

README.md



ver 2.1.4 PyPI Download python 3.6 license GPLv3 BlackArch Linux built with ❤️❤️

**usbrrip** (inherited from "USB Ripper", not "USB R.I.P.") is an open source forensics tool with CLI interface that lets you keep track of USB device artifacts (aka USB event history, "Connected" and "Disconnected" events) on Linux machines.

Table of Contents:

- [Description](#)
- [Quick Start](#)
- [Screenshots](#)
- [Git Clone](#)
- [Dependencies](#)
  - [System Log Structure](#)
  - [DEB Packages](#)
  - [PIP Packages](#)
  - [Portable](#)
- [Installation](#)
  - [pip or setup.py](#)
  - [install.sh](#)
    - [Paths](#)
    - [cron](#)
    - [uninstall.sh](#)
- [Usage](#)
  - [Synopsis](#)
  - [Help](#)
- [Examples](#)
- [Credits & References](#)
- [Post Scriptum](#)

## Description

**usbrrip** is a small piece of software written in pure Python 3 (using some external modules, though, see [Dependencies/PIP](#)) which parses Linux log files (`/var/log/syslog*`, `/var/log/messages*` or `journalctl` output, depending on the distro) for constructing USB event history tables. Such tables may contain the following columns: "Connected" (date & time), "User", "VID" (vendor ID), "PID" (product ID), "Product", "Manufacturer", "Serial Number", "Port" and "Disconnected" (date & time).

Besides, it also can:

- export gathered information as a JSON dump (and open such dumps, of course);
- generate a list of authorized (*trusted*) USB devices as a JSON (call it `auth.json`);

- search for "violation events" based on the `auth.json` : show (or generate another JSON with) USB devices that do appear in history and do NOT appear in the `auth.json` ;
- *\*when installed with -s flag\** create crypted storages (7zip archives) to automatically backup and accumulate USB events with the help of `crontab` scheduler;
- search additional details about a specific USB device based on its VID and/or PID.

## Quick Start

usbrrip is available for download and installation at [PyPI](#) (the latest version is always on GitHub, though):

```
-$ pip3 install usbrrip
```

## Screenshots

```
+ ~/~/python/usbrrip git:(master) x usbrrip events history -ed '2019-08' --user 'kali' --serial 'AA011224150718230115' 'AAJ6LE5SN1B8PC1A' --pid '6545'
```

Connected	User	VID	PID	Product	Manufacturer	Serial Number	Port	Disconnected
2019-08-09 *****	---	---	---	-----	-----	-----	---	-----
2019-08-09 06:50:50	kali	0930	6545	DataTraveler 2.0	Kingston	C86000BDB8B9EE619A2D0204	1-1	2019-08-09 06:51:07
2019-08-09 11:04:03	kali	0781	5580	Extreme	SanDisk	AA011224150718230115	1-1	2019-08-09 11:04:25
2019-08-11 *****	---	---	---	-----	-----	-----	---	-----
2019-08-11 00:14:17	kali	0dd8	3200	OnlyDisk	Netac	AAJ6LE5SN1B8PC1A	1-2	2019-08-11 00:15:28

```
[*] Shutted down at 2019-08-12 02:24:23
[*] Time taken: 0:00:01.116194
```

```
→ ~/~/python/usbrrip git:(master) x usbrrip ids search --vid '0dd8'
```

```
[*] Started at 2019-08-12 02:24:26
[02:24:26] [INFO] Getting current database version
Version: 2019.07.27
Date: 2019-07-27 20:34:05
[02:24:26] [INFO] Checking local database for update
[02:24:26] [INFO] Getting latest version and date
[02:24:29] [INFO] Local database is up-to-date
Searching for matches... Done

| Possible products:
|_ Netac Technology Co., Ltd

[*] Shutted down at 2019-08-12 02:24:29
[*] Time taken: 0:00:02.472049
```

## Git Clone

For simplicity, let's agree that all the commands where `~/usbrip$` prefix is appeared are executed in the `~/usbrip` directory which is created as a result of git clone:

```
-$ git clone https://github.com/snovvcrash/usbrip.git usbrip && cd usbrip
~/usbrip$
```

## Dependencies

---

### System Log Structure

---

usbrip supports two types of format:

1. **Non-modified** — standard `syslog` structure for GNU/Linux ( `"%b %d %H:%M:%S"` , ex. "Mar 18 13:56:07"). This type of timestamp does not provide the information about years.
2. **Modified** (recommended) — upgraded structure of system log files which provides high precision timestamps ( `"%Y-%m-%dT%H:%M:%S.%f%z"` , ex. "2019-08-09T06:15:49.655261-04:00" ).

