BadUSB — On accessories that turn evil

Karsten Nohl <nohl@srlabs.de>
Sascha Krißler <sascha@srlabs.de>
Jakob Lell <jakob@srlabs.de>

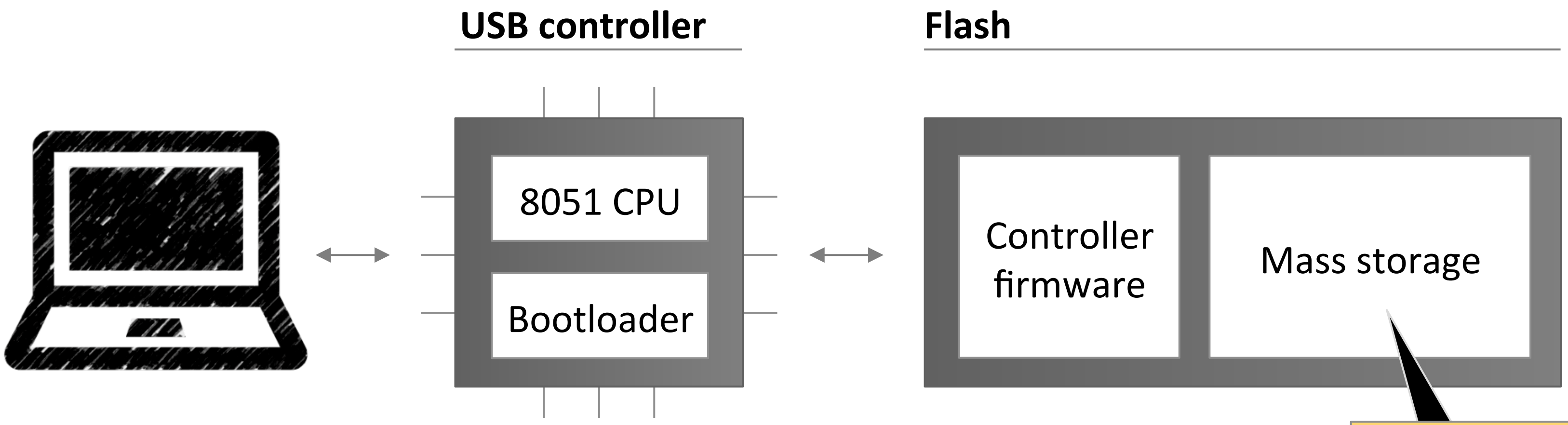
SECURITY RESEARCH LABS

Demo 1 – **USB stick takes over Windows machine**

Agenda

- **USB background**
- Reprogramming peripherals
- USB attack scenarios
- Defenses and next steps

