12 oct 2008

Les chantiers OpenBSD

Catégorie : Administration système Tags : misc

~~Retrouvez cet article dans : **Misc 21**~~

La réputation du projet OpenBSD doit autant au charisme – et au caractère – de son « Benevolent Dictator » Theo de Raadt qu'à l'attachement quasi obsessionnel de ses développeurs à la sécurité et à la qualité (l'une n'allant pas sans l'autre, et vice versa). Dès 1995, date de création du projet, Theo a décidé de privilégier la sécurité et la qualité, sans pour autant négliger la portabilité (OpenBSD 3.7, la dernière version à l'heure où nous écrivons ces lignes, est supportée sur 16 plateformes dont amd64, sparc64, et zaurus). Fort de plus de 70 développeurs dans le monde aujourd'hui, le projet OpenBSD produit une version du système d'exploitation du même nom tous les 6 mois (en mai et en novembre) et les échéances ont toujours été respectées. Et durant ses 10 ans d'existence, le projet n'a pas démenti sa forte orientation sécurité et qualité.

Malgré le nombre relativement faible de développeurs par rapport à des projets libres du même acabit, OpenBSD se donne les moyens nécessaires pour réaliser ses objectifs avec une démarche proactive envers la sécurité. Ainsi, une équipe dédiée de 6 à 12 développeurs se consacre à l'audit permanent du code source. Et tous les développeurs sont sensibilisés à la sécurité et aux principes de développement sécurisé.
En résulte un système embarquant par défaut des mécanismes sécurité (ProPolice/SSP, séparation de privilèges, StackGap, StackGhost, W^X, ld.so randomization…) et qui ne nécessite aucune configuration pour pouvoir en bénéficier.

Même si le projet annonce pompeusement sur son site web qu'il n'a connu qu'« une seule vulnérabilité à distance dans l'installation par défaut, durant plus de 8 ans ! » (qui se contente d'utiliser une machine qui n'a, par défaut, que SSH d'ouvert sur le réseau et quelques autres services peu ou pas utilisés ?), son excellent historique sécurité (les vulnérabilités sont souvent corrigées très rapidement) parle pour lui.
Si l'objectif originel du projet était – et reste – de fournir un système d'exploitation frappé du coin de la sécurité et de la qualité (l'une n'allant pas sans l'autre, rappelons-le), il a rapidement évolué vers la fourniture d'outils de même acabit destinés à une plus large audience que celle des seuls utilisateurs du système d'exploitation OpenBSD.
Ces outils, portables, sont gérés en tant que chantiers au sein du projet OpenBSD; chacun possédant sa « propre » équipe de développement (un développeur OpenBSD pouvant travailler sur plusieurs chantiers), sa section dédiée sur le site web d'OpenBSD, voire son propre site web.

Les chantiers actuels du projet OpenBSD sont les suivants :

- OpenBGPD/OpenOSPFD [1] : implémentations sécurisées des protocoles BGP version 4 et OSPF ;
- Packet Filter (PF) [2] : pare-feu stateful ;
- OpenNTPD [3] : implémentation sécurisée du protocole NTP (Network Time Protocol) version 3 ;
- OpenSSH [4] : implémentation très populaire des protocoles SSHv1 et SSHv2 ;
- OpenCVS [5] : implémentation en cours d'élaboration de CVS (Concurrent Versions System).

Le but de cet article est de présenter les quatre derniers chantiers et leurs apports à la communauté Unix en matière de sécurité, ainsi que les motivations qui ont conduit à les lancer.
Nous verrons ainsi comment OpenSSH et PF, pour citer les plus anciens, ont progressivement conquis leur place au soleil au point de devenir pour l'un (OpenSSH) un standard de fait, pour l'autre (PF) la nouvelle coqueluche des pare-feu, puis comment et pourquoi les plus récents (OpenNTPD et OpenCVS) pourraient bien suivre cette voie.

1. PF : un P comme Pare-feu et un F comme Firewall

Nous ne ferons pas au lecteur l'offense d'un tutoriel sur le filtrage de paquets : nous renvoyons aux excellents [6] et [7] celui désireux d'en apprendre ou d'en réviser les fondements.
Nous allons cependant consacrer un peu de temps à PF ou Packet Filter, l'outil de filtrage réseau qui fut longtemps l'apanage d'OpenBSD jusqu'à sa récente adoption

par le « frère ennemi » FreeBSD, puis NetBSD et DragonFlyBSD (en attendant Mac OS X ?).
A l'instar d'OpenSSH, PF a en effet débordé du cadre OpenBSD, même si son expansion (ou sa contagion ?) reste pour le moment limitée aux systèmes d'exploitation dérivés de la souche BSD.
Cet article n'étant pas un tutoriel sur PF, nous ne ferons qu'en survoler les principales caractéristiques et les points forts, en renvoyant le lecteur désireux d'approfondir le sujet à la documentation officielle [3] disponible en version originale non sous-titrée mais aussi en français.

1.1. PF : il a tout pour plaire

PF se distingue, au premier abord, des autres outils de filtrage par une syntaxe claire, intuitive et aisément compréhensible pour qui maîtrise, ne serait-ce qu'un peu, l'anglais. Ainsi, la lecture – et la compréhension – de la règle suivante :

```
block out on fxp0 from 192.168.0.1 to any
```

A l'instar de ses « concurrents », PF agit directement au niveau du noyau du système d'exploitation. Il applique sur le trafic réseau un filtrage défini par des règles. Ces règles sont généralement contenues dans un fichier de configuration (~~/etc/pf.conf~~) et chargées au démarrage de la machine, mais elles peuvent aussi être dynamiquement activées à l'aide de l'utilitaire pfctl.
Une règle décrit les caractéristiques d'un paquet – protocole utilisé, port source, etc. – et définit l'action associée à chaque paquet qui répondra aux critères ainsi définis. Les règles PF sont appliquées suivant la logique « Last match, first win » : l'action déclenchée est celle définie par la dernière règle applicable à un paquet, les règles étant lues de la première (haut) à la dernière (bas) du fichier qui les contient ou par ordre chronologique d'entrée au clavier.
Ainsi soit de la session TCP allant de 192.168.1.2:14452 et allant à 192.168.2.3:80, ainsi que l'ensemble suivant de règles :

```
1. pass in from any to any
2. pass in proto tcp from 192.168.1.0/24 \
      port >1024 to any port 80
3. block in all
```

Ces trois règles peuvent s'appliquer au paquet. Cependant, c'est la règle n°3 qui sera appliquée car elle est la dernière, même si les critères de la règle n°2 correspondent plus aux caractéristiques du paquet. En conséquence, les règles qui définissent la politique par défaut doivent être les premières du fichier de règles.
On peut cependant modifier ce comportement en utilisant le mot-clé quick. Si un paquet correspond aux critères d'une règle qui utilise ce mot-clé, PF stoppe là son analyse et déclenche l'action définie par la règle, même si dans les règles suivantes il y en a qui s'appliqueraient aussi au paquet.
La présentation de la syntaxe PF ne serait pas complète si nous ne parlions pas des

macros, listes et tableaux. Ces objets désignent différentes variables utilisées dans des règles afin de rendre la lecture du fichier qui les contient encore plus aisée.
Les macros sont tout simplement des variables définies par l'utilisateur. Elles permettent de stocker des adresses IP, des intervalles de ports, des noms d'interface. Elles sont désignées par un nom qui doit être précédé de $ lorsqu'elles sont appelées dans une règle :

```
 batman = "192.168.1.1"
robin = "192.168.2.1"
pass in on fxp0 from $batman to $robin
Les listes, quant à elles et comme leur nom l'indique, servent à stocker plusieurs valeurs :
pass in on fxp0 from { 192.168.1.1, 192.168.2.1 } to any
```

Notons qu'une macro peut être utilisée pour désigner une liste.
Les tables, enfin, sont utilisées pour manipuler de grands nombres d'adresses IP. Leur utilisation par PF est également plus rapide, ce qui améliore ses performances. De plus, leur contenu peut être modifié dynamiquement sans causer d'interruption.

Quelques exemples de tables :

```
 table <bigcave> { 192.168.1.0/24, 192.168.2.0/24 }
table <rfc1918> const \
  { 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }
table <joker> persist { 192.168.6.0/24 }
```

Les attributs ~~const~~ et ~~persist~~ indiquent respectivement que le contenu d'une table ne peut être modifié (const) et que le contenu d'une table doit rester en mémoire même si aucune règle ne l'utilise (~~persist~~).

