Manuel d'installation et d'utilisation de Google/CAPIRCA pour Telespazio (GCT)

Juan PIRON
Elève ingénieur,
4ème année *Sécurité et Technologies Informatiques*,
Insa Centre Val de Loire,
Campus de Bourges
juan.piron@insa-cvl.fr

26 juillet 2017

Table des matières

1 Prérequis **3**

2 Ce qu'est Google/CAPIRCA pour TPZ **3**

3 Ce que n'est pas Google/CAPIRCA pour TPZ **3**

4 Installation **3**
4.1 Prérequis 3
4.1.1 MariaDB 3
4.1.2 Python 4
4.1.3 SSH 4
4.1.4 Autres 5
4.2 Procédure 5
4.3 Déploiement 6

5 Utilisation **6**
5.1 Configuration du dépôt local et du client 6
5.2 Exemple 7
5.3 Recommandations 8

6 Rappels succincts sur les commandes Git **8**

1 Prérequis

Ce manuel n'a pas pour but de décrire l'architecture de Google/CAPIRCA.
Ce manuel ne remplace d'aucune manière la documentations de Google/CAPIRCA, il apporte simplement les spécifités de GCT. Il est donc nécessaire de lire le README de Google/CAPIRCA. Pour cela il existe la wiki GitHub de Google/CAPIRCA.

Toutes les adresses ips et les configurations présentes dans ce document sont fictives.

2 Ce qu'est Google/CAPIRCA pour TPZ

Capirca est un outil conçu pour utiliser des définitions communes des réseaux, des services et des fichiers de règles (politiques) de haut niveau. Cela facilite le développement et la manipulation des listes de contrôle d'accès réseau (ACL) pour diverses plates-formes. CGT est Capirca appliqué à la gestion du PGL. Il permet de rendre plus rapide la prise en charge des demandes d'ouverture de flux utilisateur.

3 Ce que n'est pas Google/CAPIRCA pour TPZ

GCT ne remplace en aucun cas le manager (Cisco ASDM). C'est à dire que toutes modifications ou suppression d'une demande ce fait avec le manager. En d'autres termes pour supprimer une ACL ajouter à l'aide de CGT :

1/ La supprimer dans le fichier .pol correspondant.

2/ Aller sur le manager et la supprimer.

Pour modifier :

3/ réécrire le term correspondant à l'ACL dans le .pol.

4/ recompiler

4 Installation

Cette section concerne l'installation de Gogs sur le serveur (le manager) et le déploiement de GCT. L'installation décrite concerne un serveur Centos 7.

4.1 Prérequis

4.1.1 MariaDB

Si la bdd n'existe pas alors l'installer :

```
sudo yum -y update
sudo yum -y install mariadb-server
sudo service mariadb start
mysql -u root -e "SET PASSWORD FOR 'root'@'localhost' = PASSWORD('password')
```

4.1.2 Python

Version 2.7 ou supérieur.

Installer pip :

```
# curl ̈https://bootstrap.pypa.io/get-pip.py ̈o ̈get-pip.py ̈
# python get-pip.py
```

Installer paramiko :

```
# yum install gcc libffi-devel python-devel openssl-devel
# pip install cryptography
# pip install paramiko
```

Installer netmiko :

```
# pip install scp
# pip install pyyaml
# pip install pytest
```

4.1.3 SSH

Avoir openssh-server sur le serveur et configuré ssh avec un utilisateur ssh git sur le PGL.

Firewall PGL :

```
hostname(config)# crypto key generate rsa modulus 1024
hostname(config)# write memory
hostanme(config# aaa authentication ssh console LOCAL
hostanme(config# username git password password
hostname(config)# ssh timeout 30
hostname(config)# ssh version 2
```

Cette example de configuration a été fait à partir de la maquette qui utilise un cisco ASA 5520. Il peut donc y avoir des différences avec le PGL. Eviter d'utiliser les certificats, sinon modifier les scripts pre-receive et post-receive. Entre le client et gogs-server :

Sur un client :

```
# ssh-keygen -t rsa
# chmod 700  /.ssh
# chmod 600  /.ssh/id_rsa
# chmod 644  /.ssh/id_rsa.pub
# chmod  /.ssh/know_hosts
```

Envoyer la clé public id_rsa.pub public du poste client au server.

