

# *Etude de la gestion en configuration des demandes de flux via le logiciel Google/CAPIRCA*

« Résumé d'activité »

Dates du stage :  
04/06/2017 au 04/08/2017

Tuteur de stage :  
Philippe Charron  
[philippe.charron@cnes.fr](mailto:philippe.charron@cnes.fr)

Enseignant référent :  
Christian Toinard  
[christian.toinard@insa-cvl.fr](mailto:christian.toinard@insa-cvl.fr)



## Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Ma gratitude s'adresse tout d'abord à mon professeur, **Mr Christian TOINARD**, tuteur de ce stage.

Je tiens à remercier vivement aussi mon maître de stage, **Mr Philippe CHARRON**, et le remercier pour son accueil et le partage de son expertise au quotidien. Grâce aussi à sa confiance, j'ai pu m'accomplir totalement dans mes missions. Il m'a été d'une aide précieuse dans les moments les plus délicats.

Je remercie encore toute l'équipe de Telespazio Guyane, dont j'ai pu apprécier l'esprit d'équipe et en particulier **Mr Christophe VIALON** et **Mr Julien MONEIN**, dont l'assistance m'a souvent été appréciable.

## Résumé

L'objectif de ce stage était l'étude de la possibilité de faire migrer les règles existantes d'un des firewalls principal (type **cisco ASA**) du **Centre Spatial Guyanais** (que l'on appellera **PGL**) dans le logiciel **google/capirca** et de mettre en place les mécanismes adéquats permettant la gestion de configuration des demandes de flux utilisateurs.

Cette demande s'articulait autour de 2 axes principaux. D'une part, la mise en place d'un mécanisme de génération des règles et leur gestion en configuration (git ou subversion ?) à partir des demandes de flux utilisateurs. D'autre part, l'adaptation des règles existantes pour les injecter dans le système **google/capirca**.

Capirca, développé par Google (d'où **google/capirca**), permet la génération d'access- lists de plusieurs types (cisco, junipers, iptables ...) et bien entendu **ciscoasa**, ce qui nous intéresse ici.

Au travers de différentes spécifications techniques, il a été proposé pour répondre à cette expression du besoin la mise en place d'un serveur git open source, **gogs** (go git service) embarquant **google/capirca** et d'une interface utilisateur de type shell/python pour la gestion des demandes de flux utilisateurs. L'interface prenait en compte la gestion des conflits qui peuvent exister entre le manager (**cisco ASDM – Adaptive Security Device Manager**), et les différents collaborateurs. En effet, les access-lists faisant partie des données ultrasensibles liées à l'activité spatiale, il est totalement interdit d'utiliser des serveurs tel que gitHub ou gitLab.

Pour des questions évidentes de sécurité et par précaution, une maquette a été mise en place pour la réalisation et la démonstration de ce projet. Cette maquette ressemble en tous points à la réalité, seules les adresses IPs sont fictives.

Le résultat de ce travail vise à permettre à tout gestionnaire présent dans le réseau du **PGL** de cloner le projet contenant l'interface de gestion et toutes les nouvelles demandes de flux. Il permet également d'en ajouter ou de modifier les existantes. Toujours à l'aide de l'interface, la nouvelle configuration peut ensuite être envoyée dans la running-config. Deux gestionnaires peuvent travailler sur les deux modes d'intégration existants (**google/capirca**, le **manager**), sans s'inquiéter des conflits pouvant être créés. Cette solution permet donc une réponse plus rapide mais aussi une gestion plus organisée des demandes de flux utilisateurs.

## Bibliographie

- <https://github.com/google/capirca/wiki>
- <https://gogs.io/docs>
- <http://www.cnes-csg.fr/web/CNES-CSG-fr/>
- <https://www.telespazio.fr/fr>
- <https://pynet.twb-tech.com/blog/automation/netmiko.html>

## Glossaire

- **ACL** : Une ACL est une liste d'Access Control Entry (ACE) ou entrée de contrôle d'accès donnant ou supprimant des droits d'accès à une personne ou un groupe.
- **ASDM** : Adaptive Security Device Manager, le manager.
- **Cisco ASA** : Depuis début 2005, Cisco a lancé une gamme de boîtiers (ASA 5510, 5520 et 5540) appelée Cisco ASA pour Adaptive Security Appliance.
- **CSG** : Centre spatial Guyanais
- **PGL** : Nom d'un firewall
- **REMUS** : réseau interne du CSG

## Table des matières

Introduction .....	2
Le Centre Spatial Guyanais (CSG) et Telespazio Guyane .....	3
Google/capirca : une solution pour une gestion organisée des demandes de flux .....	5
La gestion des demandes de flux utilisateurs .....	5
Google/capirca .....	6
Google/capirca appliqué au PGL .....	8
Gogs (Go git service).....	10
La maquette .....	11
Démonstration .....	13
Sans google/capirca.....	13
Avec google/capirca .....	14
RETEX (RETour d'EXperience).....	15
Conclusion .....	16
Annexe.....	17

## Introduction

Ce stage de 4<sup>ème</sup> année d'école d'ingénieur, d'une durée de deux mois, a consisté à mettre en place CAPIRCA pour un firewall géré par Telespazio pour le Centre Spatial Guyanais (CSG). Il s'agit d'un logiciel développé par GOOGLE pour automatiser la création des ACLs (access-list).

Ce rapport présente le travail réalisé durant mon stage au sein de Telespazio Guyane, dans le centre technique du CSG. Il s'est déroulé du 6 Juin au 6 août 2017.

Ce projet, qui s'inscrit dans le domaine de la sécurité et se situe au sein d'un lieu ultrasensible demandant de multiples procédures sécuritaires, s'intègre pleinement dans ma formation dont le cœur est la sécurité des systèmes d'information.

Ce stage m'a permis d'expérimenter de façon pratique la réalité du métier d'ingénieur dans ce secteur d'activité.

Le but de ce résumé d'activité n'est pas de faire une description exhaustive de tous les aspects techniques mis en œuvre, mais de présenter de manière claire et synthétique les différentes parties techniques.

