Linux

Linux intrusion testing methodologies, tools, and techniques

- [FR] Système des capabilities Linux

[FR] Système des capabilities Linux

Introduction et principes généraux des Capabilities

Lorsque l'on veut lancer un processus avec certains droits on peut soit le lancer en tant que root, ce qui est dangereux car le processus a maintenant tous les droits, soit lui attribuer certaines capabilities. Les capabilities ont été créées à la fin des années 90 et sont une façon, en apparence simple, de diviser les droits du root pour n'attribuer que les autorisations voulues (le terme "en apparence" sera justifié par la suite). Ceci permet de respecter le principe du moindre privilège, c'est à dire n'attribuer à un processus que les droits dont il a besoin pour éviter qu'il est trop de "pouvoir". Cette logique s'inscrit dans le système **DAC** (Discretionary Access Control) qui consiste à confiner les utilisateurs et processus en ne leur attribuant que les droits nécessaires#1. Par exemple, la capability **CAP_NET_BIND_SERVICE** autorise l'attachement d'un socket aux ports inférieurs à 1024, ce qui normalement n'est autorisé qu'aux processus systèmes. Il faut voir les capabilities comme une granularité du root.

Prenons un exemple simple : si j'essaie d'exécuter 'tcpdump' dans un terminal je vais avoir une erreur car je n'ai pas la permission :

```
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~$ tcpdump
tcpdump: wlp2s0: You don't have permission to capture on that device
(socket: Operation not permitted)
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~$
```

Maintenant, si je lance avec un sudo cela va marcher, mais en examinant le status du processus on voit que tcpdump a absolument tous les droits, ce que viole la règle du moindre privilège !

La solution est donc de donner juste les capabilities nécessaires au fonctionnement de tcpdump. En l'occurrence, tcpdump a besoin de **cap_net_raw** pour fonctionner. Voici ce que ça donne :

```
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000003fffffffff
CapBnd: 0000003fffffffff
```

```
CapInh: 0000000000000000
CapPrm: 0000000000002000
CapEff: 0000000000002000
CapBnd: 0000003fffffffff
```

```
-ideapad-110-15ISK:~$ sudo setcap cap_net_raw+ep /usr/s
bin/tcpdump
-ideapad-110-15ISK:~$
```

```
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~$ capsh --decode=2000
0x0000000000002000=cap_net_raw
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~$
```

On voit la valeur 2000. Si on capsh --decode=2000 on voit bien cap_net_raw.

Qu'est-ce qu'une Capability ? Que signifie CapEff dans le status du processus ? Ou le +ep dans la mise en place des capabilities ? Que se passe t'il si un processus engendre un processus fils ? Tant de questions auxquelles je vais tenter de répondre dans cet article. Je vais dans un premier temps expliquer de manière détaillée le fonctionnement des ensembles de capabilities ; dans un second temps éclaircir le système d'héritage ; et je finirai sur un exemple vraiment complet avec la modification du module login. Des captures d'écrans de la console et de morceaux de codes seront à chaque fois présents pour illustrer mes explications.

Actuellement la gestion des capabilities sur un processus fonctionne avec 5 ensembles : **Permitted, Effective, Inheritable, Ambient et le Bounding Set**. Nous allons, dans un premier temps, nous intéresser à Permitted et Effective.

Permitted

Permitted est l'ensemble de toutes les capabilities qui peuvent être attribuées à un processus à tout moment par le système. Pour modifier l'ensemble Permitted on va utiliser une autre facette des capabilities, qui sont les capabilities de fichiers. En effet, chaque fichier binaire (donc exécutable) possèdent lui aussi les ensembles Permitted, Effective et Inheritable (en réalité, Effective pour fichier n'est pas un ensemble mais un bit d'activation pour les capabilities, mais cela ne change pas grand-chose pour l'utilisation. C'est surtout intéressant à savoir si on s'intéresse à la gestion des capabilities par le système).

Le principe est simple : on utilise la commande bash 'setcap' (permet d'attribuer des capabilities aux fichiers binaires de cette facon : sudo setcap cap1,cap2+eip fichier) qui permet d'attribuer des capabilities aux fichiers. Comme vu précédemment, pour utiliser le service tcpdump il faut que le processus possède la capability cap_net_raw. On va donc, dans un terminal, taper la commande 'sudo setcap cap_net_raw+ep /usr/sbin/tcpdump'. Le + signifie que l'on ajoute la capability et le ep signifie qu'on l'ajoute dans le Permitted et Effective (on peut donc faire +eip pour ajouter dans les 3 ensembles).

Par contre, il existe une règle très importante sur le système des capabilities qui nous suivra tout le long : lorsqu'un setuid d'un utilisateur root vers un non-root est effectué tous les ensembles de capabilities sont effacés par sécurité...sauf à une seule et unique condition : avec la primitive 'prctl' (Prctl est une primitive de sys/prctl.h donnant accès à certains aspects de la gestion des processus. Je conseille de lire le manuel de la commande pour en savoir plus) il est possible de placer un SECUREBIT. Ce bit va permettre de conserver Permitted à travers le setuid. Cependant, le bit sera effacé au moment du setuid (utilisation unique), il faudra donc le replacer pour chaque setuid prévus dans la suite du programme. Ceci est important à retenir, ce bit aura un rôle très important par la suite. Ici, un exemple de passage de l'utilisateur 0 (root), vers le 1000 (mon numéro, non root):

```c
int
main(int argc, char *argv[])
{
    sleep(20);
    printf("Changement\n");

    setuid(1000);

    sleep(20);

    exit(EXIT_SUCCESS);
}
```

```
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

Avant

```
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
```

Après

```
int
main(int argc, char *argv[])
{
    sleep(20);
    printf("Changement\n");

    prctl(PR_SET_KEEPCAPS,1,0,0,0);

    setuid(1000);

    sleep(20);

    exit(EXIT_SUCCESS);
}
```

```
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

Après changement d'ID

On voit très bien la conservation de l'ensemble Permitted à travers le setuid.