If you use `journalctl` to manage your logs, then there's nothing to worry about (as it can convert timestamps on the fly).

Otherwise, the desired structure could be achieved by setting `RSYSLOG_FileFormat` format if you are using `rsyslog`, for example.

1. Comment out the following line in `/etc/rsyslog.conf` :

```
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
```

2. Add custom `.conf` file for usbrip:

```
-$ echo '$ActionFileDefaultTemplate RSYSLOG_FileFormat' | sudo tee /etc/rsyslog.d/usbrip.conf
```

3. *(optional)* Delete existing log files:

```
-$ sudo rm -f /var/log/syslog* /var/log/messages*
```

4. Restart the service:

```
-$ sudo systemctl restart rsyslog
```

### DEB Packages

---

- python3.6 (or newer) interpreter
- python3-venv
- p7zip-full (used by `storages` module)

```
-$ sudo apt install python3-venv p7zip-full -y
```

### PIP Packages

---

usbrip makes use of the following external modules:

- [terminaltables](#)
- [termcolor](#)
- [tqdm](#)

### Portable

---

To resolve Python dependencies manually (it's not necessary actually because `pip` or `setup.py` can automate the process, see [Installation](#)) create a *virtual environment* (optional) and run `pip` from within:

```
~/usbrrip$ python3 -m venv venv && source venv/bin/activate
(venv) ~/usbrrip$ pip install -r requirements.txt
```

Or let the `pipenv` one-liner do all the dirty work for you:

```
~/usbrrip$ pipenv install && pipenv shell
```

After that you can run `usbrrip` portably:

```
(venv) ~/usbrrip$ python -m usbrrip -h
or
(venv) ~/usbrrip$ python __main__.py -h
```

## Installation

---

There are two ways to install `usbrrip` into the system: `pip` or `setup.py`.

### pip or setup.py

---

First of all, `usbrrip` is pip installable. This means that after git cloning the repo you can simply fire up the pip installation process and after that run `usbrrip` from anywhere in your terminal like so:


```
~/usbrrip$ python3 -m venv venv && source venv/bin/activate
(venv) ~/usbrrip$ pip install .

(venv) ~/usbrrip$ usbrrip -h
```

Or if you want to resolve Python dependencies locally (without bothering PyPI), use `setup.py`:

```
~/usbrrip$ python3 -m venv venv && source venv/bin/activate
(venv) ~/usbrrip$ python setup.py install

(venv) ~/usbrrip$ usbrrip -h
```

 **Note:** you'd likely want to run the installation process while the Python virtual environment is active (like it is shown above).


### install.sh

---

Secondly, `usbrrip` can also be installed into the system with the `./installers/install.sh` script.

When using the `./installers/install.sh` some extra features become available:

- the virtual environment is created automatically;
- the `storage` module becomes available: you can set a crontab job to backup USB events on a schedule (the example of crontab jobs can be found in `usbrrip/cron/usbrrip.cron`).

 **Warning:** if you are using the crontab scheduling, you want to configure the cron job with `sudo crontab -e` in order to force the `storage update` submodule run as root as well as protect the passwords of the USB event storages. The storage passwords are kept in `/var/opt/usbrrip/usbrrip.ini`.

The `./installers/uninstall.sh` script removes all the installation artifacts from your system.


To install `usbrrip` use:

```
~/usbrrip$ chmod +x ./installers/install.sh
```

```
~/usbrrip$ sudo -H ./installers/install.sh [-l/--local] [-s/--storages]
~/usbrrip$ cd

~$ usbrrip -h
```

- When `-l` switch is enabled, Python dependencies are resolved from local `.tar` packages ( `./3rdPartyTools/` ) instead of PyPI.
- When `-s` switch is enabled, not only the usbrrip project is installed, but also the list of trusted USB devices, history and violations storages are created.

