Peripheral Security

Why ‘Peripheral’ = USB

Old days

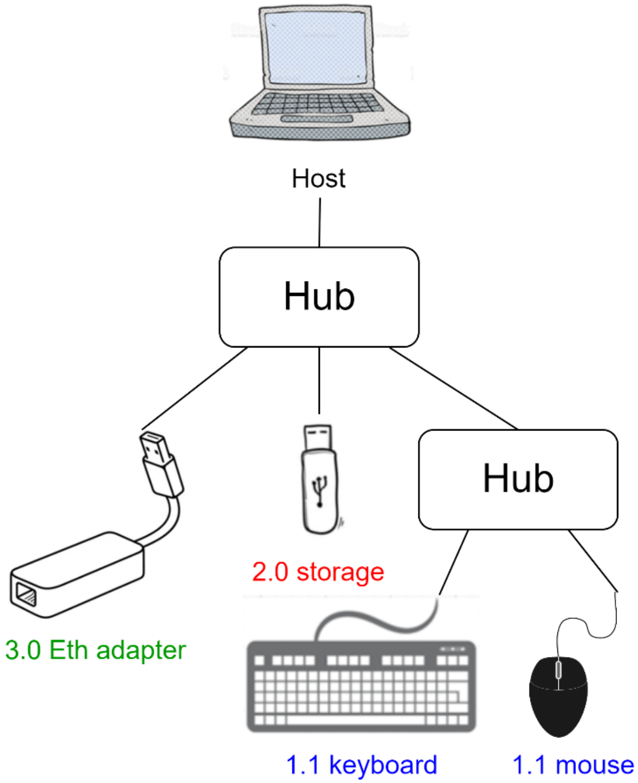
Now

USB Overview

- USB 1.0 (1996), …, USB4 (2019)
- Various speed modes, electrical interfaces & connector types

Standard		USB 1.0 1996	USB 1.1 1998	USB 2.0 2001	USB 2.0 Revised	USB 3.0 2008	USB 3.1 2013	USB 3.2 2017	USB4 2019	USB4 v2.0 2022
Max Speed	*Marketing name* (operation mode)	*Low-Speed* & *Full-Speed*		*High-Speed*		*SuperSpeed USB 5Gbps*, original: *SuperSpeed* (Gen 1)	*SuperSpeed USB 10Gbps*, original: *SuperSpeed+* (Gen 2)	*SuperSpeed USB 20Gbps* (USB 3.2 Gen 2×2)	*USB4 40Gbps* (USB4 Gen 3×2)	*USB4 80Gbps* (USB4 Gen 4)
	Signaling rate	1.5 Mbit/s & 12 Mbit/s		480 Mbit/s		5 Gbit/s	10 Gbit/s	20 Gbit/s	40 Gbit/s	80 Gbit/s

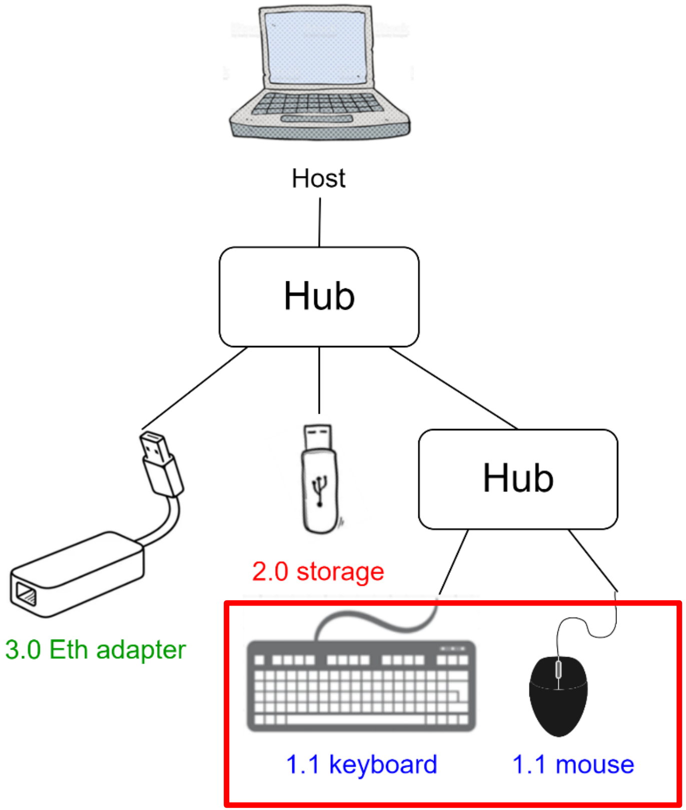
- Tree topology:
  - host = root
  - hubs = nodes
  - devices = leaves
- Backward compatibility built-in
- Host-arbitrated shared bus
- Traffic not encrypted



