Institut Universitaire de Technologie,

Aix-Marseille Université

ANNEXES
RAPPORT DE STAGE de fin de deuxième année

Bachelor Universitaire de Technologie

Spécialité Réseaux et Télécommunications

parcours cybersécurité

Création d'un nouveau réseau WIFILAIRE et Scripting Python

Hatim SBAI

Institut de Neurosciences de la Timone

Responsable entreprise : Arnaud CRUZEL

Responsable académique : Éric SOCCORCI

2023

Sommaire

1 Commande de création de VLAN switch HP

Quelques commandes de modification des switches :

hp2510g:

```
vlan 2850
   name "WIFILAIRE"
   tagged Trk1
   exit
```

hp2610:

```
vlan 2850
   name "WIFILAIRE"
   untagged 1-46
   tagged 46,51-52,Trk20-Trk21
   exit
```

hp2910:

```
vlan 2850
   name "WIFILAIRE"
   tagged Trk15
   no ip address
   exit
```

hp5406:

```
vlan 2850
   name "WIFILAIRE"
   tagged A18-A19,B19,B21-B22,C11-C12,C21,E7,F8,Trk1-Trk6,Trk9
   no ip address
   exit
```

hp 5700:

```
vlan 2850
 name WIFILAIRE
interface Bridge-Aggregation101
 port link-type hybrid
```