Anatomie d'un fichier de configuration

Si l'emplacement et le nom du fichier de configuration de PF sont librement modifiables par l'utilisateur, son contenu se doit d'être ordonné en sections qui doivent apparaître dans l'ordre suivant, le fichier commençant à sa première ligne et se terminant... à la dernière, de haut en bas :

- Options générales : elles conditionnent le comportement par défaut du filtrage ainsi que ses performances ;
- Directives de normalisation de trafic ;
- Directives de gestion de la bande passante et des priorités ;
- Règles de traduction d'adresses (NAT) ;
- Règles de filtrage.

Les définitions de macros, de tables et de listes peuvent apparaître en n'importe quel point du fichier : il suffit que ces objets soient définis avant d'être appelés dans une règle. Cependant, une bonne habitude consiste à les placer en tête de fichier.

1.2. PF : il a tout d'un « grand »

La syntaxe claire et limpide de PF n'est pas, loin s'en faut, son seul atout. Sans entrer dans les détails, nous présentons dans ce paragraphe ces petits riens qui font que PF en fait souvent plus que les autres et qui font son succès.

Le filtrage à état

Si vous avez lu [6] en particulier et MISC d'une façon générale, vous n'êtes pas sans savoir que tout pare-feu moderne qui se respecte se doit d'être stateful. PF entre bien sûr dans cette catégorie, sans quoi nous n'aurions pas l'outrecuidance de vous en parler.
Le filtrage à état peut être déclenché règle par règle à l'aide des directives ~~keep state~~, ~~modulate state~~ et ~~synproxy state~~ et s'applique à tous les protocoles.
Dans sa forme la plus simple, le filtrage à état est activé par la directive ~~keep state~~ associée à une règle de type ~~pass~~ :

```
pass in proto tcp from any to any port 80 \
  flags S/SA keep state
```

Pour le protocole TCP, la conservation de l'état ne pose pas de problème particulier puisque le protocole fournit les moyens adéquats : les drapeaux TCP.
Pour le protocole UDP qui est par définition un protocole sans état car déconnecté, PF maintient une table des sessions actives auxquelles sont affectées des durées de vie (time out).
Les messages ICMP, enfin, sont traités différemment selon qu'ils sont associés à une session active ou non. Dans le premier cas, il s'agit généralement de messages d'erreur qui sont acceptés s'ils se rapportent à une session autorisée. Dans le second, il est tout à fait possible d'utiliser la directive ~~keep state~~. Ainsi, la règle suivante active le suivi d'état pour les requêtes ICMP émises depuis le pare-feu :

```
pass out inet proto icmp all icmp-type echoreq \
  keep state
```

Les réponses à ces requêtes seront ainsi acceptées.
Les directives ~~modulate state~~ et ~~synproxy state~~ activent, elles aussi, le suivi d'état mais apportent des fonctionnalités intéressantes de protection contre les usurpations de connexions et contre les attaques par saturation de type SynFlood.
Dans le premier cas ~~– modulate state –~~ les numéros de séquence initiaux (ISN) des paquets TCP sont modifiés par PF pour être réellement aléatoires. Certains systèmes d'exploitation pèchent par la relative prédictabilité de ces numéros, ce qui entrouvre la porte à des attaques fondées sur l'usurpation de session.
Dans le second cas ~~– synproxy state –~~ le pare-feu agit comme un mandataire (proxy) entre le client et le serveur. C'est PF qui prend en charge la totalité des opérations d'établissement d'une session TCP (le fameux « 3-way handshake TCP »). La session n'est transférée au destinataire que lorsque PF a pu lui-même l'établir. L'objectif est de protéger un serveur des attaques par saturation SynFlood : seules les connexions valides sont transmises au serveur. Certes, les mauvaises langues diront toujours que c'est alors le pare-feu qui subit l'attaque. Mais d'une manière générale, un pare-feu résiste beaucoup mieux à ce type de nuisance qu'un serveur d'application.

Signalons enfin que PF fournit de nombreuses options qui permettent de configurer le fonctionnement du filtrage à état de manière très fine. Il est ainsi possible de limiter le nombre de connexions dont PF garde l'état (options max), de paramétrer la durée de vie d'une session (option time-out), et de limiter le nombre de connexions en provenance d'une même source depuis un certain laps de temps. Imaginons que nous souhaitions limiter le nombre de connexions vers un serveur et automatiquement bloquer toutes les connexions provenant d'une source qui a dépassé ce seuil. Nous allons pour cela utiliser :

- une table nommée bad_hosts pour stocker les adresses des malotrus ;
- la directive overload suivie du nom de cette table ;
- la directive flush qui provoquera le nettoyage de la table d'états et l'option global qui appliquera le nettoyage à toutes les connexions provenant de la source ainsi « black listée », même si elles n'ont aucun rapport avec le dépassement du seuil.

La mise en œuvre de cette « poubellisation » automatique repose sur les règles suivantes :

```
 block quick from <bad_hosts>
pass in proto tcp from any to $webserver port www \
  flags S/SA keep state (max-src-conn-rate 100/10, \
  overload <bad_hosts> flush global)
```

L'option max-src-conn-rate 100/10 fixe le seuil à ne pas dépasser : 100 ouvertures de connexions en 10 secondes.
Ces options permettent de contrer ou limiter l'impact des attaques par déni de service (voir [7] pour plus d'informations sur ce type d'attaques).

La traduction d'adresses IP (NAT)

Autre fonctionnalité que l'on est en droit d'attendre d'un pare-feu digne de ce nom : la traduction d'adresses, fonctionnalité qui consiste à modifier au niveau du pare-feu les adresses IP source ou destination d'un paquet, et parfois même le port destination (on parle alors de redirection).
PF fournit trois directives pour cela :

- binat : cette directive applique une traduction bidirectionnelle de type 1 pour 1. A une adresse IP originelle ne correspondra qu'une adresse IP traduite.
- nat : cette directive permet de masquer plusieurs adresses IP derrière une ou plusieurs adresses traduites. Nous avons donc là une traduction de type 1 pour N, avec N >= 1.
- rdr : cette directive effectue une redirection de trafic avec modification de l'adresse destination et éventuellement du port de destination. Cette directive permet de mettre en place des mandataires transparents par exemple.

Quelques exemples d'utilisation de ces directives :

```
# Masquage des machines du réseau batcave derrière l'adresse IP de
# l'interface fxp0
nat on fxp0 from $batcave to any -> fxp0

# Masquage des machines de batcave à l'aide d'adresses IP choisies dans
# un pool de manière alternative
nat on fxp0 from $batcave to any -> \
  { 192.168.3.1, 192.168.3.2, 192.168.3.3 } round-robin

# Traduction bidirectionnelle
binat on fxp0 from 192.168.1.1 to any -> fxp0

# Redirection du trafic HTTP vers un mandataire
rdr on fxp0 inet proto tcp from any to any port 80 -> \
  192.168.2.1 port 3128
```

Notons que ces trois directives activent de manière implicite le suivi d'état sur les connexions concernées.
Maintenant que nous avons survolé les fonctionnalités indispensables de PF, nous allons en présenter les petits « plus » qui font que PF n'est pas juste un autre pare-feu.

1.3. PF : il en fait toujours plus

Si PF ne proposait pas d'autres fonctionnalités que celles précédemment évoquées, le lecteur serait en droit de se demander quel était l'intérêt de lui consacrer tant de place dans ce magazine. Ce paragraphe a donc le double objectif de justifier le choix de PF comme sujet d'article mais surtout d'en présenter les atouts qui ont fait et font son succès.
Nous allons donc aborder les fonctionnalités de normalisation de trafic, de mise en attente et de gestion de bande passante de PF avant de terminer en beauté avec CARP et pfsync.

1.3.1. Normalisation de trafic

La normalisation de trafic est une fonctionnalité de nettoyage : elle permet de rejeter tous les paquets volontairement ou accidentellement mal formés. Elle effectue également les opérations de réassemblage des paquets fragmentés. Dans certains cas, elle permet de modifier certaines caractéristiques des paquets.
Les principaux objectifs poursuivis (et atteints) sont une réduction de la charge du filtrage en évitant de traiter ces paquets inutiles, de déjouer les attaques ou tentatives de contournement fondées sur la fragmentation de paquets et contrer les opérations de prises d'empreintes (fingerprinting).
La normalisation de trafic est activée par la directive scrub que l'on appliquera avant toute autre règle (elle se trouvera donc en tête du fichier de configuration) et si possible sur toutes les directions en appliquant un sage principe de précaution :

```
scrub in all
scrub out all
```

La directive s'applique dès lors au trafic dans les deux sens et à tous les protocoles.