Dans un premier temps est présentée l'entreprise d'accueil, Telespazio France. Ensuite seront exposées les différentes tâches réalisées au cours de ces 2 mois. Pour finir, je reviendrai sur les apports de ce stage dans ma formation, et conclurai.

## Le Centre Spatial Guyanais (CSG) et Telespazio Guyane

Le Centre National d'Etudes Spatiales (CNES) utilise à ses débuts des installations militaires implantées à Hammaguir dans le désert algérien. A l'indépendance de l'Algérie, un nouveau site doit être trouvé. Sur les 14 emplacements pressentis à travers le monde, la Guyane française arrive largement en tête car elle offre des conditions de lancement optimales. Grâce à sa large ouverture sur l'océan les lancements se font avec un maximum de sécurité tant vers l'est que vers le nord. En lançant vers l'est les lanceurs bénéficient à plein de la vitesse de la rotation de la Terre, plus importante au niveau de l'équateur. De par la proximité de l'équateur, les satellites géostationnaires minimisent les manœuvres de correction de trajectoire, économisant ainsi du carburant et augmentant notablement leur durée de vie.

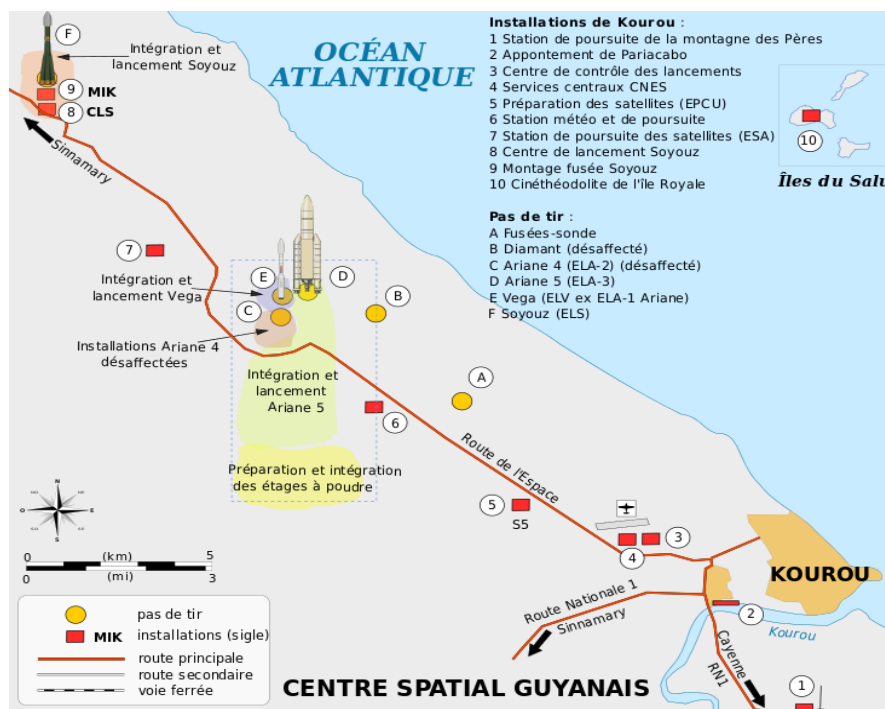


Figure 1

Pour mener à bien chacune de ses activités le Centre Spatial Guyanais est composé d'un consortium d'entreprises provenant de milieux divers. On peut citer par exemple Videlio qui s'occupe des prises de vues, ou encore Europropulsion qui se charge de la production du propergol solide (carburant). Toutes ces entreprises, parmi lesquelles Telespazio, sont réunies sur la base spatiale, dénommée CSG (Centre Spatial Guyanais), et placées sous la tutelle du CNES (Centre National d'Etudes Spatiales).

Telespazio France est organisé autour de trois métiers d'excellence :

- Les Systèmes Satellitaires et Opérations, à savoir l'exploitation des moyens et systèmes spatiaux des clients (tels que les systèmes européen EGNOS, la base spatiale de Kourou ou les moyens de télécommunications spatiales de la Défense Française...) et du développement des applications et solutions associées.



- Les Télécommunications Spatiales proposant un large portefeuille de solutions et services de connectivité.
- La Géo-Information fournissant une offre unique en imagerie spatiale radar et optique, produits et services d'observation et de surveillance de la terre et des océans.

En Guyane Française, Telespazio Guyane concentre la plupart de ses activités sur la base spatiale. Sur les 3 métiers d'excellence, on retrouve surtout les deux premiers. Une des parties de la base spatiale est appelé centre technique (CT), chaque bâtiment portant le nom d'un astre, et regroupant un service (généralement une société). Personnellement j'ai effectué mon stage dans le bâtiment Mercure, dans le service « *Groupe Réseaux, Commutation, Configuration et Supports* ». Telespazio Guyane est constitué de 2 services, le deuxième étant le service qui se charge de tout ce qui est télémessure et antenne de poursuite lanceur.

## Google/capirca : une solution pour une gestion organisée des demandes de flux

### La gestion des demandes de flux utilisateurs

Le firewall PGL centralise les règles d'accès des différents pôles de la Base Spatiale (centre technique, CDL3 etc ... Voir la figure 3 ). Il est configuré en « deny all », c'est-à-dire que, à la base, tout accès est rejeté. Toute personne, ou tout service qui désire l'ouverture d'un accès, en fait la demande à la SSI. La SSI transmet à Telespazio cette demande au travers d'une « demande d'ouverture de flux utilisateur ». Cette demande se présente sous la forme d'un fichier constitué d'une référence, des nom et prénom du demandeur etc... Mais surtout elle fait mention des différents accès à ouvrir. Sont indiqués les adresses IP, les ports, les services et le sujet de cette ouverture.

Une fois cette demande reçue par Telespazio elle est transmise à une personne que l'on nommera, dans le cadre de ce résumé d'activité, gestionnaire. Ce gestionnaire se chargera alors d'ouvrir les différents accès en ajoutant les ACL correspondantes dans le firewall PGL.

