Le projet OpenBSD : Développement d'un système sûr

Matthieu Herrb & Louis Bertrand

Septembre 2002

Introduction

- OpenBSD (http://www.openbsd.org/) est un système Unix complet basé sur 4.4BSD :
  - noyau, utilitaires, bibliothèques, démons, XFree86, etc.
  - multi plate-formes

    alpha, amiga, hp300, i386, mac86k, macppc, mvme68k, sparc, ultrasparc, vax
  - peut exécuter des binaires Linux, FreeBSD, Solaris x86 en émulation.
- Sécurité et correction du code.
  - cryptographie forte intégrée,
  - audit permanent du code,
  - sûr par défaut,
  - conforme aux standards.
- Disponible librement sous licence BSD.

Un système complet

- Internet :
  - HTTP (Apache), FTP, DNS (Bind 4), Sendmail, ALTQ, IPv6 (KAME),. . .
- Réseau local :
  - DHCP, NFS, NIS,. . .
- Sécurité :
  - Filtre de paquets PF,
  - contrôle de l'intégrité des fichiers (mtree),
  - Authentification BSD,
  - Niveaux de sécurité du noyau,
  - Flags étendus des fichiers : immutable, append-only. . .
- Cryptographie
  - Bibliothèques complètes (OpenSSL),
  - OpenSSH, Kerberos IV & V (Heimdal), IPsec, S/Key,
  - Hash des mots de passe utilisant Blowfish,
  - Support des accélérateurs matériels (Hifn, Broadcom) et des générateurs d'entropie matériels (i810,. . .)

Applications

- Le noyau et les applications de base sont maintenus par la même équipe.
  - Approche cohérente,
  - Tests intégrés.
- Mécanisme des « ports » pour les applications tierces :
  - `make install`

    téléchargement, patch, configure, compilation, packaging, installation.
  - packages pré-compilés (`ftp://ftp.openbsd.org/pub/OpenBSD/packages/`, CDROM)
  - plus de 2000 applications et bibliothèques
- Exécution d'applications binaires Linux grâce à l'émulation des appels système du noyaux Linux : Netscape, Acrobat Reader, StarOffice,. . .

Sûr par défaut

- Seuls les services et les démons indispensables sont actifs par défaut ;
  - les autres sont activés si nécessaire par l'administrateur.

- À l'opposé de la logique « rien à configurer »
  - travail supplémentaire pour désactiver les services inutiles,
  - difficile pour l'administrateur débutant,
  - ce que l'on ne connaît pas peut causer des problèmes.

« Une vulnérabilité à distance dans l'installation par défaut en presque six ans »

Licence BSD

But du projet : Système réellement librement redistribuable.

- Pas de restrictions sur l'utilisation, la modification ou la redistribution du code source ou binaire.
- Seule contrainte : laisser le copyright original.

Licence GPL (gcc,. . .) acceptable, mais minimiser et identifier clairement les composantes concernées.

Vérification exhaustive des licences des logiciels inclus puis actions :
– modification de la licence par l'auteur,
– suppression du composant,
– voire réécriture,
– lobbying auprès des organismes internationnaux.

Organisation du projet

- Démarré en 1995 (à partir de NetBSD).
- Basé au Canada (Calgary).
- Développement supporté par les utilisateurs :
  - vente de CDs, T-Shirts,. . .
  - dons,
  - support de quelques entreprises et universités (temps ingénieur, matériel,. . .),
  - Bourses DARPA, Usenix,. . .
- Une équipe de bénévoles :
  - développeurs,
  - autres contributions : tests, documentation. . .

Les développeurs du projet

- Développement international.
- Une centaine de personnes, environ 40 actifs à un instant donné.
- Leader du projet : Theo de Raadt.
- Développements cryptographie dans les pays ou cela est légalement possible (Canada,...)

Outils utilisés

CVS

– dépot principal au Canada ;
– accès par CVSup pour mirroirs complets du dépot ;
– accès en écriture pour tous les développeurs ;
– accès anonyme read-only pour tout le monde.

E-mail

– Listes publiques : `tech@openbsd.org`, `misc@openbsd.org`,. . .
– Une liste privée pour les développeurs.

Peu de listes, afin de ne pas diluer l'information.

ICB

– Forum de discussion en ligne pour les développeurs pour réactions rapides.

« Hackathons »

Calgary, Canada, Juin 2002

Rencontres régulières des développeurs autour d'événements externes (Conférences Usenix,. . .)

– Favorisent les échanges, font avancer le développement.

Programmation pour la sécurité

Paranoïaques ?
– Pas indispensable, mais ça aide !