Normalisation IP

Dans le cadre de la normalisation du protocole IP, seront rejetés par exemple :

- les paquets dont la version IP n'est pas conforme ;
- ceux dont la valeur du champ longueur d'en-tête (Header Length) est trop petite ou trop grande ;
- ceux dont la somme de contrôle (checksum) est incorrecte.

Les valeurs de certains champs peuvent aussi être modifiées :

- remise à zéro des champs d'options IP (IP Options) ;
- ajustement de la valeur du champ TTL (Time-To- Live).

Normalisation UDP

Elle s'applique après la normalisation IP. Les paquets dont la somme de contrôle (checksum) est incorrecte ou dont la taille est différente de celle indiquée dans les champs IP sont rejetés.

Normalisation ICMP

Sont rejetés :

- les messages de type Echo Request dont la destination est une adresse de multidiffusion (multicast ou broadcast) ;
- les messages dont la somme de contrôle (checksum) est incorrecte ;
- les messages de type Source Quench.

Normalisation TCP

Les opérations de normalisation effectuées sur les paquets TCP ont une fois encore pour objectif de détecter et corriger les anomalies dans les drapeaux TCP et de rejeter les paquets dont la somme de contrôle est incorrecte.

Le lecteur intéressé ou désireux d'approfondir le sujet trouvera dans [8] de quoi étancher sa soif.

1.3.2. Gestion de bande passante

Depuis la version 3.3 d'OpenBSD, PF incorpore le gestionnaire de files d'attente ALTQ, dont la fonction est de trier les paquets en fonction de la priorité qui leur a été préalablement affectée. Et vous l'aurez deviné : les priorités peuvent être attribuées dans des règles PF.

Il est ainsi possible d'agir sur l'écoulement du trafic réseau et de favoriser des protocoles ou des destinations par rapport à d'autres. Un cas « classique » d'utilisation consiste à attribuer aux protocoles interactifs comme SSH une priorité plus haute que celle de protocoles moins demandeurs de réactivité comme FTP.

La gestion de bande passante s'effectue à l'aide des directives ~~altq~~ et ~~queue~~ qui doivent apparaître dans le fichier de configuration après les règles de normalisation et avant celles de traduction d'adresses.

La première – ~~altq~~ – active la mise en file d'attente, spécifie l'algorithme de gestions des files choisi et crée une file racine d'où découleront toutes les files créées. Sa

syntaxe est la suivante :

```
altq on interface scheduler bandwith bw qlimit qlim \
  tbrsize queue { queue_list }
```

Le paramètre scheduler permet de choisir l'algorithme de gestion des priorités sur la file d'attente. Les algorithmes suivants sont actuellement disponibles : cbq pour une gestion des files par classes et priq pour une gestion par priorité, hfsc pour une gestion hybride associant bande passante et temps de traitement des paquets dans la file.

Chaque algorithme travaille à partir d'une file racine qui possède la totalité de la bande passante disponible.

- CBQ (Class Based Queuing) permet de construire une arborescence de files à partir de cette racine à la manière d'un système de fichiers Unix (à l'exception des liens). Chaque file peut avoir plusieurs sous-files et ainsi de suite. Chaque élément de cette arborescence peut avoir son quota de bande passante et ses priorités.
- PRIQ (Priority Queuing) ne permet pas de construire une arborescence aussi complexe et se contente de gérer des files rattachées à la racine, mais ne pouvant inclure de sous-files. Ces files ne possèdent de surcroît qu'un niveau de priorité et sont traitées en fonction de ce niveau de 0 à 15. Les paquets des files dont le niveau est le plus élevé sont traités en premier.
- HSFC (Hierarchical Fair Service Curve) enfin présente une approche proche de CBQ dans sa capacité à gérer une arborescence de files complexe, mais introduit dans leur traitement la notion de temps de traitement. La gestion des paquets s'effectue donc à la fois en fonction de la bande passante allouée à une file et du délai maximal de traitement des paquets de la file souhaité.

L'explication détaillée du mode de fonctionnement de ces algorithmes ferait exploser le volume de cet article, nous renvoyons donc le lecteur à [9] pour de plus amples explications.
Le paramètre ~~bandwith~~ spécifie la quantité de bande passante allouée à l'algorithme, exprimée en pourcentage ou en bits, kilo-, méga- et gigabits par secondes.
Le paramètre qlimit introduit la notion de taille de la file et indique le nombre maximum de paquets stockés (50 par défaut).
Le paramètre queue enfin donne la liste des files d'attente gérées.

Voici un exemple :

```
altq on fxp0 cbq bandwith 2Mb queue { std, ssh, ftp }
```

Cette règle active la gestion de bande passante sur l'interface fxp0, alloue 2 mégabits par seconde de bande passante à la file racine et crée trois files d'attente. Reste maintenant à gérer ces files d'attente. C'est le rôle de la directive queue dont

la syntaxe est la suivante :

```
queue name on interface bandwith bw priority pri qlimit qlim \
  scheduler sched_options { queue_list }
```

name désigne une file et doit correspondre à un nom préalablement défini dans une règle altq. bandwith désigne la quantité de bande passante attribuée à la file en question, pri sa priorité, notée de 0 à 7 pour cbq et hfsc, de 0 à 15 pour priq. Nous retrouvons ensuite des paramètres identiques à ceux utilisés pour créer la file racine et qui peuvent donc recevoir des valeurs différentes pour chaque sous-file. Il n'est cependant pas possible d'utiliser un algorithme différent de celui choisi pour la racine. Il est par contre possible de renseigner des options pour cet algorithme. Enfin, le paramètre queue_list permet de créer des sous-files dans la file considérée (sauf dans le cas d'une file gérée par priq).

Exemple:

```
 queue std bandwith 50% cbq(default)
queue ssh bandwith 25% { ssh_login, ssh_bulk }
  queue ssh_login bandwith 25% priority 4 cbq(ecn)
  queue ssh_bulk bandwith 75% cbq(ecn)
queue ftp bandwith 500Kb priority 4 cbq(borrow red)
```

Nous retrouvons bien nos trois premiers niveaux de files créées sous la racine – std, ssh et ftp – et nous avons créé deux sous-files sous la file ssh, ssh_login et ssh_bulk, ce qui permet d'affecter une priorité différente, au sein de la file ssh, aux paquets de connexion et aux paquets de session. Notons que les deux sous-files

ssh_login et ssh_bulk se partagent respectivement 25% et 75% des 25% de bande passante allouée à leur file mère, et non de la totalité de la bande passante.

Enfin, il faut maintenant diriger le trafic vers les files d'attente ad hoc. Cela se fait le plus simplement du monde dans des règles PF :

```
 pass in on fxp0 from any to any port 21 queue ftp
pass out on fxp0 from any to any port 22 \
  queue(ssh_login, ssh_bulk)
```

Et cela fonctionne aussi en mode filtrage à état :

```
pass in on fxp0 proto tcp from any to any port 22 \
  flags S/SA keep state queue ssh
```

1.3.3. Mise en haute disponibilité de pare-feu avec CARP et pfsync

La capacité à mettre des pare-feu en redondance est peut-être l'un des atouts majeurs de PF car cela répond à un besoin de plus en plus fréquemment émis par

les entreprises dès que la continuité de service est stratégique.
La redondance du service de filtrage réseau est assurée par au moins deux pare-feu dont l'un est actif et l'autre passif, prêt à prendre la main en cas de défaillance du premier.
Sous OpenBSD, on utilise pour mettre en œuvre cette redondance, outre PF, le protocole CARP et l'utilitaire pfsync. CARP assure la fonction de bascule automatique d'un pare-feu vers l'autre en cas de panne (mais ce protocole pourrait tout aussi bien assurer cette même fonctionnalité entre deux serveurs web) et pfsync assure la synchronisation des tables internes (suivi de l'état des connexions actives, de la traduction d'adresses, etc.) de chaque pare-feu afin qu'en cas de bascule, le trafic ne soit pas trop perturbé.

