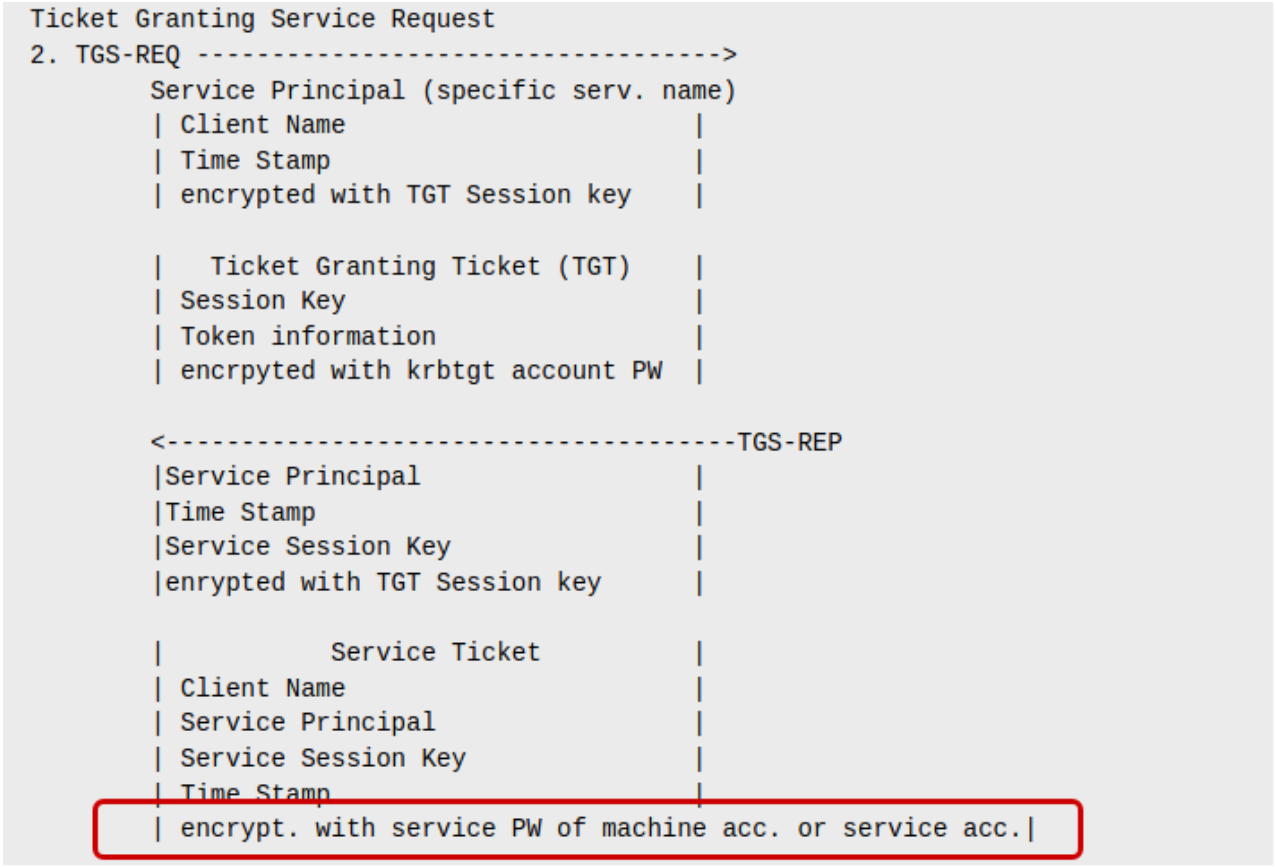
Kerberoasting

> Kerberoasting is a technique that targets service accounts with the goal of obtaining and cracking an account's password.

So what is a service account?
A service account is a normal user account (having a username, a password, etc.) for which a **Service Principal Name (SPN)** has been created. An SPN is nothing more that a unique identifier for a given service. One property of an SPN is that it always includes the name of the host that the service is running on, you can understand an SPN of being the IP & PORT description of a service.
For some more details right from the source have a read through this Microsoft Docs article: https://docs.microsoft.com/en-us/windows/win32/ad/service-principal-names.

The missing bits of *what* Kerberoasting technically is and *how* it works are best understood by answering the *why*. So *why* does Kerberoasting work? *Why* can you get a service account's password in the Kerberos protocol?
The answer to that is simple: Because the Kerberos protocol defines that a service account's password is contained in the service ticket that a user gets from the KDC in order to connect to that service.

Looking at the the Workflow above, we can find the service account's password right in the TGS-REP:



In other words that means: Every time a user requests a service ticket from the KDC in order to connect to that service in the next step, the account password of the user running that service is returned to us, wrapped in a service ticket.
Obviously that password is not contained in there as plaintext password, that's why we need to crack that password in the last step.

To recap Kerberoasting is basically a three step attack chain:

- Find a suitable service account.
  That is: A user account that has got an SPN
- Request a service ticket for that service
- Use an offline cracking tool, like hashcat or John to brute force password attempts until the password is found that can decrypt the service ticket

What makes Kerberoasting especially valuable for all attacker is that no special administrative privileges are required to conduct Kerberoasting. **Any Active Directory user is allowed to query for accounts that have an SPN set, and any Active Directory user is allowed to request a service ticket for any service.**

To run a Kerberoasting attack i recommend to use the amazing Rubeus project, e.g. with:

```
Rubeus.exe kerberoast /outfile:hashes.txt
```

The obtained service account password hash can then be cracked, e.g. using hashcat:

```
hashcat -a 0 -m 13100 hashes.txt MostCommonPasswords.list -o cracked_hashes.txt
```

Important note: The returned password hashes can come in different flavors. Rubeus will tell you which algorithm was used for encrypting the password (one reason why Rubeus is great). Make a note of this so called **EType** and ensure you use your password cracker tool with the correct EType.
For hashcat hash mode 13100 refers to EType 23 (which is the most common one).
Other ETypes are EType 17 (hashcat mode *19600*) and EType 18 (hashcat mode *19700*).
For example hashes and differences in the ETypes have a look at https://hashcat.net/wiki/doku.php?id=example_hashes

Bonus
If you read all of the above carefully you will notice that all you need to Kerberoast a user is a valid Active Directory (AD) account and a target account that has an SPN set (and thereby becomes a service user).

