RAPPORT : Analyse de Sudoers

Raphaël LOB

Été 2016

1 Parseur de sudoers

1.1 Analyse du problème

Mon dernier projet pour ce stage a été d'améliorer un parseur de configuration de sudoers.

Le sudoers est un fichier utilisé par le logiciel "sudo" permettant d'effectuer des opérations sous l'identité d'un autre utilisateur. Lors de la mise en place d'un système, l'administrateur peut mal configuré ce fichier. Une erreur de configuration peut permettre à un attaquant d'effectuer une opération sensible avec un niveau de privilège élevé. Cet outil existe donc pour établir un diagnostic du sudoers. Ce diagnostic s'appuie sur de nombreux critères parmi lesquels on trouve (cf. figure 1) :

- la mise en place de restriction par défaut (`noexec`, `env_reset`, etc.) ;
- une liste de commandes indésirables permettant une élévation de privilège ;
- une liste de commandes indésirables sans une certaine configuration ;
- l'accès possible en écriture à un binaire exécuté.

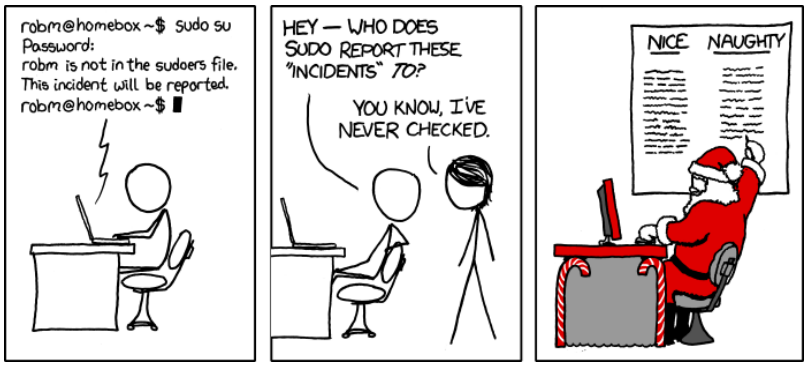

FIGURE 1 : Cette image n'a pas réellement de lien avec l'outil développé

Ce parseur est écrit en ruby et n'est qu'une fonctionnalité d'un outil beaucoup plus puissant permettant de scanner la configuration d'un système unix.

L'un des plus grands problèmes pour le parseur est que la syntaxe d'un fichier sudoers est relativement exotique (cf. listing 1).

```
#Exemple de sudoers
Defaults  syslog=auth
Defaults>root  !set_logname
Defaults:FULLTIMERS !lecture
Defaults:millert !authenticate
Defaults@SERVERS log_year, logfile=/var/log/sudo.log
Defaults!PAGERS  noexec


User_Alias FULLTIMERS = millert, mikef, dowdy
```

```
Runas_Alias OP = root, operator
#Un alias sur plusieurs lignes pour plusieurs host
Host_Alias SPARC = bigtime, eclipse, moet, anchor:\
               SGI = grolsch, dandelion, black:\
               ALPHA = widget, thalamus, foobar:\
               HPPA = boa, nag, python

#Un alias de commande sur plusieurs lignes pour plusieurs commandes
Cmnd_Alias DUMPS = /usr/sbin/dump, /usr/sbin/rdump, /usr/sbin/restore, \
                     /usr/sbin/rrestore, /usr/bin/mt, \
                     sha224:0GomF8mNN3wlDt1HD9XldjJ3SNgpFdbjO1+NsQ== \
                     /home/operator/bin/start_backups
Cmnd_Alias KILL = /usr/bin/kill

root  ALL = (ALL) ALL
%wheel  ALL = (ALL) ALL

FULLTIMERS ALL = NOPASSWD: ALL

jack  CSNETS = ALL

operator ALL = DUMPS, KILL, SHUTDOWN, HALT, REBOOT, PRINTING,\
               sudoedit /etc/printcap, /usr/oper/bin/

pete  HPPA = /usr/bin/passwd [A-Za-z]*, !/usr/bin/passwd root
```

Listing 1: Extrait de fichier sudoers

Il est en effet possible de faire des alias d'utilisateurs, de groupes, de réseaux, de commandes, de coder une commande sur plusieurs lignes.