> **Attention** : il est impossible de modifier l'ensemble Permitted d'un processus en cours d'exécution. Les librairies libcap et libcap-ng devraient le permettre grâce à certaines primitives, mais je n'ai jamais réussi à le faire. Voici un code qui devrait permettre de modifier Permitted en cours d'exécution, ainsi que les valeurs de l'ensemble Permitted avant et après modification. Avant exécution Permitted ne possède que cap_sys_nice, le code devrait rajouter cap_net_raw. Aucun changement n'a lieu, la valeur reste à 800000:

```
SigQ:   0/15065
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000000000
SigCgt: 0000000000000000
CapInh: 0000000000000000
CapPrm: 0000000000800000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

```
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~/Dropbox/StageL3/TestsDivers$ sudo
 setcap cap_sys_nice+p myecho
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~/Dropbox/StageL3/TestsDivers$ ./my
echo
Changement
```

```
#include <sys/prctl.h>
#include <cap-ng.h>

int
main(int argc, char *argv[])
{
    sleep(10);
    printf("Changement\n");

    capng_get_caps_process();
    if (capng_update(CAPNG_ADD, CAPNG_PERMITTED, (int)CAP_NET_RAW)) {
        printf("Impossible de mettre cette cap\n");
        exit(2);
    }
    capng_apply(CAPNG_SELECT_CAPS);

    sleep(10);
```

Je ne sais pas s'il s'agit d'un bug ou si cela est voulu, mais cela reste néanmoins étrange et je n'ai trouvé aucune explication concrète à ceci. Je pars donc du principe que Permitted est invariant durant l'exécution.

Effective

L'ensemble Effective représente les capabilities qui sont effectivement attribuées au processus à un instant donné et donc les capabilities qui sont en effet exploitable. Effective est un sous ensemble de Permitted. Ce qui signifie qu'une capability présente dans Effective est obligatoirement présente dans Permitted. Effective est un ensemble modifiable pendant l'exécution du programme. Cela signifie que deux solutions s'offrent à nous pour le modifier : soit utiliser setcap comme précédemment, les capabilities seront ainsi présentes dans Effective dès le début de l'exécution, soit utiliser les primitives des librairies libcap et libcap-ng (libcap et libcap-ng

sont les librairies en C permettant de travailler sur la structures des capabilities de manière simplifiée, libcap-ng est une variante de libcap rendant les modifications encore plus aisées) dans le code du programme et ainsi modifier les capabilities en cours d'exécution.

Un exemple de code qui permet de mettre cap_net_raw dans Effective en cours d'exécution. La capability était présente dans Permitted auparavant avec 'setcap'. Ensuite, deux captures de la console pour montrer l'apparition de la capability dans Effective :

```
int
main(int argc, char *ar
{
    sleep(20);
    printf("Changement\r


    capng_get_caps_process();
    if (capng_update(CAPNG_ADD, CAPNG_EFFECTIVE, (int)CAP_NET_RAW)) {
        printf("Impossible de mettre cette cap\n");
        exit(2);
    }
    capng_apply(CAPNG_SELECT_CAPS);

    sleep(20);

    exit(EXIT_SUCCESS);
}
```

Avant

```
CapInh: 0000000000000000
CapPrm: 0000000000002000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

Après

```
CapInh: 0000000000000000
CapPrm: 0000000000002000
CapEff: 0000000000002000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

En résumé, nous avons deux ensembles : Effective qui représente toutes les capabilities qui peuvent être utilisées à cet instant, et Permitted qui représente toutes les capabilities qui pourront être acquises ou héritées par le processus pendant l'exécution. Hérité ? J'ai déjà utilisé ce mot dans l'introduction, mais de quoi s'agit-il ? Et bien, nous allons voir ça tout de suite ! Et c'est ce qui va constituer la plus grande partie de ce rapport.

Le principe d'héritage

Durant l'exécution d'un processus il peut arriver que ce dernier souhaite lancer un processus fils à partir d'un fichier binaire à l'aide d'une primitive de type 'exec' (en réalité, ceci arrive très souvent, il suffit de regarder l'arborescence avec la commande 'pstree' pour s'en rendre compte). A partir de là, quelles seront les capabilities du processus fils ? Celles du fichier binaire ? Celles du processus père peut-être ? Et bien c'est un peu plus compliqué que cela, et c'est là que les ensembles Inheritable et Ambient entrent en jeu. Pour l'instant nous allons nous concentrer sur Inheritable.

Inheritable

Inheritable représente l'ensemble des capabilities que le processus père va pouvoir transmettre à son fils. Comme Effective, cet ensemble ne peut contenir que les capabilities se trouvant dans Permitted (donc le processus père peut, au maximum, donner les capabilities qui lui sont permises), sauf à une seule et unique condition : si la capability **CAP_SETPCAP** se trouve à la fois dans Permitted ET dans Effective, Inheritable pourra contenir des capabilities non présentes dans Permitted. Ceci ne fonctionne que pour Inheritable, pas pour Effective (et ne permet pas non plus de modifier Permitted en cours d'exécution).

La modification d'Inheritable est un peu plus délicate que pour les ensembles précédents et pourrait presque sembler buguée. Pour donner des capabilities Inheritable au fichier binaire qui sera exécuté dans le exec un simple 'setcap + i' suffit (avec la capability et le fichier voulu, bien entendu). Cependant, il faut que le processus père ait aussi des capabilities Inheritable. On pourrait penser que mettre des capabilities dans le Inheritable de son binaire puis de le lancer donnera les capabilities voulues au processus... Et bien non ! En réalité le binaire a les capabilities dans Inheritable (on le voit en faisant un 'getcap' dessus), mais le processus résultant ne les a pas, et ne peut donc pas les transmettre au fils lors du exec.

Voici les capabilities d'un processus pour lequel j'ai mis des capabilities dans eip. On voit qu'Inheritable n'a pas récupéré la capability :

