CC Final - ASR5 Système d'exploitation

2 heures - tout document autorisé

16 mai 2017

Afin d'obtenir tous les points il vous est demandé de justifier vos résultats. Le barème proposé est susceptible d'être modifié lors de la correction. Il n'est présent que pour vous donner une idée du poids relatif des différentes questions.

I ACL ldap (6-7pts)

On rappelle que dans l'annuaire ldap, les informations sont stockées dans un arbre (voir par exemple la figure 1). Les ACLs se composent de 4 parties :

- `<what>` spécifie l'objet cible ;
- `<who>` l'objet auquel le droit est attribué ;
- `<access>` le droit attribué (`none|auth|compare|search|read|write|manage`) ;
- `<control>` l'action à faire ensuite. Si rien n'est précisé, c'est `stop` (arrêter d'évaluer les ACLs), mais il est possible de choisir `continue` (évalue la suite de la même directive) ou `break` (passe à la directive suivante).

Attention, les parties `<who>` et `<what>` ne peuvent être définies que de manière positive en utilisant l'arbre. On peut par exemple définir « tous les employés » c'est à dire tous les descendants du nœud `cn=Users,dc=Adams,dc=Corp`, mais pas « tous les employés sauf l'administrateur ». La partie `<what>` contient une description de l'utilisateur concerné et une liste du ou des champs demandés. On peut définir :

- une liste de champs concernant tout le monde, par exemple « le mot de passe de tout le monde » (`*.userPassword`) ;
- les champs d'un groupe de personnes ou d'une personne, par exemple « toutes les informations des employés » (`"descendants de cn=Users,dc=Adams,dc=Corp".*`) ;
- une liste de champs pour un groupe ou une personne, par exemple les « informations personnelles » (`"descendants de cn=Users,dc=Adams,dc=Corp"."homePostalAddress, homePhone, street"`) ;

Pour simplifier nous allons utiliser les abréviations suivantes :

- *Administrateur* est l'utilisateur `cn=Administrateur,cn=Users,dc=Adams,dc=Corp` ;
- *machines*, descendants de `ou=Computer,dc=Adams,dc=Corp` ;
- *employés*, tous ceux qui sont sous `cn=Users,dc=Adams,dc=Corp` ;
- *RH*, tous ceux qui sont sous `ou=RH,cn=Users,dc=Adams,dc=Corp` ;
- *Ingénieurs*, tous ceux qui sont sous `ou=Ingénieurs,cn=Users,dc=Adams,dc=Corp` ;
- *Consultant*, tous ceux qui sont sous `ou=Consultant,cn=Users,dc=Adams,dc=Corp`
- *Boss*, tous ceux qui sont sous `ou=Boss,cn=Users,dc=Adams,dc=Corp` ;
- *infos employés*, tous les attributs des employés (`"descendants de cn=Users,dc=Adams,dc=Corp".*`) : Name, jpegPhoto, telephoneNumber, displayName, homePostalAddress, homePhone, street, userPassword, samAccountName, uidNumber, mail, ... ;
- *infos perso*, les attributs représentant les informations personnelles `"descendants de cn=Users,dc=Adams,dc=Corp"."homePostalAddress, homePhone, street"`).
- *compte informatique*, les attributs représentant les informations des employés relative au système `"descendants de cn=Users,dc=Adams,dc=Corp"."samaccountname, uidNumber, mail, displayName"`).

Nous avons le tableau de directives suivant :

