16 sept 2008

Anti-forensics sur systèmes de fichiers ext2/ext3

Catégorie : Sécurité Tags : misc

~~Retrouvez cet article dans : **Misc 18**~~

La première chose qu'un pirate souhaite faire quand il a réussi à rentrer sur une machine, c'est cacher les données qu'il veut laisser. Il y a maintenant ceux qui utilisent les techniques que même ma grand-mère connaît et ceux qui profitent de la structure interne du système de fichiers.
Cet article explique les différentes solutions pour cacher des données dans un système de fichiers ext2/ext3 tout en tenant compte des contraintes qui subsistent. Nous n'aborderons pas cependant le sujet des données effacées et de la manière de les récupérer qui est un tout autre sujet (celui de ne laisser aucune trace :)).

Rappels sur les systèmes de fichiers ext2/ext3

Quelques termes techniques à connaître

Il est nécessaire de connaître quelques termes qui reviendront de manière récurrente tout au long de l'article.
Le terme bloc désigne un bloc logique, c'est-à-dire un regroupement de 2, 4 ou 8 blocs physiques. Ces blocs physiques représentent l'unité du disque dur (il est divisé en blocs, pistes et secteurs) et ont une taille de 512 octets.
Un inode, avec une taille fixe de 128 octets, est un bloc particulier du disque dur regroupant les informations essentielles d'un fichier et assurant la liaison entre ce fichier et le reste du système (issu du jargon français :)).

Structure physique du système de fichiers ext2

Chaque partition ext2 est découpée en blocs de taille identique. Seul le premier bloc réservé pour la partition boot sector n'est pas utilisé par le système de fichiers. Il contient un programme qui permet d'initialiser et de lancer le système. Il a une taille fixe de 1024 octets. Le reste de la partition est organisé en groupes de blocs de taille identique qui eux-mêmes sont structurés de la manière suivante :

- le super bloc ;
- les descripteurs de groupe ;
- la table bitmap d'état d'allocation des blocs du groupe ;
- la table bitmap d'état d'allocation des inodes du groupe ;
- la table des inodes du groupe ;
- les blocs de données.

On répertorie ces structures en deux catégories : les metadata qui concernent le super bloc, les descripteurs et les différentes tables, et les data qui sont par logique les blocs de données. Ce sont sur les metadata que s'appuient les différentes techniques pour cacher des données.

La table des inodes

La table des inodes (dernière structure définie dans un groupe de blocs) est la structure qui nous intéresse le plus. Nous laissons donc de côté les autres.
Il faut savoir que tout est fichier sur un système ext2 et qu'à chaque fichier est associé un inode unique. Cela peut être un répertoire, un lien symbolique... Chaque inode a un numéro unique sachant que les inodes de 1 à 10 sont réservés, l'inode 11 représente le premier inode utilisable :

```
$ cat /usr/include/ext2fs/ext2_fs.h
[...]
/*
 * Special inode numbers
 */
#define EXT2_BAD_INO            1  /* Bad blocks inode */
#define EXT2_ROOT_INO           2  /* Root inode */
#define EXT2_ACL_IDX_INO        3  /* ACL inode */
#define EXT2_ACL_DATA_INO       4  /* ACL inode */
#define EXT2_BOOT_LOADER_INO            5  /* Boot loader inode */
#define EXT2_UNDEL_DIR_INO      6  /* Undelete directory inode */

/* First non-reserved inode for old ext2 filesystems */
#define EXT2_GOOD_OLD_FIRST_INO 11
[...]
```