 **Note:** when using `-s` option during installation, make sure that system logs do contain at least one *external* USB device entry. It is a necessary condition for usbrrip to successfully create the list of trusted devices (and as a result, successfully create the violations storage).

After the installation completes, feel free to remove the usbrrip folder.

## Paths

When installed, the usbrrip uses the following paths:

- `/opt/usbrrip/` — project's main directory;
- `/var/opt/usbrrip/usbrrip.ini` — usbrrip configuration file: keeps passwords for 7zip storages;
- `/var/opt/usbrrip/storage/` — USB event storages: `history.7z` and `violations.7z` (created during the installation process);
- `/var/opt/usbrrip/log/` — usbrrip logs (recommended to log usbrrip activity when using crontab, see `usbrrip/cron/usbrrip.cron` );
- `/var/opt/usbrrip/trusted/` — list of trusted USB devices (created during the installation process);
- `/usr/local/bin/usbrrip` — symlink to the `/opt/usbrrip/venv/bin/usbrrip` script.

## cron

Cron jobs can be set as follows:

```
~/usbrrip$ sudo crontab -l > tmpcron && echo "" >> tmpcron
~/usbrrip$ cat usbrrip/cron/usbrrip.cron | tee -a tmpcron
~/usbrrip$ sudo crontab tmpcron
~/usbrrip$ rm tmpcron
```

## uninstall.sh

To uninstall usbrrip use:

```
~/usbrrip$ chmod +x ./installers/uninstall.sh
~/usbrrip$ sudo ./installers/uninstall.sh [-a/--all]
```

- When `-a` switch is enabled, not only the usbrrip project directory is deleted, but also all the storages and usbrrip logs are deleted too.

And don't forget to remove the cron job.

# Usage

## Synopsis

```
# ----- BANNER -----

$ usbrrip banner
Get usbrrip banner.

# ----- EVENTS -----

$ usbrrip events history [-t | -l] [-e] [-n <NUMBER_OF_EVENTS>] [-d <DATE> [<DATE> ...]] [--user <USER> [<USER> ...]]
[--vid <VID> [<VID> ...]] [--pid <PID> [<PID> ...]] [--prod <PROD> [<PROD> ...]] [--manufact <MANUFACT> [<MANUFACT>
...]] [--serial <SERIAL> [<SERIAL> ...]] [--port <PORT> [<PORT> ...]] [-c <COLUMN> [<COLUMN> ...]] [-f <FILE>
<FILE> ...]] [-q] [--debug]
```

Get USB event history.

```
$ usbrp events open <DUMP.JSON> [-t | -l] [-e] [-n <NUMBER_OF_EVENTS>] [-d <DATE> [<DATE> ...]] [--user <USER>
[<USER> ...]] [--vid <VID> [<VID> ...]] [--pid <PID> [<PID> ...]] [--prod <PROD> [<PROD> ...]] [--manufact
<MANUFACT> [<MANUFACT> ...]] [--serial <SERIAL> [<SERIAL> ...]] [--port <PORT> [<PORT> ...]] [-c <COLUMN> [<COLUMN>
...]] [-f <FILE> [<FILE> ...]] [-q] [--debug]
Open USB event dump.
```

```
$ usbrp events gen_auth <OUT_AUTH.JSON> [-a <ATTRIBUTE> [<ATTRIBUTE> ...]] [-e] [-n <NUMBER_OF_EVENTS>] [-d <DATE>
[<DATE> ...]] [--user <USER> [<USER> ...]] [--vid <VID> [<VID> ...]] [--pid <PID> [<PID> ...]] [--prod <PROD>
[<PROD> ...]] [--manufact <MANUFACT> [<MANUFACT> ...]] [--serial <SERIAL> [<SERIAL> ...]] [--port <PORT> [<PORT>
...]] [-f <FILE> [<FILE> ...]] [-q] [--debug]
Generate a list of trusted (authorized) USB devices.
```

```
$ usbrp events violations <IN_AUTH.JSON> [-a <ATTRIBUTE> [<ATTRIBUTE> ...]] [-t | -l] [-e] [-n <NUMBER_OF_EVENTS>]
[-d <DATE> [<DATE> ...]] [--user <USER> [<USER> ...]] [--vid <VID> [<VID> ...]] [--pid <PID> [<PID> ...]] [--prod
<PROD> [<PROD> ...]] [--manufact <MANUFACT> [<MANUFACT> ...]] [--serial <SERIAL> [<SERIAL> ...]] [--port <PORT>
[<PORT> ...]] [-c <COLUMN> [<COLUMN> ...]] [-f <FILE> [<FILE> ...]] [-q] [--debug]
Get USB violation events based on the list of trusted devices.
```

