Using sudo with Python script

Asked 11 years, 4 months agoModified 1 year, 4 months agoViewed 231k times

61

I'm trying to write a small script to mount a VirtualBox shared folder each time I execute the script. I want to do it with Python, because I'm trying to learn it for scripting.

The problem is that I need privileges to launch mount command. I could run the script as sudo, but I prefer it to make sudo by its own.

I already know that it is not safe to write your password into a .py file, but we are talking about a virtual machine that is not critical at all: I just want to click the .py script and get it working.

This is my attempt:

```
#!/usr/bin/env python
import subprocess

sudoPassword = 'mypass'
command = 'mount -t vboxsf myfolder /home/myuser/myfolder'

subprocess.Popen('sudo -S' , shell=True,stdout=subprocess.PIPE)
subprocess.Popen(sudoPassword , shell=True,stdout=subprocess.PIPE)
subprocess.Popen(command , shell=True,stdout=subprocess.PIPE)
```

My python version is 2.6

python shell subprocess

ShareImprove this questionFollow

asked Oct 24, 2012 at 8:37

Roman Rdgz
13k 42 135 210

Is there a reason for not using /etc/fstab ? – mensi Oct 24, 2012 at 8:40

1 @mensi yes, that I am practising to learn using python for these kind of purpose – Roman Rdgz Oct 24, 2012 at 8:41

1 you need to pass the password over stdin, see this stackoverflow.com/a/165662/894872 – Eun Oct 24, 2012 at 8:42

2 If you don't know what you are doing, **avoid** shell=True . If you can't make things work without it, learn what it does and how it works (and then usually you can). – tripleee Dec 1, 2015 at 10:49

Does this answer your question? running a command as a super user from a python script – miken32 Aug 31, 2021 at 15:43

14 Answers

Sorted by:

Highest score (default)

89

Many answers focus on how to make your solution work, while very few suggest that *your solution is a **very bad** approach*. If you really want to "practice to learn", why not practice using good solutions? Hardcoding your password is learning the *wrong* approach!

If what you really want is a password-less mount for that volume, maybe sudo isn't needed *at all*! So may I suggest other approaches?

- Use /etc/fstab as mensi suggested. Use options user and noauto to let regular users mount that volume.
- Use Polkit for passwordless actions: Configure a .policy file for your script with <allow_any>yes</allow_any> and drop at /usr/share/polkit-1/actions
- Edit /etc/sudoers to allow your user to use sudo without typing your password. As @Anders suggested, you can restrict such usage to specific commands, thus avoiding unlimited
count. See this answer for more details on /etc/sudoers.

All the above allow passwordless root privilege, none require you to hardcode your password. Choose any approach and I can explain it in more detail.

As for *why* it is a very bad idea to hardcode passwords, here are a few good links for further reading:

- Why You Shouldn’t Hard Code Your Passwords When Programming
- How to keep secrets secret (Alternatives to Hardcoding Passwords)
- What's more secure? Hard coding credentials or storing them in a database?
- Use of hard-coded credentials, a dangerous programming error: CWE
- Hard-coded passwords remain a key security flaw

ShareImprove this answerFollow edited Dec 28, 2020 at 0:33 answered Jun 17, 2014 at 7:23

MestreLion
13.1k 8 68 63

2 The last point, edit sudoers is very well explained at askubuntu.com/a/155827/42796 – Pablo Marin-Garcia Apr 9, 2018 at 14:41

3 It might helpful for newcomers for you to explain *why* hardcoding the user's password is a *very bad approach*. – pdoherty926 Oct 13, 2018 at 16:57

1 @pdoherty926: I assumed it was obvious for security reasons, but you're right, might be a good idea to educate on *why*. This goes a little beyond the scope of this answer, so I'll edit it to add a few links for further reading. – MestreLion Oct 15, 2018 at 21:07