Le protocole CARP
CARP (Common Address Redundancy Protocol) est une alternative sécurisée mais surtout libre de tout droit et brevet aux protocoles HSRP et VRRP. CARP permet le partage d'une ressource commune, en l'occurrence une adresse IP, par les membres d'un groupe de redondance hébergés sur un même segment réseau.
A chaque instant, un seul membre – appelé actif – du groupe possède l'adresse commune et la conserve tant qu'il est disponible. Les autres membres sont inactifs mais prêts à prendre le relais. En cas de panne du membre actif, un nouvel actif est élu parmi les membres passifs disponibles et s'approprie l'adresse commune. CARP spécifie le mode d'élection d'un membre passif ainsi que les méthodes utilisées au sein du groupe pour tester l'état – disponible ou non – de chaque membre.

Création d'un groupe de redondance
La première étape de la mise en redondance d'un groupe de pare-feu passe par la création d'un groupe CARP.
Cela consiste à créer sur chaque membre du groupe une interface réseau virtuelle de type CARP :

```
ifconfig carpN create
```

Puis à la configurer :

```
ifconfig carpN vhid vhid [pass password] [carpdev carpdev] [advbase
advbase] [advskew advskew] [state state] ipaddress mask
```

~~carpN~~ est le nom de l'interface virtuelle, N étant un entier qui désigne le numéro de l'interface. Exemple : carp10.
~~vhid~~ est l'identifiant unique de la machine dans le groupe et prend une valeur comprise entre 1 et 255.
Un mot de passe unique connu de tous les membres du groupe de redondance peut être utilisé pour l'authentification.
~~advbase~~ indique l'intervalle exprimé en secondes entre chaque annonce faite par un membre. Par défaut, chaque membre s'annonce toutes les secondes.
L'option ~~state~~ est utilisée pour forcer l'état d'un membre du groupe (init, backup, master).

Enfin, ~~ipadress~~ et ~~mask~~ désignent respectivement l'adresse partagée et son masque.

La mise en place de ce mécanisme dans un groupe de redondance assure la bascule en cas de défaillance du membre actif. Rappelons que CARP peut être utilisé pour redonder n'importe quel type de service.
Pour assurer, dans le cas du filtrage, une bascule d'un pare-feu actif vers un passif sans (trop de) perte de données, il faut :

- que les données statiques à commencer par la configuration des deux pare-feu soient bien sûr identiques ;
- que les données dynamiques, c'est-à-dire les tables utilisées pour gérer les sessions et les traductions d'adresses entre autres, soient présentes sur les deux équipements dans un état aussi proche l'un de l'autre.

Cette dernière opération est appelée synchronisation et elle est mise en œuvre par pfsync.

pfsync
A l'instar de CARP, la mise en œuvre de pfsync passe par la création d'une interface réseau virtuelle sur laquelle seront émises les modifications des tables internes de PF.
Il faut donc, là encore, configurer cette interface sur les membres du groupe de redondance qui doivent être synchroniser. Cette création s'effectue une fois de plus par l'intermédiaire de la commande ~~ifconfig~~ :

```
ifconfig pfsyncN syncdev iface [syncpeer IP-dst]
```

~~pfsyncN~~ désigne l'interface virtuelle et N son numéro, iface désigne l'interface physique qui sert de support aux échanges de données et ~~IP-dst~~ peut optionnellement désigner l'adresse IP d'une machine dans le cas où le groupe de redondance n'est constitué que de deux machines. Par défaut, les échanges se font en multidiffusion.
Pour conclure ce paragraphe, voici comment un exemple (inspiré de l'excellente FAQ PF) de configuration de la redondance (CARP) et de la synchronisation (pfsync) entre deux pare-feu :

Pare-feu 1 (master) :

```
 ! enable preemption and group interface failover
# sysctl -w net.inet.carp.preempt=1

! configure pfsync
# ifconfig em1 10.10.10.1 netmask 255.255.255.0
# ifconfig pfsync0 syncdev em1
# ifconfig pfsync0 up

! configure CARP on the LAN side
# ifconfig carp1 create
# ifconfig carp1 vhid 1 carpdev em0 password lanpasswd state master 172.16.0.100 255.255.255.0

! configure CARP on the WAN/Internet side
# ifconfig carp2 create
# ifconfig carp2 vhid 2 carpdev em2 password netpasswd state master 192.0.2.100 255.255.255.0
```

Pare-feu 2 (backup) :

```
! enable preemption and group interface failover
# sysctl -w net.inet.carp.preempt=1

! configure pfsync
# ifconfig em1 10.10.10.2 netmask 255.255.255.0
# ifconfig pfsync0 syncdev em1
# ifconfig pfsync0 up

! configure CARP on the LAN side
# ifconfig carp1 create
# ifconfig carp1 vhid 1 carpdev em0 password lanpasswd advskew 128 state backup 172.16.0.100 255.255.255.0

! configure CARP on the WAN/Internet side
# ifconfig carp2 create
# ifconfig carp2 vhid 2 carpdev em2 password netpasswd advskew state backup 128 192.0.2.100 255.255.255.0
```

Les options ~~advskew~~ que l'on trouve sur le pare-feu 2 lui donnent une priorité d'annonce moins importante que celle du pare-feu 1. Cela est dû au fait que nous faisons du pare-feu 2 l'élément passif du groupe.

1.4. Conclusion

Il eut été illusoire de présenter PF dans son exhaustivité en aussi peu de place. Nous espérons vous avoir au pire fait comprendre l'engouement croissant pour cet outil au sein de la communauté BSD et au mieux donné l'envie d'en savoir plus, voire, rêvons un peu, de l'essayer et de l'adopter.
Vous trouverez sur le site officiel OpenBSD toute la documentation nécessaire pour une bonne prise en main de PF, depuis ses fonctions de base jusqu'aux plus évoluées, en passant par celles que nous n'avons pas pu présenter faute de place (filtrage avec authentification, filtrage basé sur le type de système d'exploitation des hôtes, etc.).

2. OpenSSH

OpenSSH est certainement le plus connu des chantiers OpenBSD. Et nous sommes convaincus que la majorité écrasante, si ce n'est la totalité, des lecteurs de MISC l'a utilisé au moins une fois dans sa vie. C'est pourquoi nous ne vous ferons pas l'affront de vous montrer comment utiliser OpenSSH. Nous allons plutôt parler un peu d'histoire et de certaines fonctionnalités méconnues.

2.1. Un peu d'histoire et plus encore

OpenSSH est très probablement l'implémentation des protocoles SSHv1 et SSHv2 la plus utilisée dans le monde... du moins si on en croit des statistiques d'utilisation collectées par le projet OpenBSD à l'aide d'un mécanisme [10] qui ressemble fortement à celui utilisé par Netcraft pour établir des statistiques d'utilisation de

différents serveurs web.
Revenons un peu en arrière et plus exactement en 1995. Cette année-là, Tatu Ylönen crée SSHv1, la première version du protocole SSH et développe SSH1, la première implémentation de ce protocole. SSH1 fut mis à disposition gratuitement à partir de juillet de cette même année et le succès fut immédiat puisque fin 1995, on estime à 20 000 le nombre de ses utilisateurs. Les demandes de support pleuvant, Tatu crée SSH Communications Security en décembre 1995 pour assurer le développement et le support commercial.
A chaque nouvelle version, SSH1 devint de moins en moins libre jusqu'à devenir complètement propriétaire. SSH2, la nouvelle implémentation correspondant à la version 2 du protocole SSH, ne connut même pas le passage par la case libre. Bien que SSH2 soit bien plus sûre que SSH1, les utilisateurs continuèrent à utiliser les versions libres de cette première implémentation.
En 1999, Björn Gronvall développa OSSH, une implémentation libre du protocole SSHv1, à partir de SSH1 1.2.12, la dernière version libre sortie des cuisines de Tatu. Quelques mois après, OpenSSH fut rapidement créée par le projet OpenBSD et intégrée par défaut à OpenBSD 2.6.
Le développeur principal d'OpenSSH est Markus Friedl qui mit en place le support des protocoles SSHv1 (dans ses deux sous-versions 1.3 et 1.5) et SSHv2. D'autres développeurs y participent, parmi lesquels on peut citer Theo de Raadt et Bob Beck.
OpenSSH connut un succès rapide. Plusieurs développeurs ont rejoint l'équipe initiale pour porter cet outil vers d'autres systèmes d'exploitation tels que Solaris, FreeBSD, Mac OS X ou GNU/Linux. Ce portage est assuré par le biais d'une surcouche de portabilité ajoutée à la version dédiée à OpenSSH. La version portable s'appelle tout naturellement… Portable OpenSSH.
OpenSSH et Portable OpenSSH sont, à quelques exceptions près, fonctionnellement identiques. Elles supportent différents mécanismes d'authentification tels que les mots de passe Unix/Linux, les bi-clés RSA/DSA, Kerberos V et S/Key. Portable OpenSSH supporte en plus PAM sur les systèmes d'exploitation possédant une implémentation PAM. La différence majeure entre ces deux versions est au niveau sécurité.
Portable OpenSSH ne bénéficie pas de l'audit de code permanent effectué par l'équipe de développeurs OpenBSD faute de ressources. Si cela peut vous rassurer, les développeurs de la version portable sont très sensibilisés à la sécurité et dans la mesure du possible, ils utilisent des techniques de développement sécurisé. Dans la suite de cet article, nous utiliserons donc OpenSSH comme dénomination commune pour les deux versions.