USB Overview

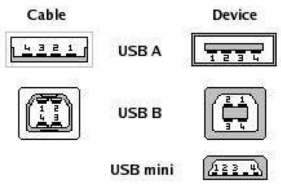
Standard		USB 1.0 1996	USB 1.1 1998	USB 2.0 2001	USB 2.0 Revised	USB 3.0 2008	USB 3.1 2013	USB 3.2 2017	USB4 2019	USB4 v2.0 2022
Max Speed	***Marketing name*** **(operation mode)**	*Low-Speed & Full-Speed*		*High-Speed*		*SuperSpeed USB 5Gbps*, original: *SuperSpeed* (Gen 1)	*SuperSpeed USB 10Gbps*, original: *SuperSpeed+* (Gen 2)	*SuperSpeed USB 20Gbps* (USB 3.2 Gen 2×2)	*USB4 40Gbps* (USB4 Gen 3×2)	*USB4 80Gbps* (USB4 Gen 4)
	Signaling rate	1.5 Mbit/s & 12 Mbit/s		480 Mbit/s		5 Gbit/s	10 Gbit/s	20 Gbit/s	40 Gbit/s	80 Gbit/s

Human Interface Devices (HIDs) are USB 1.x



Pinout and wiring

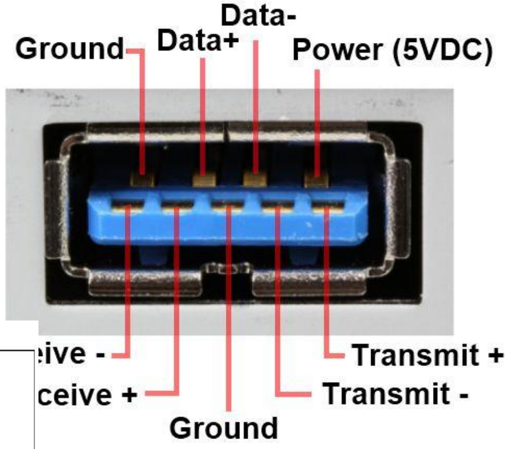
USB 1.x & 2.0



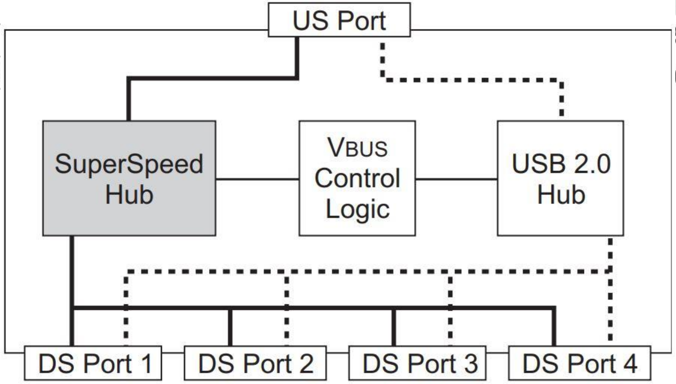
Pin	Signal	Color	Description
1	VCC	red	+5V
2	D-	white	Data -
3	D+	green	Data +
4	GND	black	Ground

WWW.MODDIY.COM

USB 3.x



USB 3.x hub



USB Built-in Security Measures

USB Implementers Forum statement on security (2014):

- "The USB-IF agrees that consumers should always ensure their devices are from a trusted source and that only trusted sources interact with their devices."
- "If consumer demand for USB products with additional capabilities for security grows, we would expect OEMs to meet that demand."

USB-IF notable members:

Why should we care?

Users Really Do Plug in USB Drives They Find

Matthew Tischer† Zakir Durumeric‡† Sam Foster† Sunny Duan†
Alec Mori† Elie Bursztein◇ Michael Bailey†

† University of Illinois, Urbana Champaign ‡ University of Michigan ◇ Google, Inc.
{tischer1, sfoster3, syduan2, ajmori2, mdbailey}@illinois.edu
zakir@umich.edu elieb@google.com

Abstract—We investigate the anecdotal belief that end users will pick up and plug in USB flash drives they find by completing a controlled experiment in which we drop 297 flash drives on a large university campus. We find that the attack is effective with an estimated success rate of 45–98% and expeditious with the first drive connected in less than six minutes. We analyze the types of drives users connected and survey those users to understand their motivation and security profile. We find that a drive's appearance does not increase attack success. Instead, users connect the drive with the altruistic intention of finding the owner. These individuals are not technically incompetent, but are rather typical community members who appear to take more recreational risks then their peers. We conclude with lessons learned and discussion on how social engineering attacks—while less technical—continue to be an effective attack vector that our community has yet to successfully address.

