University of Amsterdam

System and Network Engineering

Large Installation Administration

USB device tracking in large-scale environments

Authors:
Eddie Bijnen
Kim van Erkelens

March 28, 2014

Abstract

In this paper we will explain how one can track USB devices through a network with the help of serial numbers. Most USB devices have a serial number. When this serial number is unique, it can be used to identify the device. The researches have created a proof of concept that will track these serial numbers. When a device is plugged into a computer that is running an agent, information about the device and the machine it is plugged into, will be sent to a central server. This information can be used to track down the last user of a device. Especially, this can be used when a device is stolen or missing. Furthermore, research is conducted on how to apply this solution in large scale environments in order to use USB device tracking as part of a companies hardware inventory management.

Contents

List of Figures

1 Introduction

Universal Serial Bus (USB) devices are portable, easy to use but also frequently lost. Often an over sight, the result is company wide emails: "Where is the camera?" Not only the loss of a company asset could result in cost, also the loss of sensitive data can harm an organisation. Solutions such as hardware inventory management are used to maintain a database of company assets. However, these systems need to be kept up-to-date, and they don't provide all data relevant for USB device tracking.

The existing solutions for keeping track of USB devices, like registries on paper, are prone to errors and may not be up-to-date. Also, little research has been conducted on USB device tracking for the use case of hardware inventory management. In this report, we present a proof of concept that allows for tracking of USB devices, together with a theoretical study focused on UNIX systems.

1.1 Research Question

This study aimed to address the following research question: *How can USB devices be tracked in large-scale UNIX environments for hardware inventory management?*

In order to answer this question the following sub-questions were set:

1. What are the possibilities for uniquely identifying and tracking USB devices?
2. How does our solution scale in a large environment?
3. How can USB device tracking be tampered with?

1.2 Related Work

Studies about tracking of USB devices are mainly focused on Windows. Carvey and Altheide [1] conducted research about identifiers that are created by Windows when a USB device is connected. These identifiers can be used to uniquely identify a device. An important use case for this information is forensics in the case of data lost. They also proposed several other use cases such as finding a USB device, building a database of trusted devices, or keeping track of migration between restricted and non restricted systems. Included in this research was a script for the retrieval of USB data out of the registry.

Similar studies are mainly about forensics for finding evidence in the Windows registry for data theft via USB devices. Roy and Jain [2] described the importance of the Windows registry for forensics in case of data theft. They also created a script that can be used for a more in depth analysis by retrieving all the operations that are performed with the files on a USB storage device.

None of the studies focused on Linux. This research will contribute by conducting a study on USB device fingerprinting in UNIX environments, and combining this with hardware inventory management to be used on a large scale.

2 Background

This section describes background information relevant for this research. First, the USB standard is explained in general. Then, some characteristics of its implementation in UNIX based systems are pointed out.

2.1 USB standard

USB 2.0 is the most widely used version of the USB standard. Its specification consists of 31 documents as can be found on the official website of the USB Implementers Forum (USB-IF)[1]. This non-profit organisation is formed by the group of companies that developed the USB specification. Among those companies are multinationals like Intel, Microsoft, and IBM. The specification documents describe the architecture, data flow model, mechanical and electrical characteristics, the protocol layer, the USB device framework, and many other technical details. Relevant for this research is how the specification describes identification information of USB devices and how it's enforced by the standard.

The system where a USB device is connected to is called the USB host. The way a host communicates with a device depends on the device class the latter belongs to. A list of different device classes and their specifications can be found on usb.org[2]. Each device class describes the functionality of the devices. A class code is communicated to the USB host, and affects the drivers that are loaded by the host.

2.2 Implementation in UNIX systems

UNIX systems rely on an open-source list for looking up product names and vendors. The file that contains this list is called usb.ids, and is maintained by the Linux USB Project[3]. Various UNIX programs use this file for the creation of human-readable output.

[1] `http://www.usb.org/developers/docs/usb20_docs/`
[2] `http://www.usb.org/developers/docs/devclass_docs/`
[3] `http://www.linux-usb.org`

3 Methodology

To answer the first sub-question, theoretical research is performed together with a small experiment for testing the ability to fingerprint different types of USB devices. A proof of concept is developed to invigorate this research. Finally, benchmarking is performed on the proof of concept for the second sub-question regarding scalability of the solution.