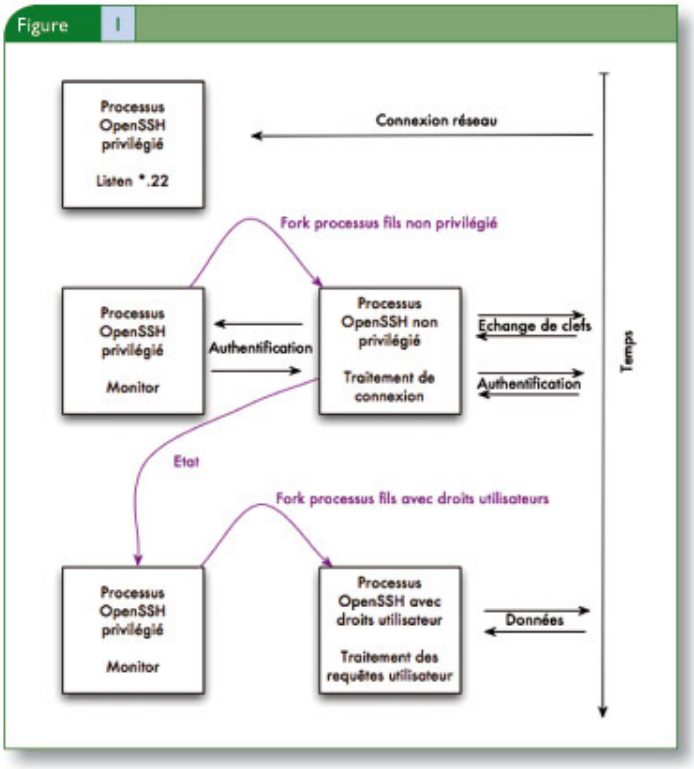
2.2. La séparation de privilèges

Pour assurer une meilleure sécurité contre les attaques visant OpenSSH, ses développeurs ont introduit dès la version 3.4 un mécanisme de séparation de privilèges également utilisé par d'autres logiciels bien connus des lecteurs de MISC tels que Postfix et vsftpd.
Mécanisme éprouvé, la séparation de privilèges est utilisée par les autres chantiers du projet OpenBSD et même certains programmes intégrés au système

d'exploitation tels que tcpdump... dans la mesure où ils interagissent avec le réseau :

```
 ttyp3:saad@beach> sudo ps ax | grep "\[priv\]"
30459 ??  Is      0:00.00 syslogd: [priv] (syslogd)
11853 ??  Is      0:00.00 pflogd: [priv] (pflogd)
32011 ??  Is      0:00.00 named: [priv] (named)
23776 ??  Is      0:00.05 ntpd: [priv] (ntpd)
25108 ??  Is      0:00.04 sshd: saad [priv] (sshd)
 6697 p1  I       0:00.02 tcpdump: [priv] (tcpdump)
 4921 C0- I       0:00.00 dhclient: fxp0 [priv] (dhclient)
```

Comme l'illustre la figure 1, la séparation des privilèges consiste à distinguer les opérations privilégiées, réduites au strict minimum, des autres opérations. Les traitements sont ainsi effectués par des processus différents avec des droits différents.



L'utilisation de la séparation de privilèges est fortement recommandée, ce d'autant plus qu'elle est facile à mettre en œuvre. Il suffit pour cela de :

- Créer un utilisateur dédié non privilégié et lui octroyer un répertoire vide tel que ~~/var/empty~~ qui fera office de cage logicielle de type chroot ;
- Compiler OpenSSH avec le support de la séparation de privilèges en passant à ~~./configure~~ les paramètres ~~--with-privsep-user=utilisateur --with-privsep-path=CHEMIN;~~

- S'assurer que la directive de configuration UsePrivilegeSeparation du fichier de configuration du serveur (sshd_config) est positionnée à yes et redémarrer le serveur sshd.

2.3. OpenSSH, un pare-feu ?

OpenSSH est tellement riche fonctionnellement que certaines de ses fonctionnalités ressemblent étrangement à celles d'un pare-feu. Par exemple, OpenSSH permet de définir quels utilisateurs et quels groupes Unix ont le droit d'accès ou pas au serveur sshd.
Pour autoriser l'accès à un nombre limité d'utilisateurs, il suffit d'utiliser la directive de configuration AllowUsers suivie d'une liste comme le montre l'exemple suivant :

```
AllowUsers zeus ares@1.2.3.4
AllowUsers athena@*.olympos.com orphu@beach.ilium.com
```

Ainsi seuls les utilisateurs suivants pourront utiliser le service SSH :

- zeus quel que soit sa provenance ;
- ares à partir de la machine d'adresse IP 1.2.3.4 ;
- athena à partir de n'importe quelle machine dont le FQDN se termine par olympos.com ;
- et enfin orphu à partir de la machine dont le FQDN est beach.ilium.com.

Pour faire l'inverse et empêcher certains utilisateurs de se connecter, on utilise la directive DenyUsers comme suit :

```
DenyUsers aphrodite hera
DenyUsers diomedes@blackship.greeks.com
DenyUsers ajax@doo?.troy.com
```

Ainsi :

- aphrodite et hera n'auront pas le droit de se connecter ;
- diomedes ne pourra pas se connecter à partir de blackship.greeks.com, mais s'il change de machine, il pourra utiliser le service ;
- ajax ne pourra pas se connecter à partir de toute machine dont le FQDN est de la forme doo?.troy.com, le caractère ? ayant la même signification que dans une expression régulière classique.

Avant que vous ne posiez la question, si un utilisateur est listé dans AllowUsers et DenyUsers, c'est le plus restrictif des deux qui l'emporte.
Le contrôle d'accès pour les groupes Unix fonctionne de manière similaire avec les directives AllowGroups et DenyGroups à l'exception près que ces deux directives n'acceptent pas l'utilisation d'adresses IP ou de FQDN à la suite des noms de groupes.
En plus de ces jeux de directives, OpenSSH peut faire du contrôle d'accès par clé lorsque l'authentification par bi-clés RSA/DSA est utilisée. Pour cela, il suffit de

préfixer chaque clé publique déclarée dans ~~authorized_keys~~ avec des directives spécifiant les contrôles imposés :

- Utilisez la directive ~~from~~ pour limiter les machines depuis lesquelles la clé privée associée à une clé publique donnée peut être utilisée pour s'authentifier sur le serveur OpenSSH. En cas de compromission d'une clé privée, cette directive permet de gagner un peu de temps pour sauver les meubles.
- La directive ~~command~~ vous permet de limiter l'accès SSH à l'exécution d'une commande.
- Utilisez ~~no-port-forwarding~~ et ~~no-x11-forwarding~~ pour interdire la création de tunnels TCP et X11 respectivement (vous pouvez toujours interdire globalement au niveau de ~~sshd_config~~ la création de tunnels TCP ou X11 avec les directives ~~AllowTcpForwarding~~ et ~~X11Forwarding~~ respectivement positionnées à ~~no~~).
- Interdisez l'attribution d'un pseudo-terminal à l'aide de la directive ~~no-pty~~.

Pour illustrer ces directives, prenons un cas commun où nous voulons restreindre l'accès d'un opérateur de sauvegarde s'authentifiant par clé à la seule exécution d'un programme (~~/usr/local/bin/bkp.sh~~) destiné à préparer la sauvegarde du jour. L'opérateur se connecte uniquement depuis la machine ~~bk.tinybits.com~~. Pour imposer ces restrictions, il suffira alors de préfixer sa clé privée comme suit :

```
from="bk.tinybits.com",command="/usr/local/bin/bkp.sh",\
no-x11-forwarding,no-port-forwarding ssh-dss AAAA...suite de la clé publique...
```

Bien sûr, ~~authorized_keys~~ ne devra pas être modifiable par l'utilisateur (accès en lecture uniquement) et ~~bkp.sh~~ ne doit pas lui permettre de s'échapper vers un shell par exemple.

