

La sécurité sous BSD

Fabien DEVAUX Nicolas BERTHEL Ludovic MARTIN
Julien LECOUFLE Vincent COUDEVILLE Guillaume PENIN
 Marc MESSIAN

Janvier 2005

Table des matières

1	Confinement	4
1.1	Systrace	4
1.1.1	Présentation	4
1.1.2	Utilisations	4
1.2	Jails	8
1.2.1	Présentation	8
1.2.2	Fonctionnement	8
1.2.3	Avantages	9
1.2.4	Autres solutions	10
1.3	PF	11
1.3.1	Historique	11
1.3.2	Avantages	11
2	Limitations	13
2.1	chflags	13
2.1.1	Présentation	13
2.1.2	Mise en place	13
2.2	securelevels	14
2.2.1	Présentation	14
2.2.2	Mise en œuvre	15
2.3	sysctl	15
2.3.1	Qu'est-ce que sysctl ?	15
2.3.2	Exemples d'utilisation	15
2.3.3	Quelques paramétrages utiles	16
2.3.4	Conclusion	16
2.4	non-exec stack (& heap)	16
2.4.1	Présentation	16
2.4.2	La politique de BSD	17
2.4.3	Les protections	18

3	Qualité	21
3.1	Honeyd	21
3.1.1	Présentation de honeyd	21
3.1.2	Caractéristiques de honeyd	21
3.1.3	Configuration de honeyd	23
3.1.4	Conclusion	25
3.2	Veriexec	25
3.2.1	Utilité	25
3.2.2	Fonctionnement	25
3.2.3	Utilisation	26
3.3	Stephanie	26
3.4	Portaudit / Audit-packages	27
3.4.1	Présentation	27
3.4.2	Fonctionnement	27
4	Confidentialité	29
4.1	/etc/ttys et login.conf	29
4.1.1	Notion de classes	29
4.1.2	Sa fonctionnalité	29
4.2	Cryptage des disques	32
4.2.1	Présentation	32
4.2.2	La mise en place	33
4.3	OTP	33
4.3.1	Le principe	33
4.3.2	Le fonctionnement	34
4.4	OpenSSH	35
4.4.1	Historique	35
4.4.2	Fonctionnalités	36
4.5	IPSec	37
4.5.1	Introduction	37
4.5.2	Services offerts par IPsec	37
4.5.3	Architecture	38
4.5.4	Gestion des échanges de clefs	39
4.5.5	Problèmes divers lié à l'utilisation de IPsec	39
4.5.6	Conclusion	40
5	Conclusion	41

Introduction

Lorsque l'on évoque "BSD"¹, les gens ignorants s'imaginent en général un système que personne n'utilise : le travail d'universitaires illuminés. Pour d'autres qui ont quelques connaissances théoriques il y a 3 mondes parfaitement distincts : FreeBSD pour la vitesse d'exécution, OpenBSD pour la sécurité et enfin NetBSD pour l'utilisation sur des architectures exotiques.

Vous verrez à l'issu de ce document que cette frontière est en fait la devise première de chaque variante mais ne signifie pas que "tel ou tel" BSD est meilleur qu'un autre. Par exemple des benchmarks récents montrent que NetBSD est en fait probablement le plus rapide dans le cadre d'une utilisation en tant que serveur.

Dans tous les cas, vous vous rendez compte je l'espère qu'il n'y a pas de BSD meilleur qu'un autre pour un serveur sécurisé. Il y a juste différentes approches pour chaque problème, par exemple le confinement fort d'un service pourra se faire à l'aide de jails sous FreeBSD, systrace sous OpenBSD et Xen sous NetBSD². Chacun avec ses avantages et inconvénients, il vous est alors possible de choisir le BSD qui vous convient le mieux en fonction de vos préférences et de l'utilisation précise que vous allez en faire.

Pour finir, je voudrais préciser que sécuriser un système BSD ne se résume pas à appliquer les différentes techniques énoncées dans ce document. Une bonne connaissance du système et une veille technologique sérieuse sont irremplaçables. Un admin sérieux considérera que les toutes premières étapes de la sécurisation d'une machine consistent à configurer³ le noyau (sur un système BSD ou Linux) et à supprimer tous les services qui ne sont pas vitaux. Ces techniques de base de l'administration ne sont même pas énoncées.

Sur ces derniers mots je vous souhaite une bonne lecture, j'espère qu'elle vous sera utile et vous donnera encore un peu plus le goût des systèmes BSD. :)

¹Berkeley Software Distribution

²systrace est aussi disponible nativement pour NetBSD

³et recompiler bien entendu

Chapitre 1

Confinement

1.1 Systrace

1.1.1 Présentation

Systrace est un outil de sécurisation méconnu à tort. Ce programme provenant d'OpenBSD est intégré nativement à OpenBSD et NetBSD mais existe pour d'autres systèmes tels que FreeBSD, Linux ou même MacOS X. La première fois qu'on appréhende ce programme on ne sait pas exactement comment l'utiliser, après quelques instants on se rends compte de la souplesse et de la puissance d'un tel outil. Les utilisations sont multiples, cet outil pourrait être vu comme le couteau suisse du confinement.

Dans la pratique, systrace est un gestionnaire d'accès aux appels système¹. Pour chaque programme exécuté sur une machine systrace permet de définir un ensemble de règles définissant les appels systèmes autorisés ou non (notamment en fonction des paramètres passés à ces fonctions) : ouverture/lecture/écriture d'un fichier ou d'une socket, allocation de mémoire, etc. . .rien ne lui échappe.

1.1.2 Utilisations

Les utilisations de cet outil ne sont limitées que par l'imagination, cependant il y a un certain nombre d'utilisations classiques, en voici quelques-unes :

Surveillance d'un programme peu fiable

Parfois vous n'avez pas les sources d'un programme mais vous aimeriez bien l'utiliser tout de même. Il est très dur de savoir ce que va faire un programme sans l'exécuter, plusieurs techniques existantes servant à examiner un programme sont connues, la majorité

¹System Calls

exécutant le programme dans un environnement isolé. Ces techniques sont plus ou moins difficiles à mettre en oeuvre et ne s'avèrent pas toujours fiables.

L'utilisation de systrace dans ce cas est un véritable jeu d'enfant, il suffit d'exécuter systrace avec le programme suspect en paramètre. À chaque appel système, systrace affiche l'appel ainsi que ces paramètres et vous demande la démarche à adopter : Deny, Permit, Deny Always ou Permit Always. Notez qu'il est aussi possible de tuer le processus à examiner à tout moment. Le fonctionnement de cet outil graphique peut-être comparée à l'utilisation d'un débogueur en mode step-by-step².

Cet outil reprends la configuration systrace existante pour le programme donné, vous évitant de répondre à des questions déjà posées lors d'une exécution précédente. Ce mode de fonctionnement permet aussi d'écrire ou de mettre à jour les règles de gestion pour le programme. Notez qu'il existe un outil supplémentaire un peu plus puissant que celui proposé par défaut pour l'affichage graphique : `gtk-systrace`[10].

Sécurisation d'un programme

Probablement l'utilisation la plus classique de systrace, une base de règles bien conçue autorisant tous les appels rentrant dans le fonctionnement "normal" d'un programme est utilisée. Systrace rejette alors automatiquement tous les autres appels. Par exemple si apache est lancé sous la surveillance de systrace et que quelqu'un arrive à lui faire exécuter du code malicieux (par exemple par l'intermédiaire d'un Buffer-Overflow, voir aussi la page 16, section 2.4 à ce sujet) alors l'appel échouera. Lire le fichier `/etc/passwd` ne fait pas partie du fonctionnement normal d'un serveur WWW, l'appel à `open()` échouera inconditionnellement dans ce cas ci. Une base de donnée de configurations peut-être trouvée sur le site du projet "Hairy Eyeball"[9], elle contient les règles de gestion pour plus de 200 programmes.

Utilisation en tant que prison (jail)

La finesse de la configuration permet les choses les plus inattendues, il est par exemple possible de générer une jail (voir page 8, section 1.2 pour plus d'informations sur les jails). Tout le monde sait que le `chroot` n'a pas été fait pour la sécurité, bien qu'il ai évolué il comporte toujours de grandes faiblesses. L'équipe de FreeBSD à donc inventé les jails afin de résoudre ce problème. Systrace réponds à tous ces besoin et plus encore, la mise en place d'une prison à l'aide de systrace demande cependant beaucoup de temps (non pas à cause de la difficulté de la mise en oeuvre, mais afin d'autoriser assez de choses il faut écrire beaucoup de règles).

²mode pas à pas ou le débogueur stoppe l'exécution du programme à chaque appel

Afin d'illustrer la mise en prison d'un utilisateur, voici un petit exemple de shell minimal. Compilez-le³ et définissez-le comme shell par défaut d'un utilisateur pour le mettre en prison. Le shell (ici *ksh*) sera exécuté sous le contrôle de *systrace* mais aussi tous les programmes lancés depuis ce shell.

jail.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    char *args[5];

    args[0] = "systrace";
    args[1] = "-t";
    args[2] = "/bin/ksh";
    args[3] = "-l";
    args[4] = NULL;

    execv("/bin/systrace",args);
    return EXIT_FAILURE;
}
```

Exemple minimaliste : imaginons que nous voulions offrir à certains usagers un shell sur un serveur. Ce shell devra être limité en fonctionnalités (copie de fichiers, listage, suppression, etc...), nous aimerions aussi que l'utilisateur ne puisse influencer que sur son répertoire HOME. Une jail ou un chroot permet de répondre plus ou moins facilement à ces attentes avec des degrés de sécurité différents. Voyons à quel point la mise en place avec *systrace* est aisée et ce, en toute sécurité :

fichier bin_ksh

```
native-fsread: filename eq "/var/mail/$USER" then permit
native-fsread: filename eq "$HOME" or filename match "$HOME/*" then permit

native-chdir: filename eq "$HOME" or filename match "$HOME/*" then permit

native-fsread: filename eq "/bin/ls" then permit
native-execve: filename eq "/bin/ls" then permit

native-fsread: filename eq "/bin/ps" then permit
native-execve: filename eq "/bin/ps" and not argv re "^ps.*(a|x)" then permit
```

³Copiez ce code dans un fichier nommé "jail.c" et tapez "gcc -o jail jail.c"

Ces lignes peuvent paraître un peu barbares pour le non-programmeur, pourtant elles sont très simple. Systrace regroupe (par défaut) les appels système faisant le même type d'opérations sous un nom commun, par exemple les fonctions comme `lstat()` ou `access()` sont regroupées sous le nom *fsread*.

Ainsi les premières lignes spécifient que le shell "ksh" a le droit d'accéder à la boîte email et aux fichiers contenus dans le répertoire HOME de l'utilisateur. La 3ème ligne empêche l'utilisateur de changer de répertoire si celui-ci n'est pas dans son HOME. Les deux lignes suivantes autorisent l'utilisateur à exécuter la commande `ls` inconditionnellement. Les deux dernières lignes autorisent l'utilisateur à exécuter la commande `ps` mais empêche l'utilisation des paramètres "x" ou "a" donnant accès à des informations sur des processus n'appartenant pas à l'utilisateur courant.

fichier `bin_ls`