Regarding the suggestion to use passwordless sudo: Hard-coding the password is bad, but adding the user to `/etc/sudoers` is almost as bad! In both cases any attacker that has access to the user account will have root access, too. – balu Dec 15, 2020 at 10:36

1 @balu realize that when you add a user to `/etc/sudoers` , you can restrict it to certain commands and use various other controls. Adding a user to `/etc/sudoers` does *not* necessarily enable root access. Although for simple mount of volumes, purpose built tools like `/etc/fstab` are better of course. – Anders Dec 23, 2020 at 16:24

50

```
sudoPassword = 'mypass'
command = 'mount -t vboxsf myfolder /home/myuser/myfolder'
p = os.system('echo %s|sudo -S %s' % (sudoPassword, command))
```

Try this and let me know if it works. :-)

And this one:

```
os.popen("sudo -S %s"%(command), 'w').write('mypass')
```

ShareImprove this answerFollow edited Oct 24, 2012 at 9:03 answered Oct 24, 2012 at 8:45

Aniket Inge
25.5k 5 52 78

1 This was my first attempt when I googled it, but doesn't work: it asks me for password at console instead of entering sudoPassword value directly – Roman Rdgz Oct 24, 2012 at 8:49

1 @RomanRdgz echo %s converts it to stdin and pipes the output of sudoPassword to sudo command's stdin. Hence it should work(and does work here) – Aniket Inge Oct 24, 2012 at 8:50

1 I imported os, then copy-pasted, and it doesn't work: keeps asking for password. In fact, If i wait and not write anything when asked, output looks like code tried to input password 3 times wrong, saying 3 times 'Sorry, try again' – Roman Rdgz Oct 24, 2012 at 8:55

26 You really should NEVER use such a line `os.system('echo %s|sudo -S %s' % (sudoPassword, command))` , cause it brings a security hole. By writing your password as shell command, it becomes accessible through `.bash_history` file and by running `history` shell command. Always pass password through stdin as it's more secure approach – thodnev Nov 7, 2016 at 23:26

5 This answer has 3 downvotes, and I'm piling up another one. **This code adds TWO vulnerabilities**: 1) recording the password in the process table which any other process can see, as said above, but also 2) **shell injection**, what if something else can set that password, and sets it to `foo$(rm -rf /*)bar` ? Do you see the problem with that. – ulidtko Nov 6, 2019 at 9:58

To pass the password to sudo's stdin:

```
sudo_password = 'mypass'
command = 'mount -t vboxsf myfolder /home/myuser/myfolder'.split()

p = Popen(['sudo', '-S'] + command, stdin=PIPE, stderr=PIPE,
          universal_newlines=True)
sudo_prompt = p.communicate(sudo_password + '\n')[1]
```

Note: you could probably configure passwordless sudo or SUDO_ASKPASS command instead of hardcoding your password in the source code.

ShareImprove this answerFollow

edited Feb 17, 2015 at 4:18

answered May 16, 2014 at 4:13
jfs
407k 199 1k 1.7k

the Popen you describe throws an error `can only concatenate list (not "str") to list` I changed it to `Popen(['sudo -S ' + command]` - That worked for me. It seems that at the time of answer this added to a list implicitly.. which is no longer allowed? or supported.. – Piotr Kula Jan 21, 2018 at 22:54

1 @ppumkin wrong. Look at the code in the answer. It has `.split()` . Compare with your code. – jfs Jan 21, 2018 at 22:56

Ohh boy yes. I missed the split() on the end.. wow late night coding. I slept on it and decided it was a bad idea to do it like this any way so going the passwordless route instead :D I just wanted something to work and was desperate – Piotr Kula Jan 22, 2018 at 14:47

How would you solve multiple commands via sudo efficiently? I want to do these commands in this order: ``` sudo mkdir Filestore sudo mount [filestore-info] Filestore sudo chmod 777 Filestore ``` Basically 3 sudo commands – DUDANF Jun 7, 2019 at 9:54