3.1 Test environment

The project group has taken a cross section of USB devices that are owned by the project members. This includes USB sticks, smart phones, Wifi dongles and drawing tablets. Each device has been tested abnormal behaviour and serial number format. Decided is to use dmesg for this experiment and for the proof of concept, because that system-log stores history and is already in a human-readable format. The workstations used in this experiment were running Ubuntu Linux 13.04.

3.2 Proof of concept: USBinv

Figure 1 depicts the general architecture of USBinv[4]. The logging clients will run on every workstation and are responsible for sending relevant information retrieved from dmesg to the database server. The information stored in the database can be viewed with the web-interface. Each part of the system is described in more detail in the following sections.

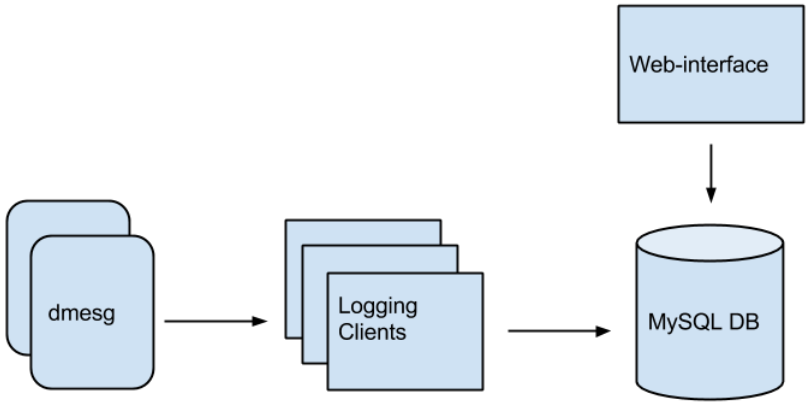

Figure 1: Architecture of the proof of concept

[4] https://github.com/kvanerkelens/lia-research

3.3 Database design

The database consists of two tables: one containing all the log events that are sent by the logging clients, and one table for the status of each device (Figure 2). Two views are created in order to display this information at the front-end of the system.

Two MySQL users are created for the USBinv scripts. The client script makes a connection with a user named workstation that has SELECT and INSERT privileges. The web-interface connects with a user named website that has SELECT privileges only.

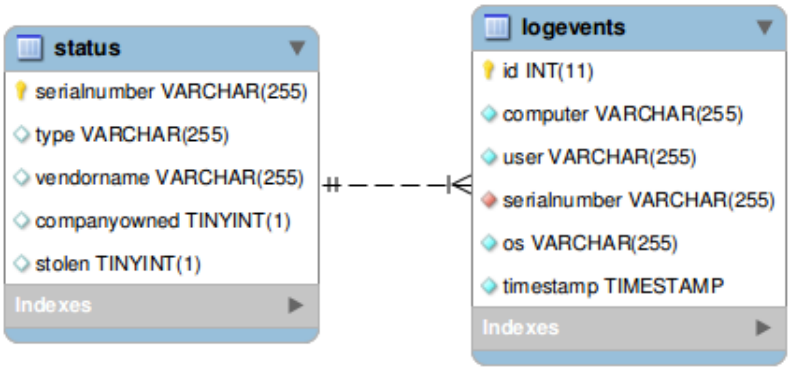

Figure 2: Database diagram for web interface

3.4 Client software

Each time a USB device is connected to the computer the serial number, product type, and manufacturer are retrieved from the dmesg and posted to the database. The database now has every instance of when that USB device has been used and this allows us to follow the USB device on our web interface.

The client software is written in Python which is supported by bash scripts for system information. For easy installation a script has been created to automate the installation process, the script is available at:
`http://usbinv.eddie4.nl/usbdevices/lia-research/install.sh`
Calling this script in the following fashion will install the USBinv.

```
wget http://usbinv.eddie4.nl/usbdevices/lia-research/install.sh |
chmod +x install.sh | ./install.sh
```

This installer will download and install the required dependencies: Python, Python-mysqldb and wget. Next, it will download the required files for the USBinv programme and install it into /etc/usbinv/ and set permissions. Lastly, it will create a rule into /etc/udev/rules.d and reload the Linux device manager. This will allow us the ability to have the script only run when a USB device is connected to the computer.

