RFC 7288 : Reflections On Host Firewalls

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 2 juillet 2014

Date de publication du RFC : Juin 2014

https://www.bortzmeyer.org/7288.html

Aujourd'hui, on met des pare-feux partout et leur présence est en général considérée, sans aucune réflexion supplémentaire, comme indispensable. Il existe des pare-feux qui se branchent sur le réseau et protègent ainsi toutes les machines du réseau ("*network firewall*"). Et il existe des pare-feux sur les machines terminales <https://www.bortzmeyer.org/terminal-host.html>, des "*host firewalls*", qui font l'objet de ce RFC. En présence d'un tel pare-feu, on peut avoir une application qui tourne, et donc consomme des ressources, mais ne peut pas communiquer avec l'extérieur puisqu'elle est bloquée par le pare-feu. Le même résultat de sécurité pourrait être atteint en ne faisant pas tourner l'application, ou en la configurant pour limiter ses possibilités. Cette approche ne serait-elle pas meilleure ?

Le RFC 2979 [1] discutait des pare-feux sur l'Internet et des propriétés qu'on attendait d'eux (et les RFC 4949 et RFC 4948 discutaient de la terminologie). Le RFC 2979 pointait notamment le risque de « faux positif », lorsqu'un pare-feu bloque accidentellement une communication légitime. Les vendeurs de pare-feux oublient toujours de mentionner ce risque. Pourtant, nombreux sont les pare-feux qui bloquent stupidement des connexions parfaitement légitimes, par exemple parce que les paquets IP contiennent des options <https://www.bortzmeyer.org/options-interdites.html>, ou des en-têtes IPv6 (cf. RFC 7045). Le RFC 4924 revient sur cette question des pare-feux en notant qu'ils sont largement responsables de l'ossification de l'Internet, et que de plus en plus d'applications consacrent beaucoup d'effort à contourner ces obstacles. En outre, les pare-feux contribuent souvent à un faux sentiment de sécurité, qui peut mener à baisser sa garde sur les machines supposées « protégées par le pare-feu » (section 6 de notre RFC). Enfin, la section 1.1 du RFC rappelle que le NAT n'est pas un vrai pare-feu <https://www.bortzmeyer.org/nat-et-securite.html>.

Le problème des pare-feux est d'autant plus aigu que la réponse classique à la question de l'utilité d'un pare-feu est « bloquer le trafic non souhaité », mais que cette réponse ne dit pas « non souhaité par qui ». Par l'utilisateur final ? Par son employeur ? Par l'administrateur réseau ? Par le

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc2979.txt

développeur ? Leurs intérêts sont souvent en conflit et on ne peut pas espérer un consensus simple entre tous ces acteurs (le RFC reprend l'excellent terme de « *"tussle"* <https://www.bortzmeyer.org/tussle-cyberspace.html> »). Pas étonnant qu'on voit souvent des « courses aux armements » entre ceux qui bloquent et ceux qui contournent le blocage.

Le pare-feu typique est souvent configuré par des règles *"allow"*/*"deny"* où on indique que tel type de trafic est autorisé ou interdit (section 2 du RFC). Voici par exemple des règles Shorewall <https://www.bortzmeyer.org/filtrage-avec-shorewall.html> où tout est interdit par défaut mais où on accepte quelques services :

```
# SMTP
ACCEPT               net       fw       tcp      25
ACCEPT               net       fw       tcp      587
# IMAP, IMAPS et POPS
ACCEPT               net        fw      tcp      143
ACCEPT               net        fw      tcp      993
ACCEPT               net        fw      tcp      995
# Nagios Remote Execution (NRPE)
ACCEPT               net:192.0.2.187  fw  TCP     5767
# SNMP
ACCEPT               net:192.0.2.187,203.0.113.170   fw  udp   161
```

Pour un pare-feu sur une machine terminale, l'utilisateur n'a pas le droit de modifier ces règles sauf dans le cas (fréquent avec les machines personnelles) où l'utilisateur est en même temps administrateur système. Par contre, des applications peuvent configurer le pare-feu, pour les systèmes où celui-ci fournit une API aux applications, ou bien s'il existe un protocole standard de changement de la configuration du pare-feu comme UPnP ou son successeur potentiel PCP (RFC 6887). Ces règles sont classées par notre RFC en deux catégories : réduction de la surface d'attaque (empêcher l'application de faire des choses que le développeur ne voulait pas qu'elle fasse) et politique de sécurité (empêcher l'application de faire des choses que l'administrateur système ne voulait pas qu'elle fasse).

La section 3 décrit la première catégorie, la réduction de la surface d'attaque. On peut se demander si cette catégorie est utile : après tout, si on ne veut pas qu'une application, par exemple, détruise des fichiers en réponse à une requête venue par le réseau, le mieux est de ne pas mettre en œuvre cette fonction. On sera alors bien plus en sécurité. Sauf qu'on n'a pas toujours le luxe d'avoir uniquement des applications bien conçues et bien écrites. Souvent, le développeur a fait des impasses (ou, en termes plus savants, a pris de la dette technique) et réécrire l'application serait trop coûteux. C'est par exemple le cas de beaucoup d'applications Web programmées à la va-vite sans souci de sécurité, en PHP ou VB.NET. Coller un pare-feu devant l'application est en général moins coûteux et plus efficace que de se plonger dans le plat de spaghettis.