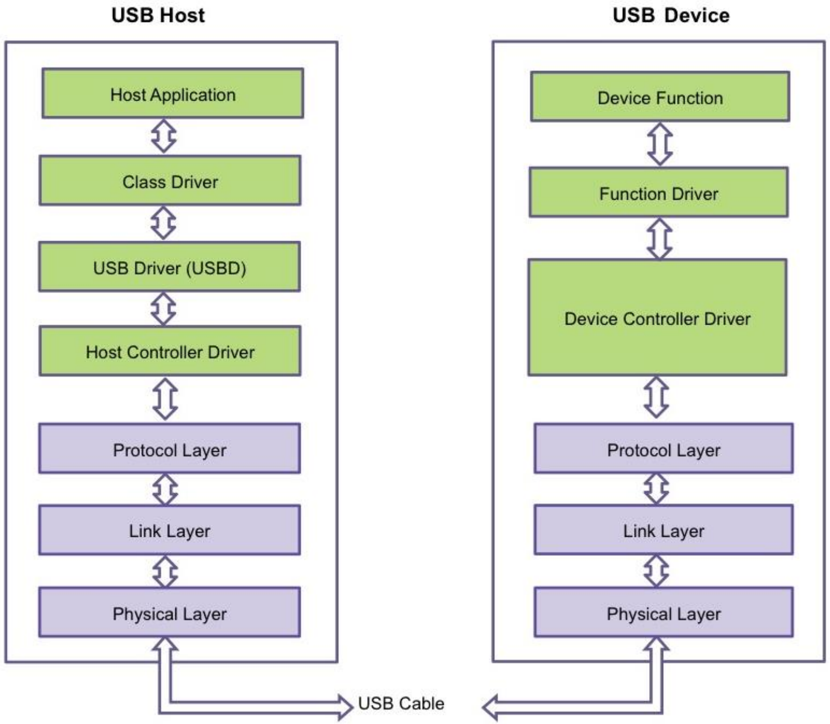
median time to connection of 6.9 hours and the first connection occurring within six minutes from when the drive was dropped. Contrary to popular belief, the appearance of a drive does not increase the likelihood that someone will connect it to their computer. Instead, users connect all types of drives unless there are other means of locating the owner—suggesting that participants are altruistically motivated. However, while users initially connect the drive with altruistic intentions, nearly half are overcome with curiosity and open intriguing files—such as vacation photos—before trying to find the drive's owner.

To better understand users' motivations and rationale, we offered participants the opportunity to complete a short survey when they opened any of the files and read about the study. In this survey, we ask users why they connected the drive, the