# ----- STORAGE -----

```
$ usbrp storage list <STORAGE_TYPE> [-q] [--debug]
List contents of the selected storage (7zip archive). STORAGE_TYPE is "history" or "violations".
```

```
$ usbrp storage open <STORAGE_TYPE> [-t | -l] [-e] [-n <NUMBER_OF_EVENTS>] [-d <DATE> [<DATE> ...]] [--user <USER>
[<USER> ...]] [--vid <VID> [<VID> ...]] [--pid <PID> [<PID> ...]] [--prod <PROD> [<PROD> ...]] [--manufact
<MANUFACT> [<MANUFACT> ...]] [--serial <SERIAL> [<SERIAL> ...]] [--port <PORT> [<PORT> ...]] [-c <COLUMN> [<COLUMN>
...]] [-q] [--debug]
Open selected storage (7zip archive). Behaves similarly to the EVENTS OPEN submodule.
```

```
$ usbrp storage update <STORAGE_TYPE> [-a <ATTRIBUTE> [<ATTRIBUTE> ...]] [-e] [-n <NUMBER_OF_EVENTS>] [-d <DATE>
[<DATE> ...]] [--user <USER> [<USER> ...]] [--vid <VID> [<VID> ...]] [--pid <PID> [<PID> ...]] [--prod <PROD>
[<PROD> ...]] [--manufact <MANUFACT> [<MANUFACT> ...]] [--serial <SERIAL> [<SERIAL> ...]] [--port <PORT> [<PORT>
...]] [--lvl <COMPRESSION_LEVEL>] [-q] [--debug]
Update storage – add USB events to the existing storage (7zip archive). COMPRESSION_LEVEL is a number in [0..9].
```

```
$ usbrp storage create <STORAGE_TYPE> [-a <ATTRIBUTE> [<ATTRIBUTE> ...]] [-e] [-n <NUMBER_OF_EVENTS>] [-d <DATE>
[<DATE> ...]] [--user <USER> [<USER> ...]] [--vid <VID> [<VID> ...]] [--pid <PID> [<PID> ...]] [--prod <PROD>
[<PROD> ...]] [--manufact <MANUFACT> [<MANUFACT> ...]] [--serial <SERIAL> [<SERIAL> ...]] [--port <PORT> [<PORT>
...]] [--lvl <COMPRESSION_LEVEL>] [-q] [--debug]
Create storage – create 7zip archive and add USB events to it according to the selected options.
```

```
$ usbrp storage passwd <STORAGE_TYPE> [--lvl <COMPRESSION_LEVEL>] [-q] [--debug]
Change password of the existing storage.
```

# ----- IDs -----

```
$ usbrp ids search [--vid <VID>] [--pid <PID>] [--offline] [-q] [--debug]
Get extra details about a specific USB device by its <VID> and/or <PID> from the USB ID database.
```

```
$ usbrp ids download [-q] [--debug]
Update (download) the USB ID database.
```

## Help

---

To get a list of module names use:

```
-$ usbrp -h
```

To get a list of submodule names for a specific module use:

```
-$ usbrp <module> -h
```

To get a list of all switches for a specific submodule use:

```
-$ usbrp <module> <submodule> -h
```


## Examples

- Show the event history of all USB devices, suppressing banner output, info messages and user interaction ( `-q` , `--quiet` ), represented as a list ( `-l` , `--list` ) with latest 100 entries ( `-n NUMBER` , `--number NUMBER` ):

```
-$ usbrp events history -ql -n 100
```