```
[...]
native-fsread: filename eq "$HOME" then permit
native-fsread: filename match "$HOME/*" then permit
native-fsread: filename match "/tmp/*" then permit , if group = wheel
native-fsread: filename eq "/tmp" then permit , if group = wheel
```

Ici nous indiquons que la commande `ls` ne pourra accéder qu'au répertoire HOME et ses sous-répertoires. Les deux dernières lignes illustrent la possibilité d'autoriser une action en fonction du groupe ou de l'utilisateur, ici nous autorisons les membres du groupe *wheel* à lister le contenu du répertoire `/tmp`⁴.

Vous remarquerez avec ces deux exemples la grande souplesse de la solution et la précision avec laquelle les permissions sont données. Il est aussi possible de spécifier les différentes erreurs renvoyées en cas de refus d'appel à une ressource, par exemple :

```
$ ls /tmp
ls: /tmp: Operation not permitted
```

Si on ajoute en fin du fichier *bin_ls* la ligne

```
native-fsread : deny[enoent]
```

on constate le changement suivant :

```
$ ls /tmp
ls: /tmp: No such file or directory
```

Alternative puissante à PrivSep

A l'aide de systrace il est possible de se passer des programmes s'exécutant avec les droits `root`. Une fonctionnalité supplémentaire est l'élévation de privilèges, systrace doit

⁴Ces deux lignes peuvent être condensées en une seule mais ont été découpées afin de préserver la mise en page de ce document

être démarré en tant que root et on lui spécifie sous quel utilisateur (et groupe) les programmes doivent être exécutés par défaut. Dans ce cas, il est aussi possible de modifier l'utilisateur (et le groupe) sous lequel va être exécuté un appel système précis. Le bien connu problème de `bind()` sur un port < 1024 (qui est réservé au root) peut alors être résolu d'une manière très élégante. Un serveur `ssh`⁵ "écoutant" le port 22 local pourra alors se voir autorisé de le faire même si il s'exécute sous un utilisateur non-privilegié en ajoutant la ligne suivante :

```
usr_bin_ssh
```

```
native-bind: sockaddr eq "inet-[0.0.0.0]:22" then permit as root
```

Rappelons que PrivSep (aka Privilege Separation) est une technique de programmation⁶ visant à séparer un programme en deux : le premier, minimal, utilise les droits du root et passe le relais dès que possible à un deuxième qui s'exécute avec les droits d'un utilisateur précis. Il est ainsi moins probable qu'un utilisateur malicieux gagne les droits du root car le processus effectuant les traitements ne les a jamais possédés. L'équipe du projet OpenBSD travaille activement à la conversion en *PrivSep* des serveurs les plus courants, un bon nombre d'entre eux sont déjà disponibles.

1.2 Jails

1.2.1 Présentation

Les jails⁷ permettent d'ajouter de la granularité aux privilèges et droits unix sur les systèmes FreeBSD (depuis la version 4.0), en reprenant le concept de chroot (une commande de base des Systèmes unix) qui permet de limiter à une arborescence donnée un processus.

Concrètement, une Jail va plus loin qu'un simple chroot et permet de confiner des programmes (en général des daemons) dans un environnement mémoire clos et sécurisé. En effet, lorsqu'un programme est placé dans une Jail, l'utilisateur de ce programme ne pourra accéder (sur le disque) qu'à la racine virtuelle créée par la Jail, et n'aura connaissance que de son propre processus ou de ses éventuels fils. L'utilisateur ne pourra donc dans aucun cas avoir accès au système hôte. Pour de plus amples informations, consultez l'excellent papier fait à ce sujet[4].

1.2.2 Fonctionnement

Lors de la création d'une jail, il faut lui attribuer une adresse IP, qui servira à la communication réseau, ainsi qu'un chemin (répertoire) qui sera la racine de l'environnement

⁵Présenté en page 35, section 4.4

⁶Et c'est pour cela qu'elle n'est pas traitée dans ce document

⁷"prison" en Français

créée par la jail.

Pour la mettre en oeuvre ensuite, il faut recréer un environnement autonome dans lequel on va enfermer le (ou les) service à isoler ainsi que ses dépendances.

Pour commencer il vous faut un répertoire contenant l'image d'un système FreeBSD complet, exactement comme pour la mise en place d'un chroot(). Il y a plusieurs façons de faire se basant soit sur la décompression des packages de base de FreeBSD (déjà compilés), soit sur la compilation du système⁸.

Ensuite il faut garder en tête qu'une jail est limitée à une seule IP (en général, un alias) il y a donc des précautions à prendre si certains serveurs s'exécutent sur le système hôte⁹ afin qu'ils ne répondent pas à des requêtes destinées à un des services de la prison.

Une fois la jail démarrée vous obtenez un shell sur cette jail, la terminaison de ce shell signifie la fermeture de la prison, la "machine virtuelle" devient alors inexistante. Il est important de garder le processus gardant la prison en vie ouvert.

Une fois le shell obtenu vous pouvez configurer cet autre système FreeBSD sans vous soucier du reste, même le programme d'installation est utilisable! Certains comportements de la jail sont paramétrables via sysctl (page 15, section 2.3) . En voici une brève présentation des MIB utiles :

security.jail.allow_raw_sockets autorise ou non les raw sockets dans la jail. Ils sont communément utilisés pour forger des paquets personnalisés.

security.jail.getfsstatroot_only seul les systèmes de fichiers montés dans la jail y seront visibles.

security.jail.set_hostname_allowed autorise le changement du hostname.

security.jail.sysvipc_allowed autorise la communication entre des processus de la jail et des processus extérieurs. Fortement déconseillé.

security.jail.jailed permet de savoir si vous êtes emprisonné.

1.2.3 Avantages

Il existe d'autres alternatives aux jails pour les systèmes Unix notamment VMware, mais ce dernier ne peut servir que de machine virtuelle de test ou encore de développement, il n'est pas envisageable de l'utiliser pour la production puisqu'il est beaucoup trop gourmand en ressource et n'as pas encore le niveau de sécurité et d'expérience des jail FreeBSD dans ce domaine. Les jails étant directement intégrées au noyau, elles sont très légères pour le système, et en créer une pour chaque programme à isoler est donc tout à fait possible.

⁸cette méthode est celle expliquée dans la page de manuel jail(8)[15]

⁹celui qui n'est pas dans une jail

1.2.4 Autres solutions

Nous allons rapidement vous présenter les alternatives possibles aux jails

VMWare

Nous l'avons déjà évoqué dans la partie précédente. Cette solution est extrêmement gourmande en ressources, il n'est vraiment pas envisageable de l'utiliser pour un serveur de production. De plus pour faire fonctionner une machine virtuelle VMWare il faut disposer d'un serveur graphique (qui n'est pas forcément conseillé pour un serveur que l'on souhaite sécurisé). Enfin un point non négligeable, cette alternative n'est pas libre.

UML (UserMode Linux)

Voilà encore une autre machine virtuelle qui crée un système au sein même de votre système, certes moins gourmand que VMWare, il reste toujours plus lourd que les jail et sa mise en place est très fastidieuse puisqu'il faut installer un système complet dans votre machine virtuelle incluant son propre système de fichier et même son propre kernel.

Xen[5]

Xen est très puissant, (peut être même trop pour la majorité des serveurs). Il n'a pas le principal défaut des solutions alternatives au jail puisqu'il est très léger mais c'est son installation à l'instar de UML qui est laborieuse puisqu'il faut là encore un deuxième système complet.

Systrace[8]

Systrace est très performant, il est peut gourmand en ressources et met à disposition les mêmes fonctionnalités que les jail. Cependant c'est au niveau de la mise en œuvre que cette méthode trouve ses limites, puisqu'il devient très compliqué si l'on souhaite fournir beaucoup de droits à l'utilisateur. (pour de plus amples informations sur Systrace référez vous à la page 4, section 1.1)

Chroot

Chroot est le précurseur de cette méthode de confinement, les jails sont d'ailleurs inspirées de chroot¹⁰. Il est donc normale que cette solution se montre aujourd'hui un peu désuète, il existe des failles connues et l'on peut donc contourner le confinement. Enfin il faut à ici aussi une nouvelle arborescence pour la machine virtuelle que l'on souhaite créer.

¹⁰certains appellent même, à tort, un système chrooté une "jail"

Chroot représente surement la solution alternative la moins sécurisée et la moins puissante mais paradoxalement la plus répandue.

1.3 PF

1.3.1 Historique

Le filtre de référence sous un système BSD est Packet Filter (appelé habituellement pf). L'unification du firewall n'est que récente :

Pf est apparu dans OpenBSD 3.0[14], les autres systèmes (NetBSD et FreeBSD) utilisant `ipf` et/ou `ipfw` pour le filtrage en accord avec `ipnat` et/ou `natd` pour le NAT¹¹. De plus, pour le QoS (et trafic shaping) `altq` était utilisé.

Actuellement pf devient la référence sous les différents systèmes BSD, il est intégré au système de base depuis FreeBSD 5.3 et fonctionne sous NetBSD 2.0 à l'aide d'un module du noyau.

1.3.2 Avantages

Les avantages de pf comparé aux outils trouvés sur d'autres Unices (ou même sous Linux) :

Syntaxe

Probablement la plus claire des solutions de ce type. un fichier `pf.conf` est en général bien plus lisible qu'un script de firewall pour iptables (netfilter). La configuration se décompose en plusieurs parties, clairement définies et pouvant être agrémentées de commentaires. Pas besoin de faire un script shell invoquant N fois le programme avec des paramètres plus ou moins obscures ni de faire appel à X programmes différents pour les différentes opérations. En effet le filtrage, la redirection et le QoS sont tous gérés par pf, il suffit donc d'apprendre une seule syntaxe. L'utilisation de macros et de tableaux (modifiables à chaud sans redémarrer toute la configuration) sont aussi un des points forts car ils permettent une lisibilité accrue et la factorisation du fichier de configuration.

Une ligne typique du fichier de configuration (section filtrage) serait :

extrait de `pf.conf`

