

RFC 4086 : Randomness Requirements for Security

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 Août 2007

Date de publication du RFC : Juin 2005

<http://www.bortzmeyer.org/4086.html>

La sécurité, c'est toujours un problème difficile. Parmi les sous-problèmes à résoudre pour rendre un système informatique sûr, la sécurité du générateur de nombres aléatoires occupe une bonne place, mais qui est souvent méconnu. Ce RFC fait donc le point sur le problème et sur les mesures à adopter lors de la mise en œuvre des protocoles IETF.

L'alerte de sécurité CVE-2007-2926 <<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-2926>> sur le générateur aléatoire de BIND est venue nous le rappeler : beaucoup de mécanismes de sécurité informatique reposent sur la génération « sûre » de nombres aléatoires. « Sûre » voulant dire ici « Difficile à prévoir par l'adversaire ». Comme tout serveur récursif DNS, BIND doit placer dans les requêtes sortantes une "query ID" que le serveur faisant autorité mettra dans sa réponse, permettant ainsi de s'assurer de la validité de la réponse (la "query ID" sert donc de "cookie"). Si les "query ID" sont prévisibles par un méchant, il peut facilement fabriquer de fausses réponses qui ont l'air vraies et ainsi empoisonner le cache du serveur récursif.

Cette faille est très classique : nombreux sont les logiciels qui utilisent, sans bien s'en rendre compte, un générateur aléatoire prédictif. Notre RFC donne une explication partielle pour cet aveuglement : les textes classiques sur les générateurs aléatoires ne les jugent que par des considérations statistiques, pas en prenant le point de vue d'un adversaire qui chercherait à trouver le nombre suivant. Notre RFC cite l'exemple d'un générateur qui, devant fabriquer des nombres de 128 bits aléatoires, produirait en moyenne 50 % de nombres égaux à zéro et 50 % de nombres aléatoires. Statistiquement, il y aurait toujours 64 bits d'entropie, ce qui peut sembler suffisant. Mais un attaquant n'aurait qu'à essayer un nombre nul pour réussir la moitié du temps et l'entropie réelle serait donc de un bit...

Ce RFC, qui succède au RFC 1750¹, est donc consacré à l'exposé des bonnes pratiques en matière de générateur aléatoire. Depuis le RFC 1750, le monde des protocoles Internet a été bouleversé puisque

1. Pour voir le RFC de numéro NNN, <http://www.ietf.org/rfc/rfcNNN.txt>, par exemple <http://www.ietf.org/rfc/rfc1750.txt>

presque tous dépendent désormais plus ou moins de la cryptographie, qui elle-même dépend souvent de générateurs aléatoires sûrs, par exemple pour fabriquer les clés de session.

Les bonnes pratiques peuvent se regrouper en deux catégories, utiliser le hasard présent dans le matériel, et utiliser des générateurs cryptographiquement forts.

Le matériel d'un système informatique est en effet source de nombreux bruits très aléatoires. Le bruit de fond du micro (section 3.2.1) ou les interruptions envoyées par le disque dur (section 3.2.2) sont deux bons exemples. Presque tous les ordinateurs ayant un tel matériel, il est recommandé de l'utiliser comme source d'entropie. (Les cartes réseaux ne sont mentionnées qu'en passant, car leurs interruptions peuvent être influencées par un tiers, surtout s'il est sur le même réseau.) Idéalement, il faudrait utiliser des dispositifs quantiques comme un ensemble d'atomes radioactifs dont on mesure la désintégration mais les PC ont plus souvent un micro qu'un compteur Geiger... (À défaut d'un tel compteur, on peut utiliser un générateur de bruit blanc comme l'Alea <<http://www.araneus.fi/products-alea-eng.html>>.)

Le RFC détaille aussi des mauvaises idées, par exemple le programmeur qui doit initialiser un générateur aléatoire avec une graine ("seed") utilise souvent l'horloge de la machine (par exemple en Python, quelque chose comme `generator = random.Random(time.time())` ou bien en C, `initstate(time(NULL), NUMSTATES);`). C'est en général très peu sûr, car les horloges ont typiquement une résolution très faible : si l'attaquant sait à quelle heure approximative le système a démarré, il peut trouver la graine et en déduire les nombres générés (section 3.4 du RFC). La section 6.1 détaille d'autres mauvaises idées, notamment celle de croire que, si on fait des manipulations très complexes des données, on est d'avantage en sécurité (en fait, c'est souvent le contraire).

Il y a beaucoup d'autres détails à prendre en compte (le RFC note que le problème est « étonnement difficile ») comme le fait qu'une machine qui démarre a accumulé peu d'entropie et est donc particulièrement vulnérable (sauf si on sauvegarde le dernier résultat du générateur lors de l'arrêt, pour l'utiliser comme graine au démarrage suivant).

Pour le programmeur qui trouve cela bien difficile, notons que le RFC, dans le traditionnel esprit IETF de se soucier des problèmes pratiques, pas juste de la théorie, a prévu une section 7 qui détaille les fonctions déjà écrites et qu'il n'y a plus qu'à utiliser comme `/dev/random` sur beaucoup de systèmes Unix. `/dev/random` est un pseudo-fichier rempli par le noyau du système en utilisant diverses sources d'entropie, dont celles fournies par le matériel. Il existe aussi un `/dev/urandom`, moins sûr (car utilisant également un générateur pseudo-aléatoire) mais non bloquant (la lecture de `/dev/random` peut durer longtemps, car le noyau n'y écrit rien tant qu'il n'a pas assez récolté d'entropie). Voici un exemple d'utilisation de `/dev/urandom` pour obtenir 512 octets aléatoires dans un fichier :

```
dd bs=512 count=1 if=/dev/urandom > /tmp/random_data
```

Ceux qui aiment lire le code source noteront que, dans les noyaux Linux 2.6, ce mécanisme est programmé dans `drivers/char/random.c` et que les commentaires très détaillés de ce fichier sont instructifs.

Les personnes allergiques aux mathématiques sont prévenues que ce RFC est un des rares documents de l'IETF à comporter des formules mathématiques (qui n'ont pas été faciles à publier en texte seul).