[prev in list] [next in list] [prev in thread] [next in thread]

List: oss-security
Subject: [oss-security] pwnkit: Local Privilege Escalation in polkit's pkexec (CVE-2021-4034)
From: Qualys Security Advisory <qsa () qualys ! com>
Date: 2022-01-25 17:57:48
Message-ID: 20220125175655.GA14958 () localhost ! localdomain
[Download RAW message or body]

Qualys Security Advisory

pwnkit: Local Privilege Escalation in polkit's pkexec (CVE-2021-4034)

==

Contents

==

==

Summary

==

We discovered a Local Privilege Escalation (from any user to root) in polkit's pkexec, a SUID-root program that is installed by default on every major Linux distribution:

"Polkit (formerly PolicyKit) is a component for controlling system-wide privileges in Unix-like operating systems. It provides an organized way for non-privileged processes to communicate with privileged ones. [...] It is also possible to use polkit to execute commands with elevated privileges using the command pkexec followed by the command intended to be executed (with root permission)." (Wikipedia)

This vulnerability is an attacker's dream come true:

- pkexec is installed by default on all major Linux distributions (we exploited Ubuntu, Debian, Fedora, CentOS, and other distributions are probably also exploitable);

- pkexec is vulnerable since its creation, in May 2009 (commit c8c3d83, "Add a pkexec(1) command");

- any unprivileged local user can exploit this vulnerability to obtain full root privileges;

- although this vulnerability is technically a memory corruption, it is exploitable instantly, reliably, in an architecture-independent way;

- and it is exploitable even if the polkit daemon itself is not running.

We will not publish our exploit immediately; however, please note that this vulnerability is trivially exploitable, and other researchers might publish their exploits shortly after the patches are available. If no patches are available for your operating system, you can remove the SUID-bit from pkexec as a temporary mitigation; for example:

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

This vulnerability is one of our most beautiful discoveries; to honor its memory, we recommend listening to DJ Pone's "Falken's Maze" (double pun intended) while reading this advisory. Thank you very much!

==

Analysis

==

pkexec is a sudo-like, SUID-root program, described as follows by its man page:

```
------------------------------------------------------------------------
NAME
       pkexec - Execute a command as another user

SYNOPSIS
       pkexec [--version] [--disable-internal-agent] [--help]

       pkexec [--user username] PROGRAM [ARGUMENTS...]

DESCRIPTION
       pkexec allows an authorized user to execute PROGRAM as another
       user. If PROGRAM is not specified, the default shell will be run.
       If username is not specified, then the program will be executed
       as the administrative super user, root.
------------------------------------------------------------------------
```

The beginning of pkexec's main() function processes the command-line arguments (lines 534-568), and searches for the program to be executed (if its path is not absolute) in the directories of the PATH environment variable (lines 610-640):

```
------------------------------------------------------------------------
 435 main (int argc, char *argv[])
 436 {
 ...
 534   for (n = 1; n < (guint) argc; n++)
 535     {
 ...
 568     }
 ...
 610   path = g_strdup (argv[n]);
 ...
 629   if (path[0] != '/')
 630     {
 ...
 632       s = g_find_program_in_path (path);
 ...
 639       argv[n] = path = s;
 640     }
------------------------------------------------------------------------
```

Unfortunately, if the number of command-line arguments argc is 0 (if the argument list argv that we pass to execve() is empty, i.e. {NULL}), then argv[0] is NULL (the argument list's terminator) and:

- at line 534, the integer n is permanently set to 1;

- at line 610, the pointer path is read out-of-bounds from argv[1];

- at line 639, the pointer s is written out-of-bounds to argv[1].

But what exactly is read from and written to this out-of-bounds argv[1]? To answer this question, we must digress briefly. When we execve() a new program, the kernel copies our argument and environment strings and pointers (argv and envp) to the end of the new program's stack; for example:

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

Clearly (because the argv and envp pointers are contiguous in memory), if argc is 0, then the out-of-bounds argv[1] is actually envp[0], the pointer to our first environment variable, "value". Consequently:

- at line 610, the path of the program to be executed is read out-of-bounds from argv[1] (i.e. envp[0]), and points to "value";

- at line 632, this path "value" is passed to g_find_program_in_path() (because "value" does not start with a slash, at line 629);

- g_find_program_in_path() searches for an executable file named "value" in the directories of our PATH environment variable;

- if such an executable file is found, its full path is returned to pkexec's main() function (at line 632);

- and at line 639, this full path is written out-of-bounds to argv[1] (i.e. envp[0]), thus overwriting our first environment variable.

