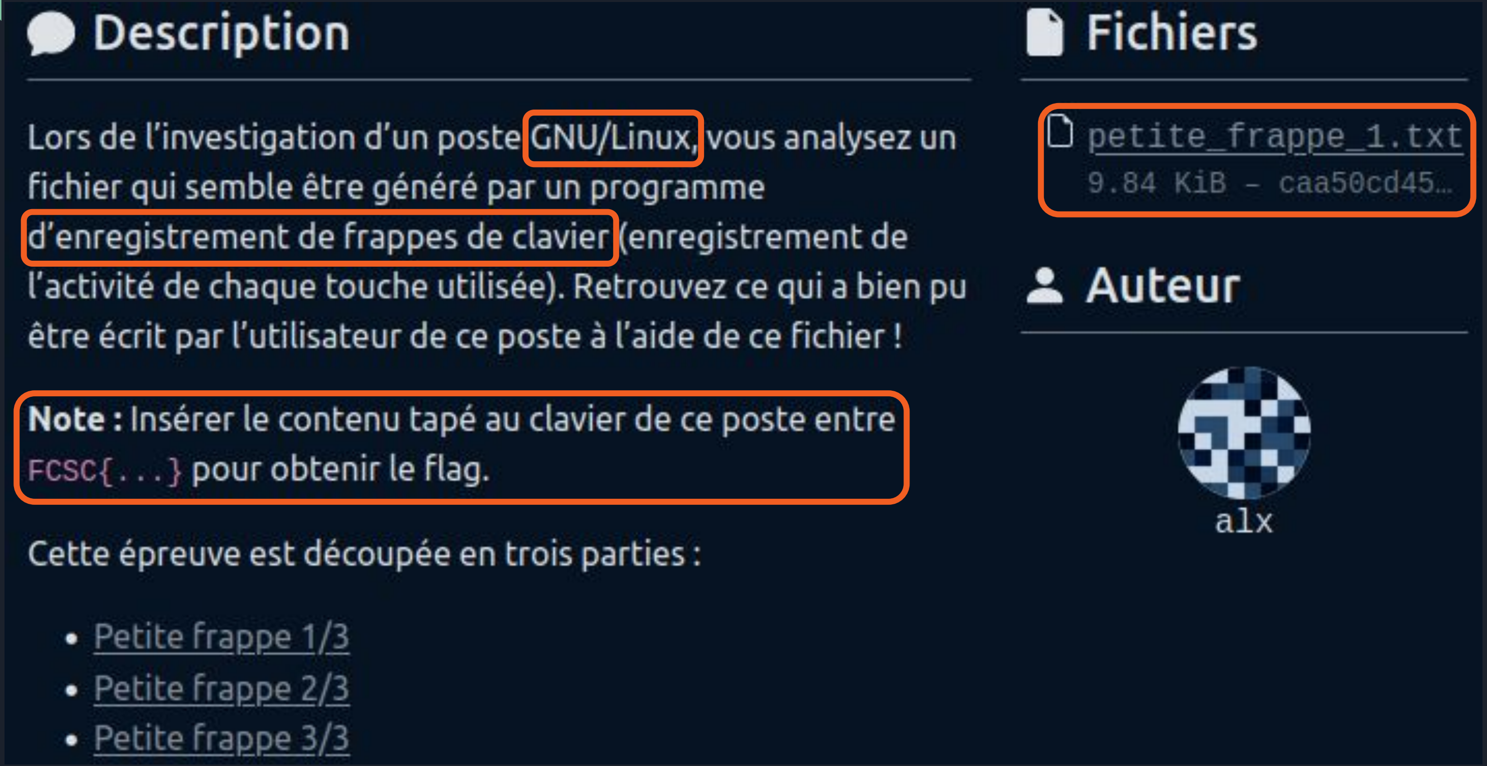
Quick Write Up: Track Petite frappe

FCSC 2020 - Yxène

Petite frappe 1/3 - Énoncé - intro



Description

Lors de l'investigation d'un poste GNU/Linux, vous analysez un fichier qui semble être généré par un programme d'enregistrement de frappes de clavier (enregistrement de l'activité de chaque touche utilisée). Retrouvez ce qui a bien pu être écrit par l'utilisateur de ce poste à l'aide de ce fichier !

Note : Insérer le contenu tapé au clavier de ce poste entre `FCSC{...}` pour obtenir le flag.

Cette épreuve est découpée en trois parties :

- Petite frappe 1/3
- Petite frappe 2/3
- Petite frappe 3/3

Fichiers

petite_frappe_1.txt
9.84 KiB – caa50cd45…

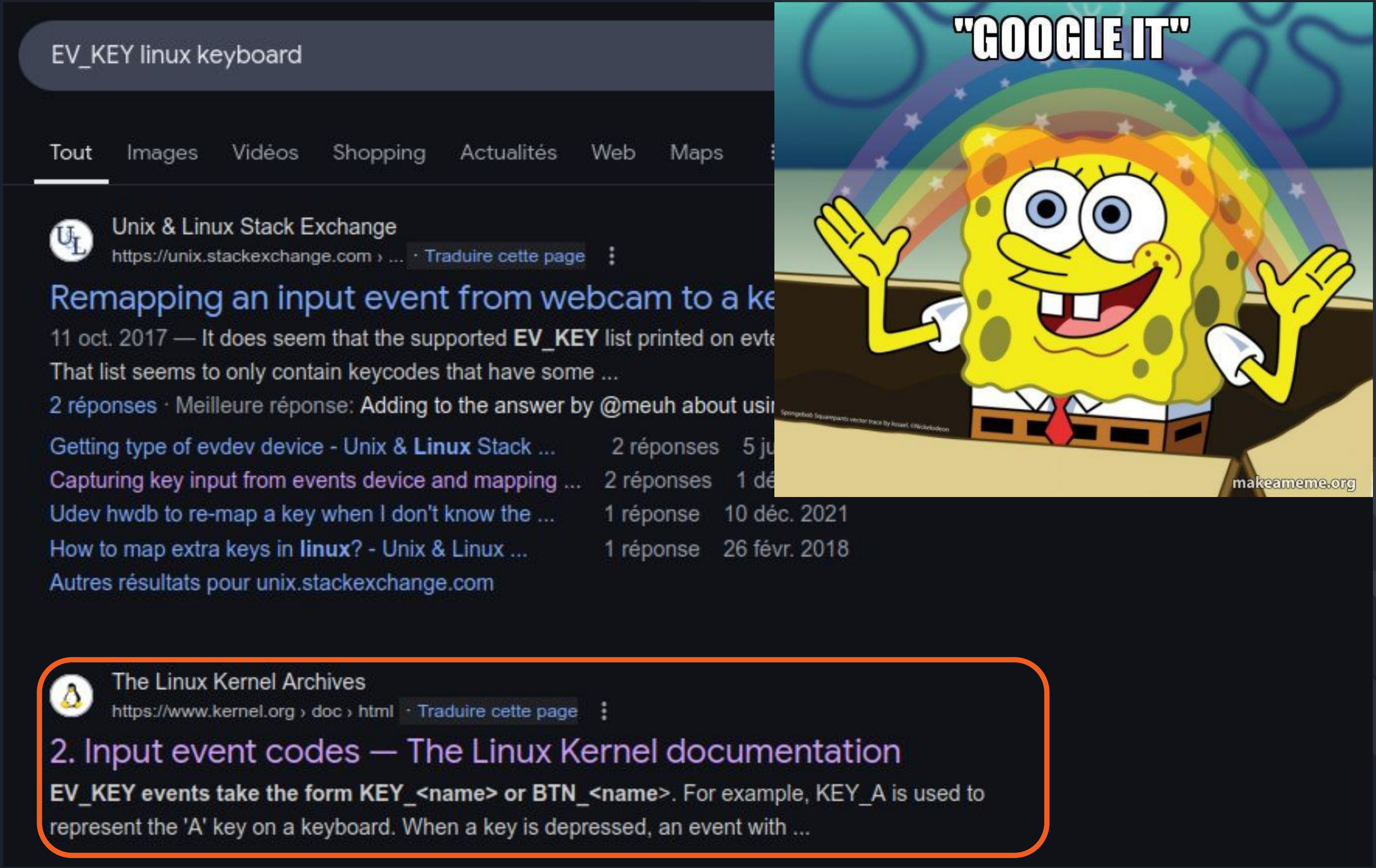
Auteur

alx

petite_frappe_1.txt

```
cat petite_frappe_1.txt
```

```
Event: time 1584656705.424839, -------------- SYN_REPORT ------------
Event: time 1584656706.404214, type 4 (EV_MSC), code 4 (MSC_SCAN), value 16
Event: time 1584656706.404214, type 1 (EV_KEY), code 22 (KEY_U), value 1
Event: time 1584656706.404214, -------------- SYN_REPORT ------------
Event: time 1584656706.508350, type 4 (EV_MSC), code 4 (MSC_SCAN), value 16
Event: time 1584656706.508350, type 1 (EV_KEY), code 22 (KEY_U), value 0
Event: time 1584656706.508350, -------------- SYN_REPORT ------------
Event: time 1584656706.674591, type 4 (EV_MSC), code 4 (MSC_SCAN), value 31
Event: time 1584656706.674591, type 1 (EV_KEY), code 49 (KEY_N), value 1
Event: time 1584656706.674591, -------------- SYN_REPORT ------------
Event: time 1584656706.774463, type 4 (EV_MSC), code 4 (MSC_SCAN), value 31
Event: time 1584656706.774463, type 1 (EV_KEY), code 49 (KEY_N), value 0
Event: time 1584656706.774463, -------------- SYN_REPORT ------------
Event: time 1584656706.926206, type 4 (EV_MSC), code 4 (MSC_SCAN), value 12
Event: time 1584656706.926206, type 1 (EV_KEY), code 18 (KEY_E), value 1
Event: time 1584656706.926206, -------------- SYN_REPORT ------------
Event: time 1584656707.023728, type 4 (EV_MSC), code 4 (MSC_SCAN), value 12
Event: time 1584656707.023728, type 1 (EV_KEY), code 18 (KEY_E), value 0
...
```



EV_KEY linux keyboard

Tout · Images · Vidéos · Shopping · Actualités · Web · Maps

Unix & Linux Stack Exchange
https://unix.stackexchange.com › ... · Traduire cette page

Remapping an input event from webcam to a ke

11 oct. 2017 — It does seem that the supported **EV_KEY** list printed on evte
That list seems to only contain keycodes that have some ...
2 réponses · Meilleure réponse: Adding to the answer by @meuh about usi

Getting type of evdev device - Unix & **Linux** Stack ...	2 réponses	5 ju
Capturing key input from events device and mapping ...	2 réponses	1 dé
Udev hwdb to re-map a key when I don't know the ...	1 réponse	10 déc. 2021
How to map extra keys in **linux**? - Unix & Linux ...	1 réponse	26 févr. 2018

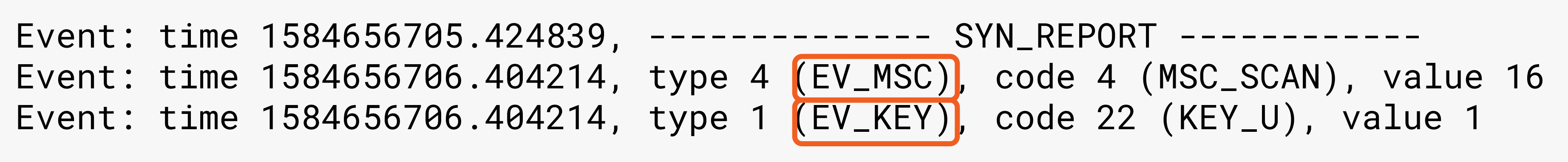
Autres résultats pour unix.stackexchange.com

The Linux Kernel Archives
https://www.kernel.org › doc › html · Traduire cette page

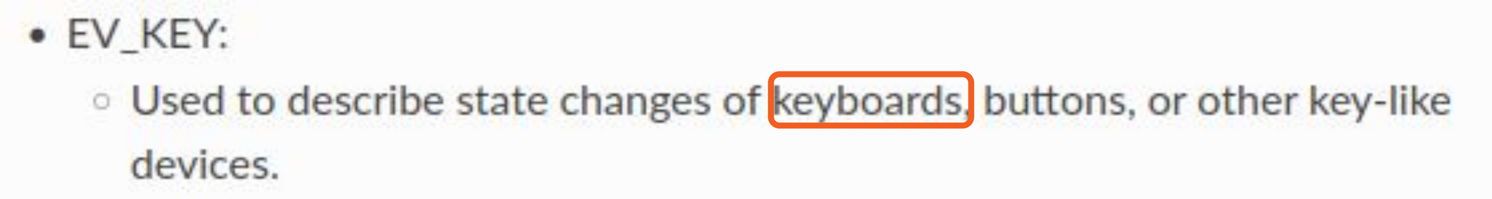
2. Input event codes — The Linux Kernel documentation

EV_KEY events take the form KEY_<name> or BTN_<name>. For example, KEY_A is used to represent the 'A' key on a keyboard. When a key is depressed, an event with ...