5

- Use -S option in the sudo command which tells to read the password from 'stdin' instead of the terminal device.
- Tell Popen to read stdin from PIPE.
- Send the Password to the stdin PIPE of the process by using it as an argument to communicate method. Do not forget to add a new line character, '\n', at the end of the password.

```
sp = Popen(cmd , shell=True, stdin=PIPE)
out, err = sp.communicate(_user_pass+'\n')
```

ShareImprove this answerFollow

edited Dec 1, 2015 at 10:46
tripleee
181k 35 286 332

answered Dec 19, 2014 at 20:03
Hubert Vijay
89 1 6

4

subprocess.Popen creates a process and opens pipes and stuff. What you are doing is:

- Start a process sudo -S
- Start a process mypass
- Start a process mount -t vboxsf myfolder /home/myuser/myfolder

which is obviously not going to work. You need to pass the arguments to Popen. If you look at its documentation, you will notice that the first argument is actually a list of the arguments.

ShareImprove this answerFollow

answered Oct 24, 2012 at 8:45
mensi
9,735 2 34 43

1 Ok, I get what I'm doing wrong, but I don't think it is possible to pass sudo's password as an argument here with subprocess.Popen(['sudo', '-S', password, command], shell=True, stdin=subprocess.PIPE). SO how can I do it? – Roman Rdgz Oct 24, 2012 at 8:52

Have a look at the linked SO question – mensi Oct 24, 2012 at 10:48

2

I used this for python 3.5. I did it using **subprocess** module.Using the password like this is very **insecure**.

The subprocess module takes command as a list of strings so either create a list beforehand using **split()** or pass the whole list later. Read the documentation for moreinformation.

```
sudoPassword = 'mypass'
command = 'mount -t vboxsf myfolder /home/myuser/myfolder'.split()

cmd1 = subprocess.Popen(['echo',sudoPassword], stdout=subprocess.PIPE)
cmd2 = subprocess.Popen(['sudo','-S'] + command, stdin=cmd1.stdout, stdout=subprocess.PIPE)

output = cmd2.stdout.read.decode()
```

ShareImprove this answerFollow

edited Oct 1, 2018 at 22:28
Ajax1234
70.7k 8 63 105

answered Apr 5, 2018 at 10:28
Nandesh
4,583 2 21 26

1

sometimes require a carriage return:

```
os.popen("sudo -S %s"%(command), 'w').write('mypass\n')
```

ShareImprove this answerFollow

edited Aug 27, 2013 at 14:16
laalto
151k 66 287 303

answered Aug 27, 2013 at 13:55
user2095717
21 1

1

Please try module pexpect. Here is my code:

```
import pexpect
remove = pexpect.spawn('sudo dpkg --purge mytool.deb')
remove.logfile = open('log/expect-uninstall-deb.log', 'w')
remove.logfile.write('try to dpkg --purge mytool\n')
if remove.expect(['(?i)password.*']) == 0:
    # print "successfull"
    remove.sendline('mypassword')
    time.sleep(2)
    remove.expect(pexpect.EOF,5)
else:
    raise AssertionError("Fail to Uninstall deb package !")
```

ShareImprove this answerFollow

answered Nov 7, 2013 at 7:24
Snap
11 1

1

To limit what you run as sudo, you could run

```
python non_sudo_stuff.py
sudo -E python -c "import os; os.system('sudo echo 1')"
```

without needing to store the password. The -E parameter passes your current user's env to the process. Note that your shell will have sudo priveleges after the second command, so use with caution!

ShareImprove this answerFollow

edited Dec 7, 2017 at 21:55

answered Dec 7, 2017 at 21:21
crizCraig
8,697 6 54 53

0

I know it is always preferred not to hardcode the sudo password in the script. However, for some reason, if you have no permission to modify /etc/sudoers or change file owner, Pexpect is a feasible alternative.