USB devices include a micro-controller, hidden from the user



USB controller

8051 CPU

Bootloader

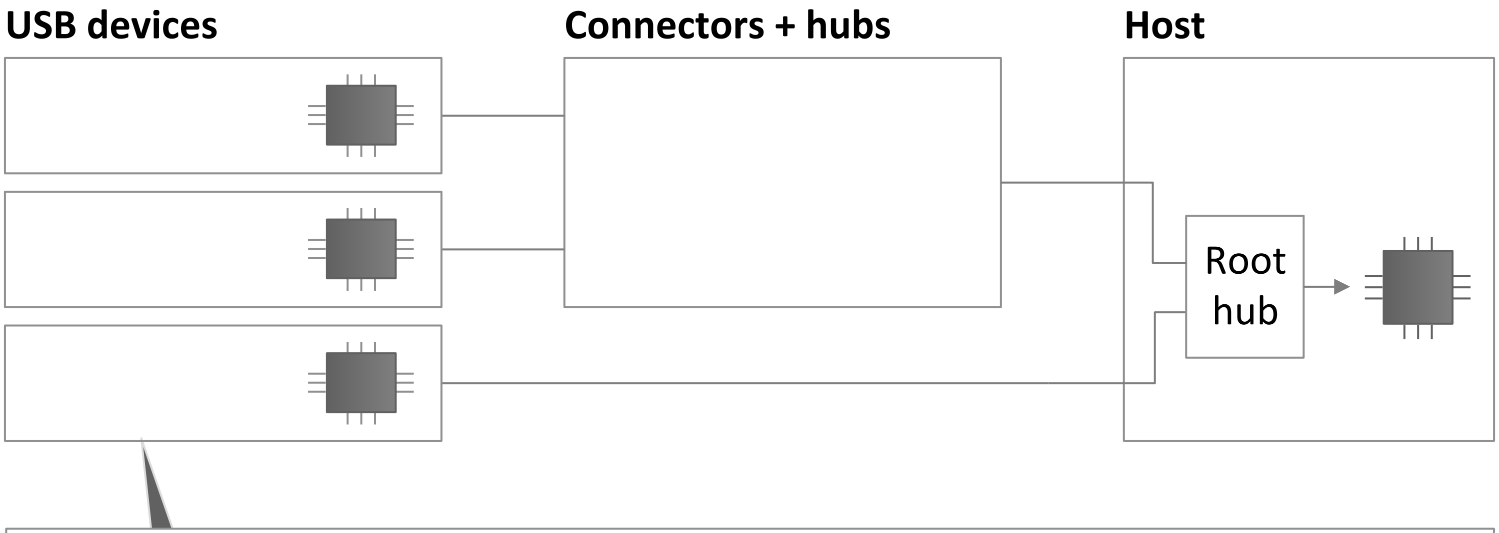
Flash

Controller firmware

Mass storage

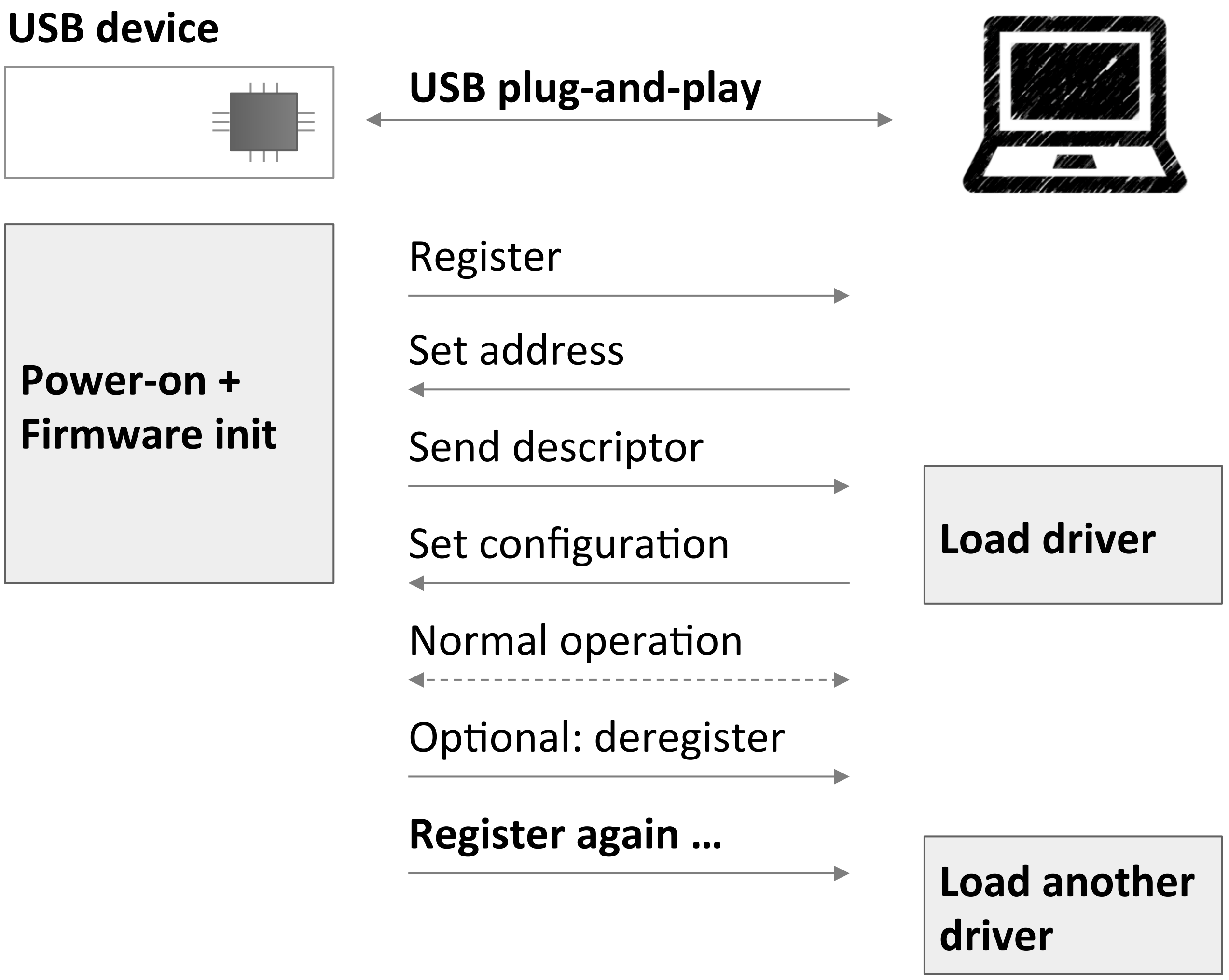
The only part visible to the user

USB devices are identified



	Examples	
Identifier	USB thumb drive	Webcam
Interface class	8 – Mass Storage	a. 1 – Audio b. 14 – Video
End points	0 – Control 1 – Data transfers	0 – Control 1 – Video transfers 6 – Audio transfers 7 – Video interrupts
Serial number	AA627090820000000702	0258A350

USB devices are initialized in several steps



USB device

USB plug-and-play

Power-on + Firmware init

Register

Set address

Send descriptor

Set configuration

Normal operation

Optional: deregister

Register again …

Load driver

Load another driver

Devices can have several identities

- A device indicates its capabilities through a descriptor
- A device can have several descriptors if it supports multiple device classes; like webcam + microphone
- Device can deregister and register again as a different device

Agenda

- USB background
- **Reprogramming peripherals**
- USB attack scenarios
- Defenses and next steps

Reversing and patching USB firmware took less than 2 months

A **Document firmware update process**

1. Find leaked firmware and flash tool on the net
2. Sniff update communication using Wireshark
3. Replay custom SCSI commands used for updates
4. (Reset bricked devices through short-circuiting Flash pins)

B **Reverse-engineer firmware**

1. Load into disassembler (complication: MMU-like memory banking)
2. Apply heuristics
   - Count matches between function start and call instructions for different memory locations
   - Find known USB bit fields such as descriptors
3. Apply standard software reversing to find hooking points

C **Patch firmware**

1. Add hooks to firmware to add/change functionality
2. Custom linker script compiles C and assembly code and injects it into unused areas of original firmware

Other possible targets

We focused on USB sticks, but the same approach should work for:

- External HDDs
- Webcams, keyboards
- Probably many more …

Agenda

- USB background
- Reprogramming peripherals
- **USB attack scenarios**
- Defenses and next steps

Demo 2 – **Windows infects USB stick which then takes over Linux machine**

Keyboard emulation is enough for infection and privilege escalation (w/o need for software vulnerability)

Challenge – Linux malware runs with limited user privileges, but needs *root* privileges to infect further sticks

Approach – Steal *sudo* password in screensaver

Restart screensaver (or *policykit*) with password stealer added via an LD_PRELOAD library