```
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~/Dropbox/StageL3/Test$ sudo setcap cap_net_raw+eip myecho
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~/Dropbox/StageL3/Test$ ./myecho
```

```
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000000000
SigCgt: 0000000000000000
CapInh: 0000000000000000
CapPrm: 0000000000002000
CapEff: 0000000000002000
0000003fffffffff
0000000000000000
```

```
capng_get_caps_process();
if (capng_update(CAPNG_ADD, CAPNG_INHERITABLE, (int)CAP_SYS_NICE)) {
    printf("Impossible de mettre cette cap\n");
    exit(2);
}
capng_apply(CAPNG_SELECT_CAPS);

/*if (prctl(PR_CAP_AMBIENT,PR_CAP_AMBIENT_RAISE,CAP_SYS_NICE,0,0)) {
    perror("Ambient impossibe\n");
    exit(1);
}*/

sleep(60);

exit(EXIT_SUCCESS);
```

```
CapInh: 0000000000800000
CapPrm: 0000000000800000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

Et un exemple avec **SETPCAP** :

```
CapInh: 0000000000000000
CapPrm: 0000000000000108
CapEff: 0000000000000108
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

```
CapInh: 0000000000800000
CapPrm: 0000000000000108
CapEff: 0000000000000108
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

Au lancement seulement **SETPCAP** et **FOWNER** sont présentes dans Permitted et Effective (**FOWNER** est juste là pour l'exemple, il n'a pas d'utilité ici). On voit sur la deuxième capture que j'ai réussi à passer **CAP_SYS_NICE** dans Inheritable. Et ceci grâce à **SETPCAP** et ce code :

Bon, c'est bien beau tout ça, on sait comment remplir l'ensemble Inheritable d'un processus qu'on veut lancer...mais maintenant, à quoi sert concrètement cet ensemble ? Il va entrer en jeu dans le calcul suivant, qui permet de connaître les capabilities qu'aura le fils :

P'(permitted) = (P(inheritable) & F(inheritable)) | (F(permitted) & cap_bset)

P'(effective) = F(effective) ? P'(permitted) : 0

P'(inheritable) = P(inheritable)

Dans ce calcul, P' représente le fils, P le père et F le fichier binaire (on retrouve le fait que l'Effective du fichier n'est qu'un bit activé ou non).

cap_bset représente quant à lui le Bounding Set dont j'ai parlé au début. Il s'agit de l'ensemble maximal des capabilities que le système peut attribuer à un processus. Cet ensemble permet simplement de créer des systèmes restreint.

Voyons ce que ça donne concrètement. J'ai donc un programme auquel je donne **CAP_SYS_NICE** dans le Permitted avec 'setcap'. Dans le code de ce programme je mets cette même capability dans Inheritable puis je lance un processus fils à l'aide d'un execve sur un binaire possédant la même capability dans Inheritable et Effective. Le processus fils ne fait rien (juste un sleep(20)). Voici le code du processus père et les capabilities du fils :

```
CapInh: 0000000000800000
CapPrm: 0000000000800000
CapEff: 0000000000800000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

```
    perror("Pas de caps proc");