La majorité des programmeurs espère que peu d'utilisateurs seront confrontés aux bugs.
– Les utilisateurs évitent les bugs,
– Les tests ne vérifient que le comportement nominal attendu.

Les attaquants font le contraire :
– injectent des données absurdes dans l'application,
– déterminent les comportements anormaux,
– abusent ces erreurs pour mettre à jour les vulnérabilités.

Il faut toujours plus de tests avec des données aléatoires.

Qualité du code

Approche volontariste (*Proactive*) :
– les erreurs de programmation sont à l'origine des vulnérabilités,
– corriger les erreurs, c'est sécuriser le système,
$\rightarrow$ audit permanent

Recherche des erreurs, pas des vulnérabilités.
– Savoir si une erreur est exploitable n'est pas important.
– Correction de l'erreur et passage à la suivante.

Bénéfice : un certain nombre d'avis de sécurité ne concernent pas OpenBSD.
– Les bugs ont été corrigés des mois (parfois des années) avant la découverte de la vulnérabilité.
– Des milliers (!) d'erreurs ont été corrigées depuis le début.

Les règles de base de la programmation sûre

- Minimiser le code critique.
- Être paranoïaque par rapport aux données d'entrée.
- Minimiser les privilèges :
  - Abandonner ses privilèges le plus tôt possible,
  - Préférer un petit programme `setuid` externe plutôt que de positionner le bit `setuid` sur une grosse application.
- K.I.S.S. → Keep It Simple and Stupid.

Exemple : authentification BSD :
concept similaire à PAM *mais* les modules sont des processus externes exécutés par `fork()`/`exec()`.

→ Pas de privilège particulier pour des applications comme par ex. `xlock`.

L'audit du code

Processus :

1. détection d'une erreur de programmation,
2. généralisation à une classe d'erreurs,
3. recherche d'autres erreurs de cette classe dans le système.

Un simple « Rechercher & Remplacer » ne suffit pas :
– différences sémantiques mineures qui peuvent être importantes,
– il est important de comprendre le code que l'on corrige.

Documentation pour les utilisateurs et les développeurs

- Le code source accessible à tous :
  - CVS anonyme.
- Les pages de manuel doivent être cohérentes avec le code :
  - inclure des tutoriels ou des how-to et des exemples ;
  - les interfaces du noyau et des bibliothèques doivent être documentées,
  - aucun code nouveau n'est intégré sans sa documentation.
- Style de codage et de documentation imposé : (`style(9)`)
  - commentaires concis et précis,
  - documentation claire des dépendances/pré-requis/hypothèses,
  - présentation homogène du code.

Le processus de release d'OpenBSD

- Une release tous les 6 mois :
  - Suppression des fonctionnalités de qualité insuffisante pour une release.
  - Verrouillage des sources un mois avant la release (correction de bugs seulement).
  - Période de test par les développeurs et les utilisateurs.
  - Les nouvelles fonctionnalités sont ajoutées rapidement après le déverrouillage $\rightarrow$ laisser le temps de stabiliser avant release suivante.
- Dilemne fonctionnalités/délais en discussion permanente.
- Des contraintes de date fixes incitent à la prudence face aux trop grands boulversements.

Intégration de nouvelles fonctionnalités

- Le code expérimental reste en dehors de l'arborescence principale ;
- intégré lorsqu'une stabilité satisfaisante et un consensus des développeurs/testeurs sont acquis.

  Harcèlement (virtuel) des développeurs pour faire avancer les développements.

- Revue mutuelle :
  - Les développeurs regardent le code des autres et donnent leur avis.
  - Pour chaque partie du système, un « propriétaire » est désigné.

Exemples de résultats du processus OpenBSD (1)

- **Chaînes de format**
  - Vulnérabilité découverte dans le serveur FTP : abus de la spécification de format du paramètre de `setproctitle()` → corrigée
  - Généralisation : autres fonctions similaires qui peuvent conduire au même type d'erreurs : `syslog()`, `errx()`, ...
  - Audit du reste des outils → plusieurs dizaines d'autres erreurs similaires corrigées (sans se préocupper de l'exploitabilité).
  - Ensuite découverte ailleurs de plusieurs nouvelles vulnérabilités de ce type déja corrigées dans OpenBSD.

Exemples de résultats du processus OpenBSD (2)

- **Fonctions de copie de chaînes de caractères**
  - Fonctions traditionnelles `strcpy()` ou `strcat()` non sûres : débordement de buffers.
  - Conseil habituel → utiliser `strncpy()` et `strncat()` qui ont un paramètre de plus pour limiter la taille.
  - L'utilisation de ces fonctions est « délicate » (risque de laisser des chaînes non terminées) et peu efficace.
  - Un audit a révélé de nombreuses erreurs dans leur utilisation.