```
pass out on dc0 inet proto tcp from $developerhosts to any port 80
```

¹¹Network Address Translation : redirection de ports

Scrub & Options

La partie Scrub permet d'assainir les paquets et les rassembler (ou les tronquer, au choix) afin d'éviter de nombreuses attaques¹². Il est aussi possible de spécifier un certain nombre d'options plus ou moins proches du noyau comme les différents délais (timeout) utilisés par la communication TCP/IP et même UDP et ICMP. Une gestion basique (limitation du nombre de nouvelles connections prises en charge) pouvant être comparée à du QoS est aussi intégrée à ce niveau, permettant de ralentir l'acceptation de connections, voir même les refuser. Ce type d'options permet de limiter finement la charge d'un service et certains types d'attaques. Il est aussi possible de modifier la gestion de la mémoire (tables utilisées pour le NAT notamment) afin de ne pas défaillir en faisant un OOM¹³ face à un réseau local de grande envergure ou une attaque.

On remarquera aussi une option très intéressante, "random-id" qui substitue le champ ID d'un paquet IP par un aléatoire. En effet une des faiblesses de beaucoup de systèmes d'exploitation est la prédiction possible de ce champ, en le réécrivant la passerelle réduit une partie des failles dues à la mauvaise implémentation de la pile tcp des systèmes d'exploitation du réseau local. Cette option ne s'applique qu'aux paquets sortants bien évidemment.

Synproxy

Pf effectue le handshake puis transmet les paquets au service concerné une fois que le handshake a été validé. Évite de nombreuses attaques et réduit la charge des services (plus de DOS par syn flooding ou avec des IP spoofées). Un scanner de ports ne finissant pas le Handshake par exemple, sera entièrement géré par pf, évitant au serveur se cachant derrière la passerelle d'allouer un processus (ou tout autre traitement plus ou moins lourd) et donc, évite sa montée en charge inutilement.

Anchors

Les points d'ancrage sont un autre point fort de pf, ils permettent d'avoir plus ou moins les fonctionnalités des chaînes de Netfilter (on envoie un paquet à une suite de règles portant un nom). Ce système permet aussi de modifier à chaud les règles de gestion à des endroits clé. un IDS¹⁴ actif peut se servir de ce système pour bloquer une IP dès le début du filtrage par exemple (c'est aussi possible avec l'utilisation de tables). Grâce à ces communications "à chaud" la communication avec des outils en mode utilisateur est facilitée ce qui amène une plus grande souplesse dans la gestion de la sécurité.

¹²Un certain type d'attaque agit en envoyant des fragments de paquets se chevauchant.

¹³Out Of Memory

¹⁴Intrusion Detection System

Chapitre 2

Limitations

2.1 chflags

2.1.1 Présentation

Dans l'optique d'une sécurité maximum du système, les développeurs des distributions BSD ont mis en place un moyen d'interdire la suppression ou la modification de fichiers. Pour cela ils existent des flags sur les fichiers qui permettent de restreindre l'accès. Ces flags sont les flags *Immutable*. Quand un fichier est immuable il ne peut être modifié ou effacé de quelque façon que ce soit jusqu'à ce que le flag *Immutable* soit désactivé. La commande qui permet de mettre en place ce paramétrage est la commande `chflags`.

2.1.2 Mise en place

La syntaxe de la commande est : `chflags option fichier`. Il existe différentes options qui permettent d'effectuer des actions sur les file flags :

- **arch** Cette option uniquement utilisable par root transforme le fichier en archive.
- **opaque** Rend le fichier "opaque" à certaine lecture (utile contre la lecture par d'autres applications) et modifiable uniquement par le propriétaire ou le root.
- **nodump** Permet d'interdire un backup du fichier via l'utilitaire `dump`. Modifiable uniquement par le owner ou le root.
- **sappnd** Instaure le flag system append-only uniquement modifiable par le root en `securelevel` supérieur à 0. Append signifie qu'on ne peut que rajouter des données et non en retirer.
- **schg** Place le flag system immutable qui, comme son nom l'indique, est censé empêcher tout évènement sur le fichier. Modifiable par le root uniquement en `secure level` niveau 0 (voir section suivante).
- **sunlnk** Permet d'interdire la suppression d'un fichier, sachant que cette option n'est modifiable que par le root.

▷ Les options `uappnd`, `uchg` et `uunlnk` sont les pendant respectifs de `sappnd`, `schg` et `sunlnk`. La différence réside dans le fait que, outre le root, le propriétaire du fichier a également le droit de modifier les file flags.

▷ Les options peuvent être précédées de “no” pour l’effet contraire.

2.2 securelevels

2.2.1 Présentation

Le noyau BSD (présent sur les 3 distributions FreeBSD, NetBSD et OpenBSD) offre la possibilité de définir des niveaux de sécurité (Secure level). Ces niveaux même s’ils sont parfois considérés comme “pas aussi parfait qu’il pourrait l’être” par certains offrent une sécurité supplémentaire et sont bien souvent suffisants pour bloquer bon nombre de “pirates bidouilleurs de procédures”.

Il existe en tout 5 niveaux de sécurité différents (numérotés de -1 à 3). Nous allons ici les décrire un après l’autre :

- 1 cette valeur instaure le mode non sécurisé (insecure mode) de façon permanente, c’est la valeur par défaut du système
- 0 il s’agit là du mode insecure énoncé précédemment, dans lequel on peut retirer les files flags *immutable*¹ et *system append only*². On peut dans ce mode effectuer des opérations de lecture/écriture sur tous les périphériques en fonction des droits définis dans `fstab`.
- 1 est le premier mode sécurisé (secure mode), les files flags qui dans le mode 0 était configurables (*immutable* et *system append only*) ne peuvent maintenant plus être désactivés, on ne peut plus accéder à `/dev/mem` et à `/dev/kmem`³ en écriture et pour finir les LKM⁴ ne peuvent plus être chargés ou déchargés.
- 2 est un mode plus sécurisé que le 1, (highly secure mode) qui est quasiment le même que le précédent, si ce n’est que seul le montage et le démontage sont autorisés sur les disques, pas d’écriture directe (formatage non-autorisé). L’horloge système ne peut pas être modifiée de plus d’une seconde à la fois (afin d’éviter les “sauts” dans le temps qui peuvent éventuellement mener à un DOS par exemple).
- 3 est le dernier, c’est le securelevel le plus “paranoïaque”, (network secure mode), il reprend les règles du mode 2 et en plus interdit de modifier les règles `ipfw`⁵ et tout ce qui consense le trafic sur le réseau.

¹empêcher tout évènement : ni effacer ni modifier

²seulement ajouter des éléments

³images de la mémoire principale de l’ordinateur

⁴Loadable Kernel Module : ajout ou retrait de fonctionnalités du kernel à chaud

⁵firewall

L'utilisateur, ou plutôt l'administrateur doit choisir parmi tout ces modes en fonction de la machine qu'il veut installer (et sécuriser). Il va s'en dire qu'une station de travail n'utilisera pas le même `securelevel` qu'un serveur (une raison toute simple est que XFree86, le serveur graphique a besoin d'accéder à `/dev/mem` alors que le niveau 1 l'interdit). Le niveau 2 très restrictif, mais offrant une sécurité optimale pour un seveur est cependant très complexe à mettre en œuvre du fait de l'impossibilité d'écrire sur les disques.

2.2.2 Mise en œuvre

Définir un `securelevel` est très simple, il suffit de modifier 2 lignes dans le fichier de configuration `/etc/rc.conf` :

fichier `/etc/rc.conf`

```
kern_securelevel_enable="YES"
kern_securelevel="X" # X définit le numéro du securelevel désiré
```

Si l'on veut augmenter le `securelevel`, on peut le faire même en cours d'exécution via `sysctl`⁶ :

```
sysctl -w kern.securelevel=X
```

Par contre pour le baisser, il faut impérativement relancer `init`⁷.

Par défaut (si le niveau n'est pas -1), `init` place le système en mode 0 en mono-utilisateur et en mode 1 en multi-utilisateurs.

2.3 sysctl

2.3.1 Qu'est-ce que sysctl ?

`Sysctl` est une interface permettant d'effectuer des changements de paramétrage sur un système FreeBSD en fonctionnement. Il permet de redéfinir un très grand nombre de paramètres du noyau contenus dans `/proc/sys` à partir d'un fichier de configuration.

2.3.2 Exemples d'utilisation

```
sysctl -a
```

Cette commande liste l'ensemble des paramètres modifiables ainsi que leurs valeurs.

```
sysctl -w net.ipv4.ip_forward=1
```

⁶voir page 15, section 2.3

⁷Sinon quelqu'un gagnant les droits `root` pourrait baisser le niveau de sécurité.

Cette commande passe le paramètre à 1 qui active le routage des paquets entre les interfaces du système.