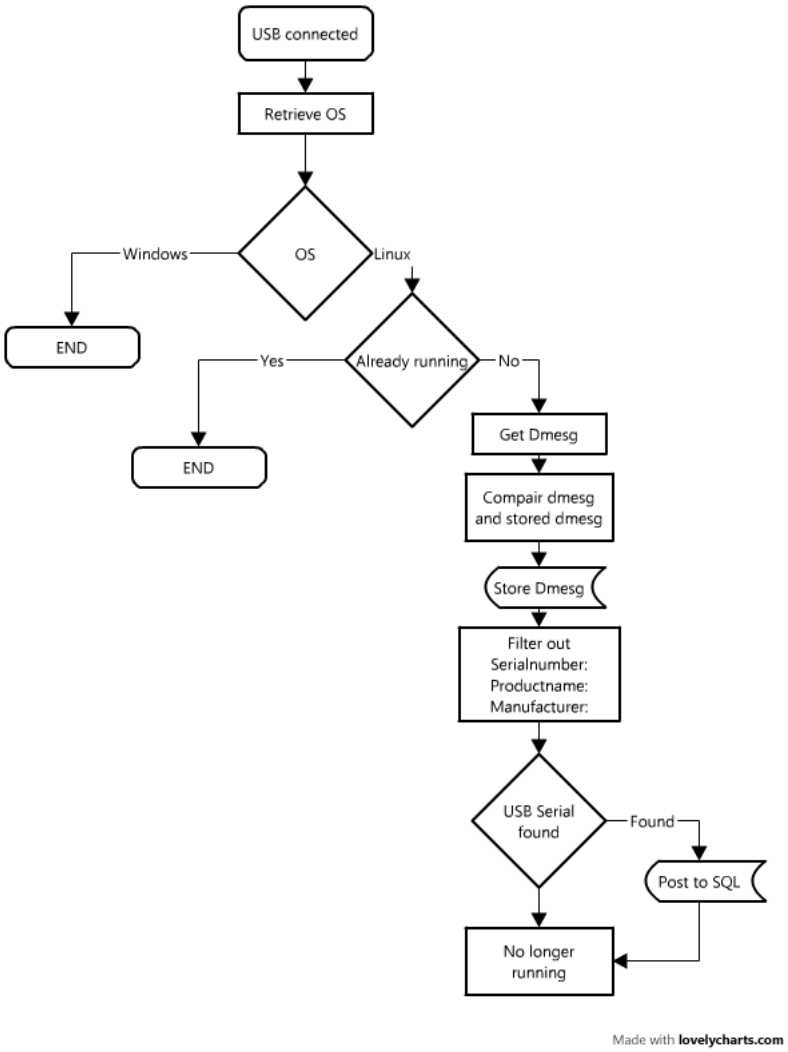

Figure 3: Flowchart

3.5 Front-end

The front-end is created with DataTables[5], an open-source jQuery plug-in that allows for sortable tables, paging and provides a search feature. DataTables is extended with a script[6] that allows for server-side processing and that uses MySQLi for connecting to the database. This script is slightly modified in order to make the serial numbers click-able. With this added function it's possible to get an overview of log events per device. Finally, the design of the interface is enhanced with Bootstrap using the code provided by DataTables[7].

[5] `http://datatables.net`
[6] `http://datatables.net/development/server-side/php_mysqli`
[7] `https://github.com/DataTables/Plugins/tree/master/integration/bootstrap/2`

The front-page of the proof of concept shows an overview of lost devices and of all the log events (Figure 4). The system currently doesn't support modification of the database, thus marking devices as lost has been done in the database directly.