Parmi les inodes réservés, ~~EXT2_BAD_INO~~ contient les pointeurs de blocs vers les blocs de données qui occupent des mauvais secteurs du disque (les blocs défectueux sont regroupés dans le répertoire ~~/lost+found~~) et ~~EXT2_ROOT_INO~~ est l'inode du répertoire racine de la partition. L'inode ~~EXT2_UNDEL_DIR_INO~~ indique le répertoire contenant les fichiers effacés qui peuvent être restaurés. Ce répertoire est caché sur le système.
Pour connaître le numéro d'inode d'un fichier, il vous suffit d'utiliser l'option ~~-i~~ de ~~/bin/ls~~ :

```
$ ls -ila /
total 32980
      2 drwxr-xr-x  20 root root        4096 2004-11-10 07:00 ./
      2 drwxr-xr-x  20 root root        4096 2004-11-10 07:00 ../
 971521 drwxr-xr-x   2 root root        4096 2005-01-29 00:09 bin/
 129537 drwxr-xr-x   2 root root        4096 2005-01-31 18:57 boot/
1101117 drwxr-xr-x   2 root root        4096 2004-03-27 00:33 cdrom/
 987713 drwxr-xr-x  13 root root       28672 2005-02-01 08:09 dev/
 388609 drwxr-xr-x  94 root root        8192 2005-02-01 11:43 etc/
      2 drwxrwsr-x   4 root staff       4096 2004-12-07 21:57 home/
 663885 drwxr-xr-x   2 root root        4096 2004-03-27 00:33 initrd/
     44 -rw-------   1 root root    33554432 2004-08-17 01:44 .journal
 194305 drwxr-xr-x   7 root root        8192 2005-01-29 00:09 lib/
     11 drwx------   2 root root       16384 2004-03-27 00:14 lost+found/
1036289 drwxr-xr-x   3 root root        4096 2004-06-28 13:51 mnt/
      2 drwxr-xr-x  22 root root        4096 2005-01-31 18:47 opt/
      1 dr-xr-xr-x  54 root root           0 2005-02-01 08:08 proc/
  97153 drwxr-xr-x  24 root root        4096 2005-02-01 11:14 root/
 372417 drwxr-xr-x   2 root root        4096 2005-01-29 00:09 sbin/
 130146 drwxr-xr-x   2 root root        4096 2004-10-13 21:40 sys/
 161921 drwxrwxrwt  10 root root       57344 2005-02-01 14:02 tmp/
```

```
 340033 drwxr-xr-x  14 root root       4096 2005-01-23 11:00 usr/
 615297 drwxr-xr-x  15 root root       4096 2005-01-12 13:43 var/
```

Les répertoires . et .. qui représentent la racine de la partition /~~.~~(partition racine de l'espace de nommage) ont bien l'inode égal à 2. Pourquoi avons-nous aussi ~~/home~~ et ~~./var~~ avec un inode égal à 2 ? Ce sont en fait les points de montage de deux partitions ext2, par conséquent ces répertoires représentent les racines de ces 2 partitions.

Tous ces inodes sont enregistrés dans une table elle-même appartenant à un groupe de blocs. Au niveau physique, une table d'inodes est une continuité de blocs.

Quant à la table elle-même, chaque inode qui la compose est représenté par la structure ~~ext2_inode~~ (toujours dans le fichier ~~linux/ext2_fs.h~~) qui contient toute l'information caractérisant un fichier. Cela concerne notamment le type du fichier, le propriétaire, les permissions et surtout des pointeurs vers les blocs de données (le contenu des fichiers).

Pour terminer la partie sur la table des inodes, un exemple de la sortie de debugfs sur le contenu de la structure inode d'un fichier en fonction de son numéro d'inode :

Un cas particulier d'inode : les répertoires

Les répertoires sont considérés comme un fichier et possèdent donc un numéro d'inode (le répertoire racine a son numéro d'inode prédéfini : ~~EXT2_ROOT_INO = 2~~). Pour chaque répertoire rencontré, ~~ext2_inode.i_mode~~ est égal à ~~EXT2_S_IFDIR~~ dans la table d'inodes et les blocs de données contiennent la liste des fichiers du répertoire et leur numéro d'inodes respectif plutôt que de contenir des données (c'est en quelque sorte, comme le dit si bien ~~the grugq~~, le DNS du système de fichiers). Cette liste est composée de structures ~~ext2_dir_entry_2~~ (~~ext2fs/ext2_fs.h~~).

Différence entre ext2 et ext3

Le système de fichiers ext3 est complètement compatible avec le système de fichiers ext2. Même les outils du package ~~e2fsprogs~~ (package qui contient les outils tels que ~~debugfs~~, ~~fsck~~, etc.) sont réutilisables sur un système ~~ext3fs~~. Pour résumer assez rapidement, le système de fichiers ext3 équivaut à ext2 plus un journal.
Sous ext3, le journal (endroit où sont stockés les logs d'activité de la journalisation) peut prendre deux formes : soit c'est un inode invisible par le système de fichiers, soit c'est un fichier caché ~~.journal~~ à la racine du système de fichiers, avec un attribut « immuable ». Tout dépend en fait de la création du système de fichiers.
Les informations enregistrées dans le journal dépendent des paramètres de montage de la partition ~~ext3~~. Pour plus de détails, reportez vous au document ~~[ext3]~~. Ce qu'il faut retenir surtout, c'est que vous avez trois méthodes pour journaliser vos données :

- Seules les structures ext2 sont enregistrées dans le journal et écrites sur le disque après les données.
- Seules les structures ext2 sont enregistrées dans le journal mais elles peuvent être écrites sur le disque avant les données.
- Tout est sauvegardé dans le journal (structure ext2 et données).