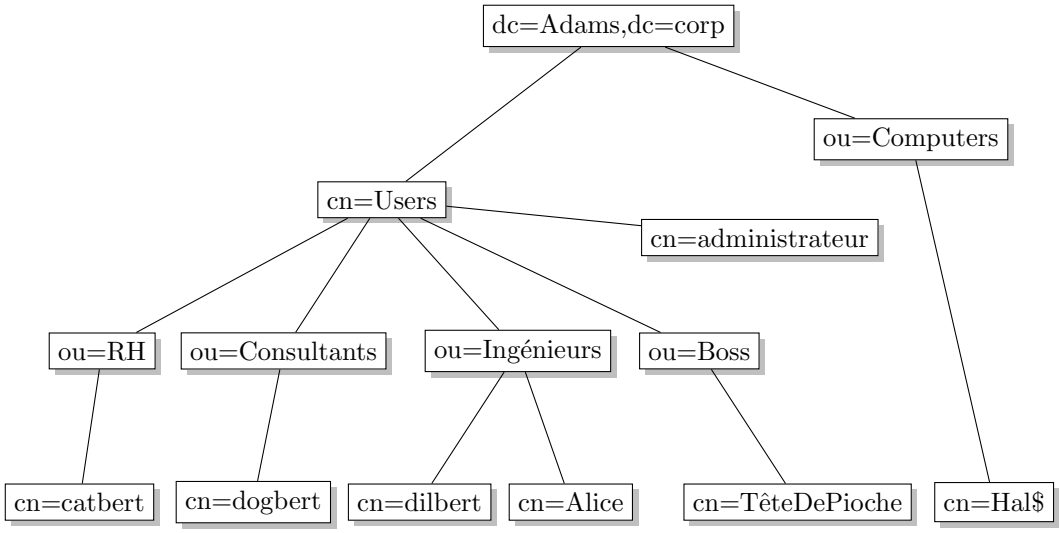

Figure 1 – Organisation de l'annuaire LDAP dans notre exemple.

directive	what	who	access	control
1	*	"Administrateur"	manage	stop
		"Boss"	read	stop
		self	write	stop
		*	none	break
2	telephoneNumber, displayName, Name	Administrateur	manage	stop
		*	read	stop
3	*.userPassword	anonymous	auth	stop
		*	none	stop
4	"infos employés"	"RH"	write	stop
		"consultants"	read	stop
		"machines"	read	stop
		*	none	break
5	*	*	*none*	*stop*

Q.I.1) - Sous la forme d'un tableau, donnez les droits obtenus par Alice, Dogbert, Catbert, Tête de Pioche et Hal$ sur les champs Name, userPassword, uidNumber et homePhone de Alice, Tête de Pioche et Catbert.

Q.I.2) - Ce tableau fait aparaitre plusieurs erreurs :
- On souhaite qu'un utilisateur puisse toujours modifier ses données. Or ce n'est pas le cas.
- On souhaite que les membre du service « RH » puissent modifier toutes les données de tous les employés (sauf le mot de passe). Or ce n'est pas le cas.

Proposez une modification pour corriger ces erreurs.

Q.I.3) - Proposer une modification des règles pour limiter l'accès des données par les machines en générale. Elles ne doivent plus avoir accès en lecture qu'aux données du *compte informatique* (définies plus haut dans le sujet). Et à aucune autre données. Cette modification ne doit pas changer les autres accès. Dans la question suivante la machine `Hal$` aura des droits particulier.

Q.I.4) - Proposez une modification des règles pour régler le cas des informations personnelles. Elle doivent être :
- *managées* par l'administrateur ;
- *lisibles* par les « boss » et la machine `Hal$` (mais seulement celle là) [1] ;
- *modifiables* par les membres du service « RH » et par l'utlisateur lui même ;
- *interdites* aux consultants, aux machines et aux utilisateurs normaux.

1. Dans cette question, la solution dépend de la question précédante. Si vous l'avez faite, votre solution doit prendre en compte la question précédante. Sinon, précisez que c'est le cas et répondez à cette question à partir du tableau de départ.

II Serveur de jeu mobile (13-14pts)

Le but de l'exercice est de programmer les échanges sur le réseau d'un serveur de jeu d'énigmes pour téléphone mobile. Ce serveur propose un jeu comme *4 images un mot* ou *94%*. Le principe de ces jeux est que le serveur propose une image au client/joueur qui envoie une ou plusieurs propositions de réponse. À chaque proposition de réponse, le serveur renvoie un résultat.

Vous ne devez faire que le code du serveur, on suppose que le programme du coté du client est capable d'interpréter les réponses pour faire avancer le joueur. De plus, vous ne devez programmer que la partie qui concerne le réseau. La génération des données à transférer (images, réponses) est cachée dans les fonctions get_next_image et get_reponse qui sont détaillées plus loin. Votre serveur doit juste s'occuper de l'échange des données durant la phase de jeux.

Attention, votre serveur doit permettre à plusieurs clients de jouer en même temps. Donc à chaque nouveau client, le serveur se duplique (`fork`) et le fils prend en charge ce client pendant que le père attend de nouveaux clients.

Le protocole suivant est défini afin que le client soit capable de lire ou d'envoyer correctement les données. Il faut donc le respecter.