port hybrid vlan 1 2000 2010 2020 2030 2040 2850 2881 3004 to 3005 tagged

port hybrid vlan 2081 untagged

port hybrid pvid vlan 2081

link-aggregation mode dynamic

2 Script Python

Script de modification que j'ai développé :

```
# Fonction pour afficher les nouvelles informations de l'utilisateur et
quitter le programme
def display_and_exit():
    pgivename = get_adattr(args.username, 'givenName')
    psurname = get_adattr(args.username, 'sn')
    pusername  = args.username
    pmail  = get_adattr(args.username, 'mail')
    try:
        pdescription = odescription = get_adattr(args.username,
'employeeType')
    except:
        pdescription = odescription = None
    try:
        pgrade = ograde = get_adattr(args.username, 'title')
    except:
        pgrade = ograde = None
    try:
        pmanager = omanager = samdb.search("%s" % get_adattr(args.username,
'manager'), scope=ldb.SCOPE_SUBTREE,
attrs=['sAMAccountName'])[0]['sAMAccountName']
    except:
        pmanager = omanager = None
    try:
        paccountExpires = oaccountExpires =
ad_timestamp(get_adattr(args.username, 'accountExpires'), False)
    except:
        paccountExpires = oaccountExpires = None
    pprimarygroup = oprimarygroup =
get_attrfattr("(&(objectCategory=group)(gidNumber=%s))" %
get_adattr(args.username, 'gidNumber'), 'sAMAccountName')
    try:
        pgroups = ogroups = get_memberof(args.username)
    except:
        pgroups = ogroups = None
    pou = oou = str(get_adattr(args.username, 'dn')).replace('CN=%s,' %
args.username, '')
    print("Nouvelles informations de l'utilisateur :")
```

```
    print_details(
        pgivename, psurname, pusername, pmail, pdescription,
        pgrade, pmanager, paccountExpires, pprimarygroup, pgroups, pou
    )
    # Demande à l'utilisateur d'appuyer sur la barre d'espace pour continuer
    res=input("Appuyez sur 'c' pour continuer...")
    if res=='c':
        sys.exit()
# Modification de l'utilisateur
def modify_nit_user():
    ### On récupère les infos des utilisateurs
    # p<variable> : print pour récap
    # o<variable> : variable récupérée de la base AD (old)
    # n<variable> : variable modifiée par l'administrateur pendant l'exécution
du script
    pgivename = ogivename = get_adattr(args.username, 'givenName')
    psurname = osurname = get_adattr(args.username, 'sn')
    pusername = ousername = args.username
    pmail = omail = get_adattr(args.username, 'mail')
    changes_to_confirm = []
    try:
        pdescription = odescription = get_adattr(args.username,
'employeeType')
    except:
        pdescription = odescription = None
    try:
        pgrade = ograde = get_adattr(args.username, 'title')
    except:
        pgrade = ograde = None
    try:
        pmanager = omanager = samdb.search("%s" % get_adattr(args.username,
'manager'), scope=ldb.SCOPE_SUBTREE,
attrs=['sAMAccountName'])[0]['sAMAccountName']
    except:
        pmanager = omanager = None
    try:
        paccountExpires = oaccountExpires =
ad_timestamp(get_adattr(args.username, 'accountExpires'), False)
    except:
        paccountExpires = oaccountExpires = None
    pprimarygroup = oprimarygroup =
get_attrfattr("(&(objectCategory=group)(gidNumber=%s))" %
get_adattr(args.username, 'gidNumber'), 'sAMAccountName')
    try:
        pgroups = ogroups = get_memberof(args.username)
    except:
        pgroups = ogroups = None
    pou = oou = str(get_adattr(args.username, 'dn')).replace('CN=%s,' %
args.username, '')
```

```
    #print_details(ogivename, osurname, ousername, omail, odescription,
omanager, oaccountExpires, oprimarygroup, ogroups, oou)
    validate = False
    while not validate:
        clear_screen()
        print_details(pgivename, psurname, pusername, pmail, pdescription,
pgrade, pmanager, paccountExpires, pprimarygroup, pgroups, pou)
        valid_ans = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "c"]
        list_columns(["1: Prénom", "2: Nom", "3: Adresse Mail", "4: Fonction",
"5: Grade", "6: Responsable", "7: Date d'expiration du compte", "8: Groupe
primaire (NE FONCTIONNE PAS)", "9: Groupe d'appartenance"])
        ans = input("\nQuel élément voulez-vous modifier (c pour valider les
changements) ?\n: ")
        while ans not in valid_ans:
            clear_screen()
            print('"%s" n\'est pas une réponse valable !' % ans)
            print_details(pgivename, psurname, pusername, pmail, pdescription,
pgrade, pmanager, paccountExpires, pprimarygroup, pgroups, pou)
            list_columns(["1: Prénom", "2: Nom", "3: Adresse Mail", "4:
Grade", "5: Fonction", "6: Responsable", "7: Date d'expiration du compte", "8:
Groupe primaire (NE FONCTIONNE PAS)", "9: Groupe d'appartenance", "c:
Appliquer les changements"])
            # list_columns(["1: Prénom", "2: Nom", "3: Adresse Mail", "4:
Statut", "5: Responsable", "6: Date d'expiration du compte", "7: Groupe
primaire", "8: Groupe d'appartenance", "9: Unité d'organisation", "c:
Appliquer les changements"])
            ans = input("\nQuel élément voulez-vous modifier (c pour valider
les changements) ?\n: ")
        if ans == "1":
            ngivename = input("\nEntrez le nouveau prénom de l'utilisateur %s
: " % ousername)
            pgivename = "%s -> %s" % (ogivename, ngivename)
            changes_to_confirm.append(('givenName', ngivename))
        if ans == "2":
            nsurname = input("\nEntrez le nouveau nom de l'utilisateur %s : "
% ousername)
            psurname = "%s -> %s" % (osurname, nsurname)
            changes_to_confirm.append(('sn', nsurname))
        if ans == "3":
            nmail = None
            while (not nmail) or (not check_valid_mail(nmail)):
                nmail = input("\nEntrez la nouvelle adresse mail de
l'utilisateur %s : " % ousername)
            pmail = "%s -> %s" % (omail, nmail)
            changes_to_confirm.append(('mail', nmail))
        if ans == "4":
            ndescription = None
            while ndescription not in Status_list:
```

```
                if (ndescription and ndescription not in Status_list):
                    print("\nLe statut '%s' ne fait pas parti des statuts
existants." % ndescription)
                else:
                    print("\nLe statut est une valeur obligatoire.")
                list_columns(Status_list)
                ndescription = input("\nEntrez le nouveau statut de
l'utilisateur %s : " % ousername)
            verboseprint("Vous avez choisi '%s' comme statut." % ndescription)
            pdescription = "%s -> %s" % (odescription, ndescription)