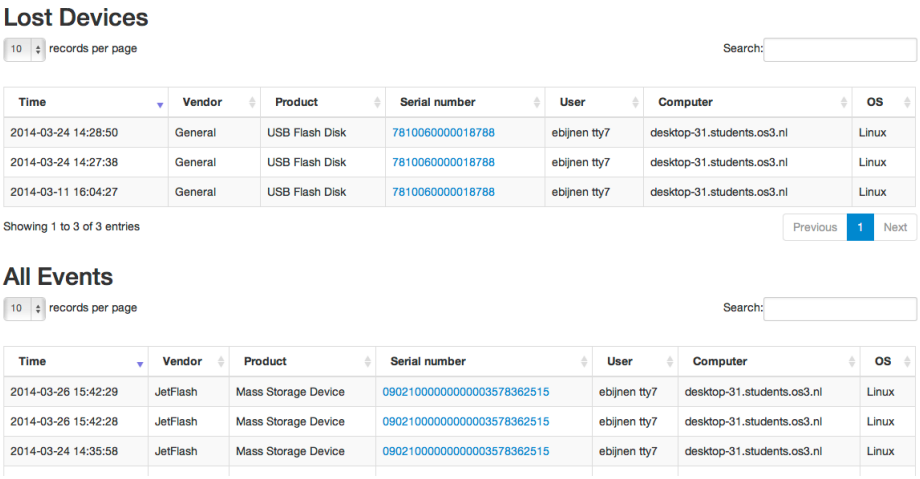

Figure 4: Proof of concept interface

3.6 Benchmark setup

To get an indication of the scalability of our application, some benchmarking has been done.

The server used for benchmarking has the following specifications:
Dell R210
Xeon L3426 @ 1.87GHz CPU
500GB 7200RPM hard disk
8GB DDR3 1333Mhz memory

MySQL 5.5 with Sysbench
1.000.000 records
4 threads

4 Results

This section describes the results for the sub-questions found in section 1.1.

4.1 Fingerprinting USB devices

Each USB device that is connected to a Linux system is registered in several places on the system. Actively connected devices are in /sys/bus/usb/devices/usb*, and /proc/bus/usb/devices or /sys/kernel/debug/usb/devices. This might differ for several distributions. Moreover, all actions are logged into the kernel ring buffer (dmesg).

The properties of a USB device are communicated to the USB host by means of device descriptors. The most important device descriptor for fingerprinting USB devices is the SerialNumber. Its purpose is to be a unique number for every device. Other useful device descriptors are the Vendor ID, and the Product ID, which can be supplied with their corresponding string descriptors for providing additional information in human readable format.

The Vendor ID is assigned by the USB-IF. The manufacturer is responsible for assigning a Product ID and SerialNumber to their devices. The SerialNumber is not mandatory according to the USB specification, except for one device type class. That means this device descriptor cannot be used in all cases for uniquely identifying USB devices. The Product and Vendor string descriptors are also optional for the manufacturer to supply. When the string descriptors aren't supplied, Linux uses the usb.ids list to display the IDs in a human-readable format.

A typical log message that is used for this experiment contains these device descriptors as shown in the example below. Line 1 contains the Vendor ID and Product ID in hexadecimal format. Line 3 and 4 are the product and vendor name in human readable form. The SerialNumber is shown on line 5.

```
1 usb 3-4.4: New USB device found, idVendor=0bb4, idProduct=0f91
2 usb 3-4.4: New USB device strings: Mfr=2, Product=3, SerialNumber=4
3 usb 3-4.4: Product: Android Phone
4 usb 3-4.4: Manufacturer: HTC
5 usb 3-4.4: SerialNumber: HT23JW105591
6 scsi10 : usb-storage 3-4.4:1.1
```

Listing 1: dmesg information from a HTC smart phone

The experiments showed that the USB devices can be divided into three groups: no serial number, non-unique serial number, and unique serial number. An overview of the results is given in Table 1. The full logging information can be found in Appendix A. The observation can be made that although a serial number is present it is not certain whether it is a unique one. To illustrate, one of the tested devices had a serial number of 12345. The proof of concept can't separate between such a serial number or unique serial numbers.

A solution for USB storage devices that don't have a unique serial number, is to write a custom serial number to the partition. This can also be automated by adding the new serial number on first insertion of the device.