2.4. Serveurs et vérifications DNS

Lorsqu'on se connecte pour la première fois à un serveur SSH, on a besoin de s'assurer que c'est bien le bon serveur et pas une machine qui usurpe l'identité de ce dernier. OpenSSH vous présente un message d'alerte avec l'empreinte de la clé publique RSA ou DSA du serveur :

```
ttyp4:saad@ark> ssh -o "VerifyHostKeyDNS yes" beach.ilium.com
The authenticity of host 'saje3.docisland.org (172.16.4.35)' can't be
established.
RSA key fingerprint is 5e:79:6d:72:7a:0d:db:28:af:d4:4d:42:74:1f:97:75.

Are you sure you want to continue connecting (yes/no)?
```

Pour vérifier si cette clé est la bonne, il faut soit disposer d'une liste des empreintes valides pour tous les serveurs sur lesquels on sera amené à se connecter ou téléphoner au BOFH (Bastard Operator From Hell ou administrateur en bon français) du serveur pour vérifier avec lui l'empreinte. Les deux solutions sont loin d'être satisfaisantes ce qui conduit trop souvent les utilisateurs à répondre yes sans vérification et en croisant les doigts.
OpenSSH offre un moyen bien plus pratique basé sur le DNS [11] pour effectuer cette vérification de manière automatique dans la mesure où l’on possède une

version récente d'OpenSSH côté client (3.4 minimum) et de BIND (9.x) côté serveur de noms responsable de la zone DNS dans laquelle est déclarée le serveur SSH. Pour utiliser ce mécanisme, il suffit d'ajouter des enregistrements de type SSHFP (BIND 9.3.0 et supérieur) ou TXT44 (versions ultérieures de BIND 9) au niveau du DNS. Ces enregistrements peuvent être générés à l'aide de l'utilitaire ssh-keygen (qui vous permet aussi de générer vos bi-clés RSA/DSA) :

```
ttyp3:saad@beach> sudo ssh-keygen -r <nomDNS> \
-f ssh_host_[rd]sa_key.pub
```

Ce qui apparaîtra dans le DNS comme suit (ici BIND 9.3.1) :

```
beach.ilium.com. IN A     172.16.4.35
                 IN SSHFP 2 1 23b9dc1d87b4e0233404607023e89b0027ca8aa3
                 IN SSHFP 1 1 63689daa30d31567c3f9d131a48ce53cf09e828f
```

Le premier enregistrement SSHFP correspond à l'empreinte de la clé publique DSA de beach.ilium.com tandis que le second correspond à l'empreinte de la clé publique RSA de ce même serveur. Notez que ce mécanisme n'est valable que pour le protocole SSHv2. D'ailleurs, si vous utilisez encore SSHv1, le silence est d'or ! Côté client, il suffira alors de se connecter en utilisant l'option -o "VerifyHostKeyDNS yes" de la commande ssh ou de renseigner cette directive dans le fichier de configuration global (ssh_config) ou individuel ($HOME/.ssh/config) du client :

```
ttyp4:saad@ark> ssh -o "VerifyHostKeyDNS yes" beach.ilium.com
The authenticity of host 'saje3.docisland.org (172.16.4.35)' can't be
established.
RSA key fingerprint is 5e:79:6d:72:7a:0d:db:28:af:d4:4d:42:74:1f:97:75.
Matching host key fingerprint found in DNS.
Are you sure you want to continue connecting (yes/no)?
```

Maintenant, on a le message : « Matching host key fingerprint found in DNS ».

2.5. Le multiplexage

Introduit par la version 3.9, le multiplexage est intéressant dans la mesure où il permet d'éviter de renégocier des clés de session à chaque connexion (SSH, SCP ou SFTP) entre un client et un serveur donnés. Une fois une première connexion ouverte entre client et serveur, celle-ci est utilisée pour véhiculer les autres connexions. Elle est alors appelée connexion principale. Voyons comment mettre en œuvre cette fonctionnalité qui permet d'économiser des ressources parfois précieuses.
Le multiplexage se configure côté client à l'aide des directives ControlMaster et ControlPath (ou les options équivalentes -M et -S). La première permet de spécifier quelle connexion sera utilisée pour véhiculer toutes les autres. Il suffit alors de la positionner à yes lors de l'établissement de la connexion principale. La seconde directive permet de spécifier une socket de contrôle utilisée côté client pour les communications inter-connexions.
Nous devons donc établir une connexion principale comme suit :

```
ttyp2:saad@ark> ssh -o "ControlMaster yes" -o
"ControlPath /tmp/mysshsock" beach.ilium.com
```

La commande suivante, plus courte, aura le même effet :

```
ttyp2:saad@ark> ssh -M -S /tmp/mysshsock beach.ilium.com
```

Ceci aura pour effet de créer une socket de contrôle sur le client :

```
ttyp8:saad@ark> ls -l /tmp/mysshsock
srw-------   1 saad  wheel  0 Aug 14 02:20 /tmp/mysshsock
```

Côté serveur, on peut voir qu'on a une seule connexion établie par l'utilisateur saad :

```
ttyp0:saad@beach> ps ax | grep sshd | grep saad
27014 ??  Is      0:00.04 sshd: saad [priv] (sshd)
 7517 ??  I       0:00.01 sshd: saad@ttyp0 (sshd)
```

Maintenant, établissons une seconde connexion SSH :

```
ttyp2:saad@ark> ssh -o "ControlPath /tmp/mysshsock" beach.ilium.com
```

La commande suivante, plus courte, aura le même effet :

```
ttyp2:saad@ark> ssh -S /tmp/mysshsock beach.ilium.com
```

Vous remarquerez que nous n'avons pas passé la directive ControlMaster qui est par défaut à no dans la première commande ni l'option -M dans la seconde commande (rappelons que ces commandes sont équivalentes). La connexion que nous venons d'établir n'est donc pas une connexion principale. Si vous aviez exécuté cette commande, vous verriez que la connexion est quasi instantanée, et pour cause ! Nous utilisons la connexion principale :

```
ttyp0:saad@beach> ps ax | grep sshd | grep saad
27014 ??  Is      0:00.04 sshd: saad [priv] (sshd)
 7517 ??  I       0:00.01 sshd: saad@ttyp0 (sshd)
```

Si maintenant, l'envie nous prenait de faire un transfert SFTP :

```
ttyp8:saad@ark> sftp -o "ControlPath /tmp/mysshsock" beach.ilium.com
Connecting to beach.ilium.com...
sftp> get IMG_1396_1.jpg
```

Et toujours côté serveur :

```
ttyp0:saad@beach> ps ax | grep sshd | grep saad
27014 ??  Is      0:00.04 sshd: saad [priv] (sshd)
 7517 ??  I       0:00.01 sshd: saad@ttyp0 (sshd)
```

Et ceci marche de manière similaire pour scp. A ce titre, saviez-vous que scp et sftp n'ont aucune connaissance du protocole SSH et n'ont de ce fait aucune fonctionnalité de sécurité ?... Ils se contentent de faire appel à ssh et sshd.

2.6. Il reste tant de fonctionnalités à découvrir !

Dommage que nous n'ayons pas assez de place. Nous aurions pu vous apprendre comment utiliser l'Agent Forwarding pour rebondir d'un serveur à un autre en utilisant le serveur intermédiaire comme mandataire d'authentification ou comment ouvrir des portes dérobées, même à travers des mandataires HTTP/HTTPS pour accéder à votre réseau domestique depuis le système d'information de votre entreprise ou comment accéder à l'Intranet d'entreprise depuis chez vous alors que vous n'avez aucune connectivité directe avec ce dernier. Mais nous en avons peut-être trop dit ! Certains lecteurs vont penser qu'OpenSSH dessert la sécurité au lieu de la servir. Ce n'est pas entièrement faux.
Pour le lecteur désireux de creuser un peu plus le sujet ou de basculer du côté obscur, les pages du manuel d'OpenSSH ainsi que le livre SSH : The Definitive Guide, 2nd Edition [12] sont d'excellentes sources d'information.