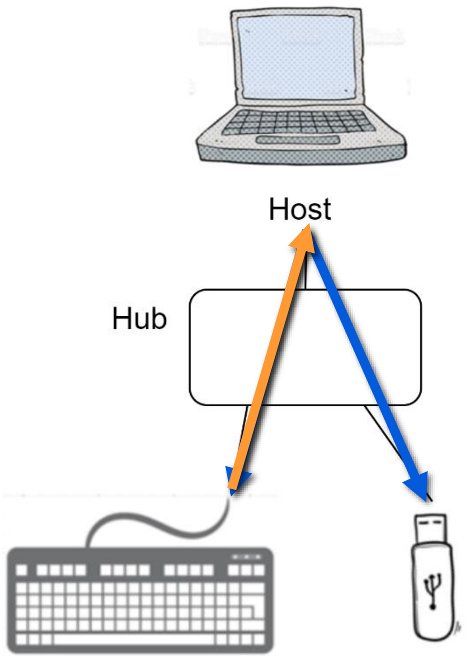
USB Communications Stack

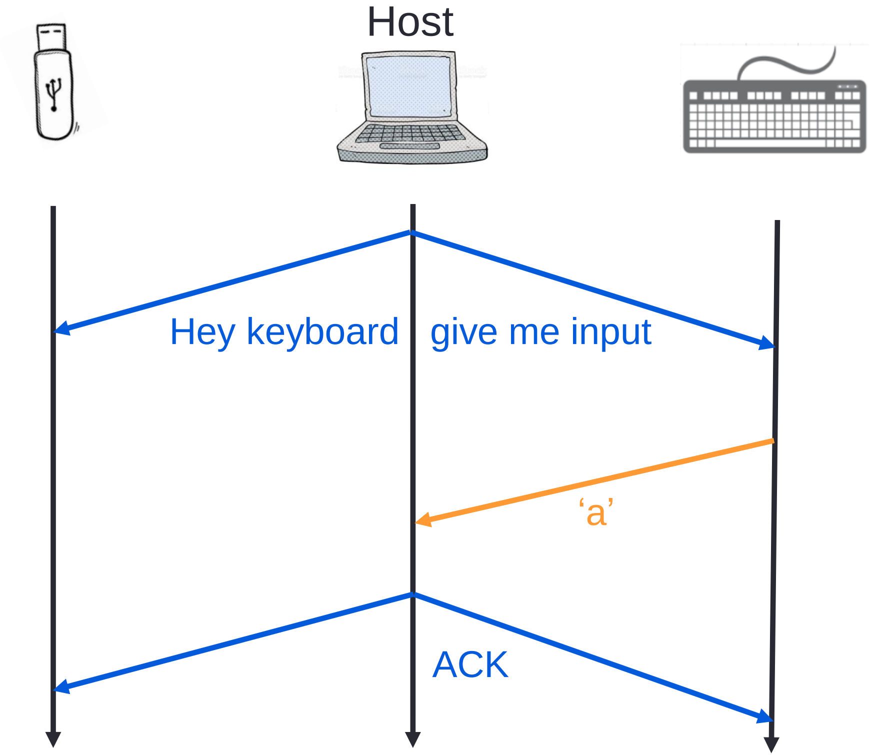
Software

Hardware

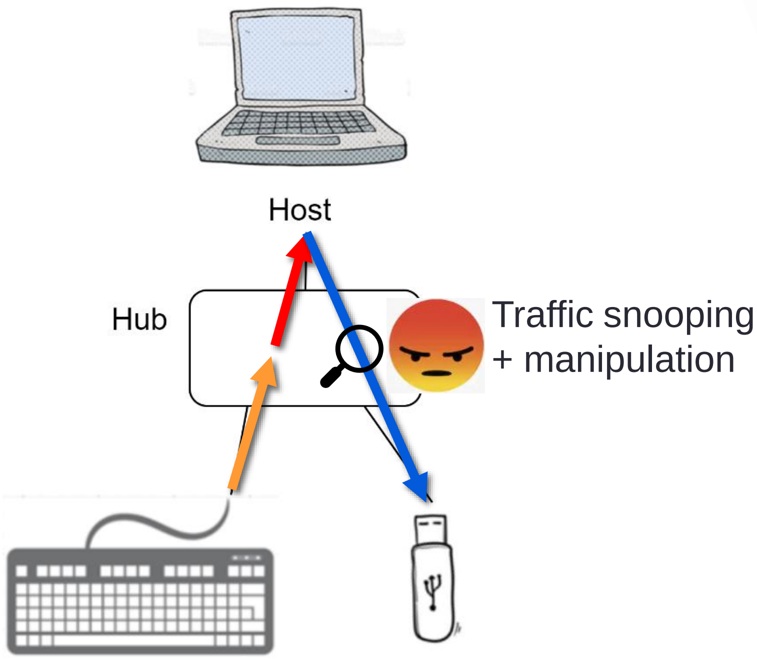


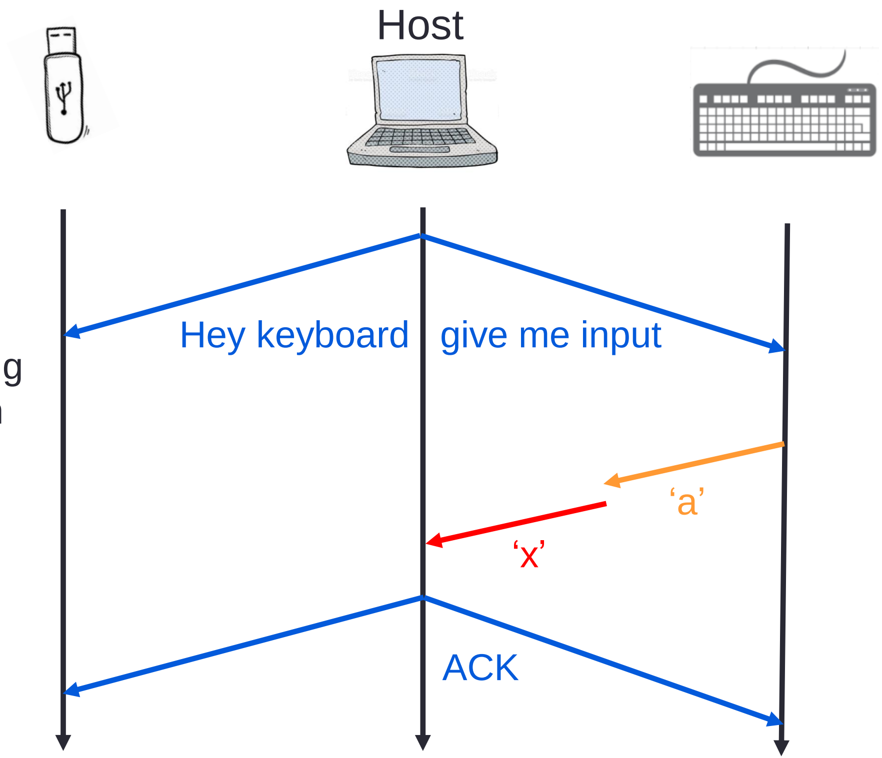