- Show the event history of the external USB devices ( `-e` , `--external` , which were *actually* disconnected) represented as a table ( `-t` , `--table` ) containing "Connected", "VID", "PID", "Disconnected" and "Serial Number" columns ( `-c COLUMN [COLUMN ...]` , `--column COLUMN [COLUMN ...]` ) filtered by date ( `-d DATE [DATE ...]` , `--date DATE [DATE ...]` ) and PID ( `--pid <PID>` [ `<PID> ...` ] ) with logs taken from the outer files ( `-f FILE [FILE ...]` , `--file FILE [FILE ...]` ):

```
-$ usbrp events history -et -c conn vid pid disconn serial -d '1995-09-15' '2018-07-01' --pid 1337 -f /var/log/syslog.1 /var/log/syslog.2.gz
```

 **Note:** there is a thing to remember when working with filters. There are 4 types of filtering available: only *external* USB events (devices that can be pulled out easily, `-e`); *by date* ( `-d` ); *by fields* ( `--user` , `--vid` , `--pid` , `--product` , `--manufact` , `--serial` , `--port` ) and *by number of entries* you get as the output ( `-n` ). When applying different filters simultaneously, you will get the following behaviour: firstly, *external* and *by date* filters are applied, then usbrp will search for specified *field* values in the intersection of the last two filters, and in the end it will cut the output to the *number* you defined with the `-n` option. So think of it as an **intersection** for *external* and *by date* filtering and **union** for *by fields* filtering. Hope it makes sense.

- Build the event history of all USB devices and redirect the output to a file for further analysis. When the output stream is NOT terminal stdout ( `|` or `>` for example) there would be no ANSI escape characters (color) in the output so feel free to use it that way. Also notice that usbrp uses some UNICODE symbols so it would be nice to convert the resulting file to UTF-8 encoding (with `encov` for example) as well as change newline characters to Windows style for portability (with `awk` for example):


```
-$ usbrp history events -t | awk '{ sub("$", "\r"); print }' > usbrp.out && encov -x UTF8 usbrp.out
```

*Remark:* you can always get rid of the escape characters by yourself even if you have already got the output to stdout. To do that just copy the output data to `usbrp.out` and add one more `awk` instruction:

```
-$ awk '{ sub("$", "\r"); gsub("\x1B\\[[0-9]*[ -/]*[@-~]", ""); print }' usbrp.out && encov -x UTF8 usbrp.out
```

- Generate a list of trusted USB devices as a JSON-file ( `trusted/auth.json` ) with "VID" and "PID" attributes containing the first *three* devices connected on November 30, 1984:

```
-$ usbrp events gen_auth trusted/auth.json -a vid pid -n 3 -d '1984-11-30'
```

 **Warning:** there are cases when different USB flash drives might have identical serial numbers. This could happen as a result of a [manufacturing error](#) or just some black hats were able to rewrite the drive's memory chip which turned out to be non-one-time programmable and so on... Anyways, «*No system is safe*». usbrp **does not** handle such cases in a smart way so far, namely it will treat a pair of devices with identical SNs (if there exists one) as the same device regarding to the trusted device list and `gen_auth` module.

- Search the event history of the external USB devices for violations based on the list of trusted USB devices ( `trusted/auth.json` ) by "PID" attribute, restrict resulting events to those which have "Bob" as a user, "EvilUSBManufacturer" as a manufacturer, "1234567890" as a serial number and represent the output as a table with "Connected", "VID" and "PID" columns:

```
-$ usbrp events violations trusted/auth.json -a pid -et --user Bob --manufact EvilUSBManufacturer --serial 1234567890 -c conn vid pid
```

- Search for details about a specific USB device by its VID ( `--vid VID` ) and PID ( `--pid PID` ):

```
~$ usbrip ids search --vid 0781 --pid 5580
```

- Download the latest version of `usb_ids/usb.ids` [database](#):

```
~$ usbrip ids download
```

## Credits & References

---

- [usbrip / Инструменты Kali Linux](#)
- [Как узнать, какие USB устройства подключались к Linux / HackWare.ru](#)
- [Linux-форензика в лице трекинга истории подключений USB-устройств / Хабр](#)
- [usbrip: USB-форензика для Линуксов, или Как Алиса стала Евой / Codeby](#)
- [Hack The Box :: Forensics Challenges](#)

## Post Scriptum

---

Yep, the banner and info messages style is inspired by the *sqlmap* project (◡\_◡;)

If this tool has been useful for you, feel free to buy me a coffee ☕