```
sysctl -p
```

Cette commande chargera le fichier de configuration */etc/sysctl.conf*.

2.3.3 Quelques paramètres utiles

net.inet.tcp.blackhole Par défaut cette OID à la valeur 0, si vous la passez à 1 vous mettez en place le dit trou noir. Vous voilà à l’abri de curieux, car cette OID, limite considérablement l’effet de nmap sur votre machine car elle empêche votre pile TCP/IP de répondre par RST(RESET), quant un port TCP est fermé.

net.inet.udp.blackhole Pareil, ici passé à 1, cet OID évite a votre serveur de répondre un ‘Port Unreachable’ à une requête sur un port UDP fermé.

security.bsd.see_other_uids Par défaut cette OID à la valeur 1, si vous la passez à 0 vous empêchez les autres utilisateurs, sauf le root, de voir les processus qui ne sont pas à eux.

net.link.ether.inet.log_arp_wrong_iface Par défaut cette OID à la valeur 1, si vous la passez à 0, elle permet de ne pas logger d’erreur quant on utilise deux ou plus interfaces Ethernet sur la même machine, sur le même réseau Ip, très utile pour les jails.

2.3.4 Conclusion

En conclusion, `sysctl` s’avère être une commande très utile car elle permet de retrouver très rapidement des paramètres systèmes importants, qu’ils agissent sur la sécurité mais également sur les performances de la machine. Cet outil permet d’éviter de longues heures de configuration !

2.4 non-exec stack (& heap)

2.4.1 Présentation

Le “buffer overflow” existe depuis toujours. Il a gagné sa notoriété en 1988 avec le ver internet de Morris. Malheureusement, la même attaque basique est toujours efficace aujourd’hui. Le type le plus commun de “buffer overflow” est la corruption de la pile. La plupart des systèmes modernes utilise la pile pour passer les paramètres des procédures et stocker les variables locales.

2.4.2 La politique de BSD

NetBSD impose son système de non-exécution de la pile sur plusieurs plateformes[1]. Il permet de mettre la pile en mode non-exécutable quand celle-ci est en écriture, ce qui rend l'exploitation du "buffer overflow" difficile, les autres BSD peuvent se comporter de la même façon mais il est nécessaire d'appliquer des patches⁸, par exemple *Stephanie* pour OpenBSD[3], traité en page 26, section 3.3.

Depuis la version 2.0, NetBSD supporte les non-exécutions de pile sur les plateformes où le matériel l'autorise. Les processus utilisant la pile (stack) et le tas (heap) sont en non-exécutable par défaut, rendant l'exploitation du "buffer overflow" difficile. NetBSD supporte également la permission `PRO_EXEC` via `mmap()` pour toutes les plateformes où le matériel différencie les accès de données et les accès d'exécutions. Aucune option est nécessaire pour l'activer, cela est toujours possible.

Exemples de buffer overflow :

source1.c

```
char shellcode[] =
"\xeb\x2a\x5e\x89\x76\x08\xc6\x46\x07\x00\xc7\x46\x0c\x00\x00\x00"
"\x00\xb8\x0b\x00\x00\x00\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80"
"\xb8\x01\x00\x00\x00\xbb\x00\x00\x00\xcd\x80\xe8\xd1\xff\xff"
"\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00\x89xec\x5d\xc3";

int main() {
    int *ret;

    ret = (int *)&ret + 2;
    (*ret) = (int)shellcode;
    return 0;
}
```

source2.c

```
char shellcode[] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

int main() {
    int *ret;

    ret = (int *)&ret + 2;
    (*ret) = (int)shellcode;
    return 0;
}
```

⁸Mise à jour du système d'exploitation

Ces exemples simples permettent de tester si la pile d'un système de type UNIX dispose d'un type de protection. Sur les BSD que nous avons testé ces programmes échouent. Sur une mandrake 10 par contre nous arrivons à obtenir un shell.

2.4.3 Les protections

Niveau noyau

Casper Dik et "Solar Designer" ont développé des patches pour Solaris et Linux afin de rendre la pile non exécutable et plus précisément à problème de la pile. Ces patches créent un espace d'adressage virtuel non exécutable pour les processus de l'utilisateur, afin que les codes "d'attaques" ne soient pas exécutés. Ils offrent des avantages comme aucune perte de performances et une protection sans re-compilation.

Le patch développé par "Solar Designer" permet de rendre la pile exécutable par détection de l'usage qui en ait fait. Le patch traite avec le signal de l'utilisateur en permettant une exécution sur la pile seulement pendant la durée du signal. Mais ces compromis permettent des intrusions potentielles par exemple une vulnérabilité du "buffer overflow".

Niveau programme

Une utilisation complémentaire : La protection "non-exetutable stack" s'entrecoupe avec celle proposée par StackGuard ; il y a des attaques qui peuvent contourner chaque technique mais elles seront attrapées par l'autre. Par exemple, un hacker peut contourner le "non-exetutable stack" en mettant son code dans le tampon séparé de la pile et utiliser simplement la pile pour repointer l'adresse de retour de son code sur le tampon. Cette forme d'attaque est stoppée par StackGuard.

Inversement, une attaque qui contournerait la protection "StackGuard" en utilisant le "buffer overflow" pour changer des pointers d'adresses d'un programme excepté l'adresse de retour, tel que les pointers de fonctions et les "longjmp buffer", qui n'ont pas besoin d'être sur la pile sera stoppée alors par la protection "non-executble stack" si elle est dans le buffer de la pile. De telles attaques sont relativement rares, mais existent.

Fonctionnement : Le patch StackGuard détecte et combat les attaques de pile en protégeant l'adresse de retour sur la pile. Le patch met un mot "canary" à coté de l'adresse de retour quand une fonction est appelée. Si le mot a été changé quand la fonction "re-viens", alors une attaque a été détectée, et le programme répond en émettant une alerte de l'intrusion dans le syslog et s'arrête.

StackGuard est implémenté comme un petit patch au générateur de code gcc, spécialement les `function_prolog()` et `function_epilog()`. `function_prolog` place les mots "canaries" dans la pile quand les fonctions commencent et `function_epilog` vérifie l'intégrité

de ces mots à la sortie des fonctions. Toutes les tentatives de corruption de l'adresse de retour sont donc détectées avant le retour de la fonction.

Recompiler vos applications avec StackGuard est un moyen efficace de lutter contre les “buffer overflow”.

GCC “proactive” : Il s'agit d'une extension de gcc. Celle-ci possède une option qui permet de protéger la pile par défaut. Il est possible de vérifier l'état de protection avec la commande `gcc -v` avec les versions 2.95.3, 3.3.1, 3.4.1. De plus, ce patch est valable depuis FreeBSD 4.3.

Cette extension permet de protéger la pile contre les attaques. Les applications écrites en C seront protégées par la méthode qui insérera automatiquement le code de protection dans l'application avant la compilation. La protection sert à détecter les “buffer overflow” et la corruption des pointeurs de variables. L'idée générale de détection des “buffer overflow” est la même que celle du patch StackGuard.

Le dispositif de protection se découpe en trois parties :

- la répartition des variables locales dans le buffer avant les pointeurs afin d'éviter leurs corruptions qui pourraient modifier les emplacements de la mémoire arbitraire.
- la copie des pointeurs d'arguments des fonctions avant les variables locales dans le buffer afin d'éviter leurs corruptions qui pourraient modifier les emplacements de la mémoire arbitraire.
- l'omission des fonctions qui diminuerait les performances.

Cette extension est utilisée comme un langage de traduction intermédiaire de gcc.

Les options de compilations `-fstack-protector`, `-fno-stack-protector` servent à activer ou désactiver la protection. Les options de compilations `-fstack-protector-all`, `-fno-stack-protector-all` servent à activer ou désactiver la protection pour toutes les fonctions et pas seulement les fonctions ayant des tableaux de caractères.

Par librairies

Les mécanismes de protection basés sur la compilation sont complètement inutiles pour les programmes que vous ne pouvez pas compiler. Dans ce cas, il existe plusieurs librairies qui ré-implémentent les fonctions dangereuses de la bibliothèque C (`strcpy`, `fscanf`, `getwd`, etc..) et assure que ces fonctions ne peut jamais écrire devant le pointer de la pile. En effet, un programme qui se veut “sécurisé” peut se lier avec la librairie de son choix mais il est parfois possible de remplacer de façon transparente tous les appels à la `libc`⁹. Malheureusement ces méthodes de défenses ont de nombreux défauts, car elles protègent contre des

⁹La librairie de base d'un unix, à peu près tous les programmes l'utilisent directement ou indirectement pour faire des affichages à l'écran, allouer de la mémoire, etc. . .

petits problèmes de sécurité et elles ne fonctionnent pas si les applications sont compilées avec l'option `-fomit-frame-pointer`.

Voici quelques exemple de bibliothèques : `libsafe`, `libverify`, `libparanoia`.

La plus répandue `libparanoia` est une bibliothèque qui intercepte les appels à des fonctions présumées sensibles et les remplace par des fonctions similaires en tous points et fonctionnalités, à la seule différence que ces fonctions sont modifiées pour empêcher toutes tentatives de corruption de la pile mémoire ce qui permet de nous prévenir des failles de type “`stack overflow`” ou `return-into-libc` très documentées et largement répandues ces dernières années.

Des bases fragiles : Les fonctions ANSI C présumées sensibles sont `strcpy`, `strcat`, `gets`, `sprintf` et `scanf` qui sont connues pour être souvent exploitées à des fins illégitimes. Le problème essentiel vient du fait que le langage C n'inclut aucune technique native de vérification lorsque nous manipulons des variables dans une mémoire tampon permettant à l'occasion d'écrire dans cette mémoire afin d'y faire exécuter diverses commandes pouvant entraîner jusqu'à la compromission du compte `root`. L'idée de base de `libparanoia` est de ne jamais exécuter de nouvelles instructions suite à ces fonctions si la pile a été modifié. On préférera alors tuer le processus ou appeler une fonction d'exit. La méthode peut paraître brutale mais ceci suppose une corruption de la mémoire tampon de ce programme et dans ce contexte entraver un instant le bon déroulement d'un programme ou d'un unique processus parait bien moins grave que de risquer la compromission complète du système.

Chapitre 3

Qualité

3.1 Honeyd

3.1.1 Présentation de honeyd

Honeyd est un démon créé par Niels Provos, utilisé pour la création de honeypots, voire de honey-net sur une plus grande échelle. Les honeypots constituent une arme de plus en plus utilisée afin de combattre les attaques des hackers. Un honeypot est une station factice, créée par un démon, et simulant l'activité d'un ordinateur sur un réseau. Pour comprendre les fonctionnalités avancées de ce genre de technologie, nous allons étudier en détail les caractéristiques du démon honeyd.

3.1.2 Caractéristiques de honeyd

Le premier but de honeyd est la détection, et plus spécialement de détecter une activité non réglementaire sur un réseau. Pour cela, il vérifie des interactions (tentatives de connexion) avec des machines qui n'existent pas. En effet, si aucun système sur un réseau ne possède une adresse IP spécifique mentionnée sur un paquet, celui-ci peut être considéré comme étant possiblement l'attaque ou la préparation à l'attaque d'une personne étrangère.

Honeyd vérifie l'activité sur des adresses IP normalement non attribuées. Si une requête est faite sur une de ces adresses, honeyd va automatiquement prendre l'identité de cette adresse non utilisée et interagir avec la machine source de cette requête. La plupart du temps, lorsque honeyd génère une alerte, il y a de très fortes probabilités que l'attaque soit bien réelle.

Par défaut, honeyd peut détecter et logger une activité sur n'importe quel port TCP ou UDP, et certaines requêtes ICMP. Le plus grand avantage de honeyd réside dans le fait qu'il est capable de simuler un service afin d'interagir avec une personne malveillante. Il est par exemple capable de simuler l'activité d'un routeur CISCO sur une machine n'ayant

rien à voir avec ce type de matériel. En plus de “répondre tel un routeur”, des service comme un faux telnet peuvent être créés par un simple script. Il est aussi possible dans certaines configurations de laisser l'accès à un réel service qui est rendu sans risque par l'utilisation de systrace (traîné en page 4, section 1.1) par exemple.

Exemple d'interaction lors de la connexion au port 23 de la machine d'adresse 192.168.1.150 qui est en fait un simple ordinateur utilisant honeyd, arpd (explication dans la section suivante) configuré afin d'émuler un routeur CISCO ainsi qu'un service telnet.