3. OpenNTPD

3.1. A la recherche du temps… perdu !

La synchronisation des horloges via NTP (Network Time Protocol) est nécessaire pour disposer d'une piste d'audit digne de ce nom.
Malheureusement l'implémentation de référence sous Unix/Linux créée par le projet NTP est loin d'être simple à mettre en œuvre. La lecture du code source n'est pas vraiment ce qu'on appelle une promenade de santé, même pour un développeur chevronné. Ce qui laisse au mieux planer un doute quant à la sécurité de ce produit. De plus, l'empreinte du programme sur le système d'exploitation est loin d'être négligeable. Enfin, la précision d'horloge apportée par ntpd est trop fine pour la plupart des besoins.
En partant de ces constats, Henning Brauer, développeur au sein du projet OpenBSD, a créé OpenNTPD en 2004. OpenNTPD est une implémentation de la version 3 du protocole NTP, décrite par la RFC 1305 [13]. Le but avoué de ce chantier est de créer un programme à faible empreinte (votre vieux VAX n'est pas prêt de prendre sa retraite) et destiné à répondre à la plupart des besoins des utilisateurs sans chercher à couvrir tous les cas d'utilisation ou fonctionnalités obscures. Et ceci sans négliger la sécurité bien entendu.
Ici aussi la séparation de privilèges, le contrôle strict des entrées/sorties et tous les autres principes de développement sécurisé adoptés par le projet OpenBSD sont mis en œuvre pour assurer une sécurité optimale.

3.2. A bas ntpd ! vive…ntpd !

La première version fonctionnelle fut intégrée de base à OpenBSD 3.6, sortie en novembre 2004. OpenNTPD est une alternative à l'implémentation du projet NTP quand la précision à la micro-seconde près n'est pas nécessaire (la précision

d'OpenNTPD est autour de 50 ms) ou lorsque l'authentification par clés partagées n'est pas requise ; fonctionnalité laissée de côté dans les versions actuelles d'OpenNTPD pour favoriser la légèreté et la facilité de mise en œuvre. En fait OpenNTPD est composé du programme ntpd (le même nom utilisé par le projet NTP) qui fait aussi bien serveur que client (en un peu plus de 2500 lignes de code) et d'un fichier de configuration (~~ntpd.conf~~), le tout marchant à la manière « set up and forget ».
Face à ces arguments de choc, une version portable a été créée pour les autres systèmes d'exploitation à la manière d'OpenSSH. Cette version est désormais disponible dans les systèmes d'installation d'applications tierces de Debian GNU/Linux, Gentoo, FreeBSD, DragonFlyBSD et même Mac OS X (via DarwinPorts).
Le fichier de configuration livré par défaut est fonctionnel dans la mesure où votre machine a un accès Internet :

```
 ttyp3:saad@beach> cat /etc/ntpd.conf
# $OpenBSD: ntpd.conf,v 1.7 2004/07/20 17:38:35 henning Exp $
# sample ntpd configuration file, see ntpd.conf(5)

# Addresses to listen on (ntpd does not listen by default)
#listen on *

# sync to a single server
#server ntp.example.org

# use a random selection of 8 public stratum 2 servers
# see http://twiki.ntp.org/bin/view/Servers/NTPPoolServers
servers pool.ntp.org
```

Comme vous pouvez le voir, la seule ligne active est la dernière. Elle indique à ntpd de choisir de manière aléatoire 8 serveurs publics de ~~stratum 2. pool.ntp.org~~ est en réalité un enregistrement DNS qui fournit plusieurs adresses de serveurs publics en round-robin. Si un ou plusieurs serveurs retenus par ntpd venaient à être indisponibles, ntpd les remplacerait dynamiquement par d'autres de façon à en avoir toujours 8.
Mais on peut choisir de se synchroniser à un seul ou plusieurs serveurs à l'aide de la directive server (on peut en mettre autant qu'on le souhaite... dans des limites raisonnables bien entendu !) :

```
 server ntp1.drool.org
server ntp2.drool.org
```

La directive ~~listen~~ permet de redistribuer le temps à d'autres services ntpd sur une ou plusieurs adresses configurées sur le serveur de temps.
Par exemple :

```
 listen 172.16.4.35
listen 192.68.4.5
```

Enfin, pour terminer ce tour d'horizon d'OpenNTPD, précisons que cette implémentation offre la possibilité d'ajuster l'horloge au démarrage si le décalage est supérieur à 180 s, sans recourir à un programme externe tel que ntpdate ou

rdate.
De plus, tous les ajustements d'horloge supérieurs à 128 ms sont journalisés via syslog.

4. OpenCVS

4.1. Introduction

OpenCVS est un chantier très récent du projet OpenBSD, auquel contribuent principalement Jean-François Brousseau, Xavier Santolaria, Joris Vink, et Jason McIntyre. Le but de ce chantier est de créer une alternative sécurisée à CVS (Concurrent Versions System), ou plus exactement de GNU CVS, un système de gestion de versions de fichiers (ou Version Control Systems – VCS – en Anglais) très répandu, particulièrement dans le monde du Logiciel libre.
Après avoir administré une base de données (repository en anglais) sous GNU CVS, Jean-François s'est frotté aux bogues bien connus de ce logiciel ainsi qu'à ses limitations. En 2003, il a commencé à réfléchir au lancement d'OpenCVS, une alternative aussi compatible que possible avec GNU CVS mais sans les bogues et les limitations de ce dernier. Le travail sur OpenCVS n'a commencé que fin 2004, quelques jours après la conférence BSDCan 2004.
Cette année, un certain nombre de vulnérabilités majeures ont été découvertes dans GNU CVS. Sous l'impulsion de Theo de Raadt, Ryan McBride et d'autres développeurs, Jean-François a commencé à travailler sur OpenCVS, les autres développeurs précités l'ont rejoint ultérieurement.
Bien que la partie cliente soit pratiquement opérationnelle, OpenCVS n'est pas encore officiellement disponible. Le projet OpenBSD espère compléter rapidement la partie serveur et intégrer le tout à OpenBSD 3.9, version prévue pour Mai 2006.

4.2. Sécurité

OpenCVS vise à être le plus sûr possible en utilisant des techniques éprouvées par le projet OpenBSD telles que le contrôle strict des entrées/sorties ainsi que l'utilisation de la séparation de privilèges. Encore une fois, rappelons au lecteur que tous les chantiers OpenBSD utilisent ce type de techniques.
Là où le bât blesse un peu, c'est au niveau de la compatibilité avec GNU CVS. Celle-ci n'est pas garantie lorsqu'il s'agit des fonctionnalités peu ou pas sécurisées de GNU CVS. Par exemple, il a été décidé de ne pas supporter la méthode d'accès « pserver » à la base de données CVS, méthode qui offre une authentification à l'aide d'un mot de passe transmis en clair sur le réseau. De plus, OpenCVS offrira un moyen puissant mais facile à utiliser pour contrôler l'accès à la base de données CVS, à la branche et au fichier près. Cette fonctionnalité est assurée par le biais de listes de contrôle d'accès.
La syntaxe de configuration de ces listes sera aussi proche que possible de celle déjà fournie par PF. Elle devrait ressembler à quelque chose comme :

```
# Repository-modifying commands
rw_cmds = { admin, commit, import, rtag, tag }

# Users with read-only access
ro_users = { user1, user3 }

# UBC hacker
ubc_hacker = { pedro }

# Example rules

allow quick all from $ubc_hacker on /cvs/src tag OPENBSD-UBC

block quick $rw_cmds from $ro_users on /cvs/src \
  allow all from any on /cvs/src
```

Un des problèmes de GNU CVS est lié aux permissions octroyées sur la base de données CVS. En effet, chaque développeur devra avoir un accès en écriture à la base de données pour modifier des fichiers via CVS. Malheureusement, si un développeur a un accès local au serveur CVS (et c'est bien souvent le cas !) il pourra modifier n'importe quel fichier sans passer par CVS et sans qu'aucun autre développeur n'en soit informé.
OpenCVS n'aura pas ce problème. Le binaire serveur contrôlera tous les accès en lecture/écriture et effectuera les opérations sur la base de données sans que les développeurs ne possèdent d'accès directement en lecture/écriture.
Enfin, et en plus des protections sécurité fournies par défaut avec le système d'exploitation, la version d'OpenCVS intégrée à OpenBSD bénéficiera de l'audit de code effectué régulièrement sur tout le code source géré par le projet OpenBSD... comme pour tous les autres chantiers.