Pour réaliser cela, le gestionnaire doit se placer au niveau du réseau privé du PGL et se connecter au manager. Le manager n'est autre que le cisco ASDM, en effet le PGL et un firewall de type cisco ASA.

Cette manipulation prend du temps, notamment car elle ne peut être réalisée de n'importe où. Elle nécessite en effet une présence physique derrière une machine précise. Ce poste, sur lequel est installé cisco ASDM, est appelé « manager ». L'ajout des règles au niveau du manager est également long et fastidieux, surtout quand on sait qu'une demande d'ouverture de flux suppose en moyenne l'ajout d'une dizaine d'ACLs.

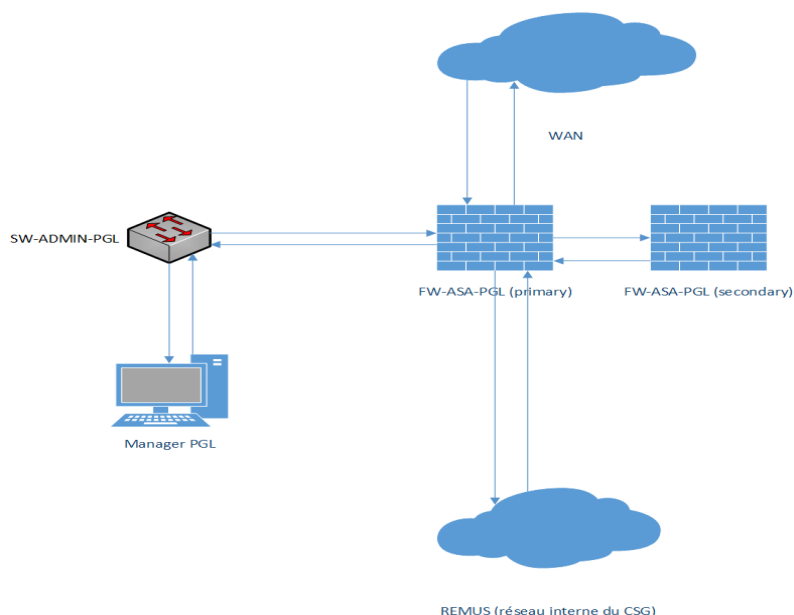


Figure 2

Le manager permet l'ajout d'ACLs, mais aussi d'objets. Un objet peut être l'IP d'un hôte, d'un réseau, d'un service (par exemple : https), d'une liste de protocole ... Ces objets correspondent bien entendu à des objets de type ciscoasa. Ils apparaissent donc dans la running config.

Or il n'est pas toujours créé d'objet pour chaque IP. Ainsi, on se retrouve souvent avec des doublons au sein du PGL, ce ne pose pas de problèmes en termes de fonctionnement. Prenons un exemple simple : le gestionnaire A traite une demande de flux à l'aide d'objets créés ou pré-existants. Le gestionnaire B fait de même avec une autre demande d'ouverture de flux. Il s'avère, toujours pour l'exemple, qu'entre ces deux demandes, 4 règles sont similaires. Etant donné que leur méthode d'ajout est différente et qu'il n'y a aucun système de détection, il y a un doublon.

Il faut ajouter concernant la technique de gestion actuelle au travers du manager que cette dernière ne permet pas de garder un visuel sur les demandes de flux traitées. En d'autres termes, on ne peut pratiquement pas retrouver une demande de flux dans la configuration du PGL avec ou sans le manager, après traitement de cette dernière.

Sinon, parmi ses aspects positifs (tels qu'une visualisation améliorée des ACLs), le manager reste le seul moyen simple de modification des configurations du PGL.

C'est dans ce cadre que l'on introduira google/capirca.

### Google/capirca

Pour pouvoir accélérer l'intégration des demandes de flux et de faciliter leur regroupement, on m'a demandé d'introduire google/capirca dans leur gestion.

Capirca est un outil conçu pour utiliser des définitions communes des réseaux, des services et des fichiers de politiques de haut niveau pour faciliter le développement et la manipulation des listes de contrôle d'accès au réseau (ACL) pour diverses plates-formes. Il a été développé par Google pour un usage interne et est désormais open source, récupérable sur Github.

Le gestionnaire retranscrit en langage de haut niveau capirca les différentes règles exprimées dans le fichier de demande d'ouverture de flux. Il crée ce que l'on appelle le policy file. Le policy ne prend que des objets (IP, services ...). Ces derniers doivent être inscrits préalablement dans des fichiers .svc pour les services et .net pour les hôtes et réseaux. Il suffit ensuite de lancer le script python aclgen.py présent dans le répertoire principal de Capirca. On obtient en sortie les ACLs désirés après que Capirca ait fait appel aux différents générateurs sélectionnés. Le seul qui nous intéresse ici étant le ciscoasa, on obtient donc qu'un seul fichier de type .asa.

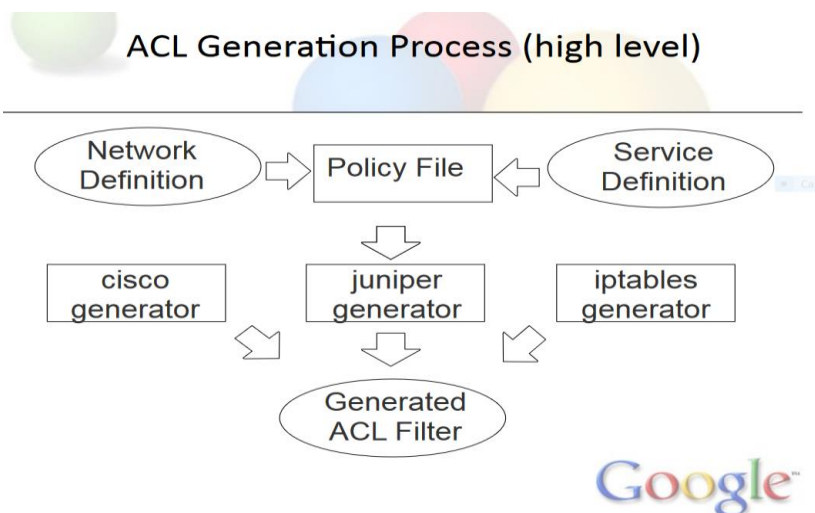


Figure 3

Un fichier de stratégie (policy file) consiste en un ou plusieurs filtres, chaque filtre contenant un ou plusieurs termes. Chaque terme spécifie les informations de base du filtre réseau, telles que les adresses, les ports, les protocoles et les actions.