Device	Type of serial number
USB drive	Unique
USB thumb drive	Unique
Smart phones	Unique
Wifi dongle	Non-unique
Pen tablet	None
Input devices	None

Table 1: Type of serial number for the tested USB devices

4.2 Scalability of USB device tracking

While designing the software our primary goal was to create a piece of software that would scale. We choose to keep all the processing on the client side. This allows us to have a single writable database server. The transfer of data is a connectionless process, which removes the need to track connections and keep state of clients. A connection is only setup when a USB device is connected. This makes the amount of USB device plug-ins per second the limiting factor for performance, instead of the amount of workstations or users. This number of plug-ins per second depends on the performance of the MySQL database.

The result from benchmarking on our current setup shows that we are able to handle 1157 read/write requests a second. Currently we are using one read and one write giving us the ability to process 579 USB plug-ins a second. Assuming that the average worker plugs in one USB device a week and works for 40 hours a week. The following calculation: 579 x Number of seconds in a work week. Gives us the number 83.376.000 employees. Some relativisation is needed, users will probably not spread evenly throughout the day. Growth of the database will slow down look up times. However the use of indexes, ageing of records and proper hardware for the database server will speedup the service. If the theoretical number of 83 million would be off by a 100 fold in production, we would still be able to track every USB device for the largest company in the Netherlands on a single server.

The front-end is able to search the database for specific users, types of devices, serial numbers and computers. The separation of views of lost/stolen devices and general updates gives a direct overview of the use of lost devices and gives information about whom are using devices at that time.

In order to use the system on a large scale, deployment of the logging clients could be automated. It is desirable to integrate deployment in updating, maintenance, or roll-out of new workstations. Our solution can be used standalone, but can also be integrated in existing inventory management systems by hooking the logging clients to the existing database.

4.3 Tampering with USB device tracking

Tracking of USB devices has one large limitation: it has to be plugged in to be detected. The programme can only detect the serial number if it's connected to a computer. If a USB device is lend-out but never used there is no record of the user ever having had the device. This is an obvious drawback that cannot be overcome. However, if the device is declared stolen and ever connected to a business computer again, for instance at home on a company laptop. These data will be logged and action can be taken.

4.3.1 Changing the serial number

Changing the serial number on a device is tricky. It is up to the vendor of the device to supply the tools and to allow it on their device. Vendors that create chips for custom/homemade USB devices typically supply a programme to change these parameter but vendors of products do not.

Devices that run an operating system that can be interfaced with are a different case. In Android there is a system file that lists the serial number, which can be modified. Changing this permanently requires an alternate init script.

4.3.2 Physical tampering

Before stealing a USB device one could connect it to a colleague's computer. This would register the colleague to be the last owner of the device. Company policy could make this colleague incorrectly responsible. If the device is never detected again, the culprit would not be caught. A possible solution is to disable registration of devices while the computer is locked. This would require that all employees lock there desktops.

4.3.3 Tampering with the software

The software is running on the local machine. This makes it inheritable vulnerable to modification. Currently, the program is only available to root. However, if users have root permissions they could manipulate the code or disable the programme altogether. It could as well be changed to inject false statements. However, the database does not allow for the removal or editing of previous records.

5 Conclusion

It is not possible for every USB device to be uniquely identified based on the device descriptors that are assigned to it. Therefore, it is impossible to differentiate two identical devices. Because the USB specification doesn't enforce a unique serial number, it's up to the vendor to decide whether to implement this. This assignment of unique serial numbers varies per manufacturer, and type of device. When it's important for an organisation to uniquely identify devices, this need to be considered at purchasing time.

The proof of concept shows that we are able to track USB devices through a network of computers that had our agent installed. Initial benchmarking has shown that our approach scales. Although this benchmark is partially based on assumptions, we feel confident that our approach would scale to large companies and perhaps multinationals with just a single server dedicated to the tracking of USB devices.

Tampering with the serial numbers of devices is a difficult task and requires extensive knowledge. It is unlikely that a person would undertake such a task. Tampering with the programme on the computer would be an easier task. However, it is risky if the software gets updated or is checked on integrity, stolen devices would get detected.