Sur le serveur :

```
# cat id_rsa.pub » /.ssh/authorized_keys
# chmod 700 /.ssh
# chmod 644 /.ssh/authorized_keys
# chmod 600 /.ssh/id_rsa
# chmod 644 /.ssh/id_rsa.pub
```

Dans /etc/ssh/sshd_config désactiver PasswordAuthentication et ainsi forcer l'utilisation des clés :

```
PasswordAuthentication no
```

4.1.4 Autres

Configurer SELinux de manière à permettre l'accès à ssh (scp) au dossier tmp pour l'échange des clés. Installer git :

```
# yum -y install git
```

Une fois Capirca importé il faut installer les prérequis dans *capirca/* à l'aide de :

```
# pip install -r requirements.txt
```

4.2 Procédure

Sur le serveur "manager" créer l'utilisateur git et définir son mot de passe :

```
# useradd -m git
```

Ajouter pour l'installation git au groupe sudo :

```
# usermod -aG wheel git
```

Ne pas oublier de l'en retirer après l'installation.

Décompresser gogsInstall.tar.gz dans */home/git* :

```
# tar -xzvf gogsInstall.tar.gz
```

Ce déplacer dans le dossier gogsInstall éditer les scripts python pre-receive et post-receive. L'adresse ip qui doit être rentrer et celle du PGL, de même que le couple id/password ssh :

```
pgl = {
  'device_type':'cisco_asa',
  'ip':'192.168.1.1',
  'username':'intern',
  'password':'tpzinsacvl',
  'secret':'tpzinsacvl',
}
```

Editer app.ini, faire correspondre l'adresse ip et le domain avec ceux du serveur :

```
[server]
PROTOCOL = http
DOMAIN = 192.168.1.10
ROOT_URL = %(PROTOCOL)s://%(DOMAIN)s:%(HTTP_PORT)s/
HTTP_ADDR = 192.168.1.10
```

Ajouter les droits d'execution au script gogsInstall.sh :

```
# chmod u+x gogsInstall.sh
```

Exécuter le script en scpécifiant le mot de passe root de mariadb comme arg1. Et scpéfier le mot de passe que vous voulez donner à l'utilisateur git de mariadb en arg2 :

```
# ./gogsInstall.sh mdp-root-mariadb mdp-git-mariadb
```

Ouvrir un navigateur et rentrer l'url : `http://ip-du-serveur:3000`

Une page d'installation s'ouvre, il suffit d'entrer le mot de passe de l'utilisateur git de la bdd mariadb et de confirmer.

4.3 Déploiement

Pour l'utilisation de Gogs se référer à sa documentation.

Créer un utilisateur "tpz". A l'aide de tpz ajouter un repository distant nommer "ACLs". Décompresser GCT.tar.gz, uploader capirca dans ACLs. Créer d'autres utilisateurs et les incorporer comme participant en "rw". Récupérer les différentes clés ssh des postes qui cloneront ACLs et les ajouter à tpz en allant dans ses paramètres. Ne pas les ajouter au repository mais bien directement dans les paramètres de l'utilisateur tpz. Enfin récuperer la wiki dans *CGT/* et l'ajouter au repository ACLs.

5 Utilisation

5.1 Configuration du dépôt local et du client

Avant tout transférer sa clé rsa (ssh) à l'admin. Ensuite se placer dans un nouveau repertoire et faire un `git init`. L'étape suivante et de configurer localement git à l'aide de :

```
# git config -global user.name "John Doe"
# git config -global user.email johndoe@example.com
```

Le nom spécifié sera celui qui apparaîtra sur les commits. Pour une demande d'ouverture de flux utilisateur créer autant de fichier .pol qu'il y a d'interfaces du PGL concernées. Dans le header le champ `target::` doit avoir pour nom celui de l'interface. Capirca ne comprend que les objets que cela soit pour les réseaux ou les services. Avant de créer un .pol il faut donc créer les objets. Pour les réseaux les ajouter en début du fichier *capirca/def/NETWORK.net* et pour les services dans *capirca/def/SERVICES.svc*. GCT récupère de manière automatique les objets présent sur le PGL, donc regarder d'abord si l'objet existe en faisant un CTRL-F à l'aide de l'ip ou du service. Les demandes au format .pol doivent se trouver dans le dossier *capirca/policies/pol*. Après avoir terminer de créer les différents fichier .pol, dans *capirca/* lancer le script : `./run.sh`

Quand on vous le demandera, entrer un commit. Tous les commits doivent être de la forme : `ajout de ref-de-la-demande` ou `modification de ref-de-la-demande`.