Au niveau implémentation, il y a entre autres comme modification deux nouveaux inodes réservés dont un qui concerne la journalisation (cf. l~~inux/ext3_fs.h~~) :

```
# ls -ialp /tmp/blaat
162599 -rw-r--r--  1 compaq compaq 6 2005-02-01 19:17 /tmp/blaat
# debugfs /dev/hda1
debugfs 1.35 (28-Feb-2004)
debugfs:  stat <162599>
Inode: 162599  Type: regular   Mode:  0644   Flags: 0x0   Generation: 2017214494
User:  1000   Group:  1000   Size: 6
File ACL: 0    Directory ACL: 0
```

```
Links: 1   Blockcount: 8
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x41ffc7a9 -- Tue Feb  1 19:17:13 2005
atime: 0x41ffc7a9 -- Tue Feb  1 19:17:13 2005
mtime: 0x41ffc7a9 -- Tue Feb  1 19:17:13 2005
BLOCKS:
(0):330468
TOTAL: 1
```

Cacher les données

Quel que soit le système de fichiers sur lequel vous souhaitez cacher des données, celles-ci ne doivent être détectées d'aucun outil forensics pour qu'elles soient efficaces. C'est évident mais cela sous-entend que vos données doivent être cachées à
des endroits du système de fichiers que les outils forensics n'analysent pas.
À cette condition s'ajoute le fait que le système de fichiers doit rester sans anomalie pour d'une part éviter d'alerter le système (nous verrons par quoi plus tard) et par conséquent l'utilisateur et d'autre part pour éviter de perdre ces fameuses données cachées.
D'une manière générale, pour que la solution fonctionne pleinement, les données cachées doivent rester présentes sur le système. Un exemple simple voire très stupide : Jean-Kevin cache ses données dans le fichier '~~/tmp/.jean-kevin~~', mais, manque de chance, le système redémarre et ce système est configuré pour nettoyer le répertoire ~~/tmp~~... Mais bon, on s'appelle pas Jean-Kevin par hasard :)

Les outils forensics

Comme nous venons de le dire, cacher des données ne veut pas dire nommer son fichier ~~/bin/.log~~ ou ~~/usr/man/man1/~~... pour qu'au premier passage de ~~chkrootkit [chkrootkit]~~ (ou un autre), les fichiers soient détectés. Certes ~~chkrootkit~~ ne détecte pas tout, mais avec un peu de temps et de patience, l'analyste post-intrusion ou quiconque arrivera à ses fins. Il pourra par exemple faire un pattern matching de différentes chaînes de caractères sur tous les fichiers ou plus simplement et avec beaucoup de réussite faire un listing complet du système de fichiers et analyser un à un les fichiers qui semblent douteux.
Des solutions moins fastidieuses existent comme faire une analyse forensics par rapport aux accès des fichiers. Il faut savoir que la structure inode d'un fichier fournit entre autres 3 informations concernant l'accès au fichier :

- ~~atime~~ : modifié par des accès en lecture (~~read()~~, ~~mmap()~~, ~~execve()~~,...)
- ~~mtime~~ : modifié par des accès en écriture ~~(write()~~, ~~truncate()~~, ~~mknod()~~,...)
- ~~ctime~~ : modification au niveau inode (~~chown()~~, ~~link()~~,...)