cap_list[0] = CAP_SETPCAP;
cap_list[1] = CAP_FOWNER;
if (cap_set_flag(caps, CAP_EFFECTIVE, 2, cap_list, CAP_SET) == -1)
    perror("Probleme effective");
if (cap_set_proc(caps) == -1)
    perror("Probleme mise en place");
sleep(30);
printf("Changement\n");
caps = cap_get_proc();
cap_list2[0] = CAP_SYS_NICE;
if (cap_set_flag(caps, CAP_INHERITABLE, 1, cap_list2, CAP_SET) == -1)
    perror("Probleme permitted");
```

Si on regarde les règles de calcul ci-dessus, on voit que les ensembles où j'ai placé les capabilities on voit que cela marche parfaitement. Il est maintenant l'heure de nous attaquer au deuxième ensemble gérant l'héritage. Abordons le deuxième ? Pourquoi faire ?

Ambient

A l'origine, seuls les trois ensembles vu précédemment existaient. Seulement, un certain nombre de défauts sont apparus et la création d'un nouvel ensemble avec un fonctionnement différent est devenu nécessaire. Ambient a vu le jour en 2015. Pour expliquer la création de l'Ambient, voici une liste sans doute non exhaustive des problèmes :

- Si le bit Effective du fichier n'est pas activé, aucune capabilities ne pourra aller dans l'Effective du processus fils... Certe le Permitted pourra se remplir mais si l'Effective non
- Les capabilities des fichiers exploitent les Extends Attributs (xattrs), sauf que certains fichiers n'exploitent pas ces attributs donc impossible d'y mettre des capabilities. On voit vite le problème, en résulte qu'il est impossible d'exécuter certaines commandes sans être root, comme la commande

‘ping’ par exemple, qui exécute certains fichiers n’exploitant pas les xattrs (on trouve un setuid(0) dans son code)

- Généralement, si on n’est pas root l’Inheritable du fichier est égale à 0. Soit pour la raison précédemment énoncée, soit parce que les développeurs n’ont pas toujours pensé à mettre toutes les capabilities nécessaires dans leurs binaires. Une solution pourrait être de remplir complètement tous les fI, mais ce n’est pas satisfaisant d’un point de vue sécurité

Ambient a pour objectif de régler ces problèmes, et va permettre de faire ce que l’héritage précédent était censé permettre de faire. Voici donc les règles de calculs maintenant :

P'(ambient) = (file get caps) ? 0 : P(ambient)

P'(permitted) = (P(inheritable) & F(inheritable)) | (F(permitted) & cap_bset) | P'(ambient)

P'(effective) = F(effective) ? P'(permitted) : P'(ambient)

P'(inheritable) = P(inheritable)

Maintenant, Ambient est toujours ajouté à Permitted et Effective, permettant un héritage permanent. Surtout qu’Ambient peut être modifié sans problème en cours d’exécution grâce à la primitive ‘prctl’, de la même façon qu’Inheritable avec libcap ou libcap-ng. Voici le lien vers un blog sur lequel se trouve un très bon code pour tester Ambient :

Reprenons le code pour tester Inheritable, mais en rajoutant l’ajout dans Ambient :

```c
int
main(int argc, char *argv[])
{
    capng_get_caps_process();
    if (capng_update(CAPNG_ADD, CAPNG_INHERITABLE, (int)CAP_SYS_NICE))
        printf("Impossible de mettre cette cap\n");
        exit(2);
    }
    capng_apply(CAPNG_SELECT_CAPS);

    if (prctl(PR_CAP_AMBIENT,PR_CAP_AMBIENT_RAISE,CAP_SYS_NICE,0,0)) {
        perror("Ambient impossibe\n");
        exit(1);
    }

    // char *newargv[] = { NULL, "hello", "world", NULL };
    char *newargv[] = { NULL };
    char *newenviron[] = { NULL };

    assert(argc == 2);  /* argv[1] identifie le
                           programme à exécuter */
    newargv[0] = argv[1];

    execve(argv[1], newargv, newenviron);
    perror("execve");   /* execve() ne retourne qu'en cas d'erreur */
    exit(EXIT_FAILURE);
}
```

```
CapInh: 0000000000800000
CapPrm: 0000000000800000
CapEff: 0000000000800000
CapBnd: 0000003fffffffff
CapAmb: 0000000000800000
```

Par contre, comme on le voit dans le calcul de l'Ambient du fils, pour qu'il récupère l'Ambient du père il faut que le fichier binaire ne contienne absolument aucune capabilities ! (La commande 'sudo setcap -r nomDuFichier' permet de les effacer). On retrouve dans cette règle la logique qui a amené à la création d'Ambient : les fichiers n'ayant pas de capabilities bloquent la transmission.

De plus, Ambient respecte l'invariant comme quoi il ne peux contenir que les capabilities se trouvant à la fois dans Permitted et Inheritable, impossible autrement. Une tentative de remplir Ambient avec Inheritable manquant :

```
guillaume@guillaume-Lenovo-ideapad-110-15ISK:~/Dropbox/StageL3/TestsDivers$
./mvecho
Ambient impossibe
: Operation not permitted
```

Une petite précision : même si Ambient est un ensemble servant à l'héritage, la capability SETPCAP n'agit pas dessus. Même en la positionnant dans Permitted et Effective il est impossible de passer des capabilities absente de Permitted dans Ambient.

Le code suivant ajoute **SETPCAP** à Effective (Permitted est ajouté avec 'setcap') puis tente d'ajouter une autre capability à Inheritable et Ambient. La première capture de console montre le statut du processus au début et la deuxième après l'ajout de la deuxième capability : Inheritable s'ajoute bien (comme vu précédemment) mais Ambient renvoie une erreur.

```
CapInh: 0000000000800000
CapPrm: 0000000000000108
CapEff: 0000000000000108
```