```
attacker $telnet 192.168.1.150
Trying 192.168.1.50...

Users (authorized or unauthorized) have no explicit or
implicit expectation of privacy. Any or all uses of this
system may be intercepted, monitored, recorded, copied,
audited, inspected, and disclosed to authorized site,
and law enforcement personnel, as well as to authorized
officials of other agencies, both domestic and foreign.
By using this system, the user consents to such
interception, monitoring, recording, copying, auditing,
inspection, and disclosure at the discretion of authorized
site.

Unauthorized or improper use of this system may result in
administrative disciplinary action and civil and criminal
penalties. By continuing to use this system you indicate
your awareness of and consent to these terms and conditions
of use. LOG OFF IMMEDIATELY if you do not agree to the
conditions stated in this warning.

User Access Verification

Username: cisco
Password:

% Access denied
```

Cette curiosité de la part d'une personne malveillante sera détectée et loggée par honeyd. Verifions comment honeyd inscrit cette tentative d'intrusion dans un log :

```
Jan 3 11:23:32 marge honeyd[22885]: Connection request: \  
  (192.168.1.10:2783 - 192.168.1.150:23)  
Jan 3 11:23:32 marge honeyd[22885]: Connection established: (192.168.1.10:2783 \  
  - 192.168.1.150:23) <-> /usr/bin/perl scripts/router-telnet.pl  
Jan 3 11:23:47 marge honeyd[22885]: \  
  E(192.168.1.10:2783 - 192.168.1.150:23):  
  
Attempted login: cisco/cisco  
Jan 3 11:23:47 marge honeyd[22885]: Connection dropped \  
  with reset: (192.168.1.10:2783 - 192.168.1.150:23)
```

Ici, nous voyons assez clairement la tentative de connection à un routeur cisco émulé par honeyd, avec des informations sur l'origine de la requête ainsi que l'activité de la machine suspecte.

Une dernière fonctionnalité très importante de honeyd est la possibilité d'émuler un système d'exploitation au niveau noyau. La grande différence avec d'autres honeypots, est la possibilité qu'offre honeyd de contourner les nmap et xprobe des personnes malveillantes en utilisant les bases de données de fingerprint de ces deux applications. Ainsi, `nmap -O` retournera un type de machine qui est la machine émulée par honeyd et non le système d'exploitation réel de la machine.

3.1.3 Configuration de honeyd

2 outils sont nécessaires : `arpd` et `honeyd`. Honeyd ne peut pas tout faire seul, il a besoin de `arpd` afin d'écouter sur les adresses non utilisées et de renvoyer les attaques vers honeyd. Honeyd ne réalise en lui-même que les interactions avec les attaquants.

```
arpd 192.168.1.0/24  
  
honeyd -p nmap.prints -f honeyd.conf 192.168.1.0/24
```

Pour cet exemple, le processus `arpd` va écouter sur toutes les adresses non utilisées du réseau `192.168.1.0/24`. Toutes les activités concernant ces adresses seront redirigées vers honeyd. En ce qui concerne honeyd, le paramètre `-p nmap.prints` renvoie vers la bases des fingerprints de nmap. Le deuxième paramètre `-f honeyd.conf` est le fichier de configuration de honeyd. Voici un exemple :

honeyd.conf

```
## Honeyd configuration file ##
### Windows computers (default)
create default
set default personality "Windows NT 4.0 Server SP5-SP6"
set default default tcp action reset
add default tcp port 110 "sh scripts/pop.sh"
add default tcp port 80 "perl scripts/iis-0.95/main.pl"
add default tcp port 25 block
add default tcp port 21 "sh scripts/ftp.sh"
add default tcp port 22 proxy $ipsrc:22
add default udp port 139 drop
set default uptime 3284460
### Cisco router
create router
set router personality "Cisco 4500-M running IOS 11.3(6) IP Plus"
add router tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set router default tcp action reset
set router uid 32767 gid 32767
set router uptime 1327650
# Bind specific templates to specific IP address
# If not bound, default to Windows template
bind 192.168.1.150 router
```

Tout d'abord, ce fichier crée des templates, c'est à dire des types de machines que nous voulons émuler. Ces templates définissent les comportements des systèmes d'exploitation que nous souhaitons émuler. Tout d'abord, nous sélectionnons la "personnalité" du template. C'est la simulation de la pile TCP/IP du système qui est concernée.

template default c'est un serveur windows NT service pack 4 et 5

template cisco c'est un routeur Cisco 4500-M utilisant IOS 11.3(6) IP Plus

Par défaut, nous avons sélectionné le rejet des paquets tcp avec `set default default tcp action reset`. De plus, nous avons associé des services factices sur certains des ports, ou encore une action spécifique

add default tcp port 110 "sh scripts/pop.sh" simule un serveur pop sur le port 110

add default tcp port 25 block bloque toute activité sur le port 25

add default tcp port 22 proxy \$ipsrc renvoie la requête vers la machine de l'attaquant!!!

Les paramètres utilisables, repris sur la requête effectuée par la personne malveillante.

\$ipsrc ip source

\$ipdst ip de destination

\$sport port source

\$dport port de destination

Enfin, il est possible d'utiliser une adresse IP spécifique pour une machine virtuelle.

`bind 192.168.1.150 router` signifie qu'une machine virtuelle de type router aura l'adresse 192.168.1.150, toutes les autres machines auront le type default

3.1.4 Conclusion

Nous avons donc pu remarquer l'ensemble des possibilités assez impressionnantes rendues possibles par ce service. Ce type de défense est désormais de plus en plus utilisé sur les réseaux et constitue une barrière de protection supplémentaire non négligeable.

3.2 Veriexec

3.2.1 Utilité

Il sert à vérifier certaines spécifications d'un fichier lorsqu'il est lu ou exécuté. Il empêche à un programme non désiré de démarrer comme un sniffer ou un DDOS. Il permet aussi de détecter un trojan ou un virus qui aura infecté un exécutable et ainsi changer son code. Car les droits donnés à un fichier ne suffisent pas si le code a été détecté s'il a été modifié car il gardera les mêmes droits. Et ainsi on ne sait pas si le programme que l'on croit exécuter est celui qui s'exécute vraiment.

A l'aide d'outils comme Tripwire on peut les détecter. Mais il fonctionne grâce à un démon qui vérifie continuellement si un fichier ne contient pas un virus. Ce qui demande beaucoup de ressources et ainsi diminue les performances de l'ordinateur. Ou alors il faut vérifier fréquemment le contenu du disque sous peine d'avoir un virus.

3.2.2 Fonctionnement

Une empreinte de fichier est un moyen de détecter si le fichier n'a pas été modifié. Son principe est d'effectuer un calcul sur le contenu du fichier ainsi s'il change le calcul ne sera plus le même. Ce calcul est souvent un MD5 sur le fichier ou alors un SHA1¹.

Une liste d'empreinte de chaque exécutable est stockée et le système charge dans le noyau cette liste au démarrage du système pour accélérer la vérification. Au début de veriexec (appelé alors signed exec) cette liste restait sur le disque mais avec cette nouvelle idée la performance de vérification a été améliorée de 70%. Lors de l'exécution d'un fichier, veriexec le vérifie en recalculant son empreinte. Pour cela il recalcule un MD5 sum du fichier et le compare à l'empreinte présente dans le kernel.

¹méthodes de hashage

Mais il se pose toujours le problème des fichiers exécutés dynamiquement dans un script. Pour palier à ce problème “ILS” ont du modifier le code de la fonction `an_open()` pour quelle aussi vérifie le code quelle exécute. Script perl???

3.2.3 Utilisation

Après son installation, on lance le script `/usr/share/examples/verixecctl/gen_sha1`. Celui-ci effectue une recherche des exécutables présents sur le disque puis il effectue une empreinte de chaque fichier exécutable. Ceux-ci sont stockés dans un fichier qu’il faut déplacer (Ex : signature) vers `/etc/` (et de préférence faire une copie sur une disquette pour garder une sauvegarde). Il faudra alors modifier le fichier de configuration du démarrage du système pour y ajouter la ligne suivante :

`/etc/rc.conf`

