Théorie des langages — Devoir

(http://dpt-info.u-strasbg.fr/~alain/compilation/)

Alain KETTERLIN (alain@dpt-info.u-strasbg.fr)

Premier semestre 2005-2006

On se propose d'écrire un analyseur syntaxique pour un langage de définition des règles d'un firewall. Le but est d'avoir un ensemble de règles qui testent différentes caractéristiques d'un paquet, afin de décider si le paquet doit être transmis ou rejeté. Dans ce devoir, nous allons nous inspirer de l'outil `iptables` que vous avez étudié l'année dernière. Le but ici n'est pas de faire du *firewalling*, mais de définir un langage qui soit simple à utiliser, et qui permette d'écrire des règles suffisamment subtiles sans être obligé d'utiliser directement les commandes habituelles. (En option, il vous est possible de réfléchir à la traduction de votre langage en une suite de commandes pour `iptables`.)

Une **configuration** est définie par un ensemble de chaînes, chaque chaîne constituant un point de traitement d'un paquet. Chaque chaîne est définie indépendamment.

Une **chaîne** est définie par un test, appliqué à chaque paquet qui la traverse, et qui décide du sort du paquet. (Si vous pensez à `iptables`, notez que nous ignorons ici la notion de table.) On écrira :

nomchaine = *test*;

Un **test** peut être de deux différentes formes. La première forme, la plus simple, est celle d'une décision (c'est-à-dire qu'on ne teste rien du tout, on décide immédiatement). Les seules décisions possibles sont : ACCEPT, DROP, REJECT. La seconde forme, plus complexe, est de la forme :

if *condition* then *action* else *action*

Le mot-clé `then` est optionnel. Par contre, la partie `else ...` est obligatoire (il ne peut pas y avoir de test sans action associée dans tous les cas). Chacune des actions des branches `then` ou `else` peut être soit simple soit complexe : on peut donc « imbriquer » les `if-then-else` à n'importe quel niveau de profondeur.

Une **condition** est une expression logique portant sur les caractéristiques du paquet testé. Il y a deux types de conditions. Les conditions simples portent sur diverses propriétés du paquet (adresses, etc. – voir le tableau 1 ci-dessous). Les conditions complexes sont des expressions logiques utilisant les opérateurs `&&`, `||` et `!` (avec une signification équivalente à celle qu'ils ont en C). La priorité est donnée à la négation (`!`), puis à la conjonction (`&&`), et la disjonction (`||`) possède la plus faible priorité. Les deux opérateurs binaires sont associatifs à gauche (c'est une convention, dans la pratique cela n'a aucune importance). Les expressions peuvent également être entourées de parenthèses.

Voici un exemple de test :

```
if ip source turing.u-strasbg.fr then
    if ( ip destination 130.79.0.0/16
         && tcp destination 1-1023 )
       || ( tcp destination { http, https } ) then
        accept
    else
        reject
else if ip source { 130.79.6/24, 130.79.4.0/23 } then
    accept
else if tcp destination 1-1023 then
    accept
else
    drop
```

Pour ce devoir vous devez, en vous aidant des outils `flex` et `bison`, écrire un programme capable de lire une description de configuration. En particulier, l'analyseur syntaxique doit construire en mémoire une structure de données contenant toutes les informations de la configuration.

Dans un deuxième, vous devez choisir une « extension » du langage, c'est-à-dire que vous devez ajouter quelque chose qui en facilite ou en étende l'usage. Vous devrez expliquer comment vous augmentez votre langage, et comment vous modifiez les différents analyseurs pour prendre en compte votre extension.

Optionnellement, dans un troisième temps, vous *pouvez* écrire une fonction qui transforme une structure de données telle que celle produite par votre analyseur syntaxique en un fichier de configuration pour `iptables` (ou un autre système de votre choix).

Condition	Signification
`ip laddrip`	adresse IP source ou destination
`ip source laddrip`	adresse IP source
`ip destination laddrip`	adresse IP destination
`udp lport`	port UDP source ou destination
`udp source lport`	port UDP source
`udp destination lport`	port UDP destination
`tcp ...`	(idem `udp`)
Exemples de notations pour `laddrip`	
`130.79.7.1`	décimale pointée
`130.79.7.1/255.255.254.0`	décimale masquée
`130.79.7.1/23`	décimale CIDR
`{130.79.6.11, 130.79.6.15}`	liste de sous-réseaux
`turing.u-strasbg.fr`	nom DNS complet
`turing/255.255.254.0`	nom (local) masqué
`turing.u-strasbg.fr/23`	nom + CIDR
`{ ditx27, ditx31, ditx80/24 }`	liste de noms
Exemples de notations pour `lport`	
`80`	forme numérique
`1-1023`	intervalle numérique
`{ 22, 80, 443, 993 }`	liste numérique
`http`	nom *well-known*
`{ ssh, http, https, imaps }`	liste de noms

TAB. 1 – Conditions simples et notations

Ce devoir est à réaliser individuellement. Votre rapport devra contenir les informations suivantes :

1. une description de la grammaire du langage de base ;
2. une indication, pour chaque symbole de la grammaire, du type de sa valeur sémantique ;
3. une description de la structure de données utilisée pour représenter un mémoire la configuration ;
4. une description détaillée ce votre extension du langage (syntaxe, impact sur les différents fichiers, etc.) ;
5. si vous avez effectué la transformation en un ensemble de règles utilisables par un outil « réel », une description formelle des algorithmes utilisés.

Vous joindrez en annexe de votre rapport les fichiers source de votre application.

Le rapport est à rendre sur papier avant le :

13 janvier 2006 à 16^{h}.

Vous pouvez également m'envoyer par courrier électronique une archive de vos fichiers, avant cette date. Je ne pourrais accepter aucun retard : il vaut donc mieux me rendre quelque chose d'incomplet que d'essayer de me rendre votre rapport après la date.