2.2 Python virtualenv - custom Python versions

Overview

This is a short guide on using Python virtual environments to achieve isolated instances of specific Python versions.

TL;DR

For those that just want the commands, scroll to the end for everything needed without explanation.

Why is this important?

With the retirement of Python2, many exploits and code blocks you will encounter working your way through machines will no longer work on Kali. If you try and install the dependencies for Python2, you will likely break something else that you will run into down the line (or immediately).

Your options are to either:

1. Convert the exploit to Python3
2. Run in a container
3. Run in a virtual environment
4. Possibly break your install by installing / removing Python and its libraries.

A basic environment

First up, install the required packages. I'm using `Kali` but this will likely be fine for any Debian based distribution.

Installing packages

```
┌──(kali㉿kali)-[~]
└─$ sudo apt-get install virtualenv
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python3-distlib python3-filelock python3-virtualenv
The following NEW packages will be installed:
  python3-distlib python3-filelock python3-virtualenv virtualenv
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 239 kB of archives.
After this operation, 1,057 kB of additional disk space will be used.
```

```
Do you want to continue? [Y/n] y
<snip>
Setting up virtualenv (20.2.1+ds-1) ...
Processing triggers for man-db (2.9.3-2) ...
Processing triggers for kali-menu (2021.1.2) ...
```

Creating the environment folder

Next, we need to make a folder to hold the environment. This would normally be where we install our script. For example, below I'll make a folder called `python` and inside I could have `exploit.py` to keep things organised.

For this example, let's place our end goal as creating a `Python2` virtual environment as our exploit won't work on `Python3`

```
┌──(kali㉿kali)-[~]
└─$ mkdir python
┌──(kali㉿kali)-[~]
└─$ cd python

┌──(kali㉿kali)-[~/python]
└─$
```

Creating the environment

So our folder is made, time to create the environment.

```
┌──(kali㉿kali)-[~/python]
└─$ virtualenv venv
created virtual environment CPython3.9.1.final.0-64 in 578ms
  creator CPython3Posix(dest=/home/kali/python/venv, clear=False,
no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle,
wheel=bundle, via=copy, app_data_dir=/home/kali/.local/share/virtualenv)
    added seed packages: pip==20.3.1, setuptools==51.0.0, wheel==0.36.1
  activators
BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActi
vator,XonshActivator
```

Can you see the problem we are about to run into? If not, it will be clear in a sec.

Activating the environment

To activate the environment, we use the `source` command. Once it's activated, `(venv)` will appear in the terminal. That's the name we set when we created it using `virtualenv venv`.

```
┌──(kali㉿kali)-[~/python]
└─$ source venv/bin/activate
```

Checking the version

So we have created and activated the environment, let's check the version of Python within the container.

```
(venv) ┌──(kali㉿kali)-[~/python]
└─$ python -v
<snip>
Python 3.9.1 (default, Dec  8 2020, 07:51:42)
<snip>
>>>
```

No good, it has defaulted to `Python 3.9.1`. That won't help us with the goal of running `Python2` exploits.

Deactivate the virtual environment

As it's not what we want, type `deactivate` to leave the environment.

Installing a custom version of Python

So, let's go all in. Time to install a custom version of Python that won't break our current install.

Finding Python

We are going to build from source, to do that, we need the source code. You can find a repo holding all of it here:

https://www.python.org/ftp/python/

I'm going to install the most modern Python2 version which is `Python 2.7.18`

The repo is at https://www.python.org/ftp/python/2.7.18/ and the link to the source is https://www.python.org/ftp/python/2.7.18/Python-2.7.18.tgz

Downloading and compiling custom python

Make and move to the folder

```
┌──(kali㉿kali)-[~/]
└─$ mkdir tmp && cd tmp
```

Download Python

```
┌──(kali㉿kali)-[~/tmp]
└─$ wget http://www.python.org/ftp/python/2.7.18/Python-2.7.18.tgz
```

Extract it and move into the folder

```
┌──(kali㉿kali)-[~/tmp]
└─$ tar zxvf Python-2.7.18.tgz && cd Python-2.7.18
```

Configure it:

This is important. The `--prefix` is where it will install. In this example, as I am the `kali` user, it will install to `/home/kali/opt/python-2.7.18`. We are also going to make sure `pip` is installed to handle packages

```
┌──(kali㉿kali)-[~/tmp/Python-2.7.18]
└─$ ./configure --prefix=$HOME/opt/python-2.7.18 --with-ensurepip=install
```

Now that is done, it's time to compile it. You may be some `warnings` but that's OK.

```
┌──(kali㉿kali)-[~/tmp/Python-2.7.18]
└─$ make && make install
```

If that all worked out, the custom version of Python is compiled and installed at `$HOME/opt/python-2.7.18`. Keep in mind you don't need to keep it there, I typically store everything in `/opt/` on Kali when installing apps.

Using custom Python in a Virtual Environment

This part is pretty simple. Just need to make a new folder, create, and active the environment with a slightly different create command than we have seen before.

Create and move into the folder

```
┌──(kali㉿kali)-[~]
└─$ cd ~ && mkdir python2.7 && cd python2.7
┌──(kali㉿kali)-[~/python2.7]
```

Create the environment with custom Python

We are going to use the `-p` argument with the `virtualenv` command to define the path to Python we want to use within our environment. As we installed it into `$HOME/opt/python-2.7.18` we need to use that. We are after the `python` binary within the `bin` folder.

```
┌──(kali㉿kali)-[~/python2.7]
└─$ virtualenv -p /home/kali/opt/python-2.7.18/bin/python venv
created virtual environment CPython2.7.18.final.0-64 in 571ms
  creator CPython2Posix(dest=/home/kali/python2.7/venv, clear=False,
no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle,
wheel=bundle, via=copy, app_data_dir=/home/kali/.local/share/virtualenv)
    added seed packages: pip==20.2.4, setuptools==44.1.1, wheel==0.35.1
  activators
BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActi
vator
```

Activate the environment

Run the source command:

```
┌──(kali㉿kali)-[~/python2.7]
└─$ source venv/bin/activate

(venv) ┌──(kali㉿kali)-[~/python2.7]
└─$
```

Check the version:

```
(venv) ┌──(kali㉿kali)-[~/python2.7]
└─$ python -v
<snip>
Python 2.7.18 (default, Dec 18 2020, 18:13:23)
[GCC 10.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Look at that, we have a winner. We are now running `Python 2.7.18` which is the custom version of Python we installed. You can now copy over your exploit and install the packages needed with `pip`

TL;DR

For those of you that don't want the text and just want the commands, the block below will setup an environment for you.

```
# Make a working folder
mkdir ~/tmp && cd ~/tmp
```

```
# Download Python
wget http://www.python.org/ftp/python/2.7.18/Python-2.7.18.tgz

# Extract Python
tar zxvf Python-2.7.18.tgz && cd Python-2.7.18

# Configure Python install folder and install pip
./configure --prefix=$HOME/opt/python-2.7.18 --with-ensurepip=install

# Compile and install Python
make && make install

# Make a folder for python2.7 venv
mkdir ~/python2.7-venv && cd  python2.7-venv

# Create the environment
virtualenv -p /home/kali/opt/python-2.7.18/bin/python venv

# Activate the environment
source venv/bin/activate

# Deactivate the environment
deactivate
```