So hat if you create a new SPN and assign it to an arbitrary user?
The answer is: You will be able to Kerberoast that user now.

If you hold write access to a user's properties (*GenericAll* or *GenericWrite* Access) you can set an SPN for that user and make it Kerberoastable.

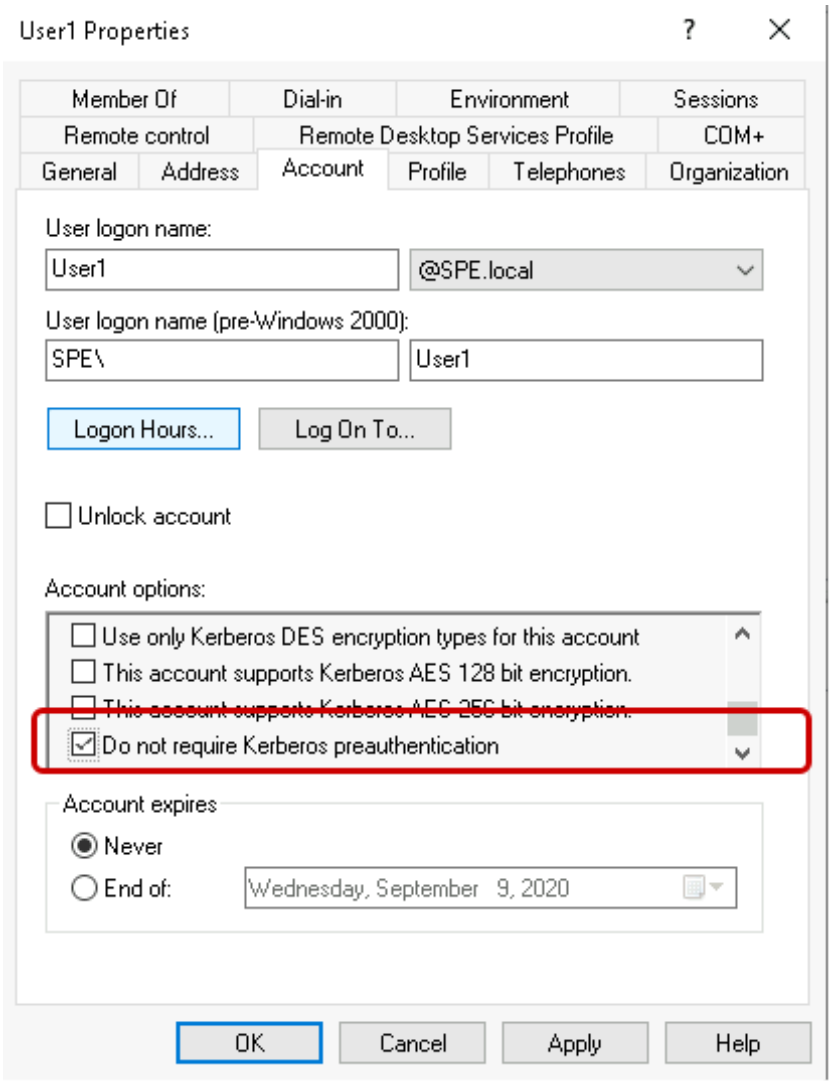
AS-REP Roasting

AS-REP Roasting is a little less known, but follows the same idea Kerberoasting is based on, which is: Crack a user's password contained in a ticket.

In contrast to Kerberoasting in AS-REP Roasting not service account is targeted, but a "normal" user account aka. a user account without an SPN.
The mechanics of AS-REP Roasting are completely the same as with Kerberoasting and i will describe the *what* and the *how* just as i did above, by explaining the *why*.
Long story short: You can crack a user's password because it's given to you in the AS-REP response:

The catch: You can't just request an AS-REP for any user in the domain, because you need to authenticate first. In the image above you will see that the AS-REQ also contains the user's password hash, meaning you need to know the user's password before you can get an AS-REP for that user. This is often referred to as **preauthentication**.

Obviously if i need to know a user's password in the first place this attack is pretty useless.
The reason this attack exist is that although preauthentication is enabled by default for all user accounts, it can be disabled, allowing every domain user to obtain a hash of the user's password.



So in essence AS-REP Roasting is an attack consisting of these three steps:

- Find a suitable user account.
  That is: A user account that does not require preauthentication
- Get a AS-REP packet for that user
- Use an offline cracking tool, like hashcat or John to brute force password attempts until the user password is found.

Just like with Kerberos this attack can be run with Rubeus…

```
Rubeus.exe asreproast /outfile:hashes.txt /format:hashcat
```

… and hashcat

```
hashcat -a 0 -m 18200 hashes.txt MostCommonPasswords.list -o cracked_hashes.txt
```

Important note: Just as with Kerberoasting the hashcat's hash mode 18200 refers to **EType 23**, see: https://hashcat.net/wiki/doku.php?id=example_hashes

Other Posts