```
verixec="YES"
```

ou alors

`/etc/rc.local`

```
verixecctl /etc/signature
```

en fonction de votre système.

Cela chargera la liste au démarrage. Il faut mettre le niveau de sécurité du kernel (voir page 14, section 2.2) à 2 pour qui bloque l’exécution de programmes dont on ne connaît pas l’empreinte. Car le niveau 1 ou 0 ne fera que déclencher des warnings.

Par contre l’emploi de X est alors refusé car il utilise la mémoire vidéo `/dev/mem` qu’il modifie. Mais comme `verixec` est utilisé sur un serveur il n’y a donc pas de X à lancer.

3.3 Stephanie

Stephanie est un paquetage permettant de renforcer OpenBSD. Elle ajoute plusieurs caractéristiques de sécurité dans OpenBSD que beaucoup d’administrateurs ou/et utilisateurs aimeraient avoir sur leur système. Stephanie contient également des solutions de prévention et des solutions de réduction des dégâts.

Les caractéristiques suivantes sont apportées par Stephanie pour OpenBSD 3.6 :

les utilisateurs de confiance (Trusted users) : Stephanie vous permet de changer dynamiquement un groupe comme “groupe courant de confiance”. Ainsi il est plus simple de maintenir la confiance d’un groupe en ajoutant ou en enlevant un utilisateur de ce groupe.

VeriExec : Vérification de l’intégrité des programmes exécutés, l’organisation des objets mémoires et l’ouverture des fichiers. Complètement remanié depuis les anciennes

versions, utilisant maintenant les tables de hachage. Vous pouvez consulter la page 25, section 3.2 pour de plus amples informations bien que cette implémentation puisse différer sur certains points.

Les patches d'exécution de confiance (TPE Trusted Patch Executing) : prévention contre les exécutions des fichiers définis comme “non fiable”. (non possédé et “écrivable” par le root seul).

Les processus privés (Process privacy) : Prévention contre les processus qui veulent obtenir des informations sur d'autres processus où le propriétaire est différent.

Les secrets de l'utilisateur (Userland privacy) : Protection contre les utilisateurs qui veulent obtenir des informations sur d'autres utilisateurs en ligne.

3.4 Portaudit / Audit-packages

3.4.1 Présentation

Le système NetBSD est, comme nous le savons tous, probablement le plus portable du monde. De cette volonté d'être portable est né le projet *pkgsrc*[12], le système de packages de NetBSD. Ce système de packages fonctionne pour toutes les architectures supportées par NetBSD mais il ne s'arrête pas là. Il fonctionne sur la majorité des systèmes d'exploitation, de Linux aux autres BSD en passant par Windows. Le but premier de l'équipe de NetBSD est d'être portable mais beaucoup d'autres éléments sont traités de façon aussi importante, nous pourrions résumer pour simplifier que tout ce qui s'associe à la qualité du logiciel tiens à coeur aux développeurs de NetBSD.

Un des éléments les plus importants faisant la qualité d'un logiciel est la sécurité. Pour le noyau et le système de base il existe des listes de diffusions pour les rares security advisory² paraissant (comme il y en a très peu, ces failles sont aussi annoncées sur la page d'accueil du site lorsqu'elles sont détectées³

3.4.2 Fonctionnement

Un manque se faisait alors sentir, sur la majorité des systèmes la plupart des programmes ne font pas partie du système de base mais proviennent plutôt de *pkgsrc*. Il fallait donc un moyen efficace de vérifier si les programmes installés ne comportent pas de failles.

²alertes de sécurité

³en général elles sont détectées bien avant qu'un exploit ne soit diffusé (ceci est du à la non trivialité des failles trouvées).

Vous y avez pensé, l'équipe NetBSD l'a fait

Comme vous commencez à vous en douter, `audit-packages` et l'outil qui remplit ces fonctions, il est en général utilisé pour effectuer un audit de sécurité (basique) régulier sur le système afin de reporter les failles éventuelles. Une base de donnée de failles doit être régulièrement mise à jour, celle-ci est maintenue par une équipe NetBSD ou de façon indirecte par des contributeurs éventuels.

Utilisation automatisée

Après avoir téléchargé la dernière version de cette base de données (cette tâche est automatisée⁴) il est possible à tout moment de vérifier les failles du système. Le nom de chaque paquet vulnérable est affiché, suivi du type de faille et d'un lien pour avoir plus d'informations sur la faille de ce programme. Le type de faille permet assez facilement de se rendre compte de la gravité de la faille et donc de l'urgence associée.

Utilisation implicite

Notez également que lors de l'installation d'un programme en utilisant `pkgsrc` le système s'arrêtera si il doit installer un programme vulnérable, un message explicite vous indique la démarche à suivre si vous souhaitez l'installer tout de même (dans le cas d'une machine non connectée à un réseau ou de façon limitée par exemple).

Une idée reprise

Bien que `pkgsrc` fonctionne sous ce système aussi, l'équipe de FreeBSD a préféré porter un outil similaire pour leur système de paquets⁵ natif. Le nom choisit est `portaudit` et à un fonctionnement identique.

⁴À l'installation la ligne à ajouter dans la crontab du système est donnée

⁵appelé ports

Chapitre 4

Confidentialité

4.1 /etc/ttys et login.conf

4.1.1 Notion de classes

`nomDeClasse|option=valeur :booleen :variableDEnvironment :`

Ce fichier définit des classes et des restrictions sur celle-ci. Une classe permet de créer un ensemble d'utilisateur comme les groupes au sens UNIX. Il est possible de faire hériter une classe grâce à l'option `tc=nomDeClasses` et ainsi récupérer les propriétés.

Il existe une classe, `Default`, qui sont automatiquement attribués les utilisateurs qui n'appartiennent pas à un groupe. Il est aussi possible. on peut aussi créer une classe `root` pour limiter les ressources du `root` et augmenter les protections de son authentification.

Un utilisateur peut aussi si il le désire créer un fichier `login.conf` qui décrira la classe "me" et ainsi se limiter lui-même mais il faut que les règles ne vont pas à l'encontre des règles définies dans `login.conf`. Mais cette fonctionnalité est surtout utilisée pour définir des variables d'environnement.

4.1.2 Sa fonctionnalité

Ce fichier permet de faire des limites de ressources, initialiser des variables d'environnement, définir des règles d'authentification, et comptabilité. Ainsi chaque classe aura ses propres règles. Nous vous recommandons l'excellent document écrit par eberkut qui couvre ce sujet[7].

limiter des ressources

cputime limite du temps CPU.

datasize limite de la taille du segment data d'un processus

maxproc limite le nombre d'utilisateur de cette classe

memoryuse limite la taille disponible pour un processus dans la RAM et SWAP

openfiles limite le nombre de fichiers que peut ouvrir un processus

sbsize limite la quantité mémoire des sockets buffers, c'est à dire pour les transmissions sur le réseau. cela empêche certain DDoS comme le SYN floods.

stacksize limite la taille maximale de la pile d'un processus

filesize limite la taille maximale d'un fichier de l'utilisateur.

vmemoryuse limite la quantité de mémoire virtuel attribué à un processus.

Chaque une de ces options peut être suffixé par max (limite hard) et cur (limite soft). Sans précision la valeur sera pour max et cur. Ces options doivent être suivi d'un égal et de la valeur et de son unité (b, kb, mb, gb, tb, ou m pour minute, s pour seconde).ex : `datasize=1kb` :

Variables d'environnement

LANG spécifie la langue

manpath spécifie le chemin d'accès défaut des man.

nologin spécifie le chemin d'accès à un éventuel fichier 'nologin' qui sera afficher si une tentative d'ouverture de session abouti sur /sbin/nologin.

requirehome attribut booléen, permet de requérir un répertoire /home valide afin d'autoriser l'ouverture de la session. Ceci permet d'ajouter une couche de sécurité en obligeant chaque utilisateur à passer par un processus d'authentification complet.

priority définit la priorité initiale des processus lancés par les utilisateurs de la classe. Permet le classement par priorité de processus suivant les classes.

setenv permet de spécifier une liste de variables d'environnement séparées par des virgules.

shell indique le shell à exécuter après l'authentification d'un utilisateur. Cet attribut est prioritaire sur celui indiqué dans /etc/passwd.

term détermine le type de terminal

umask spécifie la valeur du masque de création de fichier.

timezone permet d'indiquer la valeur de la variable \$TZ (fuseau horaire).

welcome définit le fichier contenant le message de bienvenue.

Les variables d'authentification

minpasswordlen permet d'imposer une longueur minimale pour un mot de passe.

mixpasswordcase entraînera la génération d'alertes de la part de passwd, si le mot de passe ne comporte que des minuscules, le rendant faible et vulnérable aux attaques par dictionnaire.

passwd_format permet de spécifier l'algorithme utilisé pour le stockage des mots de passe.

Md5 est conseillé mais il existe aussi 'blf' pour utiliser Blowfish.

login-backoff indique le nombre de tentatives de login infructueuses au cours d'une unique connexion avant que login ne commence à introduire des délais entre chaque tentatives afin de réduire l'efficacité d'une attaque par brute-force.

login-tries comme son prédécesseur sauf que lui ferme la connexion à la fin.

ttys.allow permet de spécifier une liste de tty utilisables par les membres d'une classe.

ttys.deny à l'inverse, la liste de tty ne seront pas utilisables.

host.allow permet de spécifier une liste d'hôtes (adresses IP ou nom d'hôte) depuis lesquels on autorise les logins.

host.deny permet de banir une liste de d'hotes.

times.allow permet de spécifier les horaires d'autorisation de connexion. EX MoWeFr0300-2000 => autorise un accès le lundi(Mo), mercredi(We), vendredi(Fr) de 3H à 20H.

times.deny et inversement il est prioritaire sur times.allow

comptabilité

accounted permet d'activer ou non les options de comptabilité pour cette classe.

passwordtime permet de définir la durée de validité des mots de passe

warnpassword permet d'indiquer quand il sera nécessaire d'avertir l'utilisateur avant d'atteindre passwordtime.

expireperiod spécifie la durée au bout de laquelle un compte lié à une classe expire.

warnexpire vous pouvez décider combien de temps avec l'expiration de son compte un utilisateur sera prévenu.

graceexpire permet d'indiquer pendant combien de temps un utilisateur peut encore se connecter avant l'expiration définitive.

autodelete permet de définir la durée après expiration au bout de laquelle un compte expiré est effectivement supprimé. L'expiration du compte n'implique en effet aucunement sa suppression automatique.

daytime, weektime et monthtime permettent quant à eux de spécifier la durée maximale de connexion d'un membre d'une classe, à l'échelle du jour, de la semaine et du mois respectivement.

wartime définit le moment auquel il un utilisateur est averti, avant que le temps de session n'ait expiré.

gracetime permet de laisser un sursis à ce même utilisateur avant de couper la connexion. Ces valeurs sont exprimées en temps avant que la limite ne soit atteinte.

sessiontime spécifie la durée maximale que peut atteindre une session.

sessionlimit spécifie le nombre maximal de sessions concurrentes, en limitant le nombre de tty simultanément ouverts possible.

▷ `ttys.accounted`, `ttys.exempt`, `host.accounted`, `host.exempt` ont les mêmes sens et syntaxes que les variables d'authentification `ttys.*` et `host.*`, mais pour la comptabilité.

Après chaque modification de `login.conf`, il est nécessaire de mettre à jour la base de données des utilisateurs afin de mettre en place la nouvelle politique, grâce à `cap_mkdb`, comme ci-dessous :

Mise à jour de la base de données

```
# cap_mkdb /etc/login.conf
```

4.2 Cryptage des disques

4.2.1 Présentation

Parler aussi de la solution sous OpenBSD : un