Souvent, après la création d'un nouveau fichier, le pirate a tendance à vouloir modifier les temps d'accès au fichier pour tromper l'ennemi. La commande ~~touch~~ avec l'option ~~-r~~ ou ~~-d~~ le permet :

```
# touch -r /bin/ls blaat
# ls -alp /bin/ls
-rwxr-xr-x  1 root root 75948 2004-07-16 13:37 /bin/ls
# ls -alp blaat
-rw-r--r--  1 compaq compaq 6 2004-07-16 13:37 blaat
```

Seulement, quelle que soit l'option utilisée, le jour de ~~ctime~~ ou ~~atime~~ reste inchangé :

```
# debugfs -f cmd.fs
[...]
ctime: 0x4204ab58 -- Sat Feb  5 12:17:44 2005
atime: 0x4204ab06 -- Sat Feb  5 12:16:22 2005
mtime: 0x40f7bde1 -- Fri Jul 16 13:37:05 2004
[...]
```

Les données peuvent donc être facilement retrouvées. Si l'on connaît approximativement la date

d'intrusion sur le système, une recherche à partir de cette date sur atime, ctime et mtime des inodes des fichiers permettra de découvrir les fichiers créés par le pirate au moment de son intrusion. L'utilisation de debugfs pour récupérer ces informations étant trop fastidieuse, il existe des outils, tel que mactime, issu de la solution forensics sleuthkit [sleuthkit], qui automatisent le processus.

Dans ce cas-là, les outils forensics remplissent bien leur rôle : donner les informations nécessaires sur les systèmes de fichiers pour conclure ou non d'une intrusion sur le système. Cependant, si nous ne savons pas où chercher et comment chercher, ces outils deviennent vite inefficaces. Et même s'ils faisaient plus que donner des informations (automatisation de l'analyse forensics), ils seraient malgré tout défaillants du fait qu'ils n'analysent pas dans son intégralité la structure du système de fichiers.

Un exemple concret

Il existe un bug (ou un défaut d'implémentation, peu importe) dans la libc qui fait que seuls les inodes >= 1 sont pris en compte. Et les outils forensics font la même erreur. Dans sleuthkit par exemple, il considère qu'aucun bloc de données ne peut être alloué à un inode avant l'inode fs->first_inum = EXT2FS_FIRSTINO = 1

```
$ cat sleuthkit-1.73/src/fstools/ext2fs.c
[...]
        /*
         * Sanity check.
         */
        if ((inum < fs->first_inum) || (inum > fs->last_inum))
                error("invalid inode number: %lu", (ULONG) inum);
[...]
```

Par conséquent, si vous créez un fichier avec un inode = 0, il ne sera visible d'aucun utilitaire lié à la libc et d'aucun outil forensics. Cette solution anti-forensics est une solution parmi tant d'autres. Elle prouve cependant que les outils forensics contiennent des failles dans leur implémentation.

Techniquement, pour créer un fichier avec un inode = 0, le plus simple est de créer un fichier normal et de modifier son numéro d'inode. Ce qui implique qu'il faut repérer son inode dans la table des inodes et par conséquent localiser cette table dans un groupe de blocs. Connaissant l'offset de l'inode, il sera ensuite possible de modifier son numéro d'inode et d'accéder aux blocs de données de cet inode (chaque inode contient les pointeurs vers les blocs de données).

Ne pas être détecter par fsck

L'exemple de l'inode = 0 est un exemple parmi tant d'autres pour montrer que les outils forensics sont une barrière à franchir afin que les solutions anti-forensics soient efficaces.

Il est ensuite nécessaire de s'assurer que le système de fichiers ne soit pas altéré, au risque de perdre nos données, de rendre le système inutilisable ou bien que l'utilitaire fsck disponible dans e2fsprogs détecte une anomalie dans le système de fichiers.

Petit rappel sur fsck : il est utilisé pour vérifier l'intégrité du système de fichiers et le réparer en cas d'anomalie. Il peut être lancé manuellement ou bien au démarrage du système à intervalles réguliers.

Ces intervalles dépendent en fait du nombre de montage de la racine et par conséquent du nombre de démarrage de la machine. Ces informations sont présentes dans le super bloc et servent à conserver un système de fichiers stable et non corrompu, une des grandes particularités de ext2.

```
# debugfs
debugfs 1.35 (28-Feb-2004)
```

```
debugfs:  open /dev/hda1
debugfs:  stats
[...]
Filesystem state:         clean
[...]
Last mount time:          Fri Feb  4 08:06:21 2005
Last write time:          Fri Feb  4 08:06:21 2005
Mount count:              4
Maximum mount count:      20
Last checked:             Thu Feb  3 14:05:40 2005
Check interval:           15552000 (6 months)
Next check after:         Tue Aug  2 15:05:40 2005
[...]
```