More precisely:

- if our PATH environment variable is "PATH=name", and if the directory "name" exists (in the current working directory) and contains an executable file named "value", then a pointer to the string "name/value" is written out-of-bounds to envp[0];

- or, if our PATH is "PATH=name=.", and if the directory "name=." exists and contains an executable file named "value", then a pointer to the string "name=./value" is written out-of-bounds to envp[0].

In other words, this out-of-bounds write allows us to re-introduce an "unsecure" environment variable (for example, LD_PRELOAD) into pkexec's environment; these "unsecure" variables are normally removed (by ld.so) from the environment of SUID programs before the main() function is called. We will exploit this powerful primitive in the following section.

Last-minute note: polkit also supports non-Linux operating systems such as Solaris and *BSD, but we have not investigated their exploitability; however, we note that OpenBSD is not exploitable, because its kernel refuses to execve() a program if argc is 0.

==
Exploitation
==

Our question is: to successfully exploit this vulnerability, which "unsecure" variable should we re-introduce into pkexec's environment? Our options are limited, because shortly after the out-of-bounds write (at line 639), pkexec completely clears its environment (at line 702):

```
------------------------------------------------------------------------
 639       argv[n] = path = s;
 ...
 657   for (n = 0; environment_variables_to_save[n] != NULL; n++)
 658     {
 659       const gchar *key = environment_variables_to_save[n];
 ...
 662       value = g_getenv (key);
 ...
 670       if (!validate_environment_variable (key, value))
 ...
 675     }
 ...
 702   if (clearenv () != 0)
------------------------------------------------------------------------
```

The answer to our question comes from pkexec's complexity: to print an error message to stderr, pkexec calls the GLib's function g_printerr() (note: the GLib is a GNOME library, not the GNU C Library, aka glibc); for example, the functions validate_environment_variable() and log_message() call g_printerr() (at lines 126 and 408-409):

```
------------------------------------------------------------------------
  88 log_message (gint     level,
  89              gboolean print_to_stderr,
  90              const    gchar *format,
  91              ...)
  92 {
 ...
 125   if (print_to_stderr)
 126     g_printerr ("%s\n", s);
------------------------------------------------------------------------
 383 validate_environment_variable (const gchar *key,
 384                                const gchar *value)
```

```
{
...
          log_message (LOG_CRIT, TRUE,
                       "The value for the SHELL variable was not found the /etc/shells file");
          g_printerr ("\n"
                      "This incident has been reported.\n");
```

g_printerr() normally prints UTF-8 error messages, but it can print messages in another charset if the environment variable CHARSET is not UTF-8 (note: CHARSET is not security sensitive, it is not an "unsecure" environment variable). To convert messages from UTF-8 to another charset, g_printerr() calls the glibc's function iconv_open().

To convert messages from one charset to another, iconv_open() executes small shared libraries; normally, these triplets ("from" charset, "to" charset, and library name) are read from a default configuration file, /usr/lib/gconv/gconv-modules. Alternatively, the environment variable GCONV_PATH can force iconv_open() to read another configuration file; naturally, GCONV_PATH is one of the "unsecure" environment variables (because it leads to the execution of arbitrary libraries), and is therefore removed by ld.so from the environment of SUID programs.

Unfortunately, CVE-2021-4034 allows us to re-introduce GCONV_PATH into pkexec's environment, and to execute our own shared library, as root.

Important: this exploitation technique leaves traces in the logs (either "The value for the SHELL variable was not found the /etc/shells file" or "The value for environment variable [...] contains suscipious content"). However, please note that this vulnerability is also exploitable without leaving any traces in the logs, but this is left as an exercise for the interested reader.

For further discussions about pkexec, GLib, and GCONV_PATH, please refer to the following posts by Tavis Ormandy, Jakub Wilk, and Yuki Koike:

https://www.openwall.com/lists/oss-security/2014/07/14/1
https://www.openwall.com/lists/oss-security/2017/06/23/8
https://hugeh0ge.github.io/2019/11/04/Getting-Arbitrary-Code-Execution-from-fopen-s-2nd-Argument/

Acknowledgments

We thank polkit's authors, Red Hat Product Security, and the members of distros@openwall for their invaluable help with the disclosure of this vulnerability. We also thank Birdy Nam Nam for their inspiring work.

Timeline

2021-11-18: Advisory sent to secalert@redhat.

2022-01-11: Advisory and patch sent to distros@openwall.

2022-01-25: Coordinated Release Date (5:00 PM UTC).

[prev in list] [next in list] [prev in thread] [next in thread]