```
setcap cap_sys_nice+ie myec
guillaume@guillaume-Lenovo-
ecve mvecho
```

Enfin, lorsque qu'un setuid allant d'un id root vers un id non root est effectué, Ambient est

inévitablement effacé, impossible de le conserver (mais si Permitted a été conservé avec un **SECUREBIT**, il est toujours possible de le remplir après le setuid).

Maintenant qu'Ambient existe et semble bien fonctionner, on est en droit de se demander à quoi sert encore Inheritable et pourquoi il n'a pas été supprimé. Et bien, au vu de mes recherches et de mes tests, je pense effectivement qu'Inheritable ne sert plus à grand-chose... Il est toujours là car une phase de transition est toujours nécessaire et ce système est très important sur les kernels Unix. C'est aussi pour ça qu'Ambient a toujours besoin d'Inheritable pour pouvoir se remplir, mais je pense que ce dernier est amené à disparaître.

Il ne reste plus qu'une chose à voir avant de passer à l'exemple concret : les capabilities d'utilisateur.

Les capabilities d'utilisateur

Et oui, je n'en ai pas encore parlé pour ne pas tout mélanger, mais les utilisateurs peuvent aussi avoir des capabilities qui leur sont attribuées. En théorie (le mot théorie est important ici, on va y revenir plus tard) cela permet de donner des capabilities au programme login lorsque l'utilisateur s'authentifie (login récupère les capabilities de l'utilisateur). Ensuite login exécute le bash, qui devrait donc irriter de ses capabilities et ainsi, tous programmes lancé par l'utilisateur depuis le bash devrait avoir les capabilities du bash (donc de l'utilisateur).

Pour donner des capabilities à un utilisateur, rien de bien compliqué : il suffit de modifier le fichier capability.conf se trouvant généralement dans le dossier `/etc/security`. La modification se fai comme les exemples déjà positionnés : pour chaque utilisateur, sur une ligne mettre la suite de capabilities voulues séparées par des virgules, suivi du nom de l'utilisateur :

```
## user guillaume
cap_net_raw,cap_net_admin       guillaume
```

La lecture dans ce fichier se fait lorsque login appelle PAM.

> "

> PAM (Pluggable Auhentification Modules) est un système permettant, je cite «l'intégration de différents schémas d'authentification de bas niveau dans une API de haut niveau, permettant de ce fait de rendre indépendants du schémales logiciels réclamant une authentification » (https://fr.wikipedia.org/wiki/Pluggable_Authentication_Modules). Pour faire plus simple, cela permet, lors de l'authentification de l'utilisateur, de charger et d'attribuer à chaque service ses droits et autorisations. De plus, PAM permet de modifier très facilement tous ces paramètres sans avoir à recompiler tous le programme, et c'est indépendant du kernel ! Chaque service voulant disposer de PAM possède un fichier dans /etc/pam.d où tous les paramètres et librairies à appeler sont rentrés.

Donc, lorsque login appelle PAM il fait appelle à la librairie pam_cap.so qui devrait permettre de lui donner les capabilities et ensuite pouvoir les transmettre. Mais comme je l'ai dis, Ambient est récent et n'est donc pas encore implémenter partout, laissant certains services sans possibilité d'héritage...et c'est le cas du login. On a beau rentrer toutes les capabilities que l'on veut dans capability.conf, aucune n'est récupérée par le bash. On essaie avec **CAP_NET_RAW** pour tcpdump :

```
## user guillaume
cap_net_raw             guillaume
## "everyone else" gets no inheritable capabilities (restrictive)
config>
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
guillaume@guillaume-Lenovo-ideapad-110-15ISK:/etc/security$ tcpdump
tcpdump: wlp2s0: You don't have permission to capture on that device
(socket: Operation not permitted)
```

Comme on le voit, tcpdump ne fonctionne pas avec la capability requise. On est donc toujours obligé de faire un setcap pour avoir tcpdump très satisfaisant.

Je vais exploiter cette situation dans un exemple concret et implémenter Ambient dans le module login et pam_cap.so, et ainsi permettre au login de donner ses capabilities au bash. Une fois les capabilities dans l'Ambient du bash tous les programmes lancés par l'utilisateur auront ses capabilities.

Exemple concret : le module login

Comme dit précédemment, login utilise la librairie pam_cap.so pour obtenir ses capabilities. Il convient donc d'aller regarder le code de cette librairie se trouvant dans le package libcap pour comprendre ce qu'il s'y passe. Voici donc des captures d'écrans du code mettant en place les capabilities ainsi que les explications nécessaires :

> “ static int set_capabilities(struct pam_cap_s *cs)
> {
> cap_t cap_s;
> ssize_t length = 0;
> char *conf_icaps;
> char *proc_epcaps;
>
> ...la mémoire nécessaire à la structure,
>
> conf_icaps va récupérer les capabilities de l'utilisateur dans le fichier

capability.conf grâce à la fonction read_capabilities_for_user écrite juste avant (j'ai fais de multiples tests dessus, elle fonctionne parfaitement),

proc_ecaps va récupérer la version textuelle de cap_s pour que ce soit plus facilement exploitable

```
    /*
     * This is a pretty inefficient way to combine
     * capabilities. However, it seems to be the most straightforward
     * one, given the limitations of the POSIX.1e draft spec. The spec
     * is optimized for applications that know the capabilities they
     * want to manipulate at compile time.
     */

    combined_caps = malloc(1+strlen(CAP_COMBINED_FORMAT)
                +strlen(proc_epcaps)+strlen(conf_icaps));
    if (combined_caps == NULL) {
    D(("unable to combine capabilities into one string - no memory"));
    goto cleanup_epcaps;
    }
    _
  }
#endif /* DEBUG */

    if (cap_s == NULL) {
    D(("no capabilies to set"));
    } else if (cap_set_proc(cap_s) == 0) {
    D(("capabilities were set correctly"));
    ok = 1;
    } else {
    D(("failed to set specified capabilities: %s", strerror(errno)));
    }

cleanup_epcaps:
    cap_free(proc_epcaps);

cleanup_icaps:
     pam_overwrite(conf_icaps);
```

Nous const e choisir l'ensemble eritable risque de eu avec 'prctl'. Il c orte de récupérer **flag** et ainsi remp nc mon code :