Revenons à la solution anti-forensics précédemment évoquée. Passe-t-elle à travers les mailles du filet de ~~fsck~~ ? Malheureusement non. Il détecte les fichiers avec un numéro d'inode = 0 comme invalide du fait que l'inode 0 n'est pas autorisé à avoir des blocs de données qui lui sont attachés. Dans le code de ~~fsck~~, cela se présente de cette manière :

```
$ cat e2fsck/pass1.c
[...]
void e2fsck_pass1(e2fsck_t ctx)
{
[...]
pctx.errcode = ext2fs_get_next_inode(scan, &ino, &inode);
[...]
while (ino) {
                [...]
        if (ino == EXT2_BOOT_LOADER_INO) {
                if (LINUX_S_ISDIR(inode.i_mode))
                        problem = PR_1_RESERVED_BAD_MODE;
        } else if (ino == EXT2_RESIZE_INO) {
                if (inode.i_mode &&
                        !LINUX_S_ISREG(inode.i_mode))
                        problem = PR_1_RESERVED_BAD_MODE;
        } else {
                if (inode.i_mode != 0)
                        problem = PR_1_RESERVED_BAD_MODE;
                }
        [...]
}
```

Pour résumer, il récupère la liste des numéros d'inodes et regarde leurs valeurs. S'ils ne correspondent à aucun numéro d'inode réservé (~~EXT2_BAD_INO < ino < EXT2_UNDEL_DIR_INO~~) mais qu'il est toujours inférieur au premier numéro d'inode non réservé (~~EXT2_GOOD_OLD_FIRST_INO~~), il vérifie si la valeur du ~~i_mode~~ de l'inode est différente de 0. Cela signifierait que le fichier est créé. C'est le cas pour le fichier avec l'inode = 0.

Un exemple concret : les bad blocks

Nous avons vu que la solution de cacher nos données dans un fichier avec inode = 0 fonctionne à moitié, ce qui évidemment nous satisfait ... à moitié. ~~Grugq~~ a implémenté une technique qui était à l'époque (nous verrons pourquoi) pleinement fonctionnelle runefs ~~[runefs]~~.

Sa solution découle du problème d'implémentation dans l'outil forensic « The Coronor's Toolkit »1 (TCT) développé par Dan Farmer et Wietse Venema ~~[tct]~~ (l'ancêtre de ~~sleuthkit~~ en fait). Ce dernier faisait l'erreur dans la version 1.09 de considérer qu'aucun bloc de données ne pouvait être alloué à un inode avant l'inode ~~EXT2_ROOT_INO~~ (inode 2) :

```
    /*
     * Sanity check.
     */
    if (inum < EXT2_ROOT_INO || inum > ext2fs->fs.s_inodes_count)
```

```
        error("invalid inode number: %lu", (ULONG) inum);
```

Sauf qu'avant l'inode ~~EXT2_ROOT_INO~~, il existe l'inode ~~EXT2_BAD_INO~~ des bad blocks. Cet inode sert à référencer les data blocks qui occupent des mauvais secteurs du disque dur. Qui dit inode dit blocs de données alloués, ce qui veut dire qu'il est possible d'y enregistrer des données. Nos données seront donc cachées et TCT ne verra rien ;

```
 # df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda1               429490    154468    252848  38% /
# ./mkrune -v /dev/hda1
+++ bb_blk +++
    bb_blk->start = 205509
        bb_blk->end = 212992
        bb_blk->group = 25
        bb_blk->size = 7484
+++
rune size: 7M
# df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda1               429490    161983    245333  40% /
```

L'espace pour cacher nos données est créé c'est-à-dire que des blocs de données sont alloués à l'inode 1. C'est vérifiable avec l'utilitaire ~~debugfs~~ :

1 On retrouve le même Sanity check, vu avec Sleuthkit

```
# debugfs
debugfs 1.27 (8-Mar-2002)
debugfs:  open /dev/hda1
debugfs:  stat <1>
Inode: 1   Type: bad type    Mode:  0000   Flags: 0x0   Generation: 0
User:     0   Group:     0   Size: 7663616
File ACL: 0    Directory ACL: 0
Links: 0   Blockcount: 14968
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x3bd5eee8 -- Wed Oct 24 00:27:52 2001
atime: 0x4204c2d4 -- Sat Feb  5 13:57:56 2005
mtime: 0x4204c2d4 -- Sat Feb  5 13:57:56 2005
BLOCKS:
(0-11):205509-205520, (IND):285, (12-267):205521-205776, [...] (7180-7435):212689-212944, (IND):640, (7436-7483):212945-212992
TOTAL: 7515
```