Un fichier de stratégie comprend une ou plusieurs sections d'en-tête, chaque section d'en-tête étant suivie d'un ou plusieurs termes. Une section d'en-tête est généralement utilisée pour spécifier un filtre pour une direction donnée

De plus, le langage de haut niveau prend en charge les "fichiers inclus", dont le code est injecté dans la policy à l'emplacement spécifié.

Chaque filtre est identifié par une section d'en-tête. La section d'en-tête sert à définir le type de filtre, un descripteur ou un nom, une direction (le cas échéant) et un format (ipv4 / ipv6).

Par exemple, l'en-tête simple suivant définit un filtre qui peut générer des résultats pour les formats cisco, juniper, iptables ,ciscoasa ....

```

header {
  comment:: "Example header for juniper and iptables filter."
  target:: juniper edge-filter
  target:: speedway INPUT
  target:: iptables INPUT
  target:: cisco edge-filter
  target:: ciscoasa edge-filter
}
  
```

Chaque fichier de sortie prend le nom de la target spécifiée avec l'extension qui est fonction du type de target (ex : edge-filter.asa). Sauf, configuration différente des paramètres dans le script aclgen.py, les filtres de sortie générés se trouvent dans capirca/filters. Les fichiers de haut-niveau doivent quant à eux se trouver dans capirca/policies/pol.

Les termes définissent les règles de contrôle d'accès dans un filtre. Une fois que le filtre est défini dans les sections d'en-tête, il est suivi d'un ou plusieurs termes. Les termes sont entre

parenthèses et utilisent des mots-clés pour spécifier la fonctionnalité d'un contrôle d'accès spécifique.

Une section de terme commence par le mot-clé, suivi d'un nom de terme. Des crochets d'ouverture et de fermeture suivent, y compris les mots-clés et les jetons pour définir l'appariement et l'action du terme de contrôle d'accès.

Les mots-clés se divisent en deux catégories, ceux-ci doivent être pris en charge par tous les générateurs de sortie et ceux qui sont éventuellement pris en charge par chaque générateur. Les mots clés facultatifs sont destinés à offrir une flexibilité supplémentaire lors de l'élaboration de stratégies sur une plate-forme cible unique.

```
term permit-to-web-servers {  
  destination-address:: WEB_SERVERS  
  destination-port:: http  
  protocol:: tcp  
  log :: syslog  
  action:: accept  
}
```

Comme soulevé plus haut, le script de génération de Capirca `aclgen.py` est configurable, c'est-à-dire que l'on peut spécifier le répertoire des fichiers de haut-niveau et le répertoire de sortie où seront stockés les filtres, d'autres paramètres tels que « `shader` » peuvent être activés. Activer « `shader` » et « `verbose` » permet d'avertir le gestionnaire de l'existence de doublon au sein d'un même fichier `.pol`.

Nous allons voir à présent la solution proposée pour l'intégration de Capirca à la gestion des ACLs pour le PGL.

### Google/capirca appliqué au PGL

Capirca ne peut être intégré directement tel quel dans le cadre du PGL. Le but n'étant pas de repartir à zéro, mais d'intégrer Capirca au système existant. Le système existant (évoqué en introduction) est un manager (cisco ASDM). La première partie du travail a donc été de savoir comment réaliser l'intégration.

Plusieurs solutions se proposaient. La première qui paraissait sur le moment la plus logique, était de récupérer par script toutes les access-lists existantes de la running-config du PGL et de les recréer en langage de haut niveau capirca. Plus exactement, un fichier par nom d'access-list. En effet, les access-lists sont nommées par interfaces. Interfaces qui correspondent à des réseaux du Centre Spatial. Cette solution s'est vite révélée imparfaite. Tout d'abord, malgré le fait que le langage de haut-niveau de Capirca est bien plus lisible que du « bas niveau » ASA, une liste de plusieurs centaines de terms est loin d'être aussi lisible qu'une interface cisco ASDM. Ensuite elle correspondait à un « court-circuit » du système ASDM ce qui n'est absolument pas le but désiré.