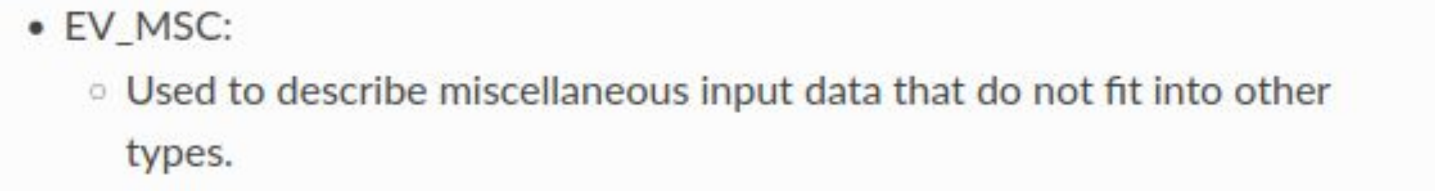
Doc Linux: Input event codes - Event types



```
Event: time 1584656705.424839, -------------- SYN_REPORT ------------
Event: time 1584656706.404214, type 4 (EV_MSC), code 4 (MSC_SCAN), value 16
Event: time 1584656706.404214, type 1 (EV_KEY), code 22 (KEY_U), value 1
```

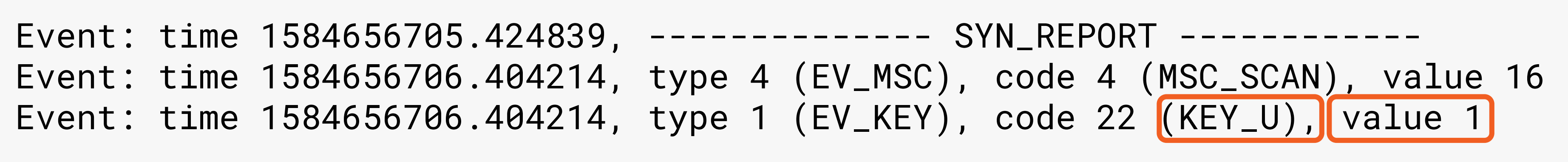


- EV_KEY:
  - Used to describe state changes of keyboards, buttons, or other key-like devices.

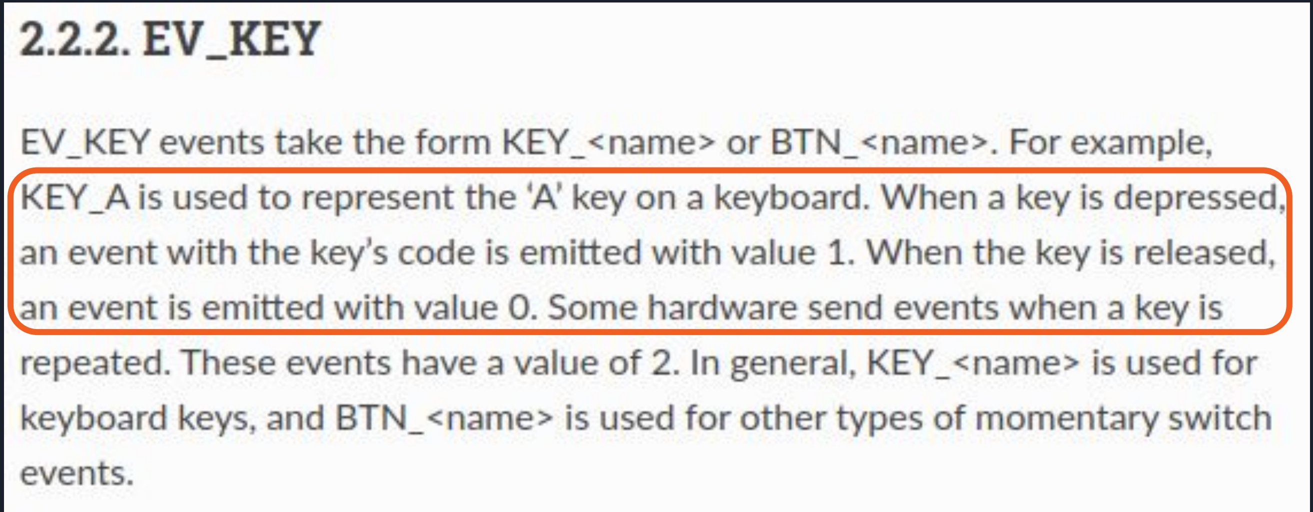


- EV_MSC:
  - Used to describe miscellaneous input data that do not fit into other types.

Doc Linux: Input event codes - EV_KEY



```
Event: time 1584656705.424839, -------------- SYN_REPORT ------------
Event: time 1584656706.404214, type 4 (EV_MSC), code 4 (MSC_SCAN), value 16
Event: time 1584656706.404214, type 1 (EV_KEY), code 22 (KEY_U), value 1
```



2.2.2. EV_KEY

EV_KEY events take the form KEY_<name> or BTN_<name>. For example, KEY_A is used to represent the 'A' key on a keyboard. When a key is depressed, an event with the key's code is emitted with value 1. When the key is released, an event is emitted with value 0. Some hardware send events when a key is repeated. These events have a value of 2. In general, KEY_<name> is used for keyboard keys, and BTN_<name> is used for other types of momentary switch events.

Le faire à la main en 30 sec
Coder pendant 10 min
imgflip.com

script1_3.py

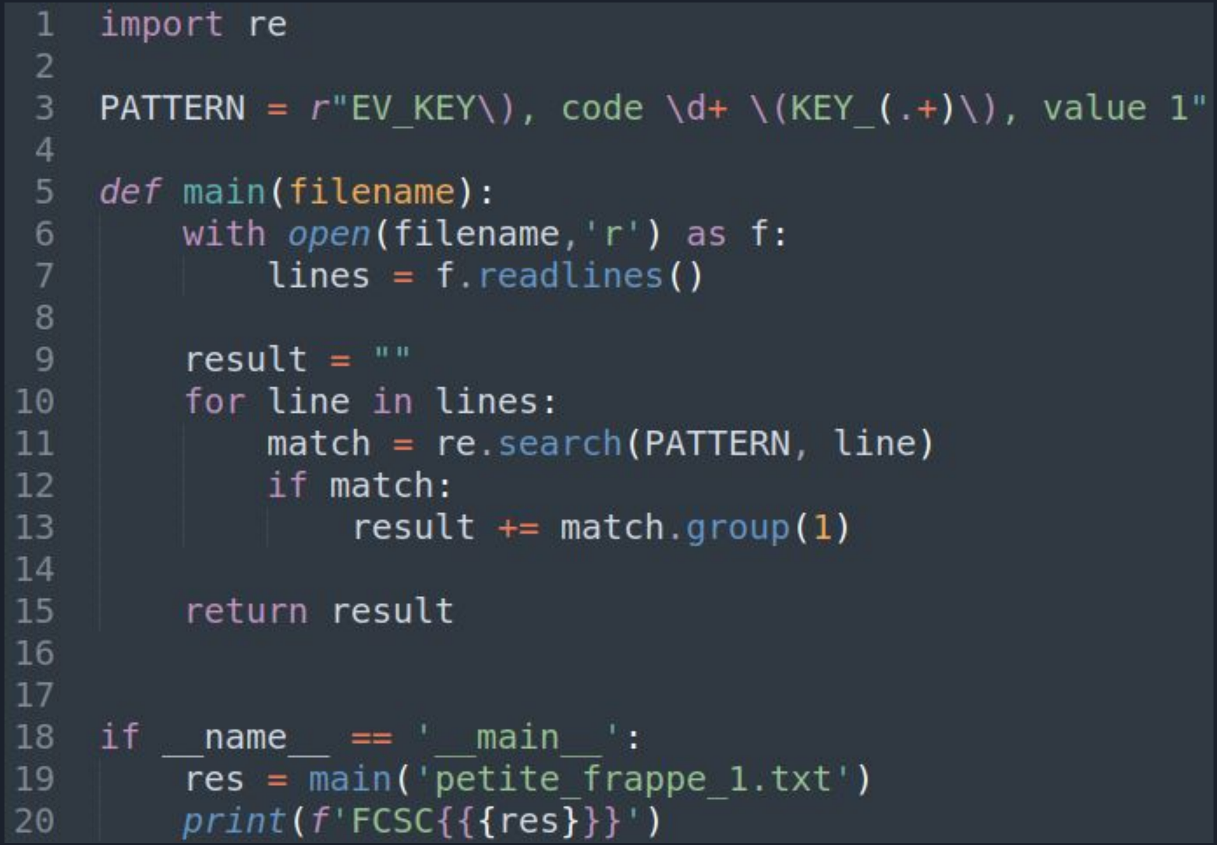
Motif à chercher →

Lecture du fichier →

Parcours ligne par ligne →

Ajout de la touche pressée au résultat →

Affichage du flag →



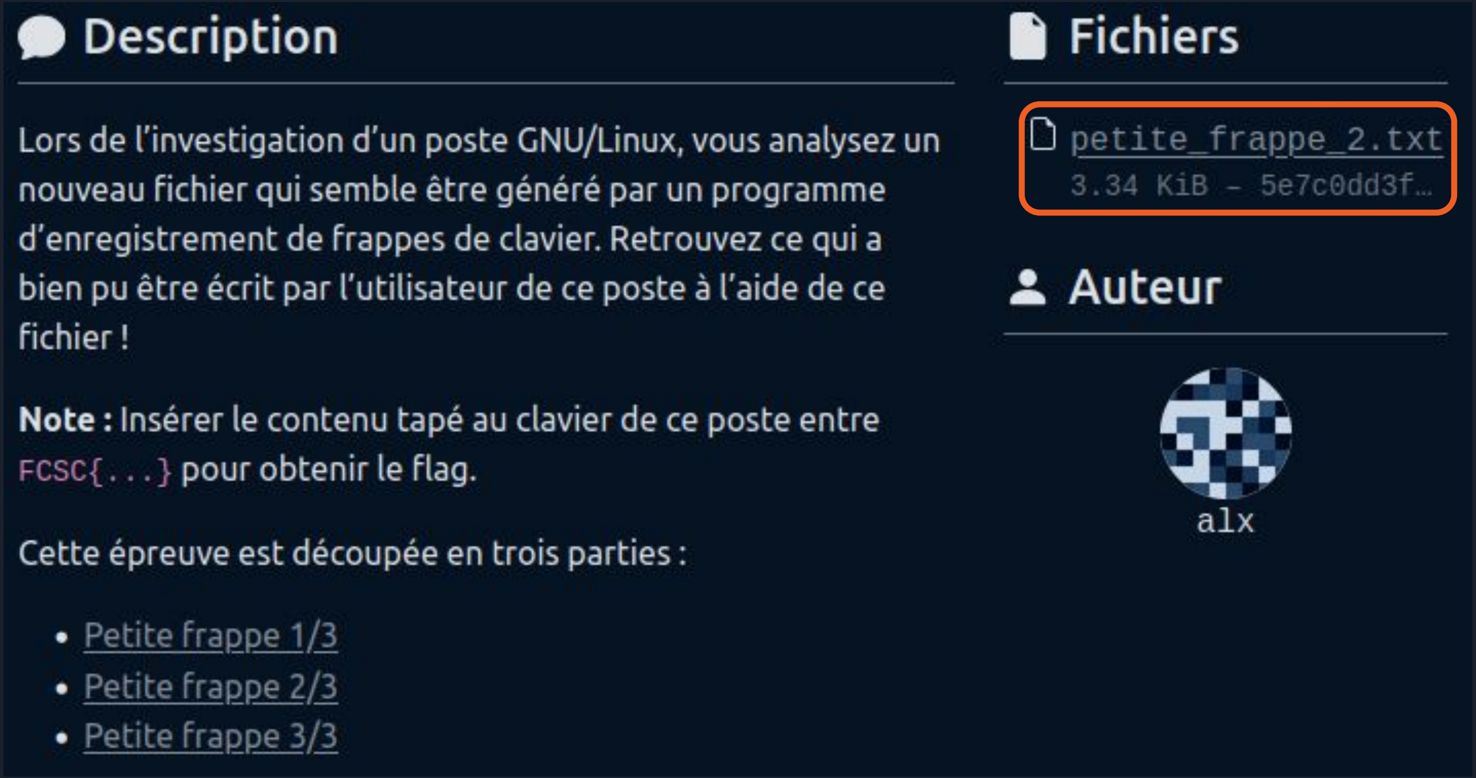
```python
import re

PATTERN = r"EV_KEY\), code \d+ \(KEY_(.+)\), value 1"

def main(filename):
    with open(filename,'r') as f:
        lines = f.readlines()

    result = ""
    for line in lines:
        match = re.search(PATTERN, line)
        if match:
            result += match.group(1)

    return result


if __name__ == '__main__':
    res = main('petite_frappe_1.txt')
    print(f'FCSC{{{res}}}')
```

Flag 1/3

```
python3 script1_3.py
```

```
FCSC{UNEGENTILLEINTRODUCTION}
```

Petite frappe 2/3 - Énoncé - ⭐



Description

Lors de l'investigation d'un poste GNU/Linux, vous analysez un nouveau fichier qui semble être généré par un programme d'enregistrement de frappes de clavier. Retrouvez ce qui a bien pu être écrit par l'utilisateur de ce poste à l'aide de ce fichier !

Note : Insérer le contenu tapé au clavier de ce poste entre `FCSC{...}` pour obtenir le flag.