```
sysctl vm.swapencrypt.enable = 1
```

pour crypter le swap + un autre truc pour crypter les disques “normaux” mais ya rien de standard, donc on en parle pas, j’ai ajouté une petite remarque pour la crypto sous FreeBSD.

Lien convivial (pour parler des quelques autres soluces pas vraiment “de base”) : ▷ http://www.infoanarchy.org/wiki/index.php/Hard_Disk_Encryption

La confidentialité est un niveau de la sécurité très important. Protéger ses données contre des intrusions, dont le but est récupérer ses mêmes données, est un point essentiel dans une démarche de sécurisation d’un système informatique. Un des moyens de limiter cette récupération est de rendre illisible les données pour toute personne étrangère au système.

C’est dans cette optique que les systèmes BSD disposent d’outils qui cryptent les partitions. De plus, un des avantages de cela est de pouvoir crypter la partition swap du système. La partition swap est une partition du disque dur sur laquelle est stockée une partie de la mémoire vive lorsque celle-ci est saturée. Cette partition est donc très importante car elle contient de nombreuses informations utilisées par le système. Il peut être donc intéressant de ne pas laisser ces données lisibles s’il y avait une intrusion dans notre système.

Nous ne parlerons pas de la solution de FreeBSD “GBDE” qui est indiquée comme peu testée et plutôt expérimentale.

4.2.2 La mise en place

Ainsi depuis NetBSD 2.0, il existe un utilitaire qui code une ou des partitions choisi par l'utilisateur : CryptoGraphic Disk Driver (CGD). Celui permet de coder la partition en utilisant différents algorithmes qui ne seront pas détaillés ici.

Tout d'abord il faut générer le fichier de configuration qui sera utilisé par le système pour crypter la partition. La commande `cgdconfig -g` permet la création du fichier. L'option `-o nomFichier` est le chemin du fichier créé. L'option `-V none` spécifie la méthode de vérification. L'option `-k randomkey`, la méthode de génération de la clé et enfin `aes-cbc`, l'algorithme de cryptage.

Voici un exemple de création du fichier de configuration :

```
cgdconfig -g -V none -k randomkey aes-cbc
```

Le mode `randomkey` permet de générer une clé de cryptographie différente à chaque démarrage de la machine. Cette technique a aussi comme avantage de ne pas faire connaître cette clé. Cependant il existe d'autres modes, tel le mode `storekey` qui stocke la clé dans le fichier de configuration. La méthode de vérification, quant à elle, permet de déterminer si la clé est correcte. Par défaut, le fichier de configuration est `/etc/cgd/cgd.conf`. Il est aussi possible de spécifier la taille de la clé à générer.

Voici un exemple pour spécifier le nom du cgd et le device associé (la partition) :

```
cgdconfig cgd0 /dev/wd0e
```

où `/dev/wd0e` est la partition swap et `cgd0` est le nom du cgd.

Lorsque le système démarrera, il utilisera le cryptage de données suivant les configurations citées. L'option `-U` permet de désactiver le cryptage.

4.3 OTP

4.3.1 Le principe

Lors du processus de sécurisation d'un système, l'élément principal est d'éviter toute intrusion. Or l'authentification est un des points d'entrée d'un système. Il s'agit donc d'apporter une sensibilisation toute particulière à l'authentification.

Les OTP (One-Time Passwords) apportent une sécurité accrue lors de l'authentification, notons cependant que pour FreeBSD 5 et supérieur OTP a été remplacé par OPIE¹, la démarche est plus ou moins la même mais le nom des commandes change, consultez le handbook[16] ou la page de manuel[15] pour les détails.

Une technique des pirates pour accéder à une machine est de capter les trames qui circulent sur un réseau. Il pourra aisément récupérer des données confidentielles comme

¹One-time Passwords In Everything

des mots de passe dans une séquence de login et la rejouer pour pouvoir s'authentifier sur la machine.

Le principe des mots de passe uniques et non réutilisables est de ne pas pouvoir permettre à quelqu'un d'utiliser un mot de passe qu'il a récupéré s'il a déjà été utilisé. Le fait que le mot de passe ne s'utilise qu'une seule fois réduit considérablement le fait de pouvoir accéder à une machine sans en avoir l'autorisation.

Le one-time password a un fonctionnement simple : il permet à un client de générer de façon sûre un mot de passe nouveau et différent grâce à un dialogue avec la machine sur laquelle il veut se connecter. Ce système est basé sur une fonction de hachage irréversible.

Le système permettant la génération de OTP est appelé **S/Key**. Il est inclus en standard dans les distributions de FreeBSD, OpenBSD et NetBSD, mais est de plus en plus utilisé par d'autres systèmes.

4.3.2 Le fonctionnement

Pour utiliser l'authentification avec OTP, l'utilisateur a besoin de plusieurs mots de passe : le premier est le mot de passe de base unix (utilisé habituellement pour s'identifier), le suivant est le mot de passe non réutilisable qui sera généré par le programme, et enfin le mot de passe secret qui servira à obtenir le mot de passe non réutilisable. Celui-ci représente une suite de six mots anglais court qui permet de pouvoir être mémorisée facilement.

Initialisation depuis une connexion sécurisée

Tout d'abord, l'utilisateur va devoir initialiser le système S/Key. Pour cela, il utilise la commande `keyinit` sur une connexion sécurisée. Après avoir demandé le mot de passe secret, le programme vous donne le mot de passe non réutilisable.

Initialisation depuis une connexion non sécurisée

Si vous n'utilisez pas de connexion sécurisée, vous pouvez utiliser la commande d'initialisation `keyinit -s`. Vous fournissez alors la valeur du compteur d'itération et la valeur du "grain de sel", clé permettant la génération des mots de passe. Tapez Entrée pour utiliser le "grain de sel" aléatoire du système.

Puis passez sur une connexion sécurisée et utilisez la commande `key` avec les mêmes paramètres que précédemment (valeur du compteur et "grain de sel"). Le programme vous donne alors votre mot de passe non réutilisable.

exemple de mot de passe non réutilisable :

```
HULL NAY YANG TREE TOUT VETO
```

Connexion à partir d'une invite de session

Lors de la connexion par une invite de session sur une machine via telnet par exemple, si la machine a été configurée pour l'utilisation de mot de passe non réutilisable², l'invite vous donne le compteur et le "grain de sel" puis vous demande votre mot de passe. Pour pouvoir générer le mot de passe, nous utilisons une machine sûre et le programme `key` (paramètres : compteur et "grain de sel" récupérés de l'invite de session précédente). Celui-ci nous donne notre mot de passe non réutilisable que l'on utilisera dans l'invite de session précédente.

Enfin, il est possible de générer plusieurs mots de passe en même temps si vous n'avez pas de machine sécurisée à disposition. Pour cela, utilisez la commande