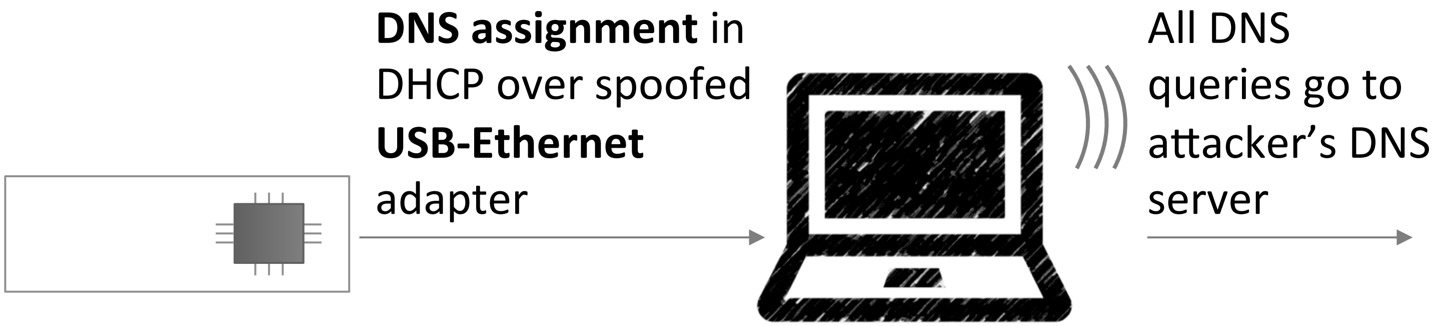
▷

- User enters password to unlock screen
- Malware intercepts password and gains root privileges using *sudo*

Privilege escalation module will be submitted to Metasploit

Demo 3 – **USB thumb drive changes DNS settings in Windows**

Network traffic can be diverted by “DHCP on USB”