```
static int set_capabilities(struct pam_cap_s *cs)
{
    cap_t cap_s;
    ssize_t length = 0;
    char *conf_icaps;
    char *proc_epcaps;
    cap_value_t *cap_list1;
    cap_value_t *cap_list2;
    cap_value_t *cap_listFinale;
    int ok = 0;

    cap_s = cap_get_proc();
```

```
 * New version. It seems to be more efficient.
 * Now we can put capabilities in the Permitted, Inheritable and Ambient set.
 * This solution need to use the libcap-ng librarie
```

conf_icaps. Si conf_icaps est à none, la liste prend NULL, si il est à all, la liste prend tout proc_epcaps. Les boucles avec le strtok sont là pour ne récupérer que les capabilities dans proc_epcaps (proc_epcaps est un string de la forme "= cap,cap,cap+eip").

```
int cpt1 = 0;
    /* Extract caps in string format from proc_pcaps to cap_value_t format */

for (str1 = proc_epcaps; ;str1 = NULL) {
    token = strtok(str1,TOK_START);
    if (token == NULL || !strcmp(token,"e") || !strcmp(token,"i") || !strcmp(token,"p")
    || !strcmp(token,"ei") || !strcmp(token,"ep") || !strcmp(token,"ip") || !strcmp(token,"eip"))
        break;

    for (i = 0,str2 = token; ;i++,str2 = NULL) {
        subtoken = strtok(str2,TOK_FLOAT);
        if (subtoken == NULL)
            break;
        /* Intersection between conf_icaps and proc_epcaps */
        for (i = 0;i<cpt1;i++) {
            for (int j = 0;j<cpt2;j++) {
                if (cap_list1[i] == cap_list2[j]) {
                    cap_listFinale[cptFinale] = cap_list1[i];
                    cptFinale++;
                }
            }
        }

        free(cap_list1);
        free(cap_list2);
        }
        /* Set the capabilities in the Ambient set */
        char *cap_string, *capToCap_ng;
        int stringToCap;
        int i,j;

        capToCap_ng = malloc(sizeof(char*) * sizeof(cap_listFinale));
        for (i = 0;i<cptFinale;i++) {
            cap_string = cap_to_name(cap_listFinale[i]);
            /* We need this loop because the "#defined" to attribute an integer
                is without CAP_ */
            for (j = 4;cap_string[j-1] != '\0';j++) {
                capToCap_ng[j-4] = cap_string[j];
            }
            stringToCap = capng_name_to_capability(capToCap_ng);

            if (prctl(PR_CAP_AMBIENT,PR_CAP_AMBIENT_RAISE,stringToCap,0,0)) {
                D(("Ambient problem - maybe check P and I capabilities set"));
            }
        }

        free(capToCap_ng);
        ok = 1;
        } else {
        D(("failed to set specified capabilities: %s", strerror(errno)));
        }
```

Je teste mon nouveau code dans un fichier de test et je compare les résultats avec ceux del'ancien code. Effectivement, maintenant ça marche : Inheritable et Ambient sont remplis.

```
CapInh: 0000000000000000 CapInh: 0000000080000000
CapPrm: 0000000080000008 CapPrm: 0000000080000008
CapEff: 0000000000000000 CapEff: 0000000000000000
CapBnd: 0000003fffffffff CapBnd: 0000003fffffffff
CapAmb: 0000000000000000 CapAmb: 0000000080000000
```

Avant Après

Bien, maintenant que ce code a été modifié, il est temps de l'implémenter.

> **Attention** : je vous conseille très fortement d'effectuer ce qui va suivre dans une machine virtuelle et de régulièrement faire des snapshots...ça vous évitera de rester enfermer en dehors de votre session.

Pour compiler et installer, suivez les instructions du README du package libcap. Ensuite, il va falloir modifier l'appel à pam_cap.so dans PAM. L'appel à pam_cap.so se trouve sur la couche auth de PAM, dans le fichier common-auth (dossier `/etc/pam.d`), et l'appel à common-auth dans logi est fait sous la forme d'un include :

```
# and here are more per-packag/etc/pam.d/common-authditional" block)
auth    optional                        pam_cap.so
# Standard Un*x authentication. /etc/pam.d/login
@include common-auth
```

Une fois le nouveau pam_cap.so installé (en suivant le méthode du README), la librairie se trouve dans /lib64/security. Il est donc nécessaire de modifier common-auth :

```
# and here are more per-package modules (the "Additional" block)
auth    optional                        /lib64/security/pam_cap.so
```

Je conseille de laisser optional, ça évitera que login plante pour un module qui ne le nécessite pas.

Nous avons finit d'implémenter notre nouvelle librairie, il est temps de tester ça. Pour se faire, nous allons devoir passer en mode console (le passage en mode console sera expliqué en annexe). En effet, l'utilisation de l'interface graphique fait que login n'est pas utilisé, c'est un dérivé qui est lancé (systemd-logind), et il change suivant la distribution Linux utilisée. Mes modifications ne sont donc fonctionnelles qu'en mode console.

On redémarre donc la VM en console et lorsque l'on regarde le statut du processus bash...et bien nous n'avons pas les capabilities !

