

PROTOCOLE NETSOUL

1 Introduction

Ce document est écrit suite aux recents changements dans l'authentification du protocole NetSoul. Vous y trouverez tous les changements et tout le nécessaire à l'écriture d'un client NetSoul.

- Serveur : ns-server.epita.fr
- Port : 4242

2 Authentification

Voici la chaine reçue à la connexion :

```
salut 1616 132889038d1baf8dd501aa11cebfff35 10.242.42.22 59557 1174399694
```

salut message de politesse

1616 fd du client

132889038d1baf8dd501aa11cebfff35 hash md5, non utilisé dans le cas présent

10.242.42.22 IP du client

59557 port du client

1174399694 timestamp de connexion du client

La chaine envoyée pour démarrer l'identification :

```
auth_ag ext_user none -
```

auth_ag commande

ext_user specifie le type d'authentification, ici externe

none non utilisé

- non utilisé

Les réponses seront toujours du type :

```
rep 002 -- cmd end
```

rep signifie qu'il y a une réponse

002 code retour de la réponse

- séparateur

cmd end message associé

La ligne de login :

```
ext_user_klog YIICQQYJKoZIICAQ[...]uqkIXI+I= NetBSD location bocal data
```

ext_user_klog la commande

YIICQQYJKoZIICAQ[...]uqkIXI+I= le token kerberos en base64

NetBSD le type de connexion

location la localisation

bocal le groupe de l'utilisateur

data data

Le serveur vous répond encore une fois (si tout se passe bien)

```
rep 002 -- cmd end
```

La connexion est établie et authentifiée.

2.1 Obtenir son token Kerberos

Dans tout le processus d'authentification, le token kerberos reste le plus complexe à obtenir.

La GSSAPI permet de récupérer aisément ce token. L'appel à seulement deux fonctions sont nécessaires pour obtenir ce token.

gss_import_name qui convertit le nom du service en principal (service/hostname@realm)

gss_init_sec_context qui initialise un contexte de sécurité, c'est à ce moment là que le token kerberos est extrait.

Des URLs vers des documentations sur ces deux fonctions sont fournies en annexes. Le nom de service pour NetSoul est "host@ns-server.epitech.net", de meme un exemple de code sera fourni en annexes.

Le type du token est aussi important, lors de l'initialisation du contexte, un nouveau token est généré portant le nom du service. Celui doit être du type "des3-cbc-sha1" (un klist -v permet d'obtenir toutes ces informations).

```
peach# klist -v
Credentials cache: FILE:/tmp/krb5cc_4202
    Principal: shine@EPITECH.NET
[...]
Server: host/ns-server.epitech.net@EPITECH.NET
Ticket etype: aes256-cts-hmac-sha1-96, kvno 1
[...]
```

Ce ticket sera invalidé.

```
peach# klist -v
Credentials cache: FILE:/tmp/krb5cc_4202
    Principal: shine@EPITECH.NET
[...]
Server: host/ns-server.epitech.net@EPITECH.NET
Ticket etype: des3-cbc-sha1, kvno 1
[...]
```

Ce ticket sera validé.

Le "etype" peut être forcé dans le fichier /etc/krb5.conf (ou C :\Windows\krb5.ini) pour cela, voir le manuel de référence.

Les données ainsi obtenues doivent être chiffrées en base64 pour être envoyées au serveur NetSoul.

2.2 Outil de développement

Pour le développement en C la libgssapi et la libkrb5 sont disponibles sur le PIE (NetBSD et Linux).

En Perl il existe le module GSSAPI.pm.

Sous Windows, le MIT Kerberos fournit un SDK qui permet d'utiliser la gssapi de la même manière que sous UNIX.

3 Autres commandes

D'autres commandes sont disponibles sur NetSoul.

ping pour rester connecté sur le serveur

list_users lister les utilisateurs

user_cmd pour toutes les commandes usuelles (**who**, **msg_user**)

state pour changer le status

3.1 user_cmd

La commande "user_cmd" est suivie d'autres commandes :

attach pour accéder aux ressources du PIE

who rechercher un ou plusieurs utilisateurs

msg_user envoyer un message à un utilisateur

watch_log_user surveiller les différents états d'un utilisateur

3.2 Syntaxe

Toutes les données passées en paramètres des commandes doivent être url_encodées.
Les commandes ciblant plusieurs utilisateurs peut être exécutée en une seule.

3.2.1 who

Voici l'exemple ici de la commande "who"

```
user_cmd who {shine,jog,rockyluke,elbarto}
```

La commande pourrait ne viser qu'une seule cible :

```
user_cmd who rockyluke
```

Voici la réponse du serveur NetSoul.

```
user_cmd 170:user:1/100:shine@10.242.42.81:NetBSD:location:bocal | who 170 shine
10.242.42.81 1175006532 1175006559 100 1 location bocal
connection:1175006532 data
user_cmd 170:user:1/100:shine@10.242.42.81:NetBSD:location:bocal | who 1351 rockyluke
10.242.42.244 1174946371 1175006532 3 1 ~ hellsing bocal
actif:1175005027 dotnetSoul%5b.netSoul%200.99.30%20-%20Beta%5d%20
user_cmd 170:user:1/100:shine@10.242.42.81:NetBSD:location:bocal | who 702 jog
10.242.42.19 1174946346 1175006324 1 4 _ _ bocal
actif:1175006090 ns_user_exe-1.042
user_cmd 170:user:1/100:shine@10.242.42.81:NetBSD:location:bocal | who 693 shine
10.242.42.22 1174946346 1175006322 1 4 _ _ bocal
idle:1174946358 ns_user_exe-1.042
user_cmd 170:user:1/100:shine@10.242.42.81:NetBSD:location:bocal | who 372 elbarto
10.242.42.91 1174946343 1175006318 1 4 NetBSD_wse bocal_r02p01 bocal
idle:1175006308 ns_user_exe-1.042
user_cmd 170:user:1/100:shine@10.242.42.81:NetBSD:location:bocal | who rep
002 -- cmd end
```