Cette épreuve est découpée en trois parties :

- Petite frappe 1/3
- Petite frappe 2/3
- Petite frappe 3/3

Fichiers

petite_frappe_2.txt
3.34 KiB – 5e7c0dd3f…

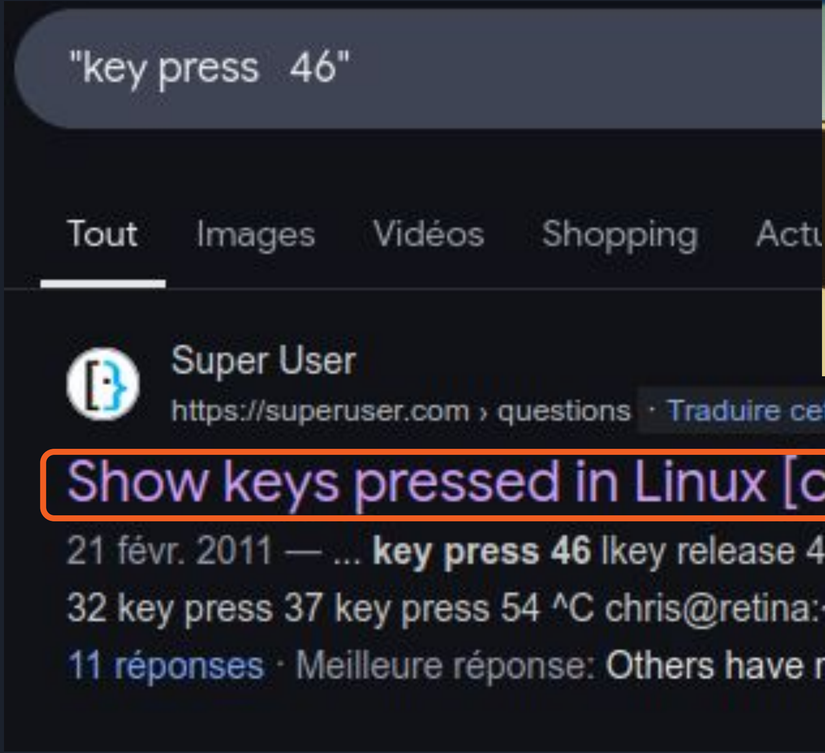
Auteur

alx

petite_frappe_2.txt

```
cat petite_frappe_2.txt
```

```
key press   46
key release 46
key press   24
key press   65
key release 24
key release 65
key press   39
key release 39
key press   32
key release 32
key press   46
key release 46
key press   30
key release 30
...
```

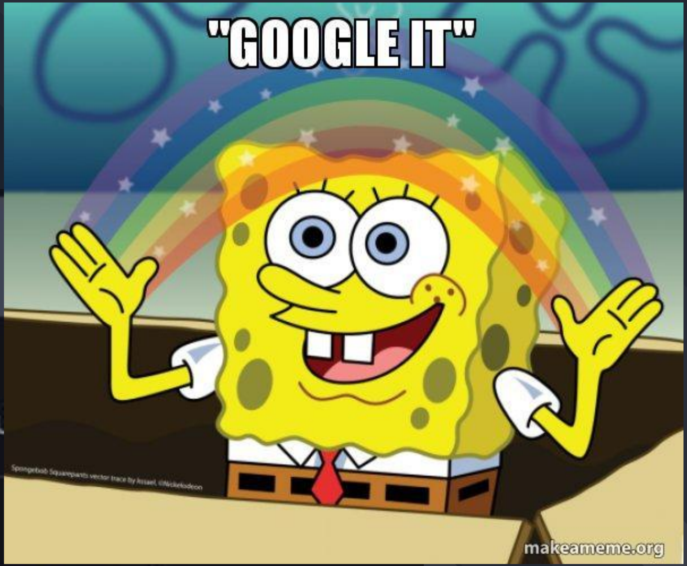


"key press 46"

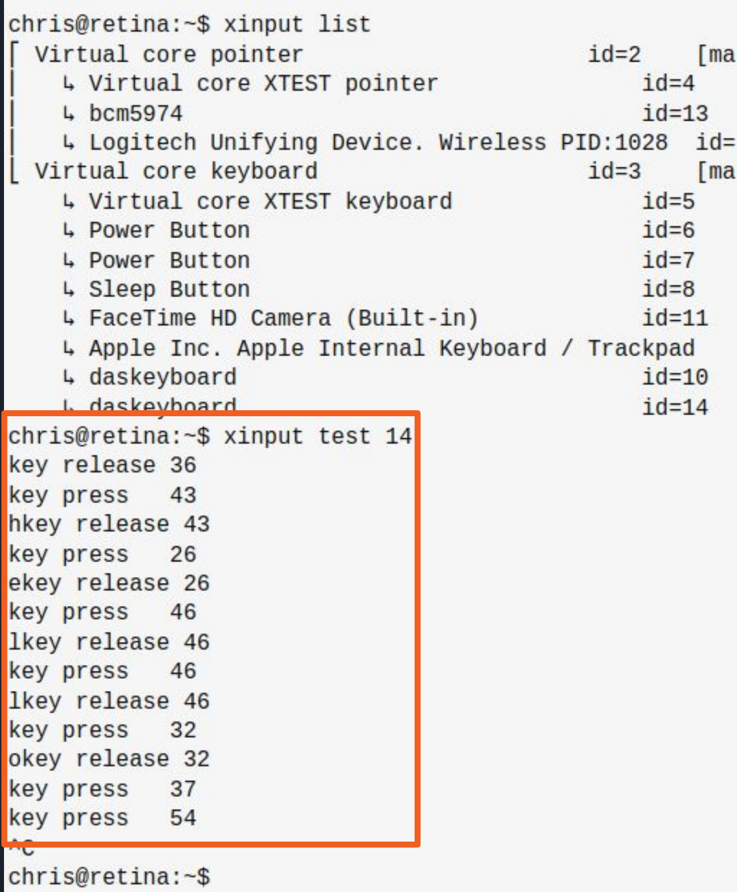
Super User
https://superuser.com › questions · Traduire cette page

Show keys pressed in Linux [closed]

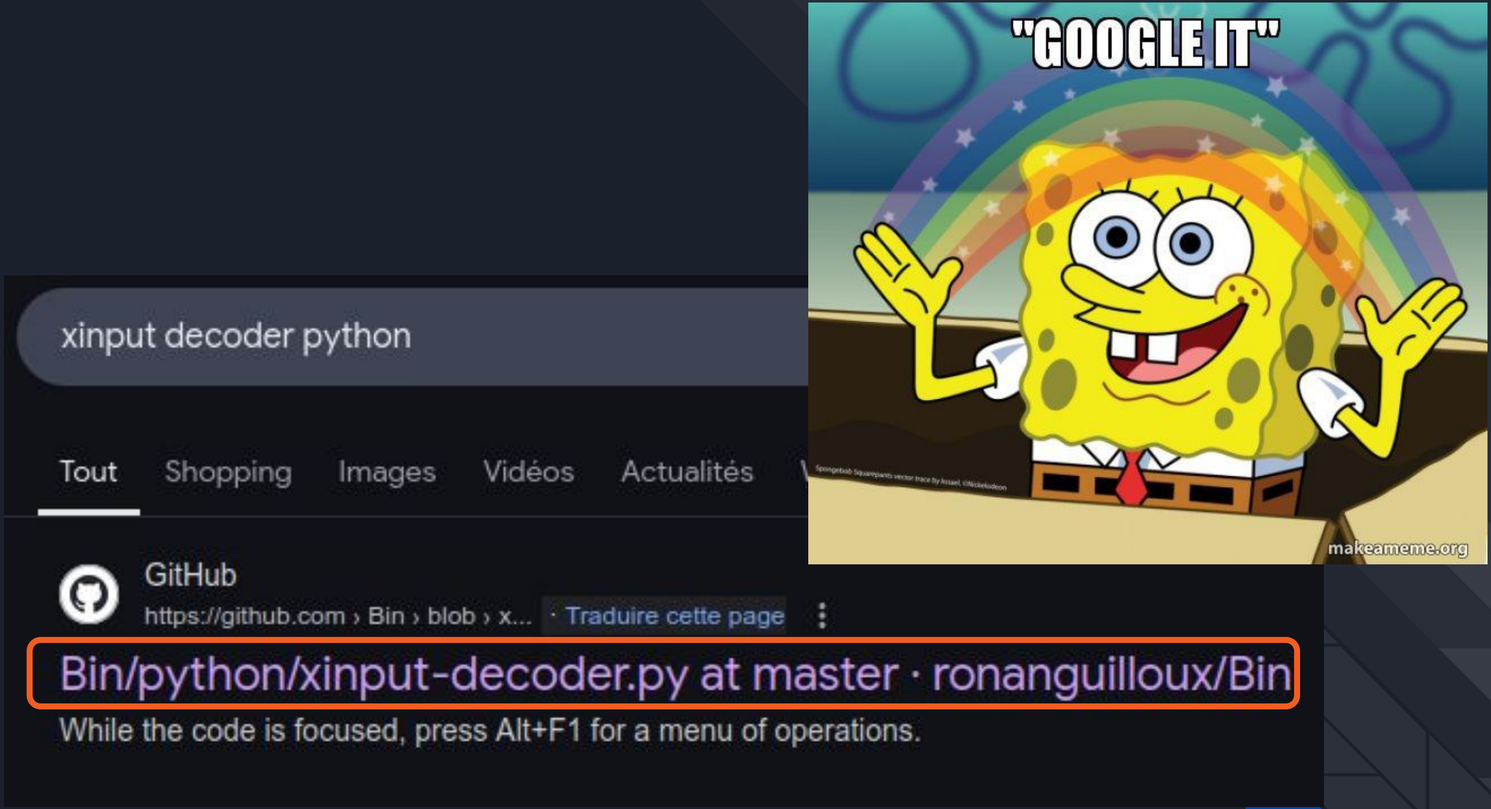
21 févr. 2011 — ... **key press 46** lkey release 46 **key press 46** lkey relea
32 key press 37 key press 54 ^C chris@retina:~$ Share.
11 réponses · Meilleure réponse: Others have mentioned xev, which is gc



xinput test



```
chris@retina:~$ xinput list
⎡ Virtual core pointer                    	id=2	[ma
⎜   ↳ Virtual core XTEST pointer              	id=4
⎜   ↳ bcm5974                                 	id=13
⎜   ↳ Logitech Unifying Device. Wireless PID:1028	id=
⎣ Virtual core keyboard                   	id=3	[ma
    ↳ Virtual core XTEST keyboard             	id=5
    ↳ Power Button                            	id=6
    ↳ Power Button                            	id=7
    ↳ Sleep Button                            	id=8
    ↳ FaceTime HD Camera (Built-in)           	id=11
    ↳ Apple Inc. Apple Internal Keyboard / Trackpad
    ↳ daskeyboard                             	id=10
    ↳ daskeyboard                             	id=14
chris@retina:~$ xinput test 14
key release 36
key press   43
hkey release 43
key press   26
ekey release 26
key press   46
lkey release 46
key press   46
lkey release 46
key press   32
okey release 32
key press   37
key press   54
^C
chris@retina:~$
```



xinput decoder python

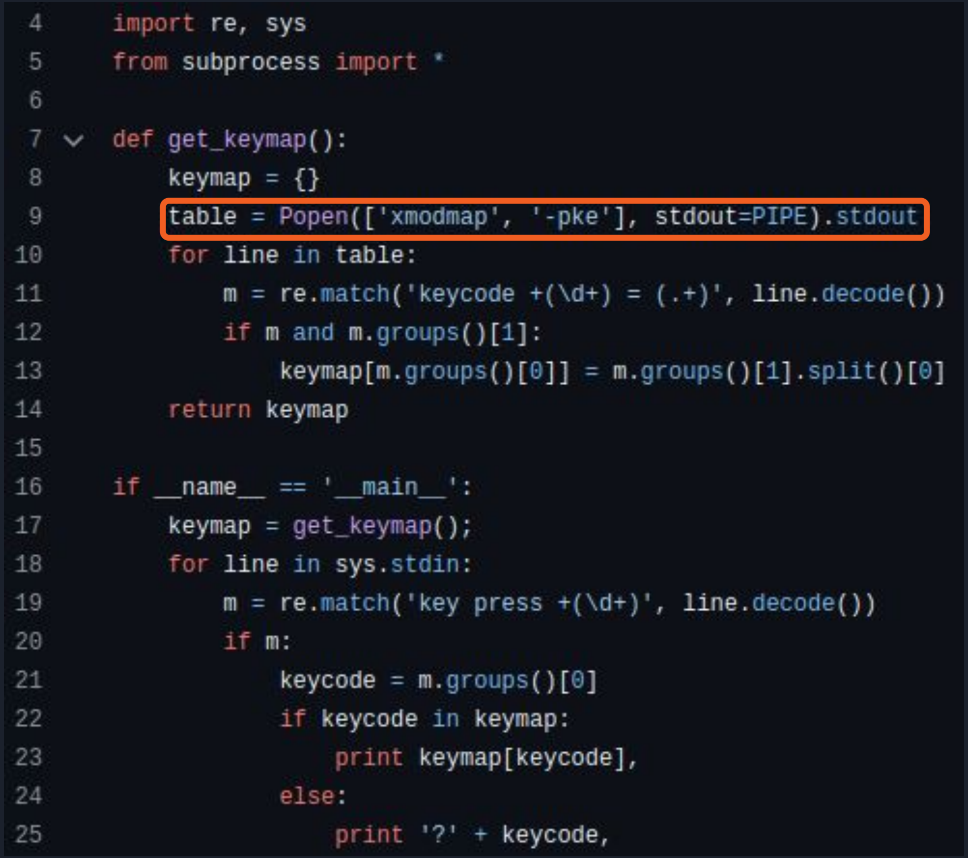
GitHub

https://github.com › Bin › blob › x... · Traduire cette page

Bin/python/xinput-decoder.py at master · ronanguilloux/Bin

While the code is focused, press Alt+F1 for a menu of operations.