- Le client commence par envoyer un identifiant (qu'il a obtenu lors de son installation). Cet identifiant est envoyé sous la forme d'un mot dont la taille est exactement 128 caractères.
- Une fois cet identifiant transmis, le client peut envoyer des commandes :
  - next\n pour demander une nouvelle énigme (une image) ;
  - prop un_mot\n pour proposer la solution un_mot ;
  - quit\n pour terminer la session de jeu.
- À chaque demande du client, le serveur doit envoyer une réponse :
  - à la commande next le serveur répond par deux choses : une taille d'image (un entier codé en chiffres décimaux seul sur une ligne) suivit de l'image elle même, c'est à dire un tableau d'octets de la bonne taille.
  - à la commande prop le serveur doit répondre par une chaine de caractères seule sur une ligne qui est le résultat ;
  - à la commande quit le serveur doit répondre bye\n et couper la socket de discussion avec le client ;
  - en, cas d'erreur, si par exemple le client envoie une commande erronée, le serveur doit envoyer Error\n.

Vous ne devez pas vous préoccuper du jeu lui même. En fait, vous disposez de 2 fonctions dans lesquelles le déroulement du jeu est caché :

- image get_next_image(const vector<char> &id_cli) qui retourne la prochaine image à transférer au client dont l'identifiant est id_cli . image est une classe contenant 2 méthodes utiles :
  - uint image::size() donne la taille de l'image.
  - const char* image::data() les données de l'image.
- string get_reponse(const vector<char> &id_cli, const string &proposition) qui retourne la réponse à envoyer au client id_cli s'il propose la solution proposition. Cette fonction peut renvoyer une chaine vide en cas d'erreur (par exemple si on propose une solution alors qu'aucune image n'a été envoyée). Cette réponse ne contient jamais de retour charriot \n.

Attention, le travail demandé est décomposé en plusieurs questions, mais vous pouvez très bien rendre un code unique qui répond à toutes les questions si cela vous semble plus facile.

Q.II.1) - Donnez le code du serveur qui accueille les clients, se duplique (`fork`) et lance une fonction de traitement des clients.

Q.II.2) - Donnez le code de la fonction de traitement des clients qui assure tout les échanges avec le client jusqu'à la fermeture de la socket.

Q.II.3) - Faites en sorte que le programme ne génère pas de *processus zombis*.

II.1 Petit manuel de fonctions

Pour faire le travail vous pouvez utiliser les fonctions de la librairie standard des sockets en C ou celles de la librairie `socklib` vue en TP.

Pour la librairie standard :

- **int read(int fd, char *buf, size_t count);**
  read() demande à lire au plus count octets sur fd, et à placer le résultat dans le tampon buf.

Retourne le nombre d'octets qui ont été effectivement lus, qui peut être inférieur à la limite donnée pour cause de non-disponibilité (-1 en cas d'erreur, 0 en fin de fichier).

— **int write(int fd, char *buf, size_t count);**
write() écrit count octets depuis le buffer buff. Retourne le nombre d'octets qui ont été effectivement écrits (ou -1 en cas d'erreur).

Attention, vous ne pouvez pas mélanger l'utilisation de la fonction de lecture de la librairie standard (**read**) avec celle de la librairie socklib.

Pour la librairie socklib

— Dans l'espace de nom socklib :
  — int CreeSocketServeur (const std::string &p) Cree une socket d'attente pour le serveur sur le port p.
  — int CreeSocketClient (const std::string &serveur, const std::string &port)Crée une socket de client en se connectant à un serveur.
  — int AcceptConnexion (int s) Accepte un nouveau client qui s'est connecté à la socket d'attente s.

  Ces 3 fonctions renvoient une socket comme résultat.
— Dans la classe BufferedReaderWriter :
  — BufferedReaderWriter (int fd) le constructeur.
  — void **write**(char *data, int offset, int taille) envoie des données situées entre &data[offset] et &data[offset+taille].
  — void **write**(const vector<char> &data) envoie des données du vector data.
  — void **write**(const string &msg) envoie la chaine de caractère msg.
  — vector<char> **read**() lit des octets et les renvoie sous la forme d'un vecteur.
  — vector<char> read_all(int taille) lit taille octets et les renvoie sous la forme d'un vecteur.
  — vector<char> read_until(char end='\n') lit tous les octets arrivés sur la socket jusqu'au prochain caractère end et les renvoie sous la forme d'un tableau de caractères.
  — int read_data(char *buff, int offset, int len) lit len octets exactement et les stocke dans le tableau buff à partir de la case offset.
  — string read_line(char end='\n') lit tous les octets arrivé sur la socket jusqu'au prochain caractère end et les renvoie sous la forme d'une chaine de caractères.

III Feuille de réponse

Numéro de copie (**Ne pas mettre votre nom**) :

			Droits de				
			Alice	Dogbert	Catbert	TdP	Hal$
Droits sur	Alice	Name userPassword uidNumber homePhone					
	TdP	Name userPassword uidNumber homePhone					
	Catbert	Name userPassword uidNumber homePhone					