Quelques pensées de Bernstein sur la sécurité...

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 22 janvier 2010

https://www.bortzmeyer.org/qmail-secu-10ans.html

En 2007 est paru l'article « "*Some thoughts on security after ten years of qmail 1.0*" », de Dan Bernstein, sur la sécurité des logiciels. Comme toujours avec cet auteur, l'article mélange bonnes idées, mauvaises idées et franche mauvaise foi.

Bernstein n'a pas changé sur un point, la malhonnêteté intellectuelle. Ainsi, en discutant de la sécurité des MTA, il tape uniquement sur sendmail, qui n'est plus guère utilisé sur les nouveaux sites, et ne dit pas un mot de Postfix, Exim ou Courier. Et pour cause, cela démolirait sa thèse comme quoi qmail est le seul MTA sûr...

D'autre part Bernstein se vante lourdement et tout le temps du défi de sécurité qu'il avait lancé pour qmail en mars 1997 <http://cr.yp.to/qmail/guarantee.html> : le principe est d'offrir une récompense monétaire à celui qui trouvera une faille de sécurité dans qmail. La récompense attend toujours qu'on la réclame. Mais ce genre de défis ne vaut pas grand'chose. D'une part, l'absence de signalement peut indiquer l'absence d'intérêt et pas l'absence de bogue (c'est courant pour les logiciels peu utilisés <https://www.bortzmeyer.org/securite-logiciels-peu-utilises.html>). Mais, surtout, ce genre de défis est en général soumis à des règles fixées par l'auteur et soigneusement conçues pour qu'on ne puisse jamais s'y conformer. Ainsi, le défi de sécurité qmail exclut les attaques par déni de service, de loin les plus fréquentes. Cela me rappelle un contrat de maintenance d'une entreprise privée pour s'occuper d'un serveur Windows où le contrat excluait les dommages liés aux virus. Exclure les conséquences des virus de l'administration d'une machine Windows, c'est comme exclure les dégâts des eaux du contrat de maintenance d'un sous-marin...

Mais est-il possible d'inclure les attaques par déni de service dans les conditions d'un tel défi ? Ce n'est pas une question facile : il existe deux sortes d'attaques par déni de service, celles qui reposent sur la force brute (on a une ligne rapide et une grosse machine et on écrase la victime sous le poids, par exemple en envoyant des millions de paquets) et celles qui reposent sur l'**amplification**, où un trafic faible pour l'attaquant peut faire beaucoup de dégâts. Par exemple, la faille "*dynamic update*" de BIND en juillet 2009 <https://www.kb.cert.org/vuls/id/725188> était dans cette catégorie : un seul paquet DNS pouvait stopper complètement le serveur. S'il n'y a pas de méthode générale

pour faire face aux attaques par force brute, en revanche, on peut éviter les amplifications, les attaques où l'assaillant peut obtenir des résultats sans disposer des dizaines de milliers de machines d'un botnet. Il n'y a donc pas de raison d'exclure **toutes** les attaques par déni de service. Mais on comprend mieux cette décision de Bernstein quand on voit qu'une telle faille de sécurité a déjà été trouvée pour qmail <http://insecure.org/sploits/qmail.DOS.rcpt.html>... et refusée par l'auteur. Bref, les défis de sécurité lancés par l'auteur d'un logiciel ne servent pas beaucoup, il rejetera toujours les bogues trouvées. (Une liste des bogues de qmail est disponible <http://www.dt.e-technik.uni-dortmund.de/~ma/qmail-bugs.html>, plusieurs affectant la sécurité.)

Maintenant, quelles sont les idées intéressantes dans ce papier ? Parmi les principales, il y a l'idée que les bogues dans le code sont inévitables mais ne devraient pas forcément être des failles de sécurité. Si le code, par exemple le code d'analyse des en-têtes d'un message, a une bogue, elle va empêcher l'analyse correcte desdits en-têtes mais elle ne devrait pas pouvoir mener à une attaque du système. Bernstein ne les cite pas, mais c'est une des motivations derrière les langages purs, comme Haskell, où le code, par défaut, ne peut pas avoir d'effets de bord. L'idée de base est que le code d'analyse des en-têtes devrait avoir des entrées et des sorties limitées et **aucun** accès au système (ce qui est très difficile à assurer dans un langage comme C).

Bernstein s'attaque aussi à certaines idées fausses en sécurité comme le fait de corriger les bogues (alertes de sécurité accompagnées de patches) : cela revient à résoudre les attaques d'hier, alors qu'il faudrait plutôt travailler à rendre plus difficiles celles de demain.

Comme beaucoup de programmeurs expérimentés, Bernstein met en garde contre l'optimisation prématurée, cette tendance à se préoccuper de performance sans avoir mesuré <https://www.bortzmeyer.org/mesurer-temps-execution.html> (et c'est un des rares points où il fait une auto-critique, à propos de l'optimisation du mécanisme de stockage de la liste des domaines gérés par qmail). Il plaide donc fortement pour faire passer la sécurité avant la performance, surtout si on n'a pas réellement mesuré le coût en performance d'une mesure de sécurité.

Bernstein suggère aussi des pistes pour limiter le nombre de bogues. Un exemple serait un mécanisme dans le langage de programmation pour mettre à jour automatiquement les variables « de résumé » (comme « le nombre d'éléments non nuls de la liste », sans doute du genre de ce que permettent les "*triggers*" de SQL. De même, il appelle de ses vœux des mécanismes pour étendre automatiquement les tableaux (mais sans citer les langages qui font déjà cela depuis longtemps comme Perl).

Parmi les autres idées de Bernstein, concevoir systématiquement les langages de programmation de façon à ce que les programmes bogués soient plus durs à faire que les programmes corrects. Il cite l'exemple de l'addition d'entiers. En C, par défaut, l'addition se fait modulo la taille de `int` et il faut explicitement utiliser une bibliothèque de "*bigint*" pour avoir l'addition courante. Cela devrait être le contraire, les cas où on veut vraiment une addition modulo étant rares. (Ce point a fait l'objet d'une bonne discussion sur LWN <http://lwn.net/Articles/257004/>.)

Bernstein pointe aussi du doigt les problèmes de sécurité posés par l'analyse, le fait de convertir des données non structurées en données structurées, avec ses conséquences comme le besoin d'échappement, source, par exemple, de tant d'injections SQL <https://www.bortzmeyer.org/sql-injection.html>.

Comment améliorer aujourd'hui la sécurité des logiciels ? Bernstein a plein d'idées. L'une d'elles a donné naissance à un logiciel spécifique, isolate <http://code.google.com/p/isolate/>. Le principe est de faire tourner le programme après avoir sérieusement limité ce qu'il a le droit de faire. Par exemple, sur Unix :

— L'empêcher de créer fichiers ou prises en mettant `RLIMIT_NOFILE` à 0 avec `setrlimit()`,
— Le faire tourner sous un UID spécifique,
— etc.