Si vous avez bien suivi :) vous pouvez voir que le premier bloc alloué est bien égal à ~~bb_blk->start (205509)~~, le dernier à
~~bb_blk->end (212992)~~ et la taille à ~~bb_blk->size~~ (TOTAL : 7515, proche de 7484 dû à quelques meta-data pris en compte). Éditez les informations de l'inode 1 sur un de vos systèmes sains et vous verrez qu'aucun bloc n'est alloué à cet inode.
Vous avez ensuite les utilitaires ~~runewr~~ et ~~runerd~~ pour respectivement écrire et lire des données cachées. Voici un exemple avec ~~runewr~~ vérifiable une nouvelle fois avec ~~debugfs~~ (~~debugfs~~ est vraiment notre ami :)) :

```
 # cat /etc/syslog.conf | ./runewr /dev/hda1
# debugfs
debugfs 1.27 (8-Mar-2002)
debugfs:  open /dev/hda1
debugfs:  cat
cat: Usage: cat <file>
debugfs:  dump <1> /tmp/dump
debugfs:
# head /tmp/dump
#  /etc/syslog.conf     Configuration file for syslogd.
#
#                      For more information see syslog.conf(5)
#                      manpage.
[...]
```

Du fait que fsck considère l'inode EXT2_BAD_INO comme un inode valide, il trouve normal que des blocs soient alloués à cet inode.

Contrairement à l'inode = 0 où dès le départ, il était considéré comme invalide.
Cette technique pour cacher des données était assez royale (même au redémarrage de la machine, les données étaient toujours présentes). Évidemment le code de TCT a été patché pour prendre en compte dans son analyse l'inode EXT2_BAD_INO et ne plus le considérer comme un inode invalide.
L'édition des blocs de données alloués à cet inode est donc maintenant possible avec TCT.

Les autres techniques

Toutes les techniques qui suivent ont été créées par the grugq [tho grugq], précurseur dans ce domaine.

Waffen FS

Cette technique consiste à créer un faux journal ext3 dans un système de fichiers ext2.
Techniquement, cela consiste à créer un fichier normal qui contiendra des fausses en-têtes meta-data de journal. La description de l'en-tête est présent dans le fichier linux/ext3_fs.h.
Au niveau de e2fsck, comme il supporte à la fois les systèmes de fichiers ext2 et ext3, il ne détectera aucune anomalie. Le fichier est vu comme un fichier journal ext3 normal.
Mais évidemment, des solutions telles que sleuthkit commence à fournir des outils (jcatet jls) permettant d'analyser les fichiers journaux présents.

Ky FS

Kyfs prend un fichier répertoire (rappelez vous que tout est fichier sur un système ext2 et ext3) et modifie sa structure de cette manière :

```
struct ext2_dir_entry_2 {
        inode = 0;
        reclen = <blocksize>;
        namelen = 0;
        file_type;
        name[];
}
```

Les données seront enregistrées dans name[]. Vous pouvez ensuite créer autant de répertoires avec inode = 0 que vous voulez sachant que les répertoires ne contiendront pas de fichiers.
Que ce soit fsck, le noyau ou les outils forensics, la supercherie ne sera pas découverte du fait que ces répertoires seront considérés comme des répertoires normaux ou effacés.

Data mule FS

Cette technique complète les 3 précédentes du fait qu'elle permet de cacher des données dans tous les endroits du système de fichiers pas encore pris en compte : l'espace réservé, l'espace restant dans les structures (padding), etc.
Un exemple est le super bloc. Il correspond à un bloc de 1024 octets (cf. SUPERBLOCK_SIZE dans ext2fs/ext2fs.h). Or sa structure ext2_super_block (toujours dans ext2fs/ext2fs.h) n'occupe qu'une partie de cet espace. Le but de la technique Data mule FS est de combler l'espace vide avec des données à cacher (759 octets d'après the grugq). Et bien évidemment, comme ce sont des espaces valides dans un système de fichiers, aucune anomalie ne sera détectée.