```
CapInh: 0000000000002000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

Cela signifie donc que les capabilities ont été perdues à un moment. Il faut donc aller regarder le code de login. Le programme est beaucoup trop long pour que je l'analyse en détail ici, mais on s'aperçoit dans le main on constate quelque chose de très intéressant : un setuid de 0 vers celui de l'utilisateur a lieu, entraînant ainsi la perte de capabilities.

C’est là que le **SECUREBIT** va entrer en jeu. Il va nous permettre de conserver Permitted à travers le setuid, et nous pourrons ensuite remplir Ambient et ainsi transmettre au bash. En observant le code on voit qu’une fonction nommée fork_session est appelée avant le setuid. Allons voir ce qu’il s’y passe :

> “ Ici dans le père, on trouve un appel à pam_setcred, qui est une implémentation de la fonction pam_sm_setcred se trouvant dans la librairie pam_cap et gérant les credentials. Ceci ayant lieu juste avant le setuid, il est intéressant d’implémenter le **SECUREBIT**.

```
if (child_pid) {
        /*
         * parent - wait for child to finish, then clean up session
         */
        close(0);
        close(1);
        close(2);
        free_getlogindefs_data();

        sa.sa_handler = SIG_IGN;
        sigaction(SIGQUIT, &sa, NULL);
        sigaction(SIGINT, &sa, NULL);

        /* wait as long as any child is there */
        while (wait(NULL) == -1 && errno == EINTR) ;
        openlog("login", LOG_ODELAY, LOG_AUTHPRIV);
```

Voici donc ction, juste avant in. Pour cela je vais uid :

Attention : pensez à modifier le fichier Makemodule.am pour prendre en compte libcap et libcapng avec -lcap-ng -lcap :

```
if BUILD_LOGIN
bin_PROGRAMS += login
dist_man_MANS += login-utils/login.1
login_SOURCES = \
        login-utils/login.c \
        login-utils/logindefs.c \
```

Une fois le nouveau login compilé et installé on peut vérifier le statut du bash en positionnant cap_net_raw dans capability.conf…et ça marche !

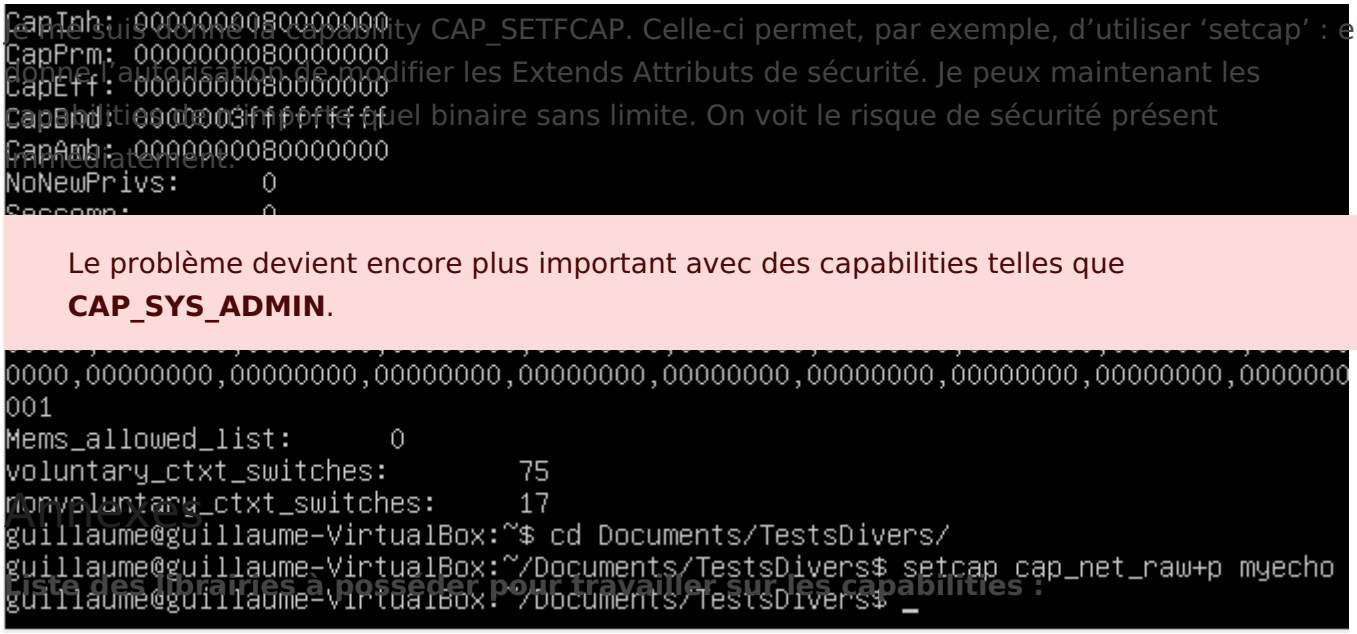
Et voici tout simplement enfin :