This research and proof of concept have demonstrated that tracking based on serial numbers is a possibility if products are bought from a manufacturer that takes serial numbers seriously. We believe that USBinv has a future as dynamic renting system or a system for tracking down lost or stolen devices.

6 Future work

6.1 Windows support

Currently the proof of concept does not support Microsoft Windows. Research has been done if Windows provides the required information. It is available it is however in a different format. Due to time restriction we chose not to expand the script to include windows support. The database and website support the addition of Windows.

6.2 Management functionalities and notifications

The front-end of the proof of concept doesn't have any functionality for modification of the database. Therefore, it is not possible to mark a device as lost. At this time are unable to send notification when a lost/stolen USB device is detected. An application could be created to monitor the database to send an alert once a new record is created for a lost/stolen device. Giving administrators the ability to respond immediately.

6.3 Offline tracking

Currently if the computer does not have an Internet connection the USB tracking information is not saved to the database and lost. Additional code could be added to the script to cache the information until the information is successfully uploaded. This would create a more complete image of who is using the devices. USB UMTS stick especially will benefit of this feature as they will primarily be plugged in when no Internet connection is available

References

[1] Harlan Carvey and Cory Altheide. “Tracking USB storage: Analysis of windows artifacts generated by USB storage devices”. In: *Digital Investigation* 2.2 (June 2005), pp. 94–100.

[2] Tanushree Roy and Aruna Jain. “Windows Registry Forensics : An Imperative Step in Tracking Data Theft via USB Devices”. In: 3.3 (2012), pp. 4427–4433.

Appendices

A Appendix: dmesg records

```
USBStick 1
[585530.898078] usb 3-4.4: new high-speed USB device number 25 using xhci_hcd
[585531.348472] usb 3-4.4: New USB device found, idVendor=090c, idProduct=1000
[585531.348477] usb 3-4.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[585531.348480] usb 3-4.4: Product: USB Flash Disk
[585531.348482] usb 3-4.4: Manufacturer: General
[585531.348485] usb 3-4.4: SerialNumber: 7810060000018788
[585531.348641] usb 3-4.4: ep 0x81 - rounding interval to 128 microframes, ep desc says 25
[585531.348645] usb 3-4.4: ep 0x2 - rounding interval to 128 microframes, ep desc says 255
[585531.349113] scsi12 : usb-storage 3-4.4:1.0
[585532.346280] scsi 12:0:0:0: Direct-Access     General  USB Flash Disk   1100 PQ: 0 ANSI
[585532.347312] sd 12:0:0:0: [sdb] 7839744 512-byte logical blocks: (4.01 GB/3.73 GiB)
[585532.347446] sd 12:0:0:0: Attached scsi generic sg1 type 0
[585532.347949] sd 12:0:0:0: [sdb] Write Protect is off
[585532.347955] sd 12:0:0:0: [sdb] Mode Sense: 43 00 00 00
[585532.348752] sd 12:0:0:0: [sdb] No Caching mode page found
[585532.348757] sd 12:0:0:0: [sdb] Assuming drive cache: write through
[585532.351629] sd 12:0:0:0: [sdb] No Caching mode page found
[585532.351634] sd 12:0:0:0: [sdb] Assuming drive cache: write through
[585532.352477]  sdb: sdb1 sdb2
[585532.354469] sd 12:0:0:0: [sdb] No Caching mode page found
[585532.354474] sd 12:0:0:0: [sdb] Assuming drive cache: write through
[585532.354477] sd 12:0:0:0: [sdb] Attached SCSI removable disk
[585532.564232] ISO 9660 Extensions: Microsoft Joliet Level 3
[585532.565414] ISO 9660 Extensions: RRIP_1991A
[585532.677781] systemd-hostnamed[7697]: Warning: nss-myhostname is not installed. Changin


USBstick 2
[585654.830063] usb 3-4.4: USB disconnect, device number 25
[585662.448499] usb 3-4.4: new high-speed USB device number 26 using xhci_hcd
[585662.470812] usb 3-4.4: New USB device found, idVendor=13fe, idProduct=3600
[585662.470817] usb 3-4.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[585662.470820] usb 3-4.4: Product: USB DISK 2.0
[585662.470822] usb 3-4.4: Manufacturer:
[585662.470824] usb 3-4.4: SerialNumber: 07B91C013A4XXXXX
[585662.471297] usb-storage 3-4.4:1.0: Quirks match for vid 13fe pid 3600: 4000
[585662.471395] scsi13 : usb-storage 3-4.4:1.0
[585663.500393] scsi 13:0:0:0: Direct-Access              USB DISK 2.0     PMAP PQ: 0 ANSI
[585663.500880] sd 13:0:0:0: Attached scsi generic sg1 type 0


HTC One X
[584641.923977] usb 3-4.4: New USB device found, idVendor=0bb4, idProduct=0f91
[584641.923982] usb 3-4.4: New USB device strings: Mfr=2, Product=3, SerialNumber=4
```

```
[584641.923985] usb 3-4.4: Product: Android Phone
[584641.923987] usb 3-4.4: Manufacturer: HTC
[584641.923990] usb 3-4.4: SerialNumber: HT23JWXXXXXX
[584641.928942] scsi10 : usb-storage 3-4.4:1.1
```