xinput-decoder.py



```
import re, sys
from subprocess import *

def get_keymap():
    keymap = {}
    table = Popen(['xmodmap', '-pke'], stdout=PIPE).stdout
    for line in table:
        m = re.match('keycode +(\d+) = (.+)', line.decode())
        if m and m.groups()[1]:
            keymap[m.groups()[0]] = m.groups()[1].split()[0]
    return keymap

if __name__ == '__main__':
    keymap = get_keymap();
    for line in sys.stdin:
        m = re.match('key press +(\d+)', line.decode())
        if m:
            keycode = m.groups()[0]
            if keycode in keymap:
                print keymap[keycode],
            else:
                print '?' + keycode,
```

xmodmap -pke

```
xmodmap -pke
```

```
...
keycode  18 = ccedilla 9 ccedilla 9 asciicircum plusminus asciicircum
keycode  19 = agrave 0 agrave 0 at degree at
keycode  20 = parenright degree parenright degree bracketright questiondown bracketright
keycode  21 = equal plus equal plus braceright dead_ogonek braceright
keycode  22 = BackSpace BackSpace BackSpace BackSpace NoSymbol NoSymbol Terminate_Server
keycode  23 = Tab ISO_Left_Tab Tab ISO_Left_Tab
keycode  23 = Tab ISO_Left_Tab Tab ISO_Left_Tab
keycode  24 = a A a A ae AE ae
keycode  25 = z Z z Z guillemotleft less guillemotleft
keycode  26 = e E e E EuroSign cent EuroSign
keycode  27 = r R r R paragraph registered paragraph
keycode  28 = t T t T tslash Tslash tslash
keycode  29 = y Y y Y leftarrow yen leftarrow
keycode  30 = u U u U downarrow uparrow downarrow
keycode  31 = i I i I rightarrow idotless rightarrow
keycode  32 = o O o O oslash Oslash oslash
keycode  33 = p P p P thorn THORN thorn
...
```

script2_3.py

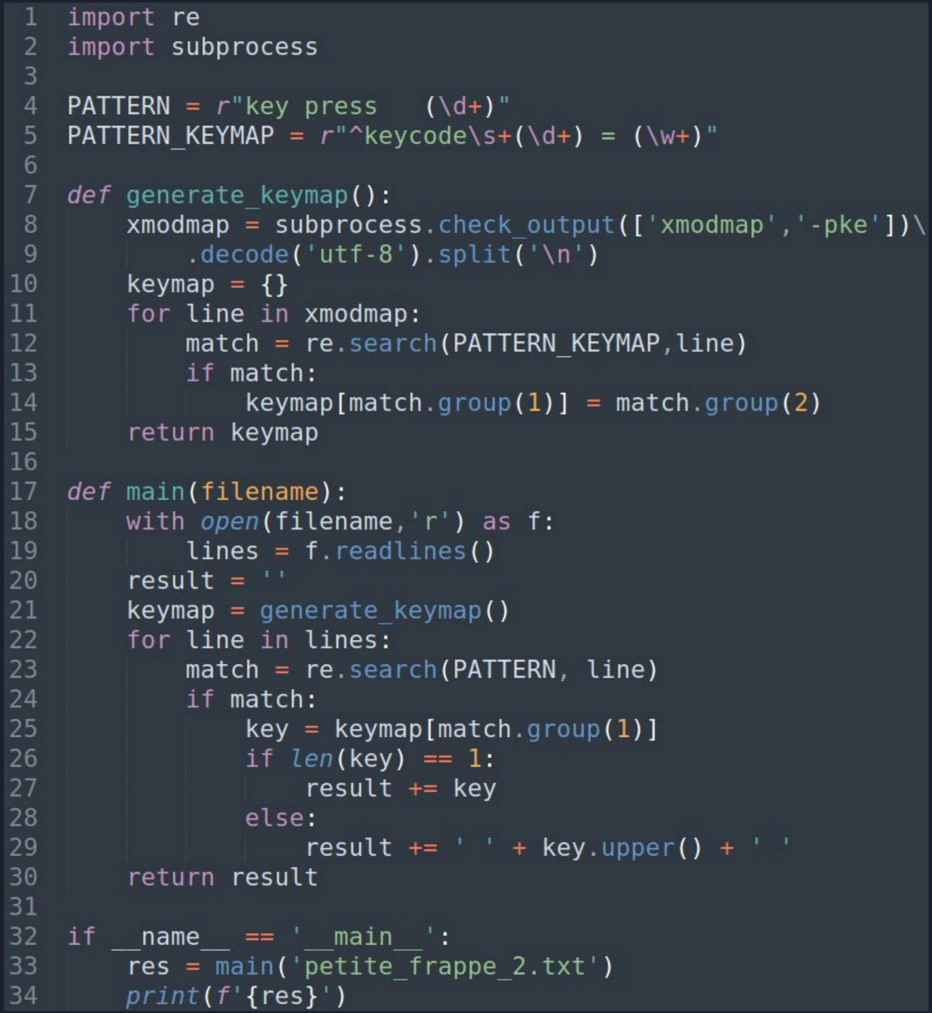
Commande *xmodmap -pke* →

Conversion en dictionnaire →

Ajout de la touche pressée au résultat →

Les touches non imprimables sont traitées différemment →

Affichage du texte →



```python
import re
import subprocess

PATTERN = r"key press   (\d+)"
PATTERN_KEYMAP = r"^keycode\s+(\d+) = (\w+)"

def generate_keymap():
    xmodmap = subprocess.check_output(['xmodmap','-pke'])\
        .decode('utf-8').split('\n')
    keymap = {}
    for line in xmodmap:
        match = re.search(PATTERN_KEYMAP,line)
        if match:
            keymap[match.group(1)] = match.group(2)
    return keymap

def main(filename):
    with open(filename,'r') as f:
        lines = f.readlines()
    result = ''
    keymap = generate_keymap()
    for line in lines:
        match = re.search(PATTERN, line)
        if match:
            key = keymap[match.group(1)]
            if len(key) == 1:
                result += key
            else:
                result += ' ' + key.upper() + ' '
    return result

if __name__ == '__main__':
    res = main('petite_frappe_2.txt')
    print(f'{res}')
```

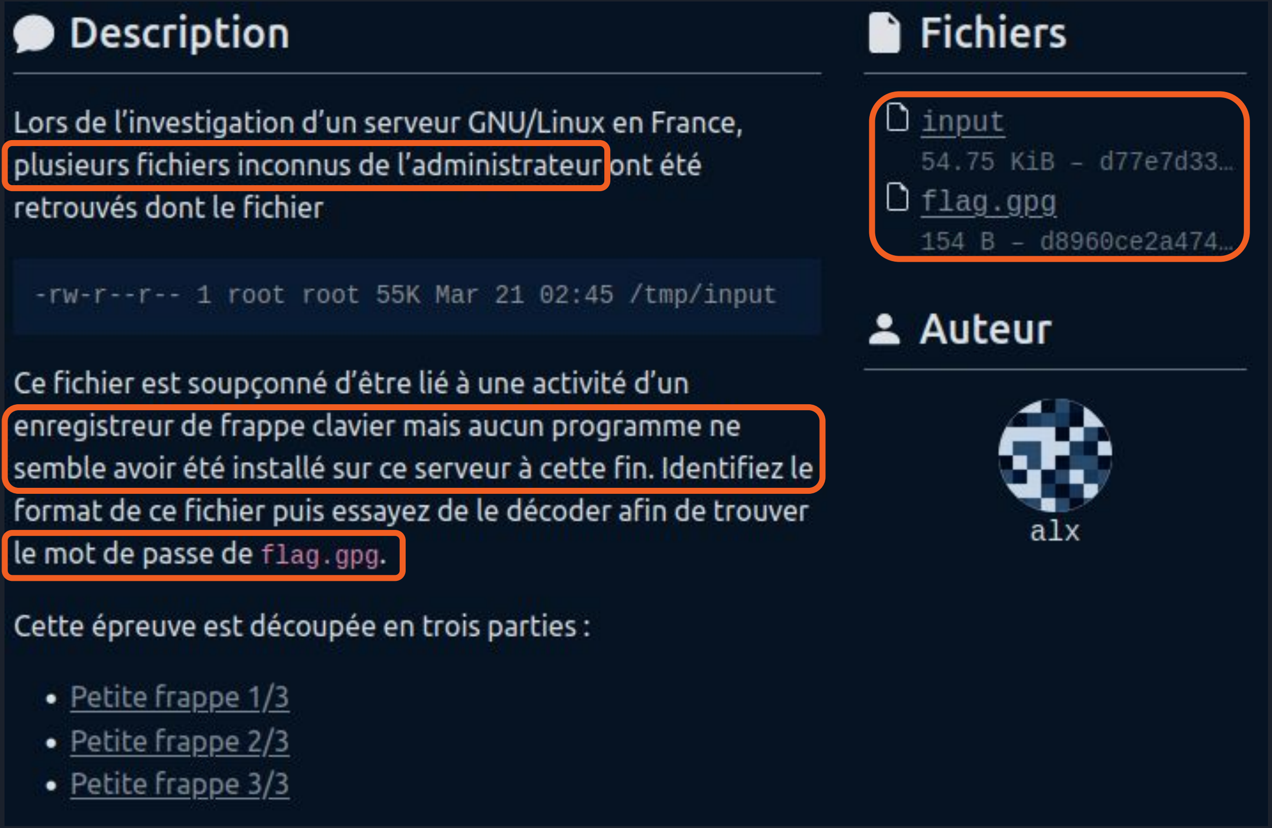
Flag 2/3

```
python3 script2_3.py
```

```
la SPACE solution SPACE avec SPACE xinput SPACE ne SPACE semble SPACE
pas SPACE super SPACE pratique SPACE a SPACE decoder SHIFT_R
SEMICOLON  SPACE le SPACE flag SPACE est SPACE un UNDERSCORE clavier
UNDERSCORE azerty UNDERSCORE en UNDERSCORE vaut UNDERSCORE deux
```

```
=> FCSC{un_clavier_azerty_en_vaut_deux}
```

Petite frappe 3/3 - Énoncé - ⭐⭐



Description

Lors de l'investigation d'un serveur GNU/Linux en France, plusieurs fichiers inconnus de l'administrateur ont été retrouvés dont le fichier

```
-rw-r--r-- 1 root root 55K Mar 21 02:45 /tmp/input
```

Ce fichier est soupçonné d'être lié à une activité d'un enregistreur de frappe clavier mais aucun programme ne semble avoir été installé sur ce serveur à cette fin. Identifiez le format de ce fichier puis essayez de le décoder afin de trouver le mot de passe de `flag.gpg`.

Cette épreuve est découpée en trois parties :

- Petite frappe 1/3
- Petite frappe 2/3
- Petite frappe 3/3

Fichiers

- input
  54.75 KiB – d77e7d33…
- flag.gpg
  154 B – d8960ce2a474…

Auteur

alx

Types de fichiers

```
file input flag.gpg
```

```
input:  data
flag.gpg: GPG symmetrically encrypted data (AES256 cipher)
```

Structure *input*

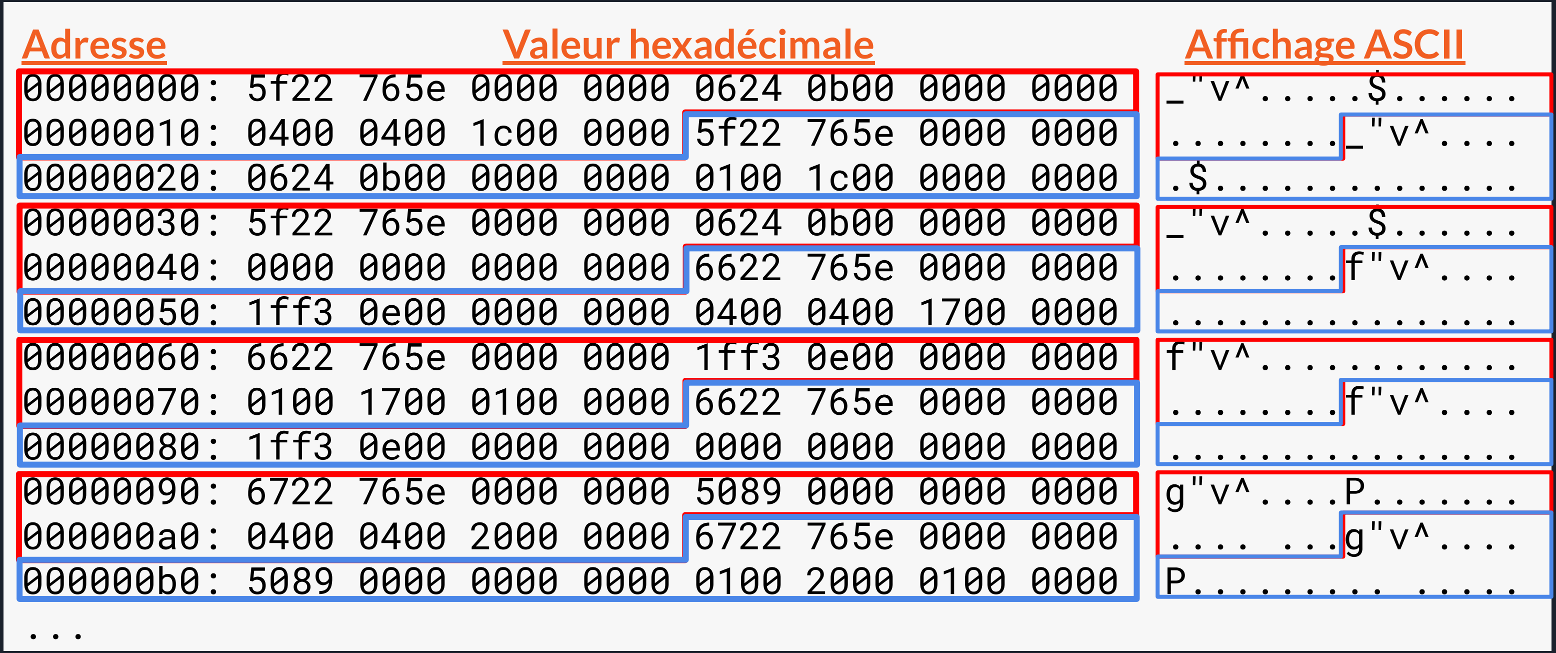
```
xxd input
```

Adresse Valeur hexadécimale Affichage ASCII

```
00000000: 5f22 765e 0000 0000 0624 0b00 0000 0000  _"v^.....$......
00000010: 0400 0400 1c00 0000 5f22 765e 0000 0000  ........_"v^....
00000020: 0624 0b00 0000 0000 0100 1c00 0000 0000  .$..............
00000030: 5f22 765e 0000 0000 0624 0b00 0000 0000  _"v^.....$......
00000040: 0000 0000 0000 0000 6622 765e 0000 0000  ........f"v^....
00000050: 1ff3 0e00 0000 0000 0400 0400 1700 0000  ................
00000060: 6622 765e 0000 0000 1ff3 0e00 0000 0000  f"v^............
00000070: 0100 1700 0100 0000 6622 765e 0000 0000  ........f"v^....
00000080: 1ff3 0e00 0000 0000 0000 0000 0000 0000  ................
00000090: 6722 765e 0000 0000 5089 0000 0000 0000  g"v^....P.......
000000a0: 0400 0400 2000 0000 6722 765e 0000 0000  .... ...g"v^....
000000b0: 5089 0000 0000 0000 0100 2000 0100 0000  P......... .....
...
```

