🔥🍗 NTLM Roasting

Intro

Until this point we already saw how many techniques que can perform when we have access to credentials, that is, ***Passwords of a user*** that can be in the form of a hash, the **TGT** and the **Session Key** of a user or anything of this sort.

But one thing that we had already seen many times is that, in order to be able to dump those credentials on memory, we are going to need to have a user with **high privileges on that local machine.** But what happens if we do not have the credentials of a user with those characteristics? Can I still **get hashes of the passwords of other users that are in the** *network infrastructure*?

Well, now we are going to see that in addition to those roasting techniques that we have already seen when we were talking about **Kerberos,** we can also take ***advantage of some weaknesses of the NTLM authentication protocol*** to get some hashes and passwords.

So, let's talk about the first section which will deal with ***poisoning certain network protocols on Windows*** and see in detail ***how to crack the hash that is generated in an interaction by NTLM***.

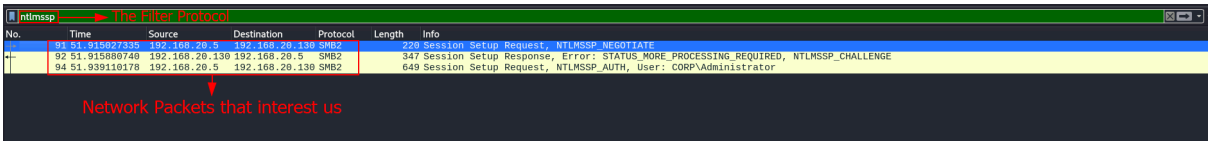
🔥🍗 NTLM Roasting on action

When a client wants to consume a service using ***NTLM*, *send the request*** to consume the services with the ***username*** and the services that it's running on a server, receive that request and send a ***nonce*** or a ***Challenge*** that is a random number. The client receives the **challenge, *encrypts*** that ***challenge with his private key*** and ***sends back the encrypted challenge***. And finally the services or the server confirm if that cipher is correct.

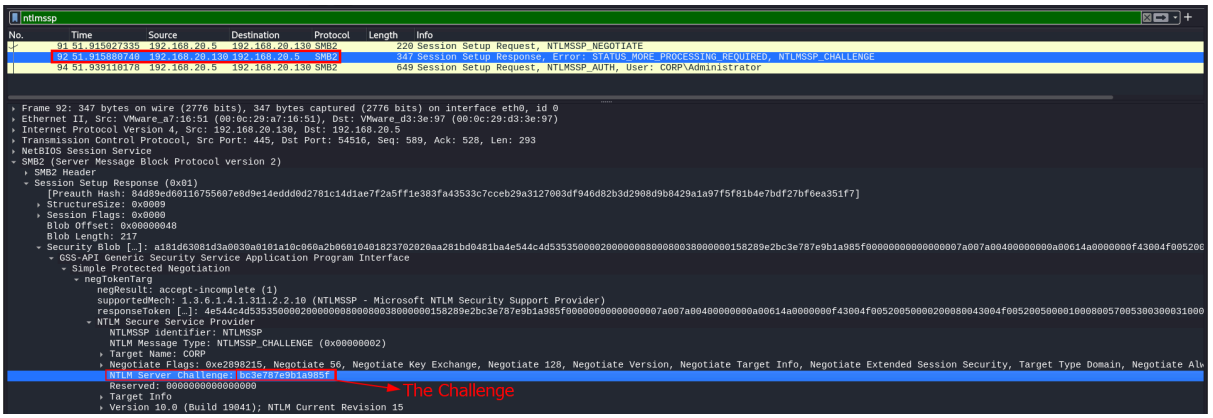
So what happens if we intercept the ***encrypted challenge*** to try to crack it? Indeed, here we can perform a **Roasting technique 🔥🐔.**

To make this we are going to need to use a Network Sniffer like **Wiresharks** and **TCP Dump**. If we are using **Wiresharks** , find ***some values*** to built the hash on a text editor following the next steps

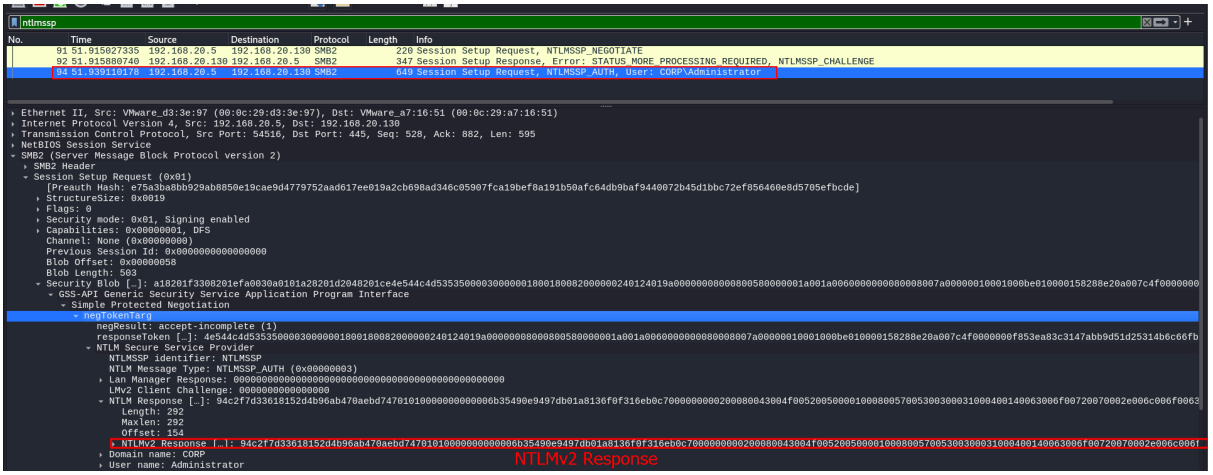
1) we have to filter our sniffer by **ntlmssp protocol** and wait until some user make a request to see the Network Packets:



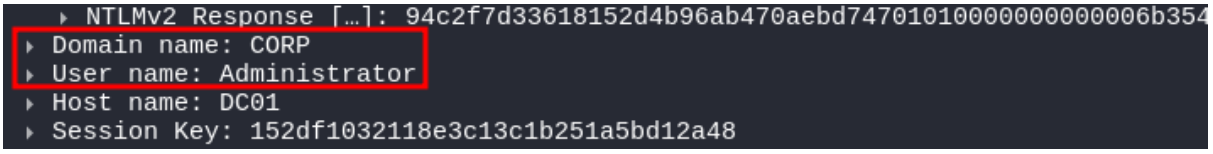
2) Now that we have captured the Network packets we to find those values starting with the ***NTLM Server Challenge*** that is in the **"second network packet"** on this way and then copy de the value



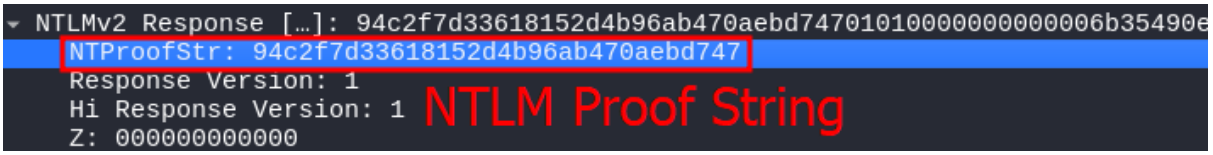
3) In the **“third network packet”** we have to find the server **NTLMv2 Response** and copy the value



4) We have to find the **username** that makes the request along the **domain** which are just below the **NTLMv2 Response** and we pass them to our text editor:



5) And Finally we will need the **NT proof string** which is just below the **NTLMv2 Response** If ***we deploy the value.***



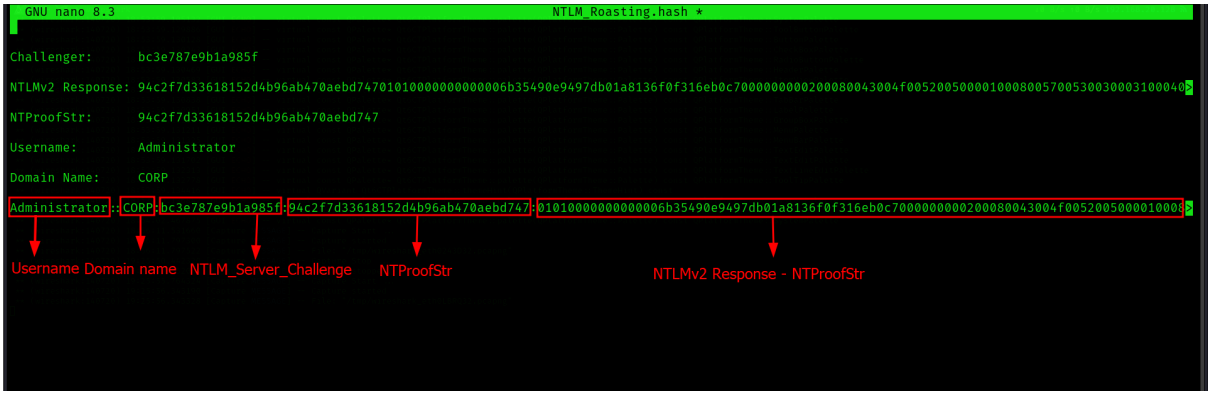
And, as we said, all theses values we must pass the to our text editor

```
Challenger:       bc3e787e9b1a985f
NTLMv2 Response:  94c2f7d33618152d4b96ab470aebd747010100000000000006b35490e9
NTProofStr:       94c2f7d33618152d4b96ab470aebd747
Username:         Administrator
Domain Name:      CORP
```

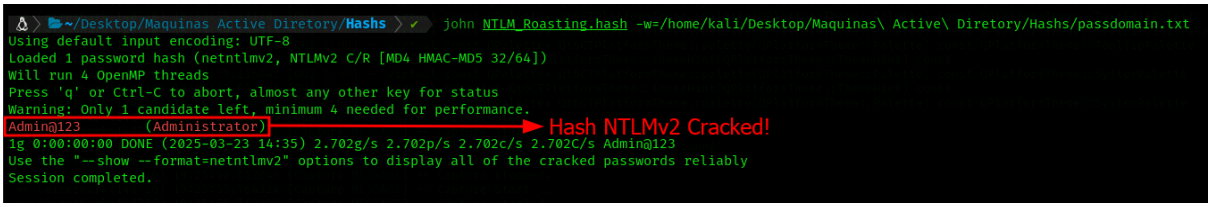
Now we need to assemble that information to create that user's NTLMv2 hash on these format

UserName::Domain:NTLM_Server_Challenge:NTProofStr:NTLMv2Response-NTProofStr

That would look like this



Now we can use tools to crack passwords on this hash (obviously leaving the text the hash already assembled).



Conclusion

This is the first way to get passwords without having to dump it from memory. If there is an authentication. If there is an ***NTLM*** **authentication** in the infrastructure, such as a user accessing a shared resource of another user, and we are able to **sniff the network traffic** 👃, we can **extract those values** 📤, then **crack the resulting hash** 🔨 and **obtain the password in plain text .**

But what happens if, for some reason, we can not sniff the network traffic. Is there another way to capture some credentials? The answer is yes and it is what we are going to see in the section ☠️ **LLMNR/NBTNS Poisoning.**