## Méthode plus autonome

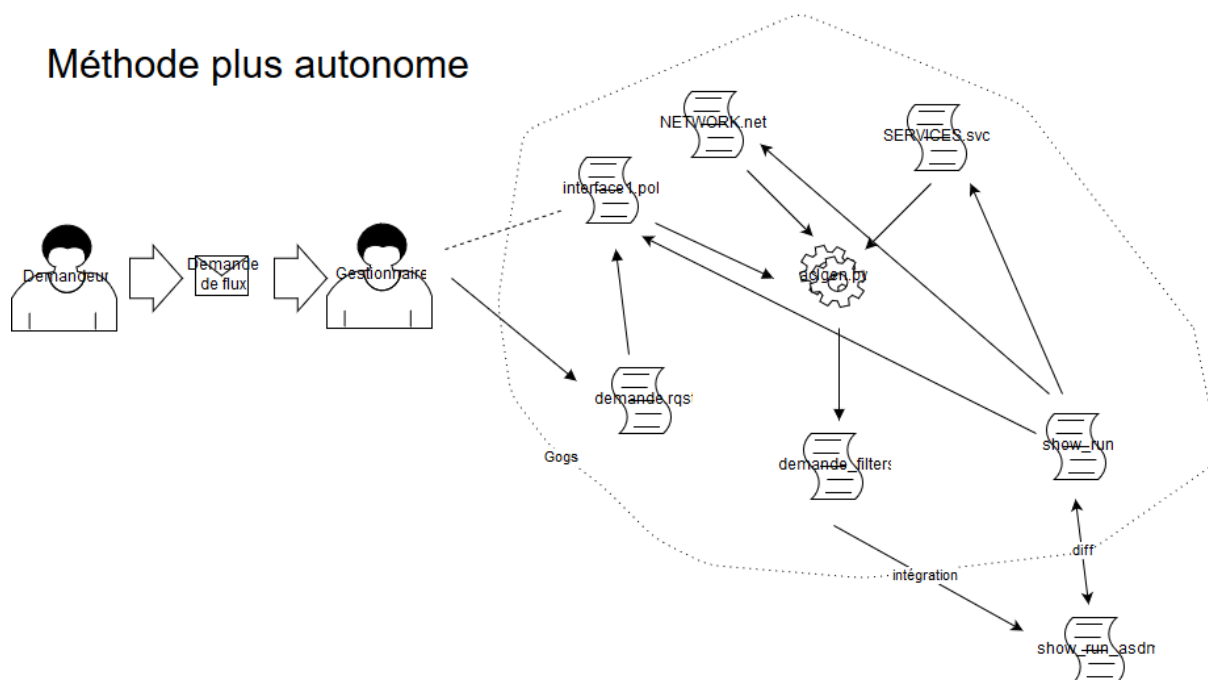


Figure 4

La seconde solution que j'ai proposée et mise en place, correspond donc à une méthode dite collaborative. Ici, seuls les objets de type network et service sont récupérés à l'aide de script python depuis la running-config. En recevant une demande d'ouverture de flux, le gestionnaire crée simplement le fichier .pol (dans le dossier capirca/policies/pol) avec le header contenant une target avec pour nom l'interface de destination souhaité. Le .pol contient également les différents terms. Il reste à créer les objets nécessaires, ou à les réutiliser s'ils sont importés.

Le fichier .pol a comme nom la référence de la demande et dans le header plusieurs commentaires sont ajoutés avec le nom et prénom du demandeur, le nom du gestionnaire etc... Dans capirca, un commentaire se note « comment :: ». Le nom des terms corespondent au nom du service sur lequel le term agit (ex : WEB). S'il y a plusieurs terms qui agissent sur le même service, ils sont alors numérotés (ex : WEB1, WEB2, WEB3).

Ainsi chaque demande d'ouverture de flux qui se présente sous la forme d'un fichier et qui détient une référence se retrouve dans Capirca dans un fichier. Cela permet de retrouver, de modifier, de gérer une demande facilement. Le manager est utilisé quant à lui pour la gestion des anciennes règles, leur suppression, ou leur modification.

Or cela crée des problèmes de concurrence. Imaginons qu'un gestionnaire crée une règle sur la manager, et qu'un deuxième crée une demande .pol qui contient une règle similaire. Il y a alors un doublon. Or l'un des buts de l'utilisation de Capirca est d'améliorer la propreté de la gestion du PGL. Pour cela il a donc fallu créer des scripts python qui, se basant sur la running-config, permettent d'éviter la majeure partie des doublons. Cette gestion est intégrée dans Capirca pour Telespazio dans le dossier diff. « Diff » permet d'éviter les doublons entre Capirca et le manager mais aussi à l'intérieur d'une même demande .pol grâce au « shader » de Capirca ainsi qu'entre chaque demande .pol.

Enfin, chaque intégration dans la running config est faite à l'aide de « push ». Push est aussi développé à l'aide la librairie Netmiko de Python.

# Méthode collaborative

```

graph LR
    Demandeur[Demandeur] --> DemandeDeFlux[Demande de flux]
    DemandeDeFlux --> Gestionnaire[Gestionnaire]
    Gestionnaire --> demande_po[demande.po]
    subgraph Gogs
        demande_po --> supsim[Supsim]
        supsim <--> NETWORK[NETWORK.net]
        supsim <--> SERVICES[SERVICES.svc]
        supsim --> general_asc[general.asc]
        general_asc -- diff --> show_run[show_run]
        show_run -- sync --> show_run_asdn[show_run_asdn]
        show_run -- push --> show_run_asdn
    end

```

Jusqu'à présent les flux peuvent être rapidement ouverts et bien rangés sous la forme de demande .pol (un fichier par demande). Ainsi, plusieurs personnes ne pouvaient pas travailler de manière collaborative. La partie suivante traitera de l'ajout de cette possibilité au travers de Gogs.

Gogs permet de créer et configurer un service Git auto-hébergé. Avec Go, cela peut se faire avec une distribution binaire indépendante dans toutes les plates-formes que Go prend en charge, y compris Linux, Mac OS X, Windows et ARM.

Le serveur Gogs est destiné à être installé sur la même machine que le manager ou du moins une machine présente sur le même réseau qui a un accès au firewall PGL. Sur le serveur est

présent un repository distant contenant Capirca pour Telespazio ( Capirca avec « sync » et « diff »). Chaque gestionnaire peut alors cloner le projet et ensuite ajouter les nouvelles demandes d'ouverture de flux ou en modifier.

Pour des raisons évidentes de sécurité ce projet n'a pu être testé directement au niveau du PGL. Tout le projet a donc été réalisé et présenté au demandeur à l'aide d'une maquette. Cela est présenté dans la partie qui suit.

### La maquette

La maquette est relativement simple, elle consiste en un firewall de type Cisco ASA, d'un client (PC) et d'un serveur Gogs. Chaque élément est sur le même réseau. Le but de cette maquette est de montrer simplement l'interaction entre les différentes parties.

Chaque liaison a été sécurisée le plus possible. Entre le serveur Gogs et le client à l'aide de l'utilisation du serveur ssh (avec certificats) intern de Gogs défini sur un port autre que le 22. Chaque gestionnaire utilise alors une adresse du type [git@gitlab.com:TPZ/Teles.git](mailto:git@gitlab.com:TPZ/Teles.git) pour cloner, pusher, puller ...