4.3. Portabilité

La portabilité sera gérée de la même manière que pour les autres chantiers du projet tels qu'OpenSSH ou OpenNTPD : en rajoutant une surcouche de portabilité à la version d'OpenCVS destinée à OpenBSD.

4.4. Pourquoi réinventer la roue ?

Certains détracteurs du projet OpenBSD pensent que ce dernier n'a de cesse de réinventer la roue et de réécrire des outils existants sans raison valable. Sans vouloir déclencher une guerre de religion, contentons-nous de dire que dans le cas précis d'OpenCVS, le code de GNU CVS a été soigneusement étudié. Il est apparu alors nécessaire de réécrire le code dans un esprit sécurité et qualité, fidèle au projet OpenBSD, au lieu d'essayer de poser des rustines ici et là qui, au mieux, n'amélioreront que légèrement la sécurité.
Mais pourquoi avoir choisi de redévelopper un outil CVS plutôt que de participer à l'une des alternatives plus récentes telles que Subversion ou Arch ? Le problème de ces alternatives récentes est justement leur jeunesse. Passer d'un VCS à un autre est loin d'être aisé. Outre les problèmes de compétences et de capitalisation, il y a aussi

des problèmes liés au changement même de la base de données. En effet, celle-ci contient toute la vie d'un projet et les outils de transition proposés sont souvent imparfaits. Et il n'est pas envisageable pour bien des projets de développement de perdre ne serait-ce qu'un iota de l'historique des changements accumulés durant des mois, voire des années !
Enfin, signalons que ces alternatives offrent beaucoup, voire trop de fonctionnalités et que les lecteurs de MISC, sensibilisés à la sécurité, savent en quoi ça se traduit (Besoin d'un coup de pouce ? Pensez surface d'attaque).

4.5. Conclusion

Le chantier OpenCVS est très prometteur. Il apportera aux projets de développement un meilleur contrôle sur la base de données CVS et une meilleure sécurité tout en restant le plus compatible possible avec GNU CVS. Ainsi seuls les administrateurs de bases de données CVS devront apprendre les nouvelles fonctionnalités sécurité d'OpenCVS. Et lorsqu'il sera prêt, OpenBSD sera bien évidemment le premier projet à l'adopter.

5. Conclusion générale

Ainsi s'achève notre tour d'horizon de quatre chantiers du projet OpenBSD. Pour certains, nous avons abordé des fonctionnalités majeures (PF) ou méconnues (OpenSSH). Pour d'autres (OpenNTPD, OpenCVS), nous avons tenté de vous donner une idée précise sur les outils fournis. Nous espérons vous avoir donné envie d'utiliser ces outils écrits dans un esprit sécurité et qualité. Vous n'êtes même pas obligés d'installer OpenBSD pour les utiliser même si ce système d'exploitation a un intérêt indéniable du point de vue de la sécurité, la fiabilité et la facilité d'utilisation.
Avec tout ça, on a – presque – failli oublier de vous dire que le projet OpenBSD est aussi le roi du textile dans le monde du Logiciel libre et des CD psychédéliques fournis avec des auto-collants époustouflants (et nous exagérons à peine !). Ces objets de désir et de culte, que vous pouvez trouver sur le web [14], aident le projet à vivre et les développeurs à boire… de la bière. Alors n'hésitez pas à contribuer à la bonne cause.

Notes

- [1] http://www.openbgpd.org/fr/
- [2] http://www.openbsd.org/faq/pf/fr/
- [3] http://www.openntpd.org/fr/
- [4] http://www.openssh.org/fr/
- [5] http://www.opencvs.org/fr/
- [6] Hors-Série n°12 et 13 de Linux Magazine « Le Firewall, votre meilleur ennemi »
- [7] E. Zwicky, S. Cooper, D. Chapman, O'Reilly, Building Internet Firewalls, 2nd Edition.

- [8] M. Handley, V. Paxson, C. Kreibich, Network Intrusion Detection : Evasion, Traffic Normalization and End-to- End Protocol Semantics.
- [9] FAQ PF : Packet Queuing and prioritization.
- [10] http://www.openssh.com/usage/fr/index.html
- [11] http://www.ietf.org/internet-drafts/draft-ietf-secsh-dns-05.txt
- [12] http://www.oreilly.com/catalog/sshtdg2/index.html
- [13] http://www.faqs.org/rfcs/rfc1305.html
- [14] https://https.openbsd.org/cgi-bin/order.eu

Lien

- Site OpenBSD : http://www.openbsd.org/fr/

~~Retrouvez cet article dans : **Misc 21**~~

Posté par (La rédaction) | Signature : Guillaume Arcas, Saâd Kadhi | Article paru dans misc

Laissez une réponse

Vous devez avoir ouvert une session pour écrire un commentaire.

« Précédent Aller au contenu »
Identifiez-vous
Inscription
S'abonner à UNIX Garden

- **Articles de 1ère page**

- FUSE, développez vos systèmes de fichiers dans l'espace utilisateur
- SVG – Dessin vectoriel dynamique (1/2)
- Le langage Ada : un peu d'assembleur
- Placement des contrôles dans une fenêtre en C++ à l'aide de wxWidgets
- Entrées/Sorties simples sur USB
- Développement web avancé avec AJAX
- Quelques techniques d'optimisation en C
- Supervision avec OCS Inventory NG et GLPI
- MISC N°41 - Janvier/Février 2009 - Chez votre marchand de journaux
- Le nouveau modèle objet de PHP5

Actuellement en kiosque :

Catégories

- Administration réseau
- Administration système
- Agenda-Interview
- Audio-vidéo
- Bureautique
- Comprendre
- Distribution
- Embarqué
- Environnement de bureau
- Graphisme
- Jeux
- Matériel
- News
- Programmation
- Réfléchir
- Sécurité
- Utilitaires
- Web

Articles secondaires

- 30/10/2008
  Google Gears : les services de Google offline

  Lancé à l'occasion du Google Developer Day 2007 (le 31 mai dernier), Google Gears est une extension open source pour Firefox et Internet Explorer permettant de continuer à accéder à des services et applications Google, même si l'on est déconnecté....

  Voir l'article...

7/8/2008
Trois questions à...

Alexis Nikichine, développeur chez IDM, la société qui a conçu l'interface et le moteur de recherche de l'EHM....

Voir l'article...
11/7/2008
Protéger une page avec un mot de passe

En général, le problème n'est pas de protéger une page, mais de protéger le répertoire qui la contient. Avec Apache, vous pouvez mettre un fichier ~~.htaccess~~ dans le répertoire à protéger....

Voir l'article...
6/7/2008
hypermail : Conversion mbox vers HTML

Comment conserver tous vos échanges de mails, ou du moins, tous vos mails reçus depuis des années ? mbox, maildir, texte... les formats ne manquent pas. ...

Voir l'article...
6/7/2008
iozone3 : Benchmark de disque

En fonction de l'utilisation de votre système, et dans bien des cas, les performances des disques et des systèmes de fichiers sont très importantes....

Voir l'article...
1/7/2008
Augmentez le trafic sur votre blog !

Google Blog Search (http://blogsearch.google.fr/) est un moteur de recherche consacré aux blogs, l'un des nombreux services proposés par la célèbre firme californienne....

Voir l'article...

- **GNU/Linux Magazine**

-
  - Lancement des demi-finales Prologin le 24 janvier
  - GNU/Linux Magazine N°112 - Janvier 2009 - Chez votre marchand de journaux
  - Édito : GNU/Linux Magazine 112
  - Les Éditions Diamond adhèrent à l'APRIL !
  - Nouvelle campagne d'adhésion de l'APRIL !

- **GNU/Linux Pratique**

  - Linux Pratique N°51 - Janvier/Février 2009 - Chez votre marchand de journaux
  - Édito : Linux Pratique N°51
  - Linux Pratique HS N°16 - Janvier/Février 2009 - Chez votre marchand de journaux
  - Édito : Linux Pratique HS N°16
  - Linux Pratique HS 16 - Communiqué de presse

- **MISC Magazine**

  - MISC N°41 : La cybercriminalité ...ou quand le net se met au crime organisé - Janvier/Février 2009 - Chez votre marchand de journaux
  - Édito : Misc 41
  - MISC 41 - Communiqué de presse
  - Les Éditions Diamond adhèrent à l'APRIL !
  - Misc HS 2 : Cartes à puce, Découvrez leurs fonctionnalités et leurs limites - Novembre/Décembre 2008 - Chez votre marchand de journaux