DNS assignment in DHCP over spoofed **USB-Ethernet** adapter

All DNS queries go to attacker’s DNS server

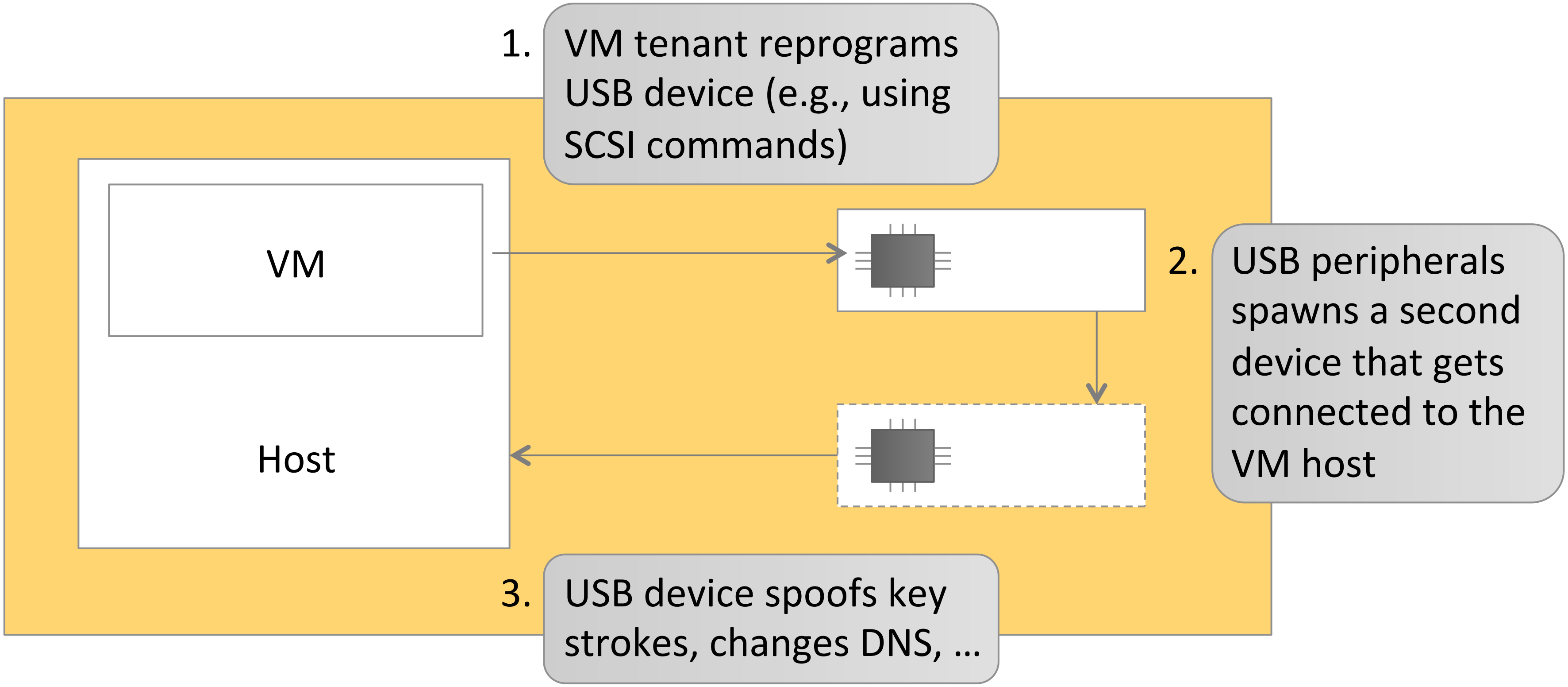
Attack steps

1. USB stick spoofs Ethernet adapter
2. Replies to DHCP query with DNS server on the Internet, but without default gateway

Result

3. Internet traffic is still routed through the normal Wi-Fi connection
4. However, DNS queries are sent to the USB-supplied server, enabling redirection attacks

Bonus: Virtual Machine break-out



1. VM tenant reprograms USB device (e.g., using SCSI commands)

VM

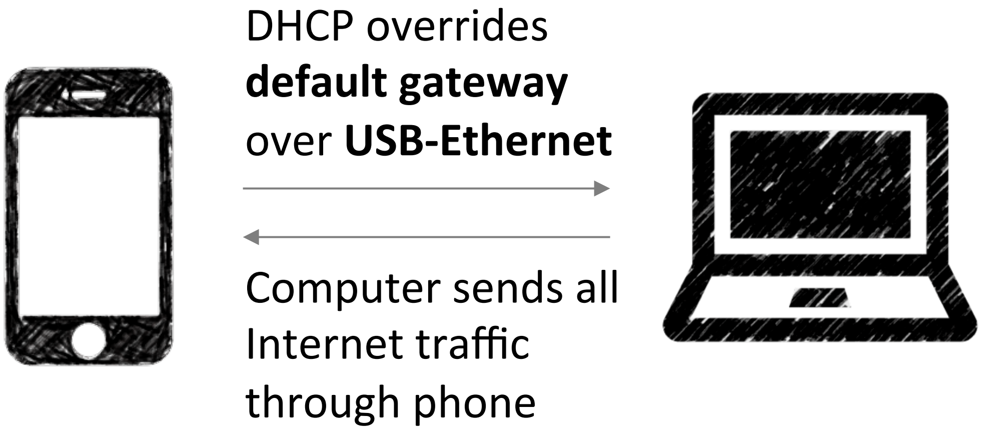
Host

2. USB peripherals spawns a second device that gets connected to the VM host

3. USB device spoofs key strokes, changes DNS, …

Demo 4 – **Android diverts data traffic from Windows machine**

“Can I charge my phone on your laptop?” – Android phones are the simplest USB attack platform



DHCP overrides **default gateway** over **USB-Ethernet**

Computer sends all Internet traffic through phone

Proof-of-concept released at: **srlabs.de/badusb**

Preparation – Android comes with an Ethernet-over-USB emulation needing little configuration

Attack – Phone supplies default route over USB, effectively intercepting all Internet traffic

Hacked by the second factor?