La réponse contient la source de la commande, un pipe et la réponse en elle même.
En décomposant cela donne :

user_cmd commande
624 identifiant unique de l'utilisateur
user type de l'utilisateur
1/100 niveau de confiance de l'authentification
shine login de l'utilisateur
10.242.42.81 adresse IP de l'utilisateur
NetBSD type de la machine
location localisation
bocal groupe de l'utilisateur
| séparateur
who commande à laquelle on répond
372 identifiant de l'utilisateur
elbarto login de l'utilisateur
10.242.42.91 adresse IP de l'utilisateur
1174946343 timestamp de connexion
1175006318 timestamp actuel
1 level d'authentification du client
4 level d'authentification de l'utilisateur
NetBSD_wse type de la machine
bocal_r02p01 localisation
bocal groupe de l'utilisateur
idle état de l'utilisateur
1175006308 timestamp du dernier changement d'état
ns_user_exe-1.042 champs data (généralement le client)

3.2.2 Messages

La syntaxe d'envoi de message est la suivante :

```
user_cmd msg_user jog msg pouet%20pouet
```

Il n'y a pas de réponse à l'envoi de messages. Comme cela est précisé plus haut, le message doit être urlencodé, c'est donc une seule chaîne qui est passée.

NetSoul vous prévient aussi lors de la réception de mail sur votre compte par la syntaxe suivante

```
user_cmd 583:user:4/1:shine@10.242.42.22:_:_:bocal | msg message%20de%20test dst=:chong_a@
```

La première partie du message est identique à celle des réponses du serveur. La seconde partie comporte la commande "msg" le message ainsi que les destinataires du message.

3.2.3 Mails

La syntaxe de réception de mails est la suivante :

```
user_cmd 0:mail:9/9:_daemon:: | new_mail -f
shine%20chong%20%3Cshine.chong%40epita.fr%3E %28lala%29
```

Les deux derniers arguments sont l'expéditeur et le sujet du mail.

3.2.4 Notification de changement de status

Il est possible de demander au serveur d'être prévenu lors d'un changement de status d'un utilisateur. À chaque connexion/deconnexion ou changement de status, le serveur enverra le status actuel de l'utilisateur.

```
user_cmd watch_log_user shine
user_cmd 491:user:1/100:shine@10.242.42.22:NetBSD:peach:bocal | login
user_cmd 491:user:1/100:shine@10.242.42.22:NetBSD:peach:bocal | state actif:1174984764
user_cmd 491:user:1/100:shine@10.242.42.22:NetBSD:peach:bocal | logout
```

3.3 Changement d'état

Le changement d'état dans NetSoul se fait avec la commande state

```
state actif:1174984764
```

state commande

actif nouvel état

1174984764 timestamp de changement de status

Il est à noter que le status par défaut est "connection", vous **devez obligatoirement** changer ce status à la connexion !

4 Annexes

- http://docs.hp.com/en/B2355-90694/gss_import_name.3.html
- http://docs.hp.com/en/B2355-90694/gss_init_sec_context.3.html

4.1 Exemple

```
#include <stdlib.h>
#include <stdio.h>
#include <gssapi/gssapi.h>

#define SERVICE_NAME ``host@ns-server.epitech.net``

void display_status(OM_uint32 min, OM_uint32 maj)
{
    gss_OID mech;
    OM_uint32 minor, status;
```

```

gss_buffer_desc      msg;

gss_display_status(&minor, min, GSS_C_GSS_CODE, GSS_C_NO_OID, &status, &msg);
puts(msg.value);
gss_display_status(&minor, maj, GSS_C_MECH_CODE, GSS_C_NO_OID, &status, &msg);
puts(msg.value);
exit(1);
}

void      import_name(gss_name_t *gss_name)
{
    OM_uint32      min, maj;
    gss_buffer_desc      buf;
    gss_OID      mech;

    buf.value = (unsigned char *) strdup(SERVICE_NAME);
    buf.length = strlen(buf.value) + 1;
    maj = gss_import_name(&min, &buf, GSS_C_NT_HOSTBASED_SERVICE, gss_name);
    if (maj != GSS_S_COMPLETE)
        display_status(min, maj);
}

unsigned char      *init_context(gss_name_t gss_name)
{
    OM_uint32      min, maj, flags, time_rec;
    gss_ctx_id_t      ctx;
    gss_name_t      name;
    gss_OID      mech_type;
    gss_buffer_t      itoken = GSS_C_NO_BUFFER;
    gss_buffer_desc      otoken;

    maj = gss_init_sec_context(&min, GSS_C_NO_CREDENTIAL, &ctx, gss_name,
                                GSS_C_NO_OID, 0, 0,
                                GSS_C_NO_CHANNEL_BINDINGS, itoken,
                                NULL, &otoken, NULL, NULL);

    if (maj != GSS_S_COMPLETE)
        display_status(min, maj);
    gss_delete_sec_context(&min, &ctx, &otoken);
}

int      main(int ac, char **av)
{
    gss_name_t      gss_name;

    import_name(&gss_name);
    init_context(gss_name);
    return (0);
}

```