Ça y est ! Nous avons enfin un héritage complet des capabilities de l’utilisateur vers le bash !

```
PTR (QM)? _ipp._tcp.local. (45)
16:13:14.123198 IP guillaume-VirtualBox.38678 > 10.0.2.3.domain: 59197+ [1au] PTR? 251.0.0.224.in-addr.arpa. (53)
16:13:14.166526 IP 10.0.2.3.domain > guillaume-VirtualBox.38678: 59197 0/0/1 (53)
16:13:14.170973 IP guillaume-VirtualBox.47765 > 10.0.2.3.domain: 18623+ [1au] PTR? 15.2.0.10.in-addr.arpa. (51)
16:13:14.181893 IP guillaume-VirtualBox.46279 > 10.0.2.3.domain: 33324+ [1au] PTR? 3.2.0.10.in-addr.arpa. (50)
16:13:14.267516 IP guillaume-VirtualBox.49270 > 10.0.2.3.hostmon: Flags [S], seq 4219690756, win 29200, options [mss 1460,sackOK,TS val 215295812 ecr 0,nop,wscale 7], length 0
16:13:14.268397 IP 10.0.2.3.hostmon > guillaume-VirtualBox.49270: Flags [S.], seq 1472001, ack 4219690757, win 65535, options [mss 1460], length 0
16:13:14.268451 IP guillaume-VirtualBox.49270 > 10.0.2.3.hostmon: Flags [.], ack 1, win 29200, length 0
16:13:15.227510 IP6 guillaume-VirtualBox.mdns > ff02::fb.mdns: 0 [2q] PTR (QM)? _ipps._tcp.local. PTR (QM)? _ipp._tcp.local. (45)
16:13:15.230393 IP guillaume-VirtualBox.54157 > 10.0.2.3.domain: 31218+ [1au] PTR? b.f.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.2.0.f.f.ip6.arpa. (101)
16:13:15.350049 IP 10.0.2.3.domain > guillaume-VirtualBox.54157: 31218 0/0/1 (101)
```

Cependant, il y a un aspect critique avec ma modification : maintenant, le bash possède toutes les capabilities de l’utilisateur, ce qui signifie que toutes les commandes lancées depuis le bash posséderont ces capabilities. Ce qui n’est pas correct vis-à-vis du principe de moindre privilège. Cela amène donc à la question : faut il garder ce nouveau système ou faut il encore le modifier

pour pouvoir décider quelles applications auront le droit d’utiliser les privilèges du bash ?



Je me suis donné la capability CAP_SETFCAP. Celle-ci permet, par exemple, d’utiliser ‘setcap’ : elle donne l’autorisation de modifier les Extends Attributs de sécurité. Je peux maintenant les capabilities sur n’importe quel binaire sans limite. On voit le risque de sécurité présent immédiatement.

> Le problème devient encore plus important avec des capabilities telles que **CAP_SYS_ADMIN**.

Liste des librairies à posséder pour travailler sur les capabilities :

- libcap-ng-dev
- libcap-dev
- libcap2
- libcap2-bin

Lien vers les packages à télécharger pour modifier pam_cap.c et login.c

- pam_cap.c : https://github.com/mhiramat/libcap
- login.c : https://github.com/karelzak/util-linux

Pour compiler, installer et passer en mode console :

Pour le paquet libcap :

> depuis le dossier principal faire : make all && sudo make install

Pour le paquet util-linux:

> depuis le dossier principal, faire : ./autogen.sh && ./configure && make pour installer login faire, depuis le dossier principal: sudo rm /bin/login && sudo cp login /bin

Modifier en root le fichier /etc/pam.d/common-auth

> remplacer l'appel de pam_cap.so par /lib64/security/pam_cap.so et vérifier que l'appel est bien optional et non required

> Avant de passer en mode console, faire un snapshot. Je n'ai pas trouvé comment revenir en mode graphique une fois en console, remettre les anciens paramètres ne fonctionne pas.

Pour passer en mode console : modifier le fichier /etc/default/grub

A la ligne `GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"` , remplacer par

`GRUB_CMDLINE_LINUX_DEFAULT="text"`

Décommenter la ligne `#GRUB_TERMINAL=console`

Depuis un terminal : `sudo update-grub && sudo systemctl set-default multi-user.target`

Redémarrer la VM

Liste des #define et #include pour mon code pam_cap.c:

Liste des #define et #include que j'ai rajouté pour mon code login.c :

```
#include <stdio.h>
#include <string.h>
#include <cap-ng.h>
#include <sys/capability.h>
#include <linux/capability.h>
#include <sys/prctl.h>
#include <linux/prctl.h>
#include <unistd.h>
#define TOK_START               "= +"
#define TOK_FLOAT               "+"
#include <sys/prctl.h>
#include <security/pam_modules.h>
#include <security/_pam_macros.h>
#define USER_CAP_FILE           "/etc/security/capability.conf"
#define CAP_FILE_BUFFER_SIZE    4096
#define CAP_FILE_DELIMITERS     " \t\n"
#define CAP_COMBINED_FORMAT     "%s all-i %s+i"
#define CAP_DROP_ALL            "%s all-i"
#define TOK_START               "= +"
#define TOK_FLOAT               "+"
```

- Papier d'origine expliquant les capabilities : https://pdfs.semanticscholar.org/6b63/134abca10b49661fe6a9a590a894f7c5ee7b.pdf
- Article traitant des capabilities : https://lwn.net/Articles/632520/
- Article explicitant l'ambient : https://lwn.net/Articles/636533/
- Article sur un blog avec un code simple mais complet permettant de tester Ambient : https://s3hh.wordpress.com/2015/07/25/ambient-capabilities/
- Article explicitant le fonctionnement de PAM : https://artisan.karma-lab.net/petite-introduction-apam
- Explication sur DAC, MAC et le lien avec les capabilities : https://connect.eddiamond.com/GNU-Linux-Magazine/GLMF-164/Les-capabilities-sous-Linux