Here is a Python function sudo_exec for your reference:

```
import platform, os, logging
import subprocess, pexpect

log = logging.getLogger(__name__)

def sudo_exec(cmdline, passwd):
    osname = platform.system()
    if osname == 'Linux':
        prompt = r'\[sudo\] password for %s: ' % os.environ['USER']
    elif osname == 'Darwin':
```

```
    child = pexpect.spawn(cmdline)
    idx = child.expect([prompt, pexpect.EOF], 3)
    if idx == 0: # if prompted for the sudo password
        log.debug('sudo password was asked.')
        child.sendline(passwd)
        child.expect(pexpect.EOF)
return child.before
```

ShareImprove this answerFollow

answered Sep 29, 2014 at 9:38

Jeremy Kao
913 10 14

0

It works in python 2.7 and 3.8:

```
from subprocess import Popen, PIPE
from shlex import split

proc = Popen(split('sudo -S %s' % command), bufsize=0, stdout=PIPE, stdin=PIPE, stderr=PIPE)
proc.stdin.write((password +'\n').encode()) # write as bytes
proc.stdin.flush() # need if not bufsize=0 (unbuffered stdin)
```

without .flush() password will not reach sudo if stdin buffered. In python 2.7 Popen by default used bufsize=0 and stdin.flush() was not needed.

For secure using, create password file in protected directory:

```
mkdir --mode=700 ~/.prot_dir
nano ~/.prot_dir/passwd.txt
chmod 600 ~/.prot_dir/passwd.txt
```

at start your py-script read password from ~/.prot_dir/passwd.txt

```
with open(os.environ['HOME'] +'/.prot_dir/passwd.txt') as f:
    password = f.readline().rstrip()
```

ShareImprove this answerFollow

edited Oct 1, 2020 at 11:32

answered Sep 30, 2020 at 19:53

evan
31 2

0

```
import os
os.system("echo TYPE_YOUR_PASSWORD_HERE | sudo -S TYPE_YOUR_LINUX_COMMAND")
```

Open your ide and run the above code. Please change TYPE_YOUR_PASSWORD_HERE and TYPE_YOUR_LINUX_COMMAND to your linux admin password and your desired linux command after that run your python script. Your output will show on terminal. Happy Coding :)

ShareImprove this answerFollow

answered Sep 14, 2022 at 17:40

user14106084

0

You can use SSHScript . Below are example codes:

```
## filename: example.spy
sudoPassword = 'mypass'
command = 'mount -t vboxsf myfolder /home/myuser/myfolder'
$$echo @{sudoPassword} | sudo -S @{command}
```

or, simply one line (almost the same as running on console)

```
## filename: example.spy
$$echo mypass | sudo -S mount -t vboxsf myfolder /home/myuser/myfolder
```

Then, run it on console

```
sshscript example.spy
```

Where "sshscript" is the CLI of SSHScript (installed by pip).

answered Oct 19, 2022 at 2:25

Singu Singuan Iap
31 1 4

0

solution im going with,because password in plain txt in an env file on dev pc is ok, and variable in the repo and gitlab runner is masked.

use .dotenv put pass in .env on local machine, DONT COMMIT .env to git. add same var in gitlab variable

.env file has:
PASSWORD=superpass

```
from dotenv import load_dotenv
load_dotenv()

subprocess.run(f'echo {os.getenv("PASSWORD")} | sudo -S rm  /home//folder/filetodelete_created_as_root.txt',
shell=True, check=True)
```

this works locally and in gitlab. no plain password is committed to repo.

yes, you can argue running a sudo command w shell true is kind of crazy, but if you have files written to host from a docker w root, and you need to pro-grammatically delete them, this is functional.

ShareImprove this answerFollow

answered Oct 26, 2022 at 14:36

Dave
1

As it’s currently written, your answer is unclear. Please edit to add additional details that will help others understand how this addresses the question asked. You can find more information on how to write good answers in the help center. – user11717481 Oct 30, 2022 at 8:16