Lorsque j'ai débuté le projet, un parseur en ruby était en place mais il n'était pas satisfaisant pour permettre un résultat fiable.

J'avais pour objectif d'adapter le code du parseur présent dans le logiciel "sudo" pour permettre une compréhension fidèle du fichier. J'ai alors découvert qu'un outil nommé visudo est contenu dans ce logiciel. Cet outil permet d'éditer correctement la configuration de la machine. Le fichier sudoers étant un fichier vital pour la majorité des utilisateurs, il est nécessaire de prendre des précautions pour son édition. En lisant la documentation lié a cet outil j'ai trouvé deux commandes pertinentes :

1. `visudo --check` qui permet d'indiquer l'intégrité du sudoers et relève certains défaut de configuration ;
2. `visudo --export myname.json` qui permet d'exporter le sudoers dans le format JSON.

Le problème de syntaxe à alors disparu. En effet dès que le fichier est approuvé par `visudo`, il suffit de l'exporter et de parser un fichier JSON avec une bibliothèque externe (JSON en ruby). Grâce à cette avancée, il est alors tout à fait possible pour le parseur de lire une configuration complexe.

Malheureusement cette avancée implique de changer la façon dont le problème est traité et rend donc l'ancien code obsolète en quasi-totalité .

1.2 Mise en place de la solution

J'ai donc récréé totalement un parseur. Celui-ci a toutes les fonctionnalités souhaités, il comprend des configurations complexes et permet un diagnostique détaillé des problèmes de configuration.

Une autre de ces fonctionnalités est de vérifier que chacun des utilisateurs ne peut pas obtenir un accès en écriture sur le binaire qu'il exécute en tant que root. En effet si l'utilisateur à un accès sur une configuration du type :

```
john ALL = /usr/bin/whoami
```

Si john a le droit d'écriture, il peut alors remplacer le binaire en place par un binaire malveillant qu'il exécutera en tant que root. C'est le cas si john a le droit d'écriture sur :

- `/`
- `/usr/`
- `/usr/bin/`
- `/usr/bin/whoami`

Il a donc été nécessaire de faire plus qu'une simple lecture du fichiers sudoers (cf. fig 2).

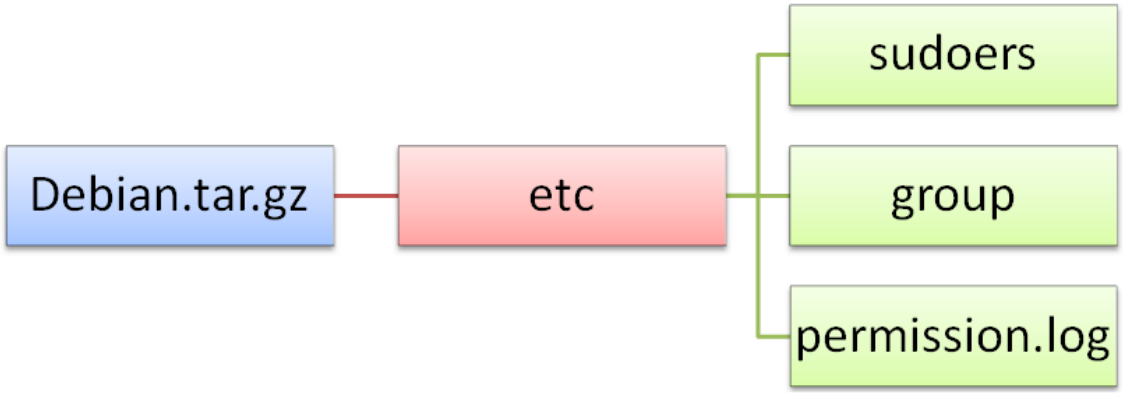

FIGURE 2 : Le script doit également parser les groupes et les permissions

Je n'ai pas eu autant de temps que je le souhaitai pour développer ce projet (1 semaine), il semble fonctionner correctement, je n'ai rencontré aucune erreur lors de son utilisation lors de mon développement.

Je remercie Nicolas pour m'avoir confié ce projet. Malgré le manque de temps, Il a été très enrichissant et m'a permis de progresser en Ruby.

1.3 Bibliographie

- figure 1 : xkcd.org