Using keyboard emulation, a virus-infected smartphone could hack into the USB-connected computer.

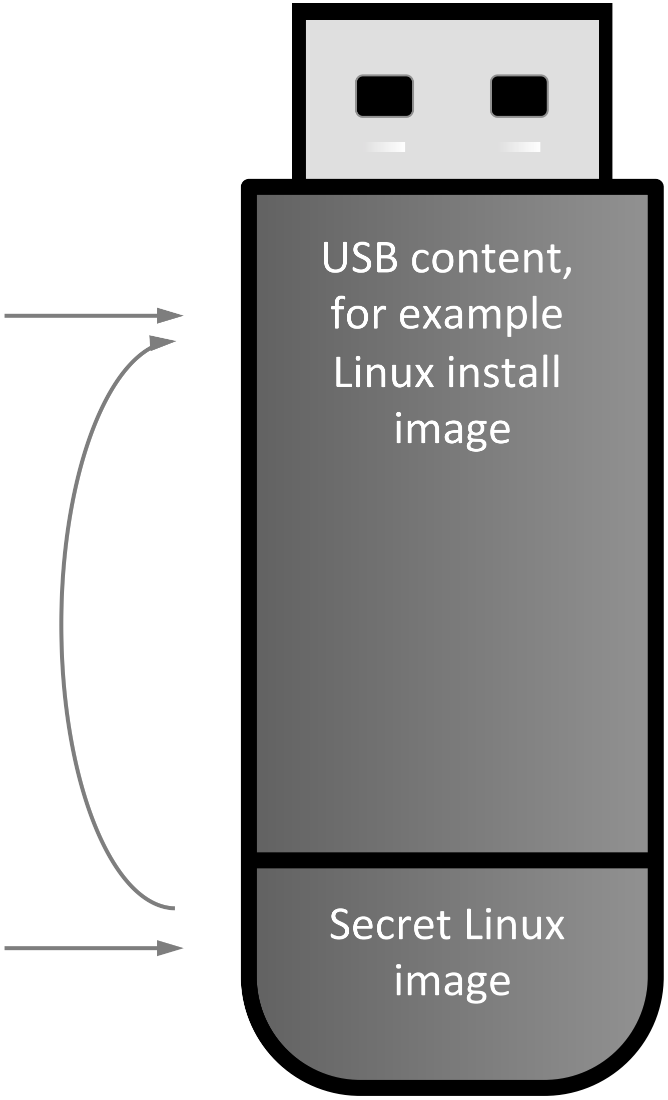
This compromises the “second factor” security model of online banking.

Boot-sector virus, USB style

Fingerprint OS/BIOS.
Patched/ USB stick firmware can distinguish Win, Mac, Linux, and the BIOS based on their USB behavior

Hide rootkit from OS/AV.
When an OS accesses the stick, only the USB content is shown

Infect machine when booting.
When the BIOS accesses the stick, a secret Linux is shown, booting a root kit, infecting the machine, and then booting from the USB content



USB content, for example Linux install image

Secret Linux image

Family of possible USB attacks is large

Attacks shown

- Emulate keyboard
- Spoof network card
- “USB boot-sector” virus

More attack ideas	Effect
Hide data on stick or HDD	External storage can choose to hide files instead of deleting them
Rewrite data in-flight	Viruses can be added to files added to storage First access by virus scanner sees original file, later access sees virus
Update PC BIOS	Emulate a keyboard during boot and install a new BIOS from a file in a secret storage area on a USB stick
Spoof display	Emulate a USB display to access security information such as Captchas and randomly arranged PIN pads

Agenda

- USB background
- Reprogramming peripherals
- USB attack scenarios
- **Defenses and next steps**