Exemples de résultats du processus OpenBSD (3)

→ Spécification et implémentation de nouvelles interfaces :

```
size_t strlcpy(buf, input, sizeof(buf));
size_t strlcat(buf, suffix, sizeof(buf));
```

- Utilisation cohérente de la taille du buffer destination,
- Assure que la chaîne résultat est correctement terminée,
- Plus intuitif,
- Code plus lisible,
- (Reste quand même quelques problèmes).

□ Adopté par d'autres systèmes (NetBSD, FreeBSD, Solaris), bientôt standardisé.

Audit des licences

Morceaux de code avec conflits de licence (pas de modification ou redistribution gratuite seule) :

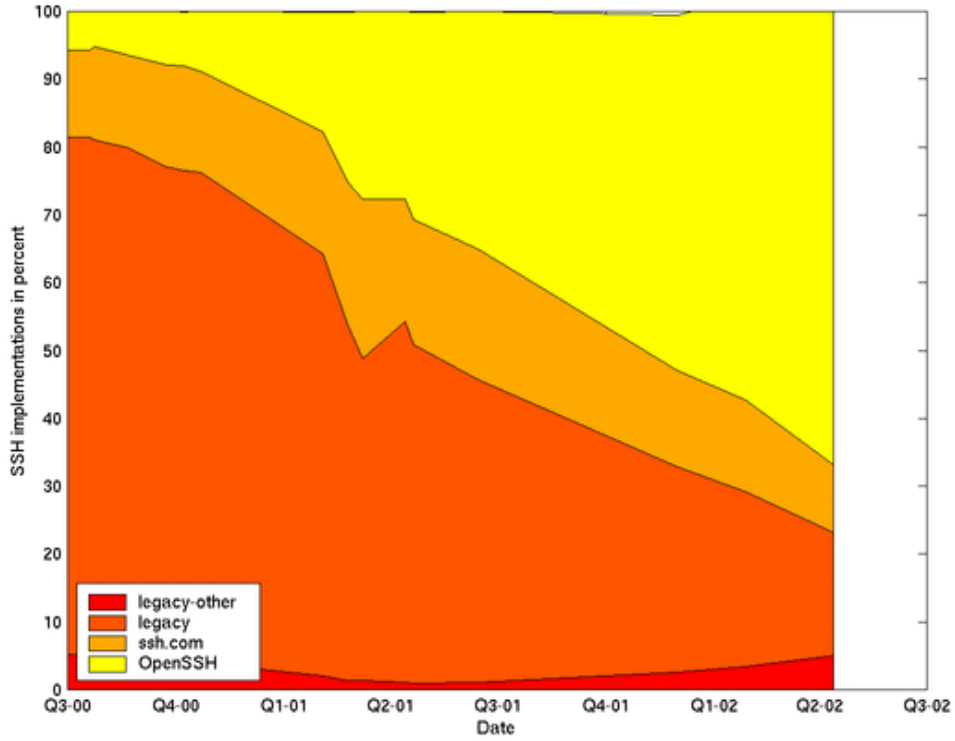
- tcpwrappers (licence modifiée par l'auteur)
- md5 (réécrit)
- rpc.pcnfsd (licence modifiée)
- lex (licence modifiée)
- pppd (licence modifiée)
- certains modules de tcpdump (licence modifiée/réécrits/suprimés)
- . . .

Plusieurs autres problèmes de licence en cours de résolution.

Nouveau ! Brevets logiciels dans OpenSSL :-(

Succès sur le front des licences (1)

- **OpenSSH**
  implémentation du protocole SSH v1.5 sur la base de la dernière version libre de SSH 1.2. Nombreux ajouts (support du protocole v2, cartes à puces,. . .)



Succès sur le front des licences (2)

- **PF**

  La licence de IPF, le filtre de paquets de OpenBSD $\leq$ 2.9 posait problème (interdiction de redistribuer le code si modification) $\Rightarrow$ développement d'un nouveau logiciel à partir de zéro.
  - maintient une certaine compatibilité avec les règles d'IPF,
  - nouvelles fonctions : nettoyage des paquets, insertion dynamique d'états (pour relais applicatifs), service d'authentification (authpf),
  - meilleures performances,
  - code plus petit.

Difficultés

Le monde n'est pas parfait...