Pour synchroniser la running-config il faut la récupérer. Face à l'impossibilité d'utiliser scp, j'ai décidé d'utiliser la librairie python Netmiko. Le serveur gogs se charge donc de récupérer la running-config (c'est « sync » qui se charge de cela). Les dernières versions d'ASA ne permettant pas d'utiliser de certificats, il est utilisé ici le niveau maximum de sécurité au travers de ssh avec identifiant et mot de passe.

Pour pouvoir récupérer cette running config, « sync » utilise les hooks de git et plus particulièrement le hook pre-receive. Sous git un hook permet d'exécuter du code (un script) avant ou après un commit, un push... Le script pre-receive est exécuté avant le traitement d'un push. Ainsi à chaque exécution de run.sh, est exécuté un push à blanc qui permet la synchronisation avec le firewall PGL du serveur Gogs (le hook pre-receive étant côté serveur). A la suite, à l'aide d'un scp le gestionnaire est synchronisé à son tour.

La maquette a donc servi de démonstration pour la solution que j'ai choisie. Dans un premier temps elle sera intégrée seulement au manager. Cela signifie que, seul depuis la machine du manager, on aura accès à Capirca. Dans un futur proche, après installation d'un VPN dans tout le bâtiment, elle devrait être étendue et ouverte à un certain nombre de postes. Dans un grand centre, tel que la base spatiale, les nouveaux projets prennent du temps à être intégrés. Comprenant la sensibilité de ce milieu, on voit mieux pourquoi. Ceci est approfondi dans la partie qui suit : RETEX.



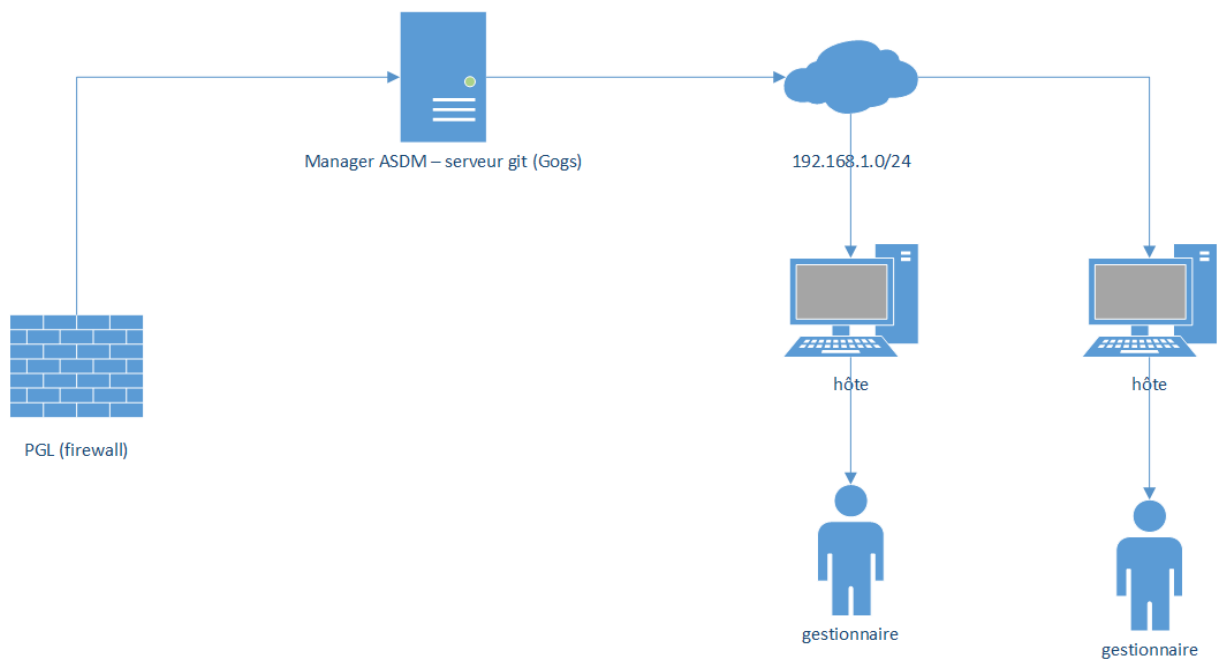



Figure 6

## Démonstration

Sans google/capirca

 <b>cnès</b> CENTRE SPATIAL GUYANAIS	Réf. : CSG-IP-1248-1231-CNES	Classe : GP
	Ed/Rév : 03/01	
	Date : 24/06/2009	
	Page : 12 /15	
	REGLES DE CREATION OU DE MODIFICATION DE FILTRES SUR LE PGL	

**Formulaire d'identification 4/4**

•Type d'interface :

Descriptif succinct :

Adresse IP source	Protocole	Port(s) source	Adresse IP destination	Port(s) destination	service de communication

Figure 7

Le gestionnaire reçoit une demande d'ouverture de flux qui se présente sous la même forme que la figure 7. Cette demande contient généralement quatre lignes, parfois plus. Chaque ligne correspond à une ACL. Il se charge ensuite de rentrer une par une chaque ligne dans le manager cisco ASDM (figure 8). Cela peut vite devenir fastidieux.