Samsung galaxy Mini

```
[584851.882537] usb 3-4.4: new high-speed USB device number 23 using xhci_hcd
[584851.899501] usb 3-4.4: New USB device found, idVendor=04e8, idProduct=6860
[584851.899506] usb 3-4.4: New USB device strings: Mfr=2, Product=3, SerialNumber=4
[584851.899509] usb 3-4.4: Product: Android
[584851.899511] usb 3-4.4: Manufacturer: Samsung
[584851.899513] usb 3-4.4: SerialNumber: 155493D140D664157630683XXXXXXX
```

Nexus 5

```
[584922.522677] usb 3-4.4: new high-speed USB device number 24 using xhci_hcd
[584922.539574] usb 3-4.4: New USB device found, idVendor=18d1, idProduct=4ee1
[584922.539579] usb 3-4.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[584922.539581] usb 3-4.4: Product: Nexus 5
[584922.539584] usb 3-4.4: Manufacturer: LGE
[584922.539586] usb 3-4.4: SerialNumber: 05cc29f73444XXXX
```

HTC desire

```
[584832.717780] usb 3-4.4: New USB device found, idVendor=0bb4, idProduct=0ff9
[584832.717784] usb 3-4.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[584832.717787] usb 3-4.4: Product: Android Phone
[584832.717789] usb 3-4.4: Manufacturer: HTC
[584832.717791] usb 3-4.4: SerialNumber: SH0CERX0XXXX
```

Nokia Lumia 920

```
[ 2209.636031] usb 3-2: new high-speed USB device number 6 using xhci_hcd
[ 2209.673281] usb 3-2: New USB device found, idVendor=0421, idProduct=0661
[ 2209.673286] usb 3-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 2209.673289] usb 3-2: Product: RM-821|Nokia Lumia 920
[ 2209.673291] usb 3-2: Manufacturer: Nokia
[ 2209.673293] usb 3-2: SerialNumber: 000000XXXXXE454F0000000000000000
```

iPhone 4s:

```
[ 2091.850771] usb 3-1: new high-speed USB device number 5 using xhci_hcd
[ 2091.868509] usb 3-1: New USB device found, idVendor=05ac, idProduct=12a0
[ 2091.868514] usb 3-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 2091.868516] usb 3-1: Product: iPhone
[ 2091.868519] usb 3-1: Manufacturer: Apple Inc.
[ 2091.868521] usb 3-1: SerialNumber: 7056c3407fa952cedae2f5ba5fXXXXXXXXXXXXXX
```