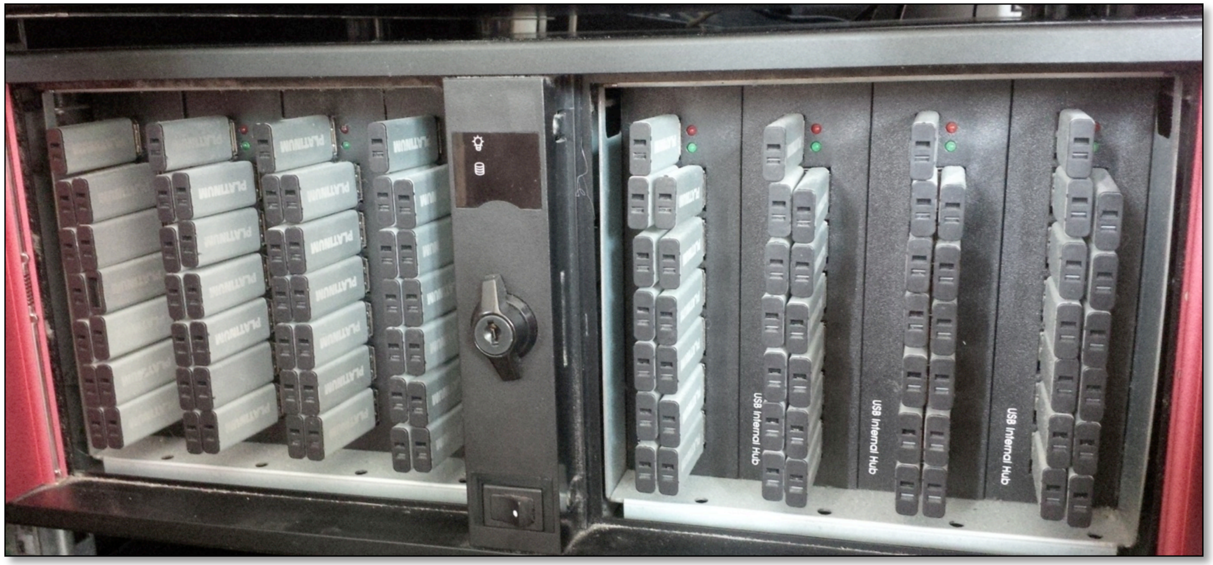
No effective defenses from USB attacks exist

Protection idea	Limitation
Whitelist USB devices	▪ USB devices do not always have a unique serial number ▪ OS's don't (yet) have whitelist mechanisms
Block critical device classes, block USB completely	▪ Obvious usability impact ▪ Very basic device classes can be used for abuse; not much is left of USB when these are blocked
Scan peripheral firmware for malware	▪ The firmware of a USB device can typically only be read back with the help of that firmware (if at all): A malicious firmware can spoof a legitimate one
Use code signing for firmware updates	▪ Implementation errors may still allow installing unauthorized firmware upgrades ▪ Secure cryptography is hard to implement on small microcontrollers ▪ Billions of existing devices stay vulnerable
Disable firmware updates in hardware	▪ **Simple and effective**

USB peripherals can also be re-programmed for constructive purposes

Idea 1 – **Speed up database queries**

- Data can be parsed on the stick before (or instead of) sending it back to the host
- Our original motivation was to speed up of A5/1 rainbow table lookups



Idea 2 – **Repurpose cheap controller chips**

- Use the reprogrammable chips for other applications than USB storage
- The flowswitch / phison project, for example, aims for a low-cost USB 3 interface for FPGAs

Take aways

- **USB** peripherals provide for a versatile **infection path**

- Once infected – through USB or otherwise – malware can use peripherals as a **hiding place**, hindering system clean-up

- As long as USB controllers are re-programmable, USB peripherals should **not be shared** with others

Questions?

usb@srlabs.de