- Manque de développeurs motivés pour le travail de fond (ce qui ne se voit pas),
- Erreurs humaines toujours possibles : cf. Failles récentes d'OpenSSH
- Progrès de l'emprise de la propriété intellectuelle (brevets logiciels)
- Support des nouveaux matériels de plus en plus dur (absence de documentation, complexité intrinsèque, mauvaise qualité)

Projets en cours

- Audits en cours :
  - des fonctions de traitement des signaux (usage incorrect de fonctions non sûres, race conditions),
  - débordements arithmétique entière
- Révocation des privilèges des démons système (et du serveur X),
- Systrace : outil de définition de politiques de contrôle d'accès aux ressources système.
- Élimination de race conditions obscures dans le noyau,
- Réintégration du code de 4.4 BSD après libération du code par Caldera,
- Meilleure résistance du noyau au manque de mémoire virtuelle,
- Support de nouveaux accélérateurs de cryptographie, et support de ces accélérateurs dans OpenSSL (et donc OpenSSH).
- Améliorations dans PF (meilleure intégration avec ALTQ,...)

Projets à plus long terme

Support IEEE 1394 (Firewire)

Nouvelles architectures : HP-PA, mvme88k, mvmeppc, SGI, mips,. . .

Multi-processeurs,

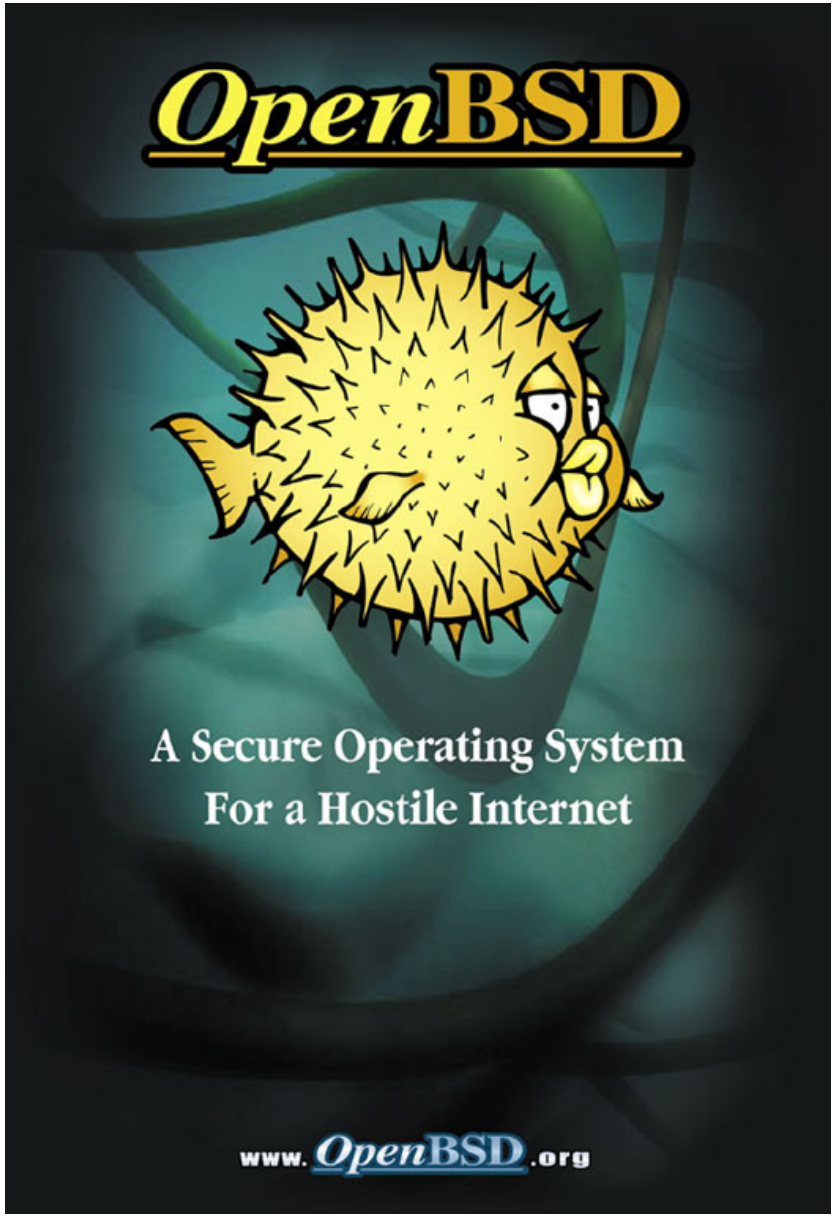
XFree86 DRI : accélération matérielle OpenGL (3D),

Intégration de fonctions de sécurité avancées de Trusted BSD,

UBC (Unified Buffer Cache),

Migration CVS → OpenCM (http://www.opencm.org/)

Et toujours : **rechercher et corriger les erreurs dans le code !**


OpenBSD
A Secure Operating System
For a Hostile Internet
www.OpenBSD.org