Wacom tablet:

```
[ 2617.103072] usb 3-3.3: new full-speed USB device number 8 using xhci_hcd
[ 2617.124466] usb 3-3.3: New USB device found, idVendor=056a, idProduct=00d4
[ 2617.124471] usb 3-3.3: New USB device strings: Mfr=1, Product=2, SerialNumber=0
```

```
[ 2617.124474] usb 3-3.3: Product: CTL-460
[ 2617.124476] usb 3-3.3: Manufacturer: Wacom Co.,Ltd.
[ 2617.173043] input: Wacom Bamboo Pen Pen as /devices/pci0000:00/0000:00:14.0/usb3/3-3/3-
[ 2617.175670] input: Wacom Bamboo Pen Finger as /devices/pci0000:00/0000:00:14.0/usb3/3-3
[ 2617.175865] usbcore: registered new interface driver wacom
```

TP-Link USB wireless dongle:

```
[ 2432.409380] usb 3-3.4: new high-speed USB device number 7 using xhci_hcd
[ 2432.434006] usb 3-3.4: New USB device found, idVendor=0cf3, idProduct=1006
[ 2432.434011] usb 3-3.4: New USB device strings: Mfr=16, Product=32, SerialNumber=48
[ 2432.434014] usb 3-3.4: Product: USB2.0 WLAN
[ 2432.434016] usb 3-3.4: Manufacturer: ATHEROS
[ 2432.434018] usb 3-3.4: SerialNumber: 12345
[ 2432.466317] cfg80211: Calling CRDA to update world regulatory domain
[ 2432.471205] cfg80211: World regulatory domain updated:
[ 2432.471208] cfg80211:   (start_freq - end_freq @ bandwidth), (max_antenna_gain, max_eir
[ 2432.471209] cfg80211:   (2402000 KHz - 2472000 KHz @ 40000 KHz), (300 mBi, 2000 mBm)
[ 2432.471210] cfg80211:   (2457000 KHz - 2482000 KHz @ 40000 KHz), (300 mBi, 2000 mBm)
[ 2432.471211] cfg80211:   (2474000 KHz - 2494000 KHz @ 20000 KHz), (300 mBi, 2000 mBm)
[ 2432.471212] cfg80211:   (5170000 KHz - 5250000 KHz @ 40000 KHz), (300 mBi, 2000 mBm)
[ 2432.471212] cfg80211:   (5735000 KHz - 5835000 KHz @ 40000 KHz), (300 mBi, 2000 mBm)
[ 2432.487214] usb 3-3.4: ath9k_htc: Firmware htc_9271.fw requested
[ 2432.487265] usbcore: registered new interface driver ath9k_htc
[ 2432.773025] usb 3-3.4: ath9k_htc: Transferred FW: htc_9271.fw, size: 51272
[ 2433.010382] ath9k_htc 3-3.4:1.0: ath9k_htc: HTC initialized with 33 credits
[ 2433.276149] ath9k_htc 3-3.4:1.0: ath9k_htc: FW Version: 1.3
[ 2433.276153] ath: EEPROM regdomain: 0x809c
[ 2433.276155] ath: EEPROM indicates we should expect a country code
[ 2433.276157] ath: doing EEPROM country->regdmn map search
[ 2433.276158] ath: country maps to regdmn code: 0x52
[ 2433.276160] ath: Country alpha2 being used: CN
[ 2433.276161] ath: Regpair used: 0x52
[ 2433.286725] ieee80211 phy0: Atheros AR9271 Rev:1
[ 2433.286742] cfg80211: Calling CRDA for country: CN
[ 2433.289409] cfg80211: Regulatory domain changed to country: CN
[ 2433.289413] cfg80211:   (start_freq - end_freq @ bandwidth), (max_antenna_gain, max_eir
[ 2433.289415] cfg80211:   (2402000 KHz - 2482000 KHz @ 40000 KHz), (N/A, 2000 mBm)
[ 2433.289417] cfg80211:   (5735000 KHz - 5835000 KHz @ 40000 KHz), (N/A, 3000 mBm)
[ 2433.289419] cfg80211:   (57240000 KHz - 59400000 KHz @ 2160000 KHz), (N/A, 2800 mBm)
[ 2433.289420] cfg80211:   (59400000 KHz - 63720000 KHz @ 2160000 KHz), (N/A, 4400 mBm)
[ 2433.289422] cfg80211:   (63720000 KHz - 65880000 KHz @ 2160000 KHz), (N/A, 2800 mBm)
```