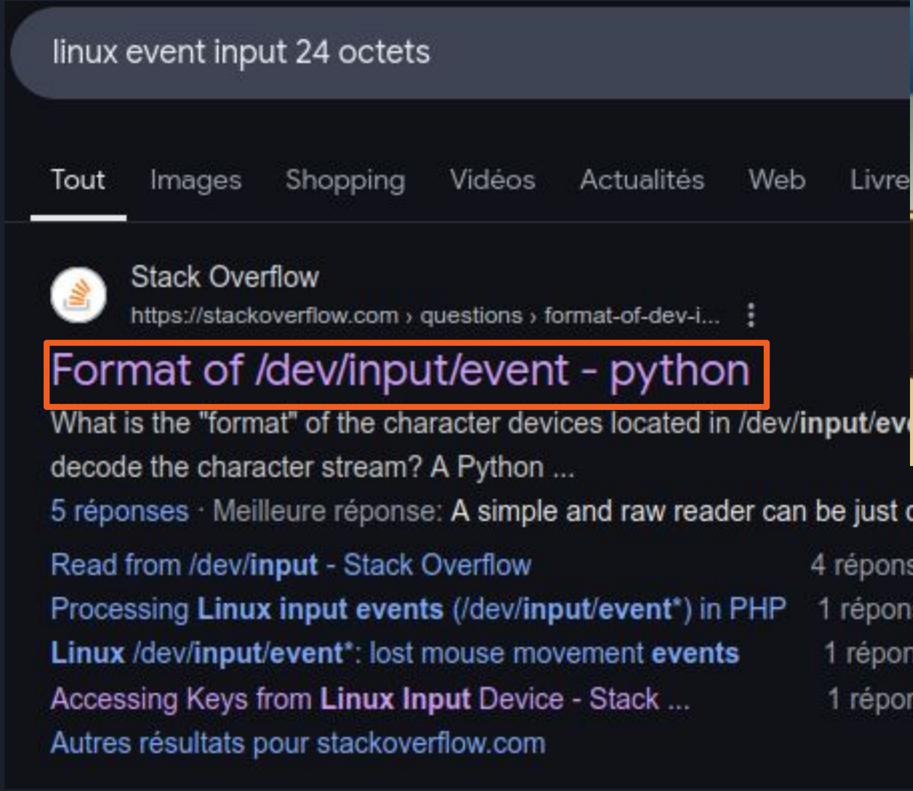
Structure *input*

```
xxd input
```



```
00000000: 5f22 765e 0000 0000 0624 0b00 0000 0000  _"v^.....$......
00000010: 0400 0400 1c00 0000 5f22 765e 0000 0000  ........_"v^....
00000020: 0624 0b00 0000 0000 0100 1c00 0000 0000  .$..............
00000030: 5f22 765e 0000 0000 0624 0b00 0000 0000  _"v^.....$......
00000040: 0000 0000 0000 0000 6622 765e 0000 0000  ........f"v^....
00000050: 1ff3 0e00 0000 0000 0400 0400 1700 0000  ................
00000060: 6622 765e 0000 0000 1ff3 0e00 0000 0000  f"v^............
00000070: 0100 1700 0100 0000 6622 765e 0000 0000  ........f"v^....
00000080: 1ff3 0e00 0000 0000 0000 0000 0000 0000  ................
00000090: 6722 765e 0000 0000 5089 0000 0000 0000  g"v^....P.......
000000a0: 0400 0400 2000 0000 6722 765e 0000 0000  .... ...g"v^....
000000b0: 5089 0000 0000 0000 0100 2000 0100 0000  P......... .....
...
```

Blocs de 24 octets



linux event input 24 octets

Tout · Images · Shopping · Vidéos · Actualités · Web · Livre

Stack Overflow
https://stackoverflow.com › questions › format-of-dev-i...

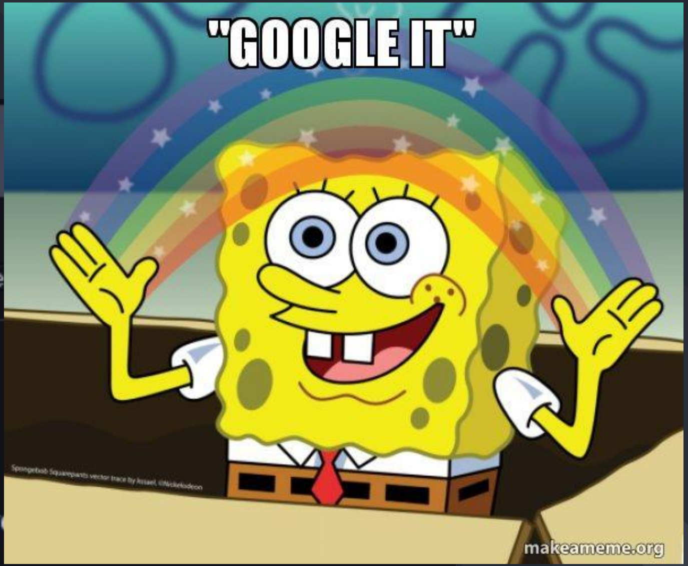
Format of /dev/input/event - python

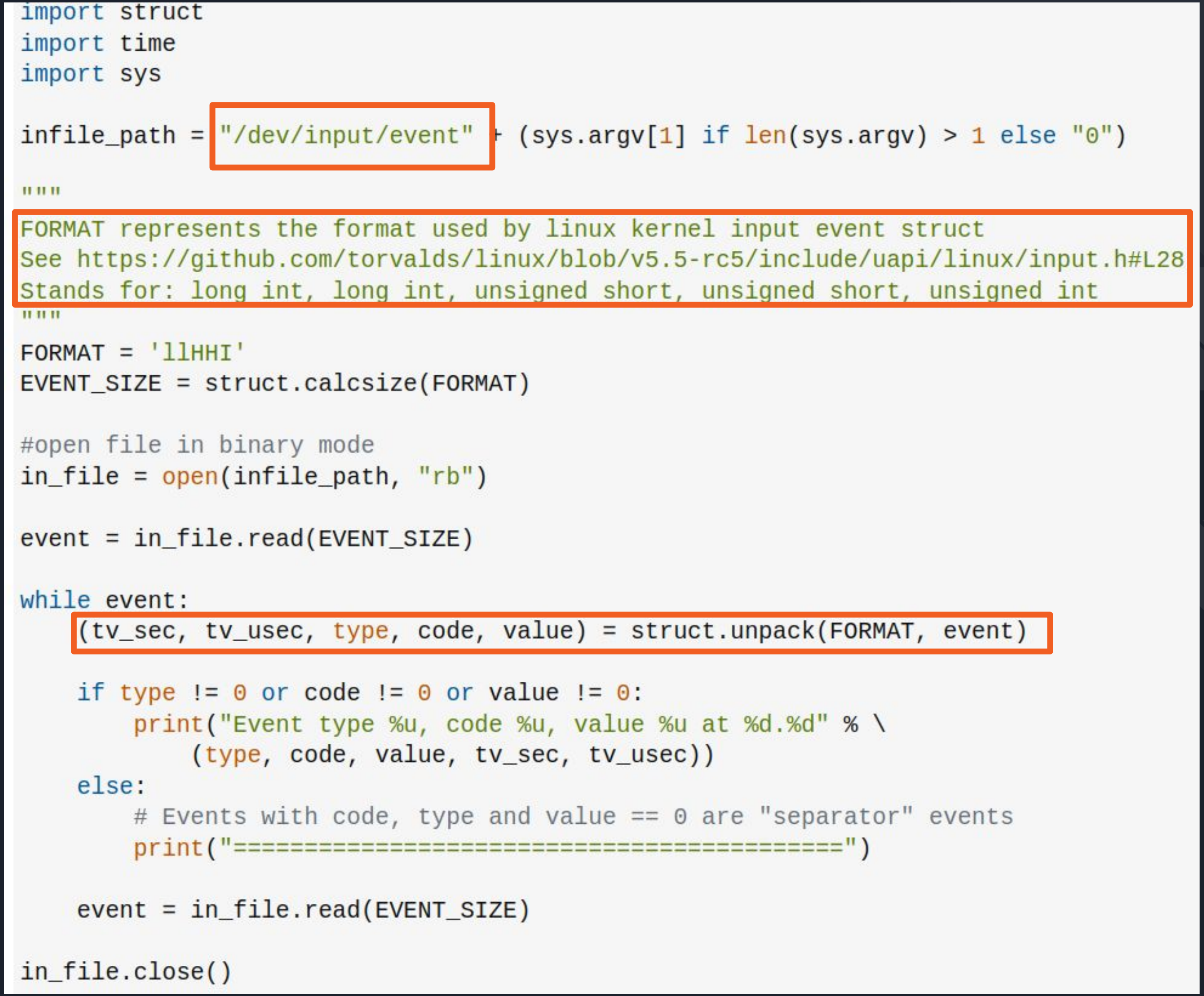
What is the "format" of the character devices located in /dev/**input**/eve
decode the character stream? A Python ...

5 réponses · Meilleure réponse: A simple and raw reader can be just done using: #!/usr/bin/python imp...

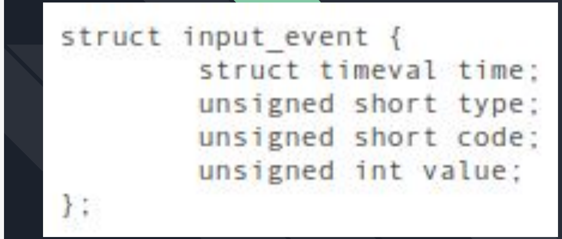
Read from /dev/**input** - Stack Overflow	4 réponses	11 avr. 2013
Processing **Linux input events** (/dev/**input**/**event***) in PHP	1 réponse	19 nov. 2023
Linux /dev/**input**/**event***: lost mouse movement **events**	1 réponse	22 avr. 2021
Accessing Keys from **Linux Input** Device - Stack ...	1 réponse	6 janv. 2014

Autres résultats pour stackoverflow.com





```python
import struct
import time
import sys

infile_path = "/dev/input/event" + (sys.argv[1] if len(sys.argv) > 1 else "0")

"""
FORMAT represents the format used by linux kernel input event struct
See https://github.com/torvalds/linux/blob/v5.5-rc5/include/uapi/linux/input.h#L28
Stands for: long int, long int, unsigned short, unsigned short, unsigned int
"""
FORMAT = 'llHHI'
EVENT_SIZE = struct.calcsize(FORMAT)

#open file in binary mode
in_file = open(infile_path, "rb")

event = in_file.read(EVENT_SIZE)

while event:
    (tv_sec, tv_usec, type, code, value) = struct.unpack(FORMAT, event)

    if type != 0 or code != 0 or value != 0:
        print("Event type %u, code %u, value %u at %d.%d" % \
            (type, code, value, tv_sec, tv_usec))
    else:
        # Events with code, type and value == 0 are "separator" events
        print("===========================================")

    event = in_file.read(EVENT_SIZE)

in_file.close()
```