Conclusion

The grugq a quasiment fait tout le tour des possibilités de cacher des données dans un système de fichiers ext2/ext3. Nous pouvons en imaginer d'autres, mais peut-être un peu tirées par les cheveux et certainement détectables par les outils forensics ou ~~fsck~~ :

- créer un fichier et marquer l'inode comme free. Il vaut mieux certainement créer le fichier avec un inode très élévé pour éviter qu'il soit écraser.
- l'inode d'indice 6 est réservé à la restauration des fichiers effacés. Un répertoire interne caché regroupe les enregistrements des fichiers effacés s'ils ont la propriété restauration, permettant ainsi une restauration après effacement. L'idée est donc d'effacer un fichier mais en prenant bien soin qu'il ait la propriété restauration.

Reste à être convaincu et à tester. Mais en fait la tendance est tout autre. On ne cherche plus comment cacher des données sur un système de fichiers ext2/ext3, on cherche plutôt comment ne pas en écrire. Mais c'est une tout autre histoire ... et un tout autre article :).

Références :

- [ext3] Michael K. Johnson - « Whitepaper: Red Hat's New Journaling File System: ext3 » http://www.redhat.com/support/wpapers/redhat/ext3/
- [chkrootkit] http://www.chkrootkit.org
- [sleuthkit] Brian Carrier - « Sleuthkit » ; http://www.sleuthkit.org
- [runefs] The grugq - « Defeating Forensic Analysis on Unix » : http://www.phrack.org/phrack/59/p59-0x06.txt
- [tct] Dan Farmer, Wietse Venema - « TCT » : http://www.fish.com/security
- [the grugq] The grugq - « The Art of Defiling: Defeating Forensic Analysis on Unix File Systems » http://www.blackhat.com/presentations/bh-europe-04/bh-eu-04-grugq.pdf

~~Retrouvez cet article dans : **Misc 18**~~

Posté par (La rédaction) | Signature : Samuel Dralet | Article paru dans 

Laissez une réponse

Vous devez avoir ouvert une session pour écrire un commentaire.

« Précédent Aller au contenu »

Identifiez-vous
Inscription
S'abonner à UNIX Garden

Articles de 1ère page

- Dos par complexité
- Petite introduction à l'électronique à l'usage des sysadmins et codeurs
- Sortez du software !
- Les dénis de service
- La mort annoncée du WEP
- nikto Tests de serveurs HTTP

- duplicity Sauvegarde chiffrée
- knl
- ffmpeg
- tor

Actuellement en kiosque :

Il y a actuellement

771 articles/billets en ligne.

Recherche

Catégories

- Administration réseau
- Administration système
- Agenda-Interview
- Audio-vidéo
- Bureautique
- Comprendre
- Distribution
- Embarqué
- Environnement de bureau
- Graphisme
- Jeux
- Matériel
- News
- Programmation
- Réfléchir
- Sécurité
- Utilitaires
- Web

Archives

- septembre 2008
- août 2008
- juillet 2008
- juin 2008
- mai 2008
- avril 2008
- mars 2008
- février 2008
- janvier 2008
- décembre 2007
- novembre 2007
- février 2007

GNU/Linux Magazine

- GLMF, partenaire de l'évènement "Paris, capitale du Libre"
- GNU/Linux Magazine 108 - Septembre 2008 - Chez votre marchand de journaux
- Edito : GNU/Linux Magazine 108
- GNU/Linux Magazine HS 38 - Septembre/Octobre 2008 - Chez votre marchand de journaux
- Edito : GNU/Linux Magazine HS 38

GNU/Linux Pratique

- Linux Pratique soutient la journée mondiale contre les brevets logiciels
- Linux Pratique, partenaire de l'évènement "Paris, capitale du Libre"
- Linux Pratique N°49 -Septembre/Octobre 2008 - Chez votre marchand de journaux
- Edito : Linux Pratique N°49
- À télécharger : Les fichiers du Cahier Web de Linux Pratique n°49

MISC Magazine

- Misc 39 : Fuzzing - Injectez des données et trouvez les failles cachées - Septembre/Octobre 2008 - Chez votre marchand de journaux
- Edito : Misc 39
- MISC 39 - Communiqué de presse
- Salon Infosecurity & Storage expo - 19 et 20 novembre 2008.
- Misc 38 : Codes Malicieux, quoi de neuf ? - Juillet/Août 2008 - Chez votre marchand de journaux