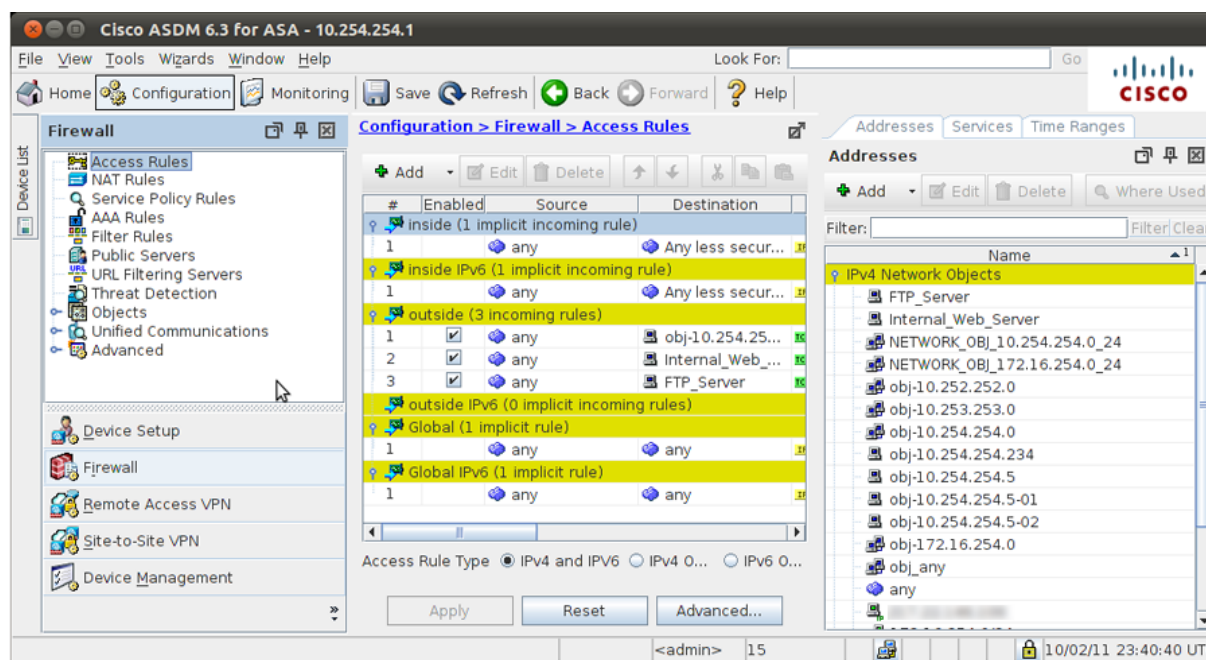


Figure 8

## Avec google/capirca

Le gestionnaire reçoit toujours la même demande d'ouverture de flux (figure 7). Mais cette fois il crée un fichier *nomdelademande.pol* (figure 9), il contient un *header* et autant de *terms* que de lignes « ACLs ». Comme pour le manager le gestionnaire crée aussi des objets mais cette fois dans les fichiers NETWORK.net et SERVICES.svc (figure 10 et 11). Après cela il lance *google/capirca pour TPZ* (nom donné après réalisation du projet et les ajouts de « sync » et « diff »), voir la figure 12. Dit comme cela, cette solution peut paraître plus fastidieuse, dans les faits il n'en est rien. Chaque demande à la même architecture, il suffit donc de faire un copier/coller d'un fichier *nomdelademande.pol* et de modifier les champs adéquats. De plus, comme dit en amont, avec *google/capirca pour TPZ* une bonne partie des doublons sont pris en charges.

```
header {
  comment:: "Demandeur : Tintin"
  comment:: "Realisee par : Cpt. Hadock"
  target:: ciscoasa Eth1 CSG iter:: juniper or
}
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
}
```

Figure 9 nomdelademande.pol

```
# DEMO
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
```

Figure 10 NETWORK.net

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

Figure 11 SERVICES.svc

```
juan@ThinkPad-T440p ~/Bureau/gogs-server/ACLs/capirca $ ./run.sh
Did you fill your policies files ? (yes/no) : yes
SYNC INFO
Wait for Gogs (Go git service) synchronisation ...
Already up-to-date.
[master 1b42450] pre-pull
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 capirca/sync/hook
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), 391 bytes | 0 bytes/s, done.
Writing objects: 100% (4/4), 391 bytes | 0 bytes/s, done.
Total 4 (delta 3), reused 0 (delta 0)
To ssh://git@192.168.1.10:2222/tpz/ACLs.git
408cfdc..1b42450 master -> master
running-config.txt 100% 2908 2.8KB/s 00:00
DIFF INFO
There are some changes (logged via syslog)
access-list Eth1 CSG remark Demandeur : Tintin
access-list Eth1 CSG remark Realisee par : Cpt. Hadock
access-list Eth1 CSG remark NTP
access-list Eth1 CSG extended permit udp 192.168.10.0 255.255.255.0 eq ntp 192.168.20
.0 255.255.255.0 eq ntp log
access-list Eth1 CSG remark SYSLOG
access-list Eth1 CSG extended permit udp host 192.168.1.2 eq syslog 192.168.30.0 255.
255.255.0 range 1024 65535 log
general.asa 100% 397 0.4KB/s 00:00
Enter the commit message: ajout de CSG-ref123.pol
[master 5bc627f] ajout de CSG-ref123.pol
10 files changed, 89 insertions(+), 18 deletions(-)
create mode 100644 capirca/diff/filter/compilation.asa
create mode 100644 capirca/diff/policies/pol/compilation.pol
```

Figure 12 exemple execution run.sh (google/capirca pour TPZ)

## RETEX (RETour d'EXperience)

Certaines difficultés ont pu apparaître au cours du projet, comme par exemple la découverte du Cisco ASA. Ce handicap s'est révélé être au final une réelle opportunité de progression et d'apprentissage du métier d'ingénieur. L'adaptation professionnelle est en effet une caractéristique essentielle de la profession.

L'appropriation du projet a été facilitée par mon parcours (DUT R&T) mais aussi par la présence d'une bonne équipe dans le service. Tout particulièrement, le responsable du groupe « Groupe Réseaux, Commutation, Configuration et Supports » m'a été d'un précieux soutien. La bonne synergie de groupe entre voisins de bureaux a permis de résoudre nombre de questions épineuses (par exemple ce qui pouvait être réalisable tout en respectant les procédures). Avant de tout coder, mon tuteur de stage, Philippe Charron, demandait une spécification technique du besoin. Cela lui permettait de savoir si j'avais bien compris les attentes. De mon côté, cela m'a permis de structurer ma pensée, de définir les objectifs. Dans l'ensemble, ce projet m'a permis de mieux définir le métier d'ingénieur, mais aussi et surtout d'apprendre à endosser ce rôle. Ce projet fut aussi l'occasion de mettre en place les mesures de sécurité apprises durant le cursus, particulièrement en sécurité réseau.