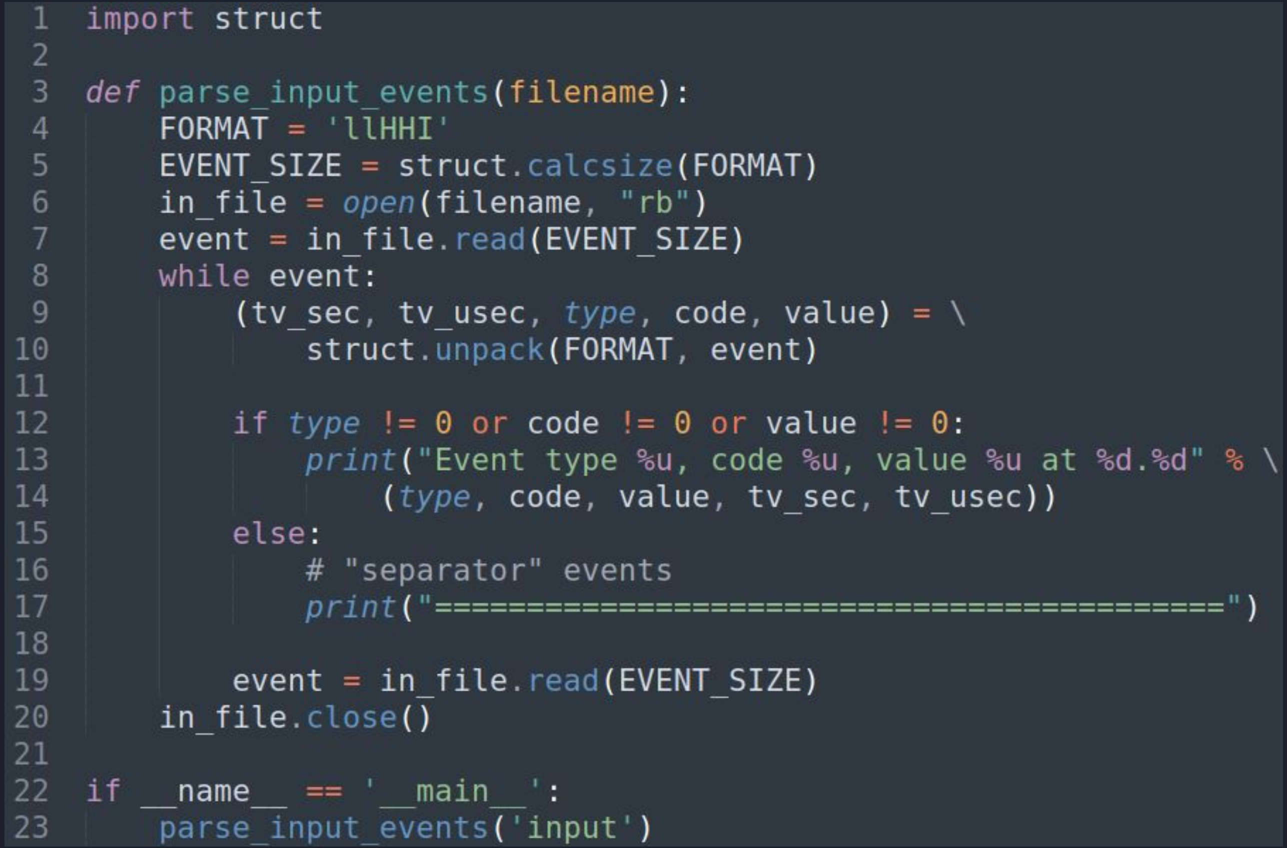
```c
struct input_event {
        struct timeval time;
        unsigned short type;
        unsigned short code;
        unsigned int value;
};
```

script3_3_v1.py

Parcours par bloc de 24 octets →

Découpage du bloc selon la → structure de *input_event*

Affichage de l'événement analysé →



```python
import struct

def parse_input_events(filename):
    FORMAT = 'llHHI'
    EVENT_SIZE = struct.calcsize(FORMAT)
    in_file = open(filename, "rb")
    event = in_file.read(EVENT_SIZE)
    while event:
        (tv_sec, tv_usec, type, code, value) = \
            struct.unpack(FORMAT, event)

        if type != 0 or code != 0 or value != 0:
            print("Event type %u, code %u, value %u at %d.%d" % \
                (type, code, value, tv_sec, tv_usec))
        else:
            # "separator" events
            print("===========================================")

        event = in_file.read(EVENT_SIZE)
    in_file.close()

if __name__ == '__main__':
    parse_input_events('input')
```

script3_3_v1.py

```
python script3_3_v1.py
```

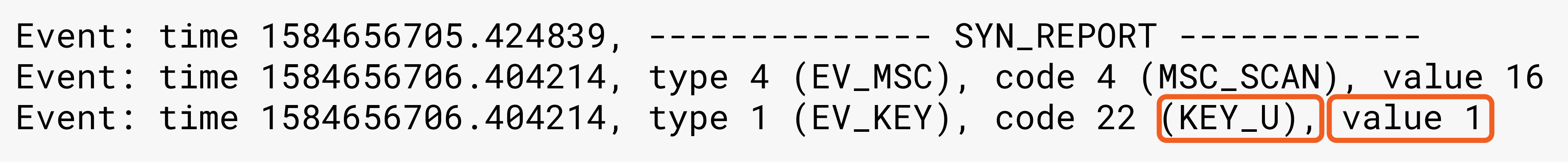
```
Event type 4, code 4, value 28 at 1584800351.730118
Event type 1, code 28, value 0 at 1584800351.730118
===========================================
Event type 4, code 4, value 23 at 1584800358.979743
Event type 1, code 23, value 1 at 1584800358.979743
===========================================
Event type 4, code 4, value 32 at 1584800359.35152
Event type 1, code 32, value 1 at 1584800359.35152
===========================================
Event type 4, code 4, value 23 at 1584800359.49404
Event type 1, code 23, value 0 at 1584800359.49404
===========================================
Event type 4, code 4, value 32 at 1584800359.132535
Event type 1, code 32, value 0 at 1584800359.132535
===========================================
Event type 4, code 4, value 28 at 1584800359.905387
Event type 1, code 28, value 1 at 1584800359.905387
===========================================
...
```

petite_frappe_1.txt

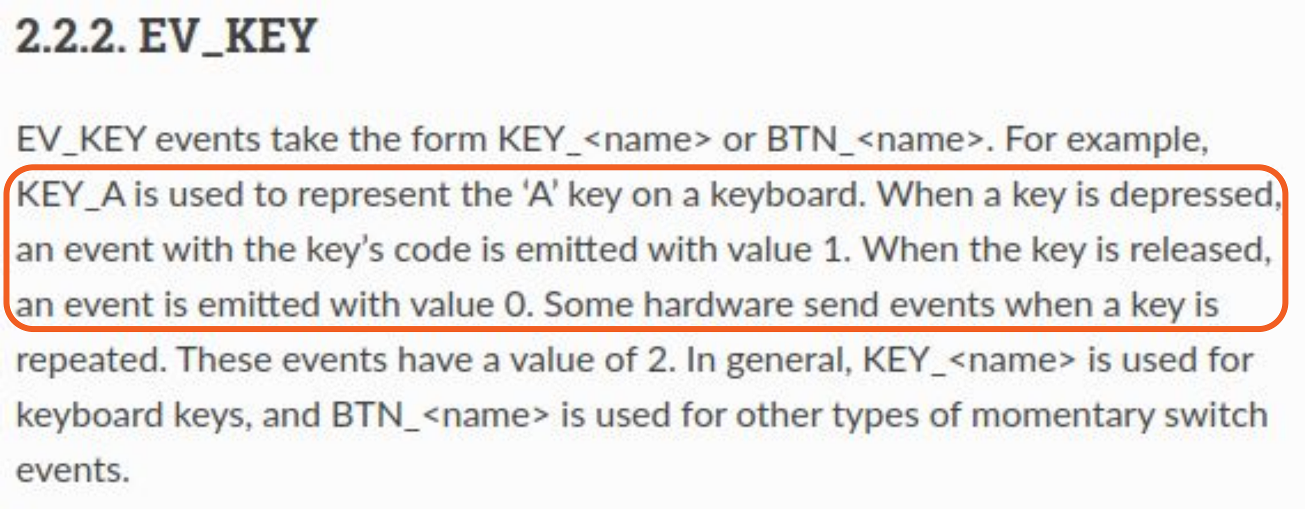
```
cat petite_frappe_1.txt
```

```
Event: time 1584656705.424839, -------------- SYN_REPORT ------------
Event: time 1584656706.404214, type 4 (EV_MSC), code 4 (MSC_SCAN), value 16
Event: time 1584656706.404214, type 1 (EV_KEY), code 22 (KEY_U), value 1
Event: time 1584656706.404214, -------------- SYN_REPORT ------------
Event: time 1584656706.508350, type 4 (EV_MSC), code 4 (MSC_SCAN), value 16
Event: time 1584656706.508350, type 1 (EV_KEY), code 22 (KEY_U), value 0
Event: time 1584656706.508350, -------------- SYN_REPORT ------------
Event: time 1584656706.674591, type 4 (EV_MSC), code 4 (MSC_SCAN), value 31
Event: time 1584656706.674591, type 1 (EV_KEY), code 49 (KEY_N), value 1
Event: time 1584656706.674591, -------------- SYN_REPORT ------------
Event: time 1584656706.774463, type 4 (EV_MSC), code 4 (MSC_SCAN), value 31
Event: time 1584656706.774463, type 1 (EV_KEY), code 49 (KEY_N), value 0
Event: time 1584656706.774463, -------------- SYN_REPORT ------------
Event: time 1584656706.926206, type 4 (EV_MSC), code 4 (MSC_SCAN), value 12
Event: time 1584656706.926206, type 1 (EV_KEY), code 18 (KEY_E), value 1
Event: time 1584656706.926206, -------------- SYN_REPORT ------------
Event: time 1584656707.023728, type 4 (EV_MSC), code 4 (MSC_SCAN), value 12
Event: time 1584656707.023728, type 1 (EV_KEY), code 18 (KEY_E), value 0
...
```

Doc Linux: Input event codes - EV_KEY

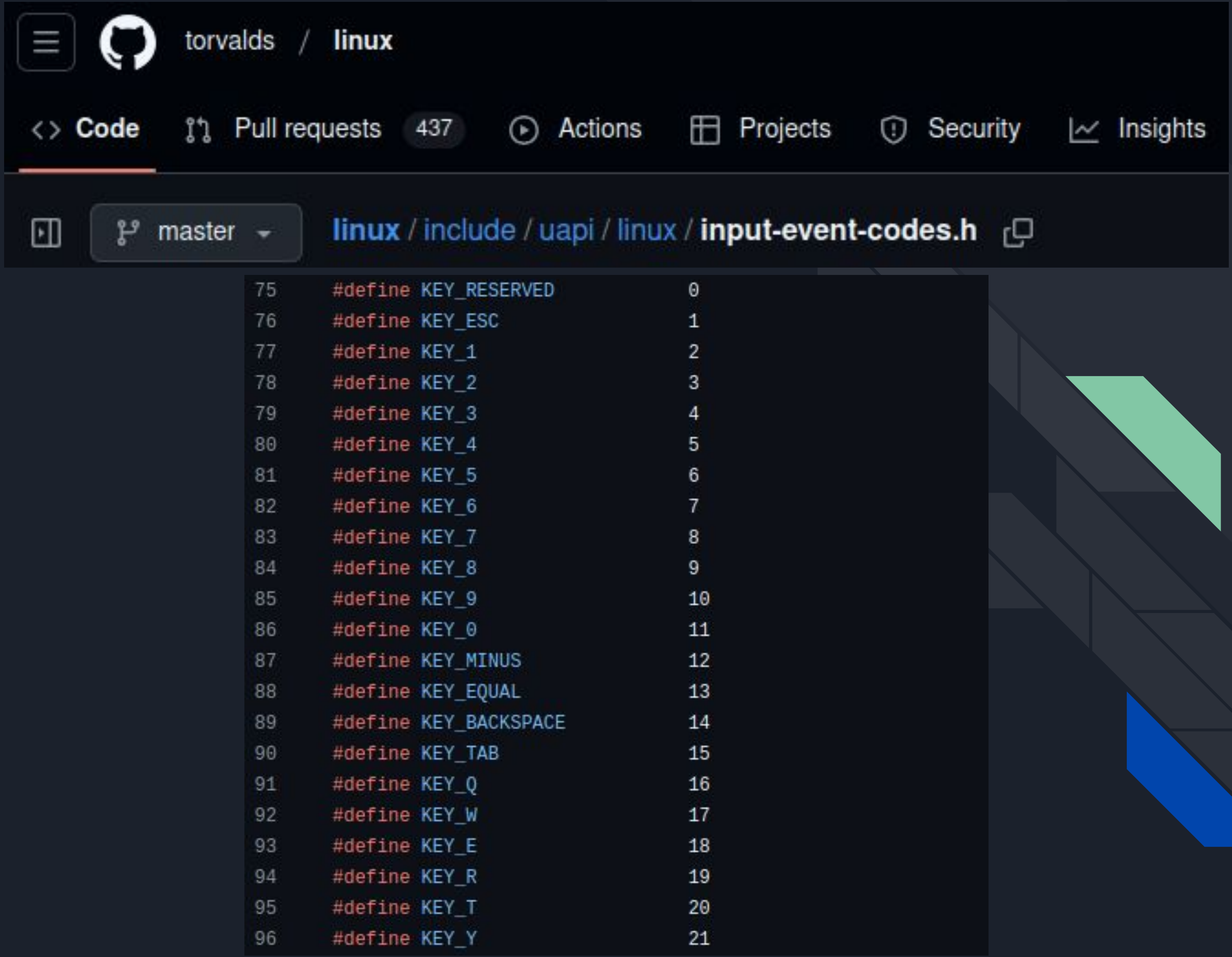


```
Event: time 1584656705.424839, -------------- SYN_REPORT ------------
Event: time 1584656706.404214, type 4 (EV_MSC), code 4 (MSC_SCAN), value 16
Event: time 1584656706.404214, type 1 (EV_KEY), code 22 (KEY_U), value 1
```