```
key -n nbPasswd rang grainDeSel
```

, où `nbPasswd` est le nombre de mot de passe à générer, `rang` le rang de la dernière itération et `grainDeSel`, votre "grain de sel" vu plus haut.

4.4 OpenSSH

4.4.1 Historique

Depuis très longtemps les utilisateurs de station UNIX avaient la volonté de transférer des fichiers entre des machines voir même d'exécuter des commandes à distance comme si il s'agissait d'une utilisation locale. Ces diverses taches étaient possibles en utilisant divers programmes dédiés, par exemple FTP pour le transfert de fichiers et telnet pour l'exécution distante. Une suite logicielle est alors venue en aide pour répondre à tous les besoins à la fois : les "r" tools³.

Les "r" tools

Un même package concentrait divers petits outils indispensables à l'utilisation de multiple machines en réseau. `rcp` permettant de copier des fichiers d'une machine A à une machine B (il se peut qu'aucune des deux machines ne soit la machine locale). `rsh` utilisé pour exécuter quelques commandes à distance et finalement `rlogin` permettant de simuler une réelle session, exactement comme une utilisation locale.

Une solution pratique mais limitée

Les réseaux se démocratisant de plus en plus, la sécurité de la communication devenait un besoin. De plus les r* tools souffraient de certains défauts : pas de compression possible, communication complètement non-sécurisée, bugs dans rsh, etc. . .

²Sous NetBSD et FreeBSD c'est automatique, sous OpenBSD il faut le préciser lors du login

³r pour remote

La solution : SSH

Un programme a alors fait son apparition : SSH. Grossièrement il permettait la même chose que les “r” tools mais de façon sécurisée. Un peu plus tard la politique à changée et le programme n’est pas resté *libre*. La communauté *BSD a réagit et à repris la dernière version libre de SSH pour en faire OpenSSH.

4.4.2 Fonctionnalités

Ce programme est disponible sur toutes les variantes de BSD mais aussi sur un grand nombre d’autres systèmes, même Microsoft Windows, qui n’est pas un UNIX, dispose de multiples versions d’OpenSSH utilisables. Ce programme a largement évolué au cours des années.

Chiffage fort des données

Non-seulement les données sont chiffrées mais vous pouvez choisir le protocole utilisé ainsi que la longueur des clé. Nous remarquerons cependant que l’algorithme 3des est utilisé par défaut tandis que le chiffage blowfish est plus rapide et réputé plus sûr.

Autorisation plus fine

OpenSSH reprends le principe de base des `.rhosts` mais l’étends de façon bien plus adaptée. Il est possible de fonctionner par clé, ainsi seul la personne disposant de la bonne clé pourra se connecter. De nombreux paramètres sont personnalisables “par clé” pour n’autoriser par exemple que certaines commandes. Il existe en réalité plusieurs protocoles, la version 2 est la meilleure à l’heure actuelle.

Configuration transparente de X11

Il est désormais possible d’exécuter des applications graphiques de façon totalement transparente. Plus besoin de modifier des variables d’environnement ou les droits d’accès aux `DISPLAY`. Si la configuration du démon ssh l’autorise un client pourra lancer une application graphique sans manipulation supplémentaire.

Plus adapté pour les lignes bas débit

La compression est maintenant une simple option, le niveau de compression est configurable afin de trouver le bon compromis en fonction de la ligne utilisée. Même la communication graphique est compressée ce qui permet d’éviter la mise en place d’une solution de type VNC dans bien des cas.

Crypte des protocoles non chiffrés

L'ajout d'une fonction "tunnel" dans OpenSSH permet de former un tunnel crypté d'une machine à une autre. Cette mise en place est triviale et permet par exemple de crypter une connections VNC ou SMTP en une seule ligne de commande. Nous remarquerons que certains programmes prévoient l'interaction avec ssh afin de sécuriser leur utilisation (par exemple : cvs).

Souplesse de configuration

Enfin nous n'allons pas tout citer dans ce document mais la configuration d'OpenSSH est très souple. Elle peut être très permissive ou très restrictive selon vos goûts, il est aussi possible d'avoir des éléments de configuration différents pour différents utilisateurs.

4.5 IPsec

4.5.1 Introduction

IPsec permet de créer des réseaux privés virtuels (VPN), c'est-à-dire de relier entre eux des systèmes informatiques de manière sûres en s'appuyant sur un réseau existant. Contrairement à d'autres (comme les tunnel SSH) cette méthode est standard (facultative en IPv4 mais obligatoire en IPv6). Les avantages supplémentaires sont l'économie de la bande passante, grâce à la compression des en-têtes des données et à l'utilisation de technique d'encapsulation relativement légère, et la protection des protocoles bas niveau comme ICMP, RIP IGMP etc...

4.5.2 Services offerts par IPsec

Les méthodes d'échange IPsec

- Une telle méthode de communication peut fonctionner suivant deux modes différents
- le mode transport : Celui-ci permet une protection des protocoles de niveau supérieur.
 - le mode tunnel : Il permet d'encapsuler des datagrammes IP dans d'autres datagrammes IP, dont le contenu est protégé.

Les protocoles utilisés par IPsec

AH (authentication header) : Ce protocole est considéré comme le plus simple faisant partie de la spécification IPsec

Il garantie :

- L’authentification : l’adresse IP de l’émetteur indiqué dans l’en-tête est bien celle de l’émetteur.
- L’unicité : les datagrammes envoyés ne peuvent être utilisés qu’une seule fois et donc ne peuvent être utilisés par un attaquant ayant intercepté le datagramme.
- L’intégrité : il garantit que les principaux champs du datagrammes IP reçus n’ont pu être modifiés durant l’émission (à l’exception de certains champs dont la valeur ne peut être prédictible lors de l’émission).

AH ne spécifie pas d’algorithme de signature particulier, ceux-ci sont décrits séparément.

Le protocole AH n’assure aucune la confidentialité des données, celle-ci sont signés mais non chiffrées.

ESP (encapsulating security payload) : Contrairement à AH, ESP ne protège pas les en-têtes des datagrammes IP utilisés pour transmettre la communication. Seules les données sont protégées. En mode transport, il assure :

- La confidentialité des données : celles-ci sont chiffrés.
- L’authentification : la partie de données émise est bien issu de l’hôte avec qui l’échange IPsec a lieu, en effet ce dernier ne peut s’identifier avec succès que s’il connaît la clé associé à la communication ESP.
- L’unicité : un datagramme déjà émis ne peut être réutilisé
- L’intégrité : les données n’ont pas été modifiées depuis leur émission

En mode tunnel : les même garantie s’applique avec deux avantages supplémentaires :

- Un attaquant observant les flux de données est incapable de savoir exactement le volume de données échangé entre deux hôtes particuliers.
- La confidentialité des données s’étend à l’ensemble des champs, y compris les en-têtes du datagramme IP encapsulé dans le datagramme protégé par ESP.

ESP, de même qu’AH, ne spécifie pas d’algorithme de signature ou de chiffrement particulier, ceux-ci sont décrits séparément.

(Datagramme IP)

4.5.3 Architecture

Fonctionnement de IPsec

IPsec peut être utilisé pour protéger uniquement certain trafic. Sur chaque système utilisant IPsec possède une SPD⁴ qui permet de préciser la politique de sécurité à appliquer au système. Chaque entrée de cette base de données est identifiée par un SPI⁵

Une communication protégée à l’aide d’IPsec est appelée SA (security association). Une SA est unidirectionnelle, ce qui signifie que pour une communication entre deux hôtes, deux

⁴security policy database

⁵security parameters index

SA doivent être activées. Une SA repose sur une unique application de AH ou de ESP, ce qui n'exclut pas pour autant une multiple encapsulation, ce qui nécessite donc, l'activation d'autant de SA nécessaire à la communication. Chaque SA est identifiée de manière unique par un SPI. Les SA actives sont regroupées dans une SAD⁶.

La SPD est consultée pendant le traitement de tout datagramme IP, elle peut soit rejeter le datagramme, soit l'accepter sans traitement IPsec, ou alors appliquer IPsec.

Un administrateur du système peut donc spécifier une politique de sécurité assez fine et détaillée en choisissant d'appliquer un traitement IPsec spécifique pour chaque flux, en fonction du port utilisé, de l'adresse IP etc... Cependant une politique commune pour le trafic vers un ensemble de systèmes permet une meilleure protection.

Architectures supportées

- Dialogue entre deux hôtes protégeant le trafic eux-mêmes.
- Dialogue entre deux LANs à l'aide de passerelles de sécurité.
- Dialogue entre deux hôtes traversant deux passerelles de sécurité.
- Dialogue entre un hôte et une passerelle de sécurité.

4.5.4 Gestion des échanges de clefs

Les services de protection d'IPsec s'appuient sur des algorithmes cryptographiques et reposent donc sur des clefs. Celles-ci peuvent être gérées manuellement dans le cas où peu d'hôtes seront amenés à engager des conversations IPsec. Si le système prend une extension plus importante il est préférable d'utiliser un procédé d'échange de clefs automatiques (comme le protocole IKE⁷ par exemple) qui est aussi considéré comme plus sûr.

4.5.5 Problèmes divers liés à l'utilisation de IPsec

Certains problèmes peuvent apparaître lors de l'utilisation d'IPsec. Tout d'abord les limitations dues à l'éventuelle gestion manuelle des clefs, comme expliqué précédemment. Des problèmes de performances apparaissent aussi lors de l'envoi et de la réception de datagrammes multicast et broadcast. Un autre problème assez important concerne l'utilisation de firewalls. En effet il n'est pas possible au code de firewalling de lire certaines données, par exemple des numéros de port, dans des données chiffrées, ou transmises dans un format qu'il ne connaît pas. Ainsi se pose le problème de savoir si le décodage de l'IPsec doit avoir lieu avant ou après l'application des règles de firewalling. D'autres problèmes apparaissent notamment lors de l'utilisation du NATS, et qui sont en cours de résolution.

⁶security association database

⁷internet key exchange

Enfin un inconvénient d'IPsec et que ce protocole ne prévoit que le convoyage sécurisé de datagrammes IP. Il existe cependant une solution à ce problème : encapsuler les données à protéger dans du PPP, lui-même transporté par IPsec. Le rôle du PPP est en effet de permettre la transmission de différents protocoles au dessus d'un lien existant. Ceci implique de pouvoir encapsuler PPP dans les datagramme IPsec, ce qui est fait par les protocoles PPTP et L2TP.

4.5.6 Conclusion

IPsec est aujourd'hui utilisé sur quelques OS populaire comme Windows, Linux, NetBSD, FreeBSD et bien sûr OpenBSD. IPsec reste assez complexe à mettre en place. Et ne pourrait se résumer à quelques clics pour une bonne configuration. Il reste donc pour le moment réservé à des experts.

Chapitre 5

Conclusion

J'espère que ce document vous a donné un aperçu des possibilités et spécificités des systèmes BSD en matière de sécurité. Vous remarquerez que chacun de ses systèmes a été pensé avec la sécurité en tête. Pour vous en convaincre, regardez les Changelogs d'une version sur une autre, vous constaterez qu'une partie non négligeable des améliorations concerne la sécurité et ce, indépendamment du système BSD considéré.

Tandis que certains systèmes se veulent polyvalents (Windows, Linux ou MacOS X par exemple), se disant simple d'utilisation, rapides et sécurisé, la réalité est tout autre. Certains objectifs prennent le pas sur les autres (sans parler des systèmes commerciaux qui ajoutent à cela la pression des dates de sortie fixe résultant de stratégies commerciales). Les systèmes BSD, bien que pouvant être utilisés à cet effet, ne se focalisent pas sur le multimédia ou le support avant les autres de la nouvelle technologie à la mode. Ils n'ont pas non-plus la vocation d'être utilisable par votre petite soeur qui est en primaire.

Ce sont plutôt des systèmes fait par des experts et destinés aux informaticiens ou passionnés. Les équipes de développement peuvent alors se concentrer sur la qualité du code source (qui a pour effet de bord la vitesse d'exécution et la sécurité) et sur les fonctionnalités vraiment utiles pour un serveur ou une machine de bureau "lambda". Le temps gagné est aussi mis à profit pour écrire des documentations claires et précises mais aussi pour répondre aux demandes des utilisateurs¹

Si vous pensez faire partie des personnes qui pourraient tirer profit d'un tel système je vous conseille d'aller visiter les sites associés et d'y télécharger librement la dernière version.

<http://www.netbsd.org/>

<http://www.freebsd.org/>

<http://www.openbsd.org/>

¹Les systèmes BSD disposent tous d'un outil, de base "send-pr", permettant de reporter un problème et éventuellement de rentrer en contact avec un développeur. Le temps de réaction suite une telle requête est étonnamment court.

Bibliographie

- [1] Site officiel de NetBSD
▷ <http://www.netbsd.org>.
- [2] Site officiel de FreeBSD
▷ <http://www.freebsd.org>.
- [3] Site officiel d'OpenBSD
▷ <http://www.openbsd.org>.
- [4] Poul-Henning Kamp & Robert N. M. Watson
Jails : Confining the omnipotent root
The FreeBSD Project, 2000.
▷ <http://docs.freebsd.org/44doc/papers/jail/jail.html>.
- [5] *Xen virtual machine monitor*
▷ <http://www.cl.cam.ac.uk/Research/SRG/netos/xen/>.
- [6] *Xen*
comparaison des performances
▷ <http://www.cl.cam.ac.uk/Research/SRG/netos/xen/performance.html>.
- [7] *Securing FreeBSD step by step*
▷ <http://www.minithins.net/warehouse/FreeBSD.txt>.
- [8] *Systrace*
▷ <http://systrace.org>.
- [9] *Hairy Eyeball*
Configurations pour systrace
▷ <http://blafasel.org/floh/he/>.
- [10] *Gtk-systrace*
▷ <http://www.citi.umich.edu/u/provos/systrace/index.html>.
- [11] *CGU : Gruik Coders United*
Support et promotion des Unices libres
▷ <http://gcu.info>.
- [12] *pkgsrc : The NetBSD Packages Collection*
▷ <http://www.pkgsrc.org>.

- [13] *FreeBSDébutant*
site francophone pour FreeBSD
▷ <http://frebsdebutant.org>.
- [14] *PF : The OpenBSD Packet Filter*
La FAQ sur PF
▷ <http://openbsd.org/faq/pf/>.
- [15] *FreeBSD Hypertext Man Pages*
Pages de manuel pour divers systèmes
▷ <http://www.freebsd.org/cgi/man.cgi>.
- [16] *FreeBSD Handbook*
Manuel d'utilisation de FreeBSD
▷ http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html.