Pour une suppresion ou modification faire comme indiqué dans *Ce que n'est pas GCT*.

Récapitulation :

1/ Rechercher ou créer les objets

2/ Créer les .pol

3/ `./run.sh`

Si lors de la compilation (abus de langage) il se produit un git "CONFLICT" se référer à la partie *Rappel succint sur les commandes git*. Avant tout essai assurez-vous d'avoir bien toutes les librairies python requises d'installé :

- gflags (`pip install python-gflags`)
- getopt
- csv
- re (`pip install regex`
- socket
- netaddr

5.2 Exemple

Voici un exemple de configuration d'une demande .pol et des objets.

```
------ .POL -------

header {
  comment:: "Demandeur : Tintin"
  comment:: "Realisee par : Cpt. Hadock"

  target:: ciscoasa Eth1_CSG
}

term NTP {
  source-address:: REMUS1
  protocol:: udp
  source-port:: NTP_SOURCE
  destination-address:: REMUS2
  destination-port:: NTP_DEST
  logging:: syslog
  action:: accept
}

term SYSLOG {
  source-address:: A_192_168_1_2
  protocol:: udp
```

```
  source-port:: SYSLOG_SOURCE
  destination-address:: SSI
  destination-port:: SYSLOG_DEST
  logging:: syslog
  action:: accept
}

------ NETWORK.net -------

REMUS1 = 192.168.10.0/24
REMUS2 = 192.168.20.0/24
REMUS10 = 192.168.100.0/24
REMUS11 = 192.168.110.0/24
A_192_168_1_2 = 192.168.1.2/32
SSI = 192.168.30.0/24

------ SERVICES.svc -------

NTP_SOURCE = 123/udp
NTP_DEST = 123/udp
SYSLOG_SOURCE = 514/udp
SYSLOG_DEST = 1024-65535/udp
Netbios-src = 135/tcp
              139/tcp
Netbios-dest = 1024-65535/tcp
```

5.3 Recommandations

Ne pas utiliser de charactères spéciaux. Ne pas utiliser de : ou d'accents. Ne pas écrire de `remark::` trop longue, la limite est de 50 charactères.

6 Rappels succincts sur les commandes Git

En cas de conflit Git c'est toujours mieux de se souvenir des commandes importantes.

`git init` : initialise un repertoire vide

`git clone https://...` : récupère un projet (correspond à un fetch + un merge)

En local :

`git add fichier.txt`

`git add -A` : tous

`git commit fichier.txt -m blablabla ou git commit -a blabla`

`git commit -amend` : modifier le dernier commit (ex : faute de frappe).

Travail avec un depôt distant :

`git remote add origin https://...` : ajout d'un depôt distant, origin correspond au nom que l'on donne au dépôt

`git remote -v` : liste les reps distants (dépôts)

`git remote rm origin` : supprime le remote origin

`git pull origin master` : mise à jour avec la branche master du dépôt origin

`git push origin master` : on envoie à la branche master du dépôt origin nos modifs

Les branches :

`git branch test` : créer une branche test en local

`git branch` : liste les branches présentes

`git push orgin test` : permet de pusher sur la nouvelle branche mais aussi de la créer sur le dépôt distant

`git checkout test` : permet de switcher entre les différentes branches

`git merge test` : depuis la branche master, permet de fusionner le travail de la branche test

`git push origin -delete test` : supprime la branche dans le rep distant

`git branch -d test` : supprime la branche localement

Le stash :

`git stash` : pour mettre de côté un travail que l'on ne veut pas commit si on doit changer de branch

`git stash apply ou git stash apply stash@{0}` : pour le récupérer au retour

`git stash list` : affiche la pile

`git stash drop stash@{0}` : supprime le stash

`git stash pop` : pour appliquer la remise puis la suprimer

Les différences :

`diff master origin/master` : différence entre la branche local et celle dans le rep distant

Revenir en arriere :

`git log` : lister les commits (et voir le num)

`git checkout <commit>` : revenir au commit indiqué comme spectateur

`git reset -hard` : revenir au dernier commit

`git reset <commit> -hard` : revenir au commit indiqué

Les différentes copies dans gits :

workspace, index, local repository, remote repository

add : workspace –> index
commit : index –> local
push : local –> remote
fetch : remote –> local
merge : local –> workspace
pull : remote –> workspace