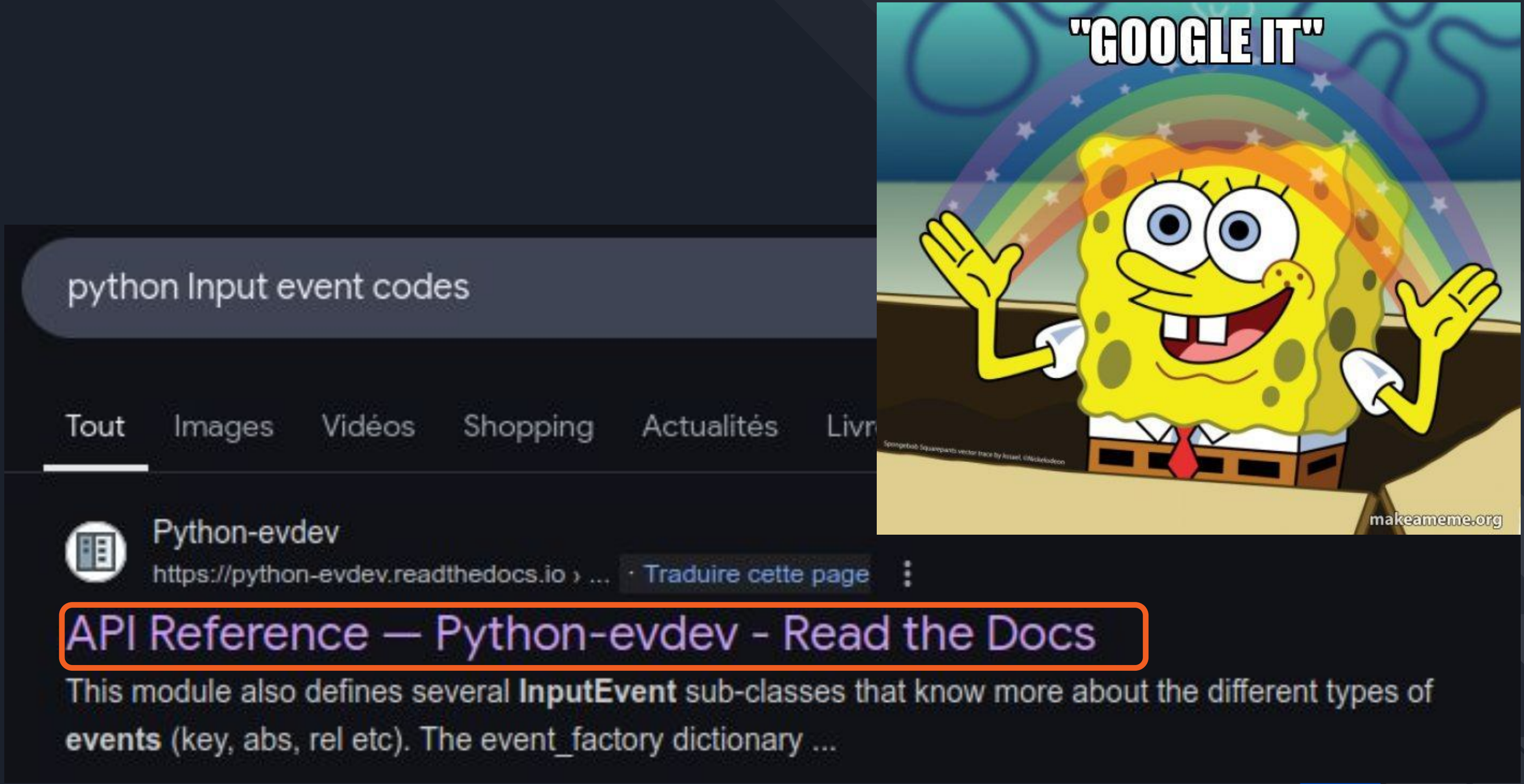
2.2.2. EV_KEY

EV_KEY events take the form KEY_<name> or BTN_<name>. For example, KEY_A is used to represent the 'A' key on a keyboard. When a key is depressed, an event with the key's code is emitted with value 1. When the key is released, an event is emitted with value 0. Some hardware send events when a key is repeated. These events have a value of 2. In general, KEY_<name> is used for keyboard keys, and BTN_<name> is used for other types of momentary switch events.



linux / include / uapi / linux / input-event-codes.h

```
#define KEY_RESERVED            0
#define KEY_ESC                 1
#define KEY_1                   2
#define KEY_2                   3
#define KEY_3                   4
#define KEY_4                   5
#define KEY_5                   6
#define KEY_6                   7
#define KEY_7                   8
#define KEY_8                   9
#define KEY_9                   10
#define KEY_0                   11
#define KEY_MINUS               12
#define KEY_EQUAL               13
#define KEY_BACKSPACE           14
#define KEY_TAB                 15
#define KEY_Q                   16
#define KEY_W                   17
#define KEY_E                   18
#define KEY_R                   19
#define KEY_T                   20
#define KEY_Y                   21
```



python Input event codes

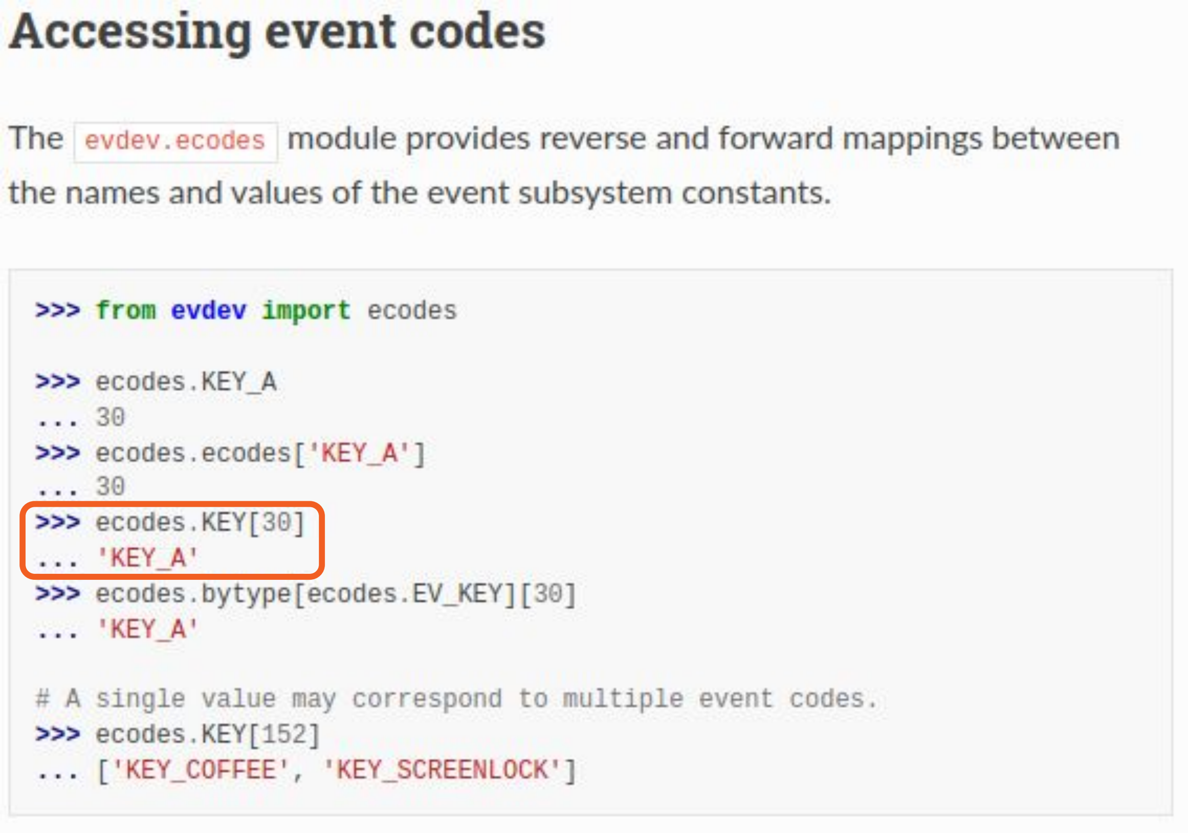
Python-evdev

https://python-evdev.readthedocs.io › ... · Traduire cette page

API Reference — Python-evdev - Read the Docs

This module also defines several **InputEvent** sub-classes that know more about the different types of **events** (key, abs, rel etc). The event_factory dictionary ...

Doc python-evdev

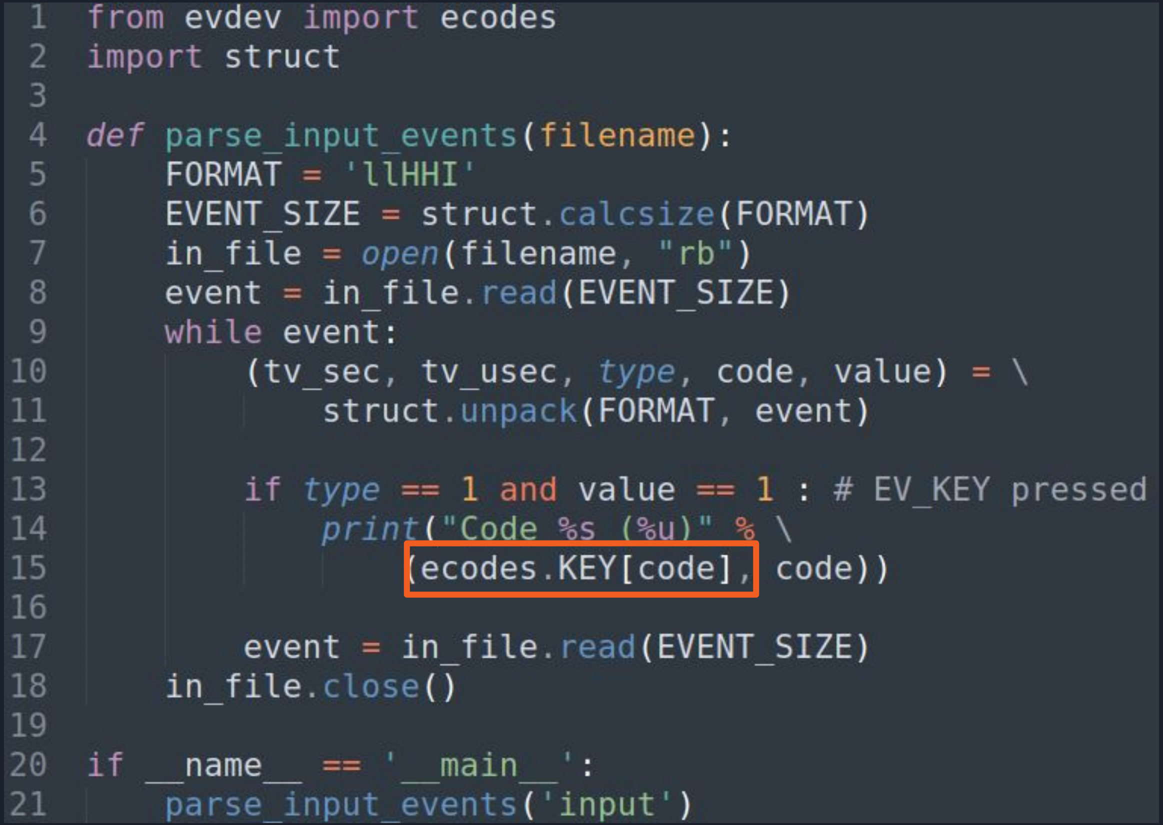


Accessing event codes

The `evdev.ecodes` module provides reverse and forward mappings between the names and values of the event subsystem constants.

```
>>> from evdev import ecodes

>>> ecodes.KEY_A
... 30
>>> ecodes.ecodes['KEY_A']
... 30
>>> ecodes.KEY[30]
... 'KEY_A'
>>> ecodes.bytype[ecodes.EV_KEY][30]
... 'KEY_A'

# A single value may correspond to multiple event codes.
>>> ecodes.KEY[152]
... ['KEY_COFFEE', 'KEY_SCREENLOCK']
```

script3_3_v2.py



```python
from evdev import ecodes
import struct

def parse_input_events(filename):
    FORMAT = 'llHHI'
    EVENT_SIZE = struct.calcsize(FORMAT)
    in_file = open(filename, "rb")
    event = in_file.read(EVENT_SIZE)
    while event:
        (tv_sec, tv_usec, type, code, value) = \
            struct.unpack(FORMAT, event)

        if type == 1 and value == 1 : # EV_KEY pressed
            print("Code %s (%u)" % \
                (ecodes.KEY[code], code))

        event = in_file.read(EVENT_SIZE)
    in_file.close()

if __name__ == '__main__':
    parse_input_events('input')
```

script3_3_v2.py

```
python script3_3_v2.py
```

```
Code KEY_I (23)
Code KEY_D (32)
Code KEY_ENTER (28)
Code KEY_G (34)
Code KEY_R (19)
Code KEY_O (24)
Code KEY_U (22)
Code KEY_P (25)
Code KEY_S (31)
Code KEY_ENTER (28)
Code KEY_P (25)
Code KEY_Z (44)
Code KEY_D (32)
Code KEY_ENTER (28)
Code KEY_Z (44)
Code KEY_ENTER (28)
Code KEY_P (25)
Code KEY_I (23)
Code KEY_N (49)
```

id

groups

???

???

ping

```
Code KEY_G (34)
Code KEY_SPACE (57)
Code KEY_RIGHTSHIFT (54)
Code KEY_2 (3)
Code KEY_9 (10)
Code KEY_8 (9)
Code KEY_COMMA (51)
Code KEY_1 (2)
Code KEY_2 (3)
Code KEY_5 (6)
Code KEY_COMMA (51)
Code KEY_2 (3)
Code KEY_5 (6)
Code KEY_0 (11)
Code KEY_COMMA (51)
Code KEY_4 (5)
Code KEY_2 (3)
Code KEY_ENTER (28)
Code KEY_S (31)
```

??

```
Code KEY_U (22)
Code KEY_D (32)
Code KEY_O (24)
Code KEY_SPACE (57)
Code KEY_Q (16)
Code KEY_P (25)
Code KEY_T (20)
Code KEY_SPACE (57)
Code KEY_U (22)
Code KEY_P (25)
Code KEY_D (32)
Code KEY_Q (16)
Code KEY_T (20)
Code KEY_E (18)
Code KEY_ENTER (28)
Code KEY_Q (16)
Code KEY_P (25)
Code KEY_T (20)
...
```

apt update ?

script3_3_v1.py

```
python script
```

id

groups

???