Comme évoqué plus haut, il reste à intégrer le projet au système existant. C'est la façon, courante ici, de faire des maquettes avant intégration. L'entreprise teste la maquette pendant une certaine durée avant son intégration (se rappeler que le PGL est un élément critique). Pour améliorer le projet, il y aurait la possibilité de mettre en place une interface plus intuitive en ajoutant par exemples des graphismes. Mais aussi la possibilité de pouvoir modifier les ACLs directement depuis les fichiers .pol qui correspondent aux demandes.

## Conclusion

Le sujet proposé était l'étude de la possibilité de faire migrer les règles existantes du PGL dans ce logiciel et de la mise en place des mécanismes adéquats permettant la gestion de configuration des demandes de flux utilisateurs

Le stage s'articule donc autour de 2 points majeurs :

- La mise en place d'un mécanisme de génération des règles et de leur gestion en configuration (git ou subversion) à partir des demandes de flux utilisateurs.
- L'adaptation des règles existantes pour les injecter dans le système.

En ce qui concerne le premier point, j'ai choisi et mis en place un serveur git au travers de Gogs. Pour le deuxième point, j'ai développé « sync » et « diff ». Au final comme réponse à l'étude, j'ai une démonstration de ma solution au travers d'une maquette qui reflète de la manière la plus précise possible le système d'intégration cible (OS, réseau ...). En tant que futur ingénieur j'ai eu la responsabilité d'une étude, c'est-à-dire de proposer une solution (démontrée) à un besoin. Pour cela un cadre était défini, celui de la base spatiale avec ses mesures particulières de sécurité à haut niveau. Malgré cela, une grande autonomie m'a été accordée. Toute solution qui répondait aux besoins et qui respectait les attentes du CSG, était accueillie favorablement.

Pour des raisons de sécurité, certaines informations n'ont pu être détaillées dans ce résumé d'activité. On peut citer par exemples le détail des systèmes utilisés, les IPs, la définition des architectures réseaux etc...

J'ai eu la chance durant ce stage de travailler sur une base spatiale, avec tout ce que cela peut signifier. Le cadre de ce projet se situe en effet au cœur de la filière sécurité informatique, la base spatiale étant un site critique à ce niveau. Je suis reconnaissant à Telespazio pour l'expérience professionnelle unique qu'elle m'a permis de vivre en Guyane Française.

## Annexe

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

# 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`

27 juillet 2017

## Table des matières

<b>1</b>	<b>Prérequis</b>	<b>3</b>
<b>2</b>	<b>Ce qu'est Google/CAPIRCA pour TPZ</b>	<b>3</b>
<b>3</b>	<b>Ce que n'est pas Google/CAPIRCA pour TPZ</b>	<b>3</b>
<b>4</b>	<b>Présentation du logiciel</b>	<b>3</b>
<b>5</b>	<b>Installation</b>	<b>4</b>
5.1	Prérequis . . . . .	4
5.1.1	MariaDB . . . . .	4
5.1.2	Python . . . . .	4
5.1.3	SSH . . . . .	4
5.1.4	Autres . . . . .	5
5.2	Procédure . . . . .	5
5.3	Déploiement . . . . .	6
<b>6</b>	<b>Utilisation</b>	<b>6</b>
6.1	Configuration du dépôt local et du client . . . . .	6
6.2	Exemple . . . . .	7
6.3	Recommandations . . . . .	8
<b>7</b>	<b>Rappels succincts sur les commandes Git</b>	<b>9</b>



## 1 Prérequis

Ce manuel n'a pas pour but de décrire l'architecture de Google/CAPIRCA. Ce manuel ne remplace d'aucune manière la documentation de Google/CAPIRCA, il apporte simplement les spécificités 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

Ne pas utiliser GCT pour une utilisation autre que celle décrite. Si le besoin et de créer des ACLs de type iptables, juniper ou encore cisco, veuillez utiliser le Google/CAPIRCA original.

## 4 Présentation du logiciel

Si vous détenez ce présent document c'est que vous avez aussi l'archive GCT.tar.gz. Cette archive contient 2 dossiers. Le premier contient les fichiers sources originaux de Google/CAPIRCA plus les dossiers sync et diff. *sync/* et *diff* contiennent les différents scripts d'adaptation de Capirca au PGL de TPZ. Sync s'occupe de la synchronisation des objets et diff d'éviter les doublons. A coter de *capirca/* il y a *gogsInstall* qui contient tous le nécessaire pour l'installation.

## 5 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.

### 5.1 Prérequis

#### 5.1.1 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')
```

#### 5.1.2 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
```

#### 5.1.3 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
hostname(config)# aaa authentication ssh console LOCAL
hostname(config)# username git password password
hostname(config)# ssh timeout 30
hostname(config)# ssh version 2
```

Cette exemple de configuration a été fait à partir de une 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/known_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
```

#### 5.1.4 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
```

## 5.2 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 -xvzf 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'exécution au script gogsInstall.sh :

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

Exécuter le script en spécifiant le mot de passe root de mariadb comme arg1. Spécifier 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.

## 5.3 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 "GCT". Décompresser GCT.tar.gz, uploader capirca dans GCT. Créer d'autres utilisateurs et les incorporer comme participant en "rw". Récupérer les différentes clés ssh des postes qui cloneront GCT 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 récupérer la wiki dans *CGT*/ et l'ajouter au repository GCT.

## 6 Utilisation

### 6.1 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 en spécifiant de l'ip ou du service. Les demandes au format .pol doivent se trouver dans le dossier *capirca/policies/pol*. Après avoir terminé 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 suppression 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 succinct sur les commandes Git*. Avant tout essai assurez-vous d'avoir bien toutes les librairies python requises d'installées :

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

## 6.2 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

```

### 6.3 Recommandations

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

## 7 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 dépôt distant :

`git remote add origin https://...` : ajout d'un dépôt 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 origin 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 supprimer

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