            changes_to_confirm.append(('description', ndescription))
        if ans == "5":
            ngrade = input("\nEntrez le nouveau grade de l'utilisateur %s : "
% ousername)
            pgrade = "%s -> %s" % (ograde, ngrade)
            changes_to_confirm.append(('grade', ngrade))
        if ans == "6":
            nmanager = input("\nEntrez le nouveau responsable de l'utilisateur
%s : " % ousername)
            while (nmanager) and (not check_exist_user(nmanager)):
                print("Le manager doit existeri dans la base AD ou être vide")
                nmanager = input("\nEntrez le nouveau responsable de
l'utilisateur %s : " % ousername)
            pmanager = "%s -> %s" % (omanager, nmanager)
            changes_to_confirm.append(('manager', nmanager))
        if ans == "7":
            naccountExpires = input("\nEntrez la nouvelle date d'expiration du
compte %s (JJ-MM-AAAA) : " % ousername)
            paccountExpires = "%s -> %s" % (oaccountExpires, naccountExpires)
            changes_to_confirm.append(('accountExpires', naccountExpires))
        if ans == "8":
            nprimarygroup = input("\nEntrez le nouveau groupe primaire de
l'utilisateur (NON FONCTIONNEL) %s : " % ousername)
            pprimarygroup = "%s -> %s" % (oprimarygroup, nprimarygroup)
            changes_to_confirm.append(('primarygroup', nprimarygroup))
        if ans == "9":
            ngroups = input("\nEntrez la nouvelle liste des groupes dont
l'utilisateur %s fait partie (syntaxe : group1,group2,group3...) : " %
ousername)
            pgroups = "%s -> %s" % (ogroups, ngroups)
            changes_to_confirm.append(('groups', ngroups))
        if ans == "c":
            validate = True

    changes_dict = {}
    change_group = {}

    #dès qu'on détecte une modif on l'ajoute dans le bon dictionnaire
    if pgivename != ogivename:
```

```
        changes_dict['givenName'] = ngivename
    if psurname != osurname:
        changes_dict['sn'] = nsurname
    if pmail != omail:
        changes_dict['mail'] = nmail
    if pdescription != odescription:
        changes_dict['description'] = ndescription
    if pgrade != ograde:
        changes_dict['title'] = ngrade
    if pmanager != omanager:
        changes_dict['manager'] = get_adattr( nmanager, 'dn' )
    if paccountExpires != oaccountExpires:
        args.expire=naccountExpires
        date=ad_timestamp(timestamp=args.expire)
        changes_dict['accountExpires'] = date
    if pprimarygroup != oprimarygroup:
        change_group['nprimarygroup'] = nprimarygroup
    if pgroups != ogroups:
        change_group['ngroups'] = ngroups


    if change_group:
        print("Il y a des changements dans les groupes (ne fonctionne pas
encore)")
        print(change_group)
        for attribute, new_value in change_group.items():
            confirmation = str(f"Voulez-vous modifier l'attribut '{attribute}'
avec la valeur '{new_value}' ? ")
            rep=yes_or_no(confirmation)
            if rep == True:
                # Effectuer les modifications de groupe ici
                if attribute == 'nprimarygroup':
                    # get_adattr(nprimarygroup, 'gidNumber')
                    # ldif...
                    gidnumber=get_adattr(args.pgroup, 'gidNumber')
                    verboseprint(gidnumber)

                    # Vérifier si la recherche renvoie des résultats
                    if gidnumber:
                        # Création du LDIF pour la modification du gidNumber
                        ldif_data = "dn:
{}\n".format(get_adattr(args.username, 'dn'))
                        ldif_data += "changetype: modify\n"
                        ldif_data += "replace: gidNumber\n"
                        ldif_data += "gidNumber: {}\n".format(gidnumber)

                    verboseprint(ldif_data)
                    confirmation_execute = str("Voulez-vous exécuter cette
modification ? ")
```

```
                    rep=yes_or_no(confirmation_execute)
                    if rep == True:
                        # Exécution de la modification en utilisant le LDIF
                        samdb.modify_ldif(ldif_data)

                elif attribute == 'ngroups':
                    for group in ogroups:
                        #supprime tout les groupe actuel puis les remplace par
les nouveaux
                        samdb.add_remove_group_members(groupname=group,
members=[args.username], add_members_operation=False)
                    groups = new_value.split(",")
                    for group in groups:
                        samdb.add_remove_group_members(groupname=group,
members=[args.username], add_members_operation=True)
                elif attribute == 'nou':
                    # Effectuer les modifications d'unité d'organisation ici
                    # ...
                    pass

                print(f"L'attribut '{attribute}' a été modifié avec succès.")
            else:
                print(f"La modification de l'attribut '{attribute}' a été
annulée.")
                break


    if changes_dict:
        print("Il y a des changements")
        print(changes_dict)
        all_changes_confirmed = False

        # Demander confirmation pour toutes les modifications en une seule
fois
        confirmation_global = str("Voulez-vous appliquer ces modifications ?
")
        rep=yes_or_no(confirmation_global)
        if rep == True:
            print(get_adattr(args.username, 'dn'))
            ldif_data = "dn: {}\n".format(get_adattr(args.username, 'dn'))
            ldif_data += "changetype: modify\n"

            # Ajouter les modifications confirmées au LDIF
            for attribute, new_value in changes_dict.items():
                if new_value!=None:
                    ldif_data = "dn: {}\n".format(get_adattr(args.username,
'dn'))
                    ldif_data += "changetype: modify\n"
                    ldif_data += "replace: {}\n".format(attribute)
```

```
                    ldif_data += "{}: {}\n".format(attribute, new_value)
                else :
                    ldif_data = "dn: {}\n".format(get_adattr(args.username,
'dn'))
                    ldif_data += "changetype: modify\n"
                    ldif_data += "add: {}\n".format(attribute)
                    ldif_data += "{}: {}\n".format(attribute, new_value)
                samdb.modify_ldif(ldif_data)
            print(ldif_data)

            print("Les modifications ont été appliquées avec succès.")

        else:
            print("Les modifications ont été annulées.")

    all_changes_confirmed = True

    display_and_exit()
```