???

ping

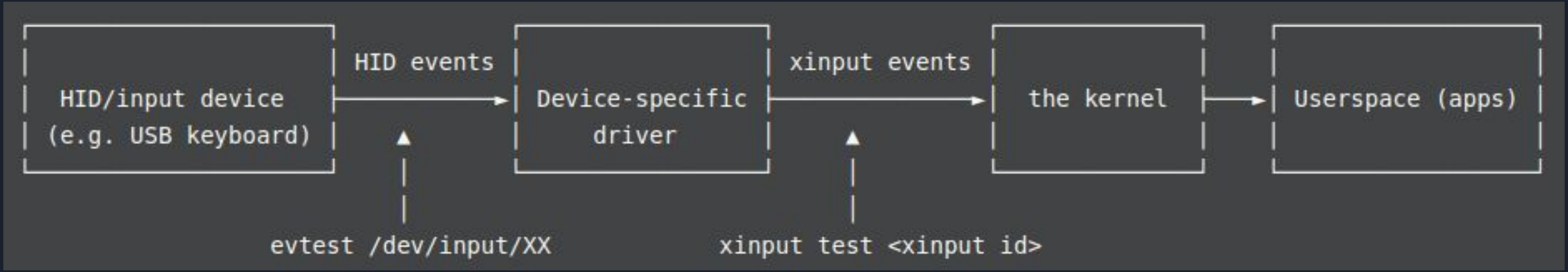
```
Code KEY_I (23)
Code KEY_D (32)
Code KEY_ENTER (2
Code KEY_G (34)
Code KEY_R (19)
Code KEY_O (24)
Code KEY_U (22)
Code KEY_P (25)
Code KEY_S (31)
Code KEY_ENTER (2
Code KEY_P (25)
Code KEY_Z (44)
Code KEY_D (32)
Code KEY_ENTER (2
Code KEY_Z (44)
Code KEY_ENTER (2
Code KEY_P (25)
Code KEY_I (23)
Code KEY_N (49)
```

```
Code KEY_2 (3)
Code KEY_ENTER (28)
Code KEY_S (31)
```

```
U (22)
D (32)
O (24)
SPACE (57)
Q (16)
P (25)
T (20)
SPACE (57)
U (22)
P (25)
D (32)
Q (16)
T (20)
E (18)
ENTER (28)
Q (16)
Code KEY_P (25)
Code KEY_T (20)
...
```

apt update ?

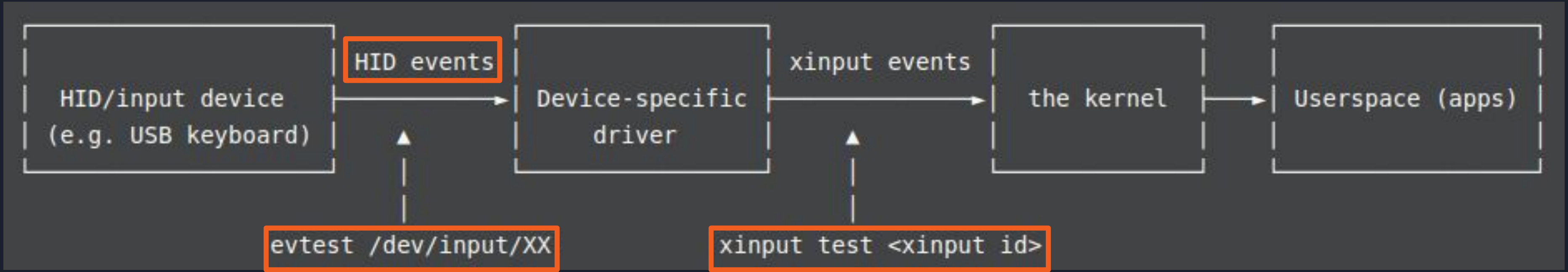
Fonctionnement entrées Linux



```
┌─────────────────────┐            ┌─────────────────┐               ┌──────────────┐    ┌──────────────────┐
│                     │ HID events │                 │ xinput events │              │    │                  │
│  HID/input device   ├──────────►│ Device-specific ├─────────────►│  the kernel  ├───►│ Userspace (apps) │
│ (e.g. USB keyboard) │            │     driver      │               │              │    │                  │
└─────────────────────┘            └─────────────────┘               └──────────────┘    └──────────────────┘
                           ▲                               ▲
                           │                               │
                           │                               │
                  evtest /dev/input/XX            xinput test <xinput id>
```

Fonctionnement entrées Linux

Petite frappe 3/3

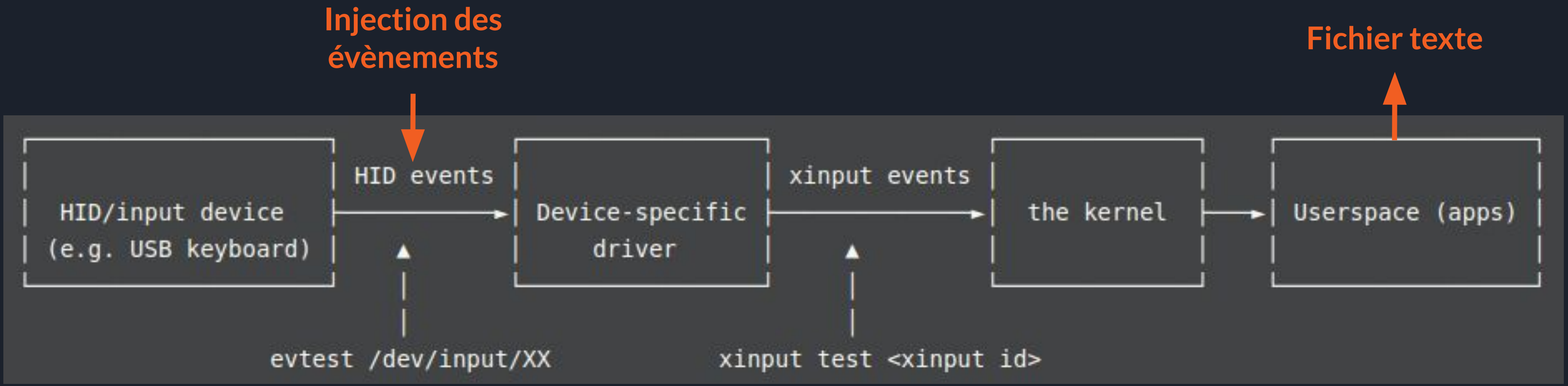


```
┌────────────────────┐            ┌───────────────┐               ┌────────────┐     ┌──────────────────┐
│                    │ HID events │               │ xinput events │            │     │                  │
│  HID/input device  ├───────────►│Device-specific├──────────────►│ the kernel ├────►│ Userspace (apps) │
│ (e.g. USB keyboard)│     ▲      │    driver     │       ▲       │            │     │                  │
└────────────────────┘     │      └───────────────┘       │       └────────────┘     └──────────────────┘
                           │                              │
                  evtest /dev/input/XX           xinput test <xinput id>
```

Petite frappe 1/3

Petite frappe 2/3

Rejeu des entrées

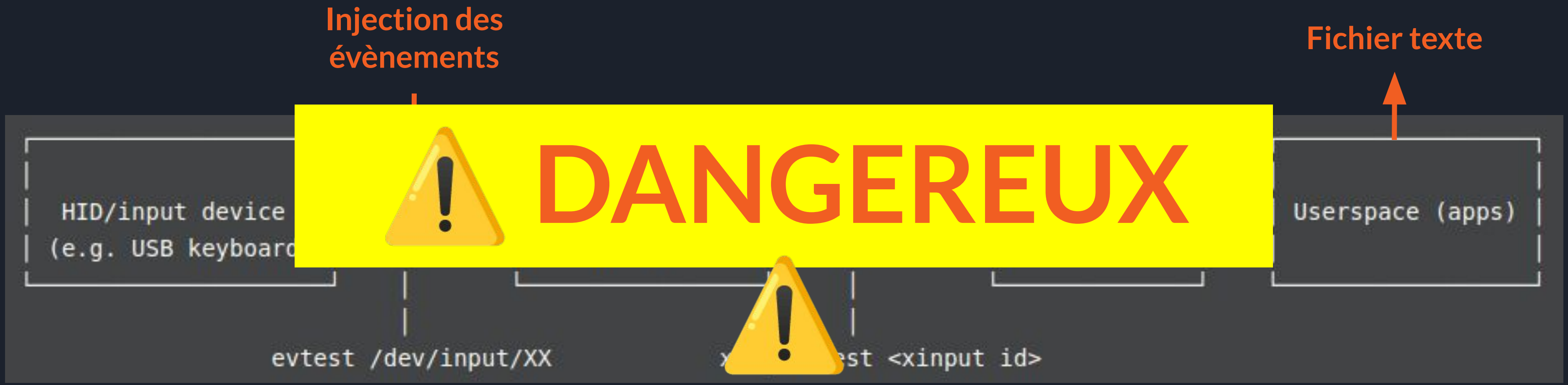


Injection des évènements

Fichier texte

```
┌──────────────────────┐              ┌──────────────────┐                ┌───────────────┐     ┌────────────────────┐
│                      │  HID events  │                  │ xinput events  │               │     │                    │
│  HID/input device    ├─────────────►│ Device-specific  ├───────────────►│  the kernel   ├────►│ Userspace (apps)   │
│ (e.g. USB keyboard)  │              │     driver       │                │               │     │                    │
└──────────────────────┘      ▲       └──────────────────┘        ▲       └───────────────┘     └────────────────────┘
                              │                                   │
                              │                                   │
                   evtest /dev/input/XX                 xinput test <xinput id>
```

Rejeu des entrées



Injection des évènements

Fichier texte

⚠ DANGEREUX

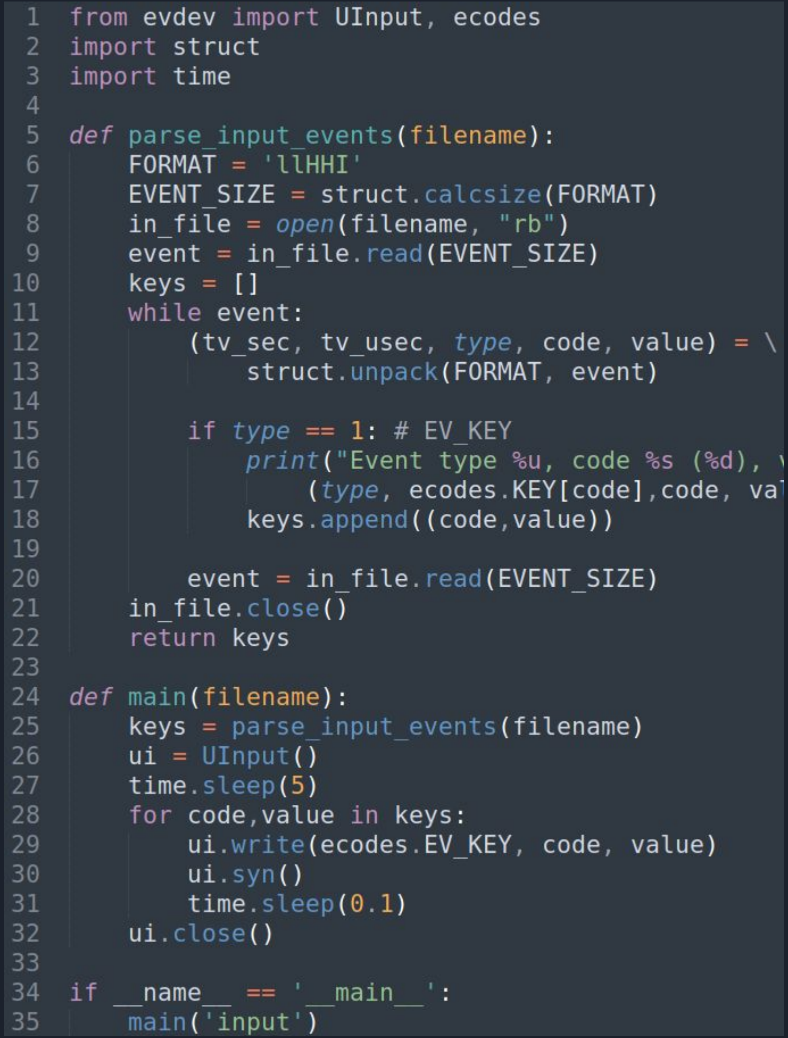
```
| HID/input device
| (e.g. USB keyboarc
                                                              | Userspace (apps) |

          evtest /dev/input/XX            x⚠est <xinput id>
```

script3_3.py

Ajout de chaque évènement de type EV_KEY (code, value) →

Attente pour nous laisser le temps d’aller dans un fichier texte →

Rejeu de chaque évènement →



```python
from evdev import UInput, ecodes
import struct
import time

def parse_input_events(filename):
    FORMAT = 'llHHI'
    EVENT_SIZE = struct.calcsize(FORMAT)
    in_file = open(filename, "rb")
    event = in_file.read(EVENT_SIZE)
    keys = []
    while event:
        (tv_sec, tv_usec, type, code, value) = \
            struct.unpack(FORMAT, event)

        if type == 1: # EV_KEY
            print("Event type %u, code %s (%d), 
                (type, ecodes.KEY[code],code, va
            keys.append((code,value))

        event = in_file.read(EVENT_SIZE)
    in_file.close()
    return keys

def main(filename):
    keys = parse_input_events(filename)
    ui = UInput()
    time.sleep(5)
    for code,value in keys:
        ui.write(ecodes.EV_KEY, code, value)
        ui.syn()
        time.sleep(0.1)
    ui.close()

if __name__ == '__main__':
    main('input')
```

script3_3.py

```
python script3_3.py
```

```
...
apt list --upgradable
man asc    i
  qcd
ls
cd Docu    Challs
git status
git rm flag:wq
git add README.md
git commit -m "Remove flag.txt"
git push
cd ..
ls
df -h
gpg --output flag --decrypt flag.gpg
Destination_Autriche_#TEAMFR
cat flag
exit
```

Déchiffrement GPG

```
gpg --output flag --decrypt flag.gpg # Destination_Autriche_#TEAMFR
cat flag
```

```
FCSC{0bec21052ae86baf149eb97ce52de0fec1d6b8e1ed827fe026f6197be07419c3}
```

Questions