Il y a aussi des cas où la dette prise par le développeur (reporter le problème à plus tard...) était justifiée par un manque de support de la part du système. Par exemple, une règle aussi simple et courante que « communication uniquement avec le lien local » est difficile à mettre en œuvre de façon portable, sur Unix, dans tous les langages de programmation. (Notez au passage que n'autoriser que les adresses IP source du même réseau que celui du serveur n'est pas suffisant, en raison de la possibilité d'usurpation d'adresse. Pour UDP, il faudrait aussi tester le TTL, cf. RFC 5082.) Il y a aussi le cas où la machine se déplace d'un réseau sûr à un réseau non-sûr. Le développeur est donc tenté de déléguer la mise en œuvre de cette politique à un pare-feu.

Même chose avec des politiques complexes, comme de ne communiquer que sur les liens les moins coûteux, sachant que l'application n'a pas forcément connaissance des coûts (sur Android, cette politique est en général purement binaire : on considère que la 3G est coûteuse et le Wifi gratuit, ce qui n'est

pas toujours vrai.) Avoir une API permettant à l'application d'exprimer ses souhaits, que le système d'exploitation traduirait en règles de pare-feu concrètes, simplifierait beaucoup les choses.

Quelles sont donc les solutions pour l'utilisateur, qui se retrouve avec une application « imparfaite », pleine de failles de sécurité ? D'abord, réparer le logiciel en corrigeant ces failles. Les partisans des pare-feux feront évidemment remarquer qu'il est plus rapide et plus efficace d'utiliser un pare-feu : déboguer un logiciel mal écrit est long et coûteux et on n'est jamais sûr d'avoir bouché tous les trous de sécurité. Il y a d'ailleurs des cas où réparer le logiciel est quasi-impossible : pas les sources pour le faire ou bien pas de mécanisme de mise à jour pour les engins qui font tourner ce logiciel. En général, un changement des règles du pare-feu peut être déployé plus facilement qu'un "*patch*".

Autre idée, ne pas utiliser le logiciel. Solution brutale mais qui résout tout. Notez que l'idée n'est pas si violente qu'elle en a l'air : certains systèmes font tourner par défaut des logiciels qui ne devraient pas tourner. Arrêter ce logiciel inutile est une solution réaliste.

Enfin, il y a le pare-feu, qui va protéger contre le logiciel mal fait. Le logiciel tournera toujours, consommant des ressources (personnellement, je ne trouve pas l'argument très convaincant : si le logiciel est juste en attente d'événements, il ne consomme pas grand'chose). Et, surtout, si un utilisateur tente de se servir du logiciel, il aura des surprises, car beaucoup de programmes ne sont pas conçus de manière robuste : ils réagissent mal quand, par exemple, certains services réseau ne leur sont pas accessibles. Parfois, cela aggrave la consommation de ressources (par exemple, le logiciel, n'ayant pas compris qu'il était bloqué par le pare-feu, réessaie en boucle). Autre problème avec les pare-feux : ils sont souvent surbloquants. Par exemple, ils bloquent tout ce qu'il ne connaissent pas et gênent donc considérablement l'innovation. Certains analysent les paquets de manière très sommaire (oubliant par exemple certaines fonctions avancées du protocole) et prennent donc de mauvaises décisions. C'est ainsi qu'ICMP est souvent bloqué, menant à des problèmes de découverte de la MTU du chemin (RFC 4890 et RFC 2979).

Deuxième catégorie de règles dans un pare-feu, celles qui visent à faire respecter une politique de sécurité. L'administrateur a décidé qu'on ne regarderait pas YouTube, les règles bloquent YouTube. Cette catégorie peut en théorie être mise en œuvre à trois endroits. D'abord, dans les applications elles-mêmes. Ça ne passe évidemment pas à l'échelle : il faudrait ajouter du code à toutes les applications (faites un peu une liste du nombre de clients HTTP sur votre machine, par exemple ; rappelez vous qu'il n'y a pas que les navigateurs Web) **et** les configurer. En pratique, il y aura toujours une application oubliée. Cette solution n'est donc pas réaliste.

Deuxième endroit possible, le pare-feu. Outre les problèmes discutés plus haut (application qui réagit mal au blocage de certains services), cela mène facilement à une course aux armements : comme les désirs de l'utilisateur final et de l'administrateur du pare-feu ne coïncident pas forcément, les applications, à la demande des utilisateurs, vont évoluer pour contourner le filtrage, qui va devenir de plus en plus intrusif et ainsi de suite jusqu'à ce que tout le monde, pris dans cette course, oublie le but originel de la politique de sécurité. Comme le disait la section 2.1 du RFC 4924, le filtrage sans le consentement (au moins passif) de l'utilisateur est difficile, et va mener à des solutions compliquées, coûteuses et fragiles, violant largement les bons principes d'architecture réseau (par exemple en faisant tout passer au-dessus de HTTP, seul protocole autorisé).

Reste le troisième endroit, qui a plutôt les faveurs du RFC : faire des politiques de sécurité un service auquel s'inscrivent les applications. L'application utilise une bibliothèque (comme les "*TCP wrappers*") ou bien parle un protocole réseau (comme PCP, RFC 6887), décrit ce qu'elle veut, et apprend si c'est autorisé ou pas, de manière propre. Cela nécessite évidemment des standards (comme PCP, qui est pour l'instant très peu déployé, et qui est de toute façon sans doute insuffisant pour gérer toutes les politiques possibles) car on n'imagine pas chaque application ajouter du code pour l'API de chaque marque de pare-feu.

Cette technique est très originale par rapport à l'approche habituelle sur les pare-feux mais c'est celle recommandée par notre RFC. Pour le résumer en une phrase, on fait trop travailler les pare-feux et pas assez les applications.