USB 1.1 & 2.0 Communication Model





Hub-in-the-Middle (on-path)



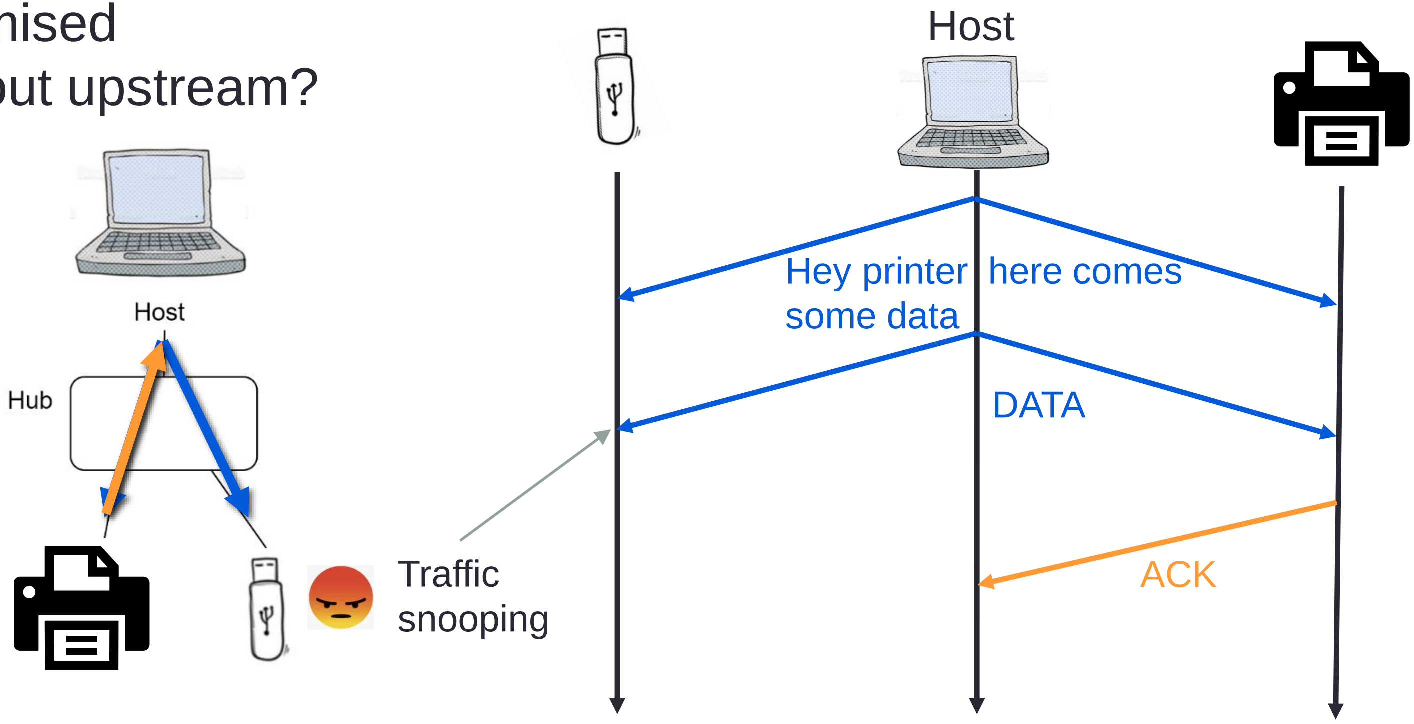


What can we do with devices?


PDF
USB Root Hub
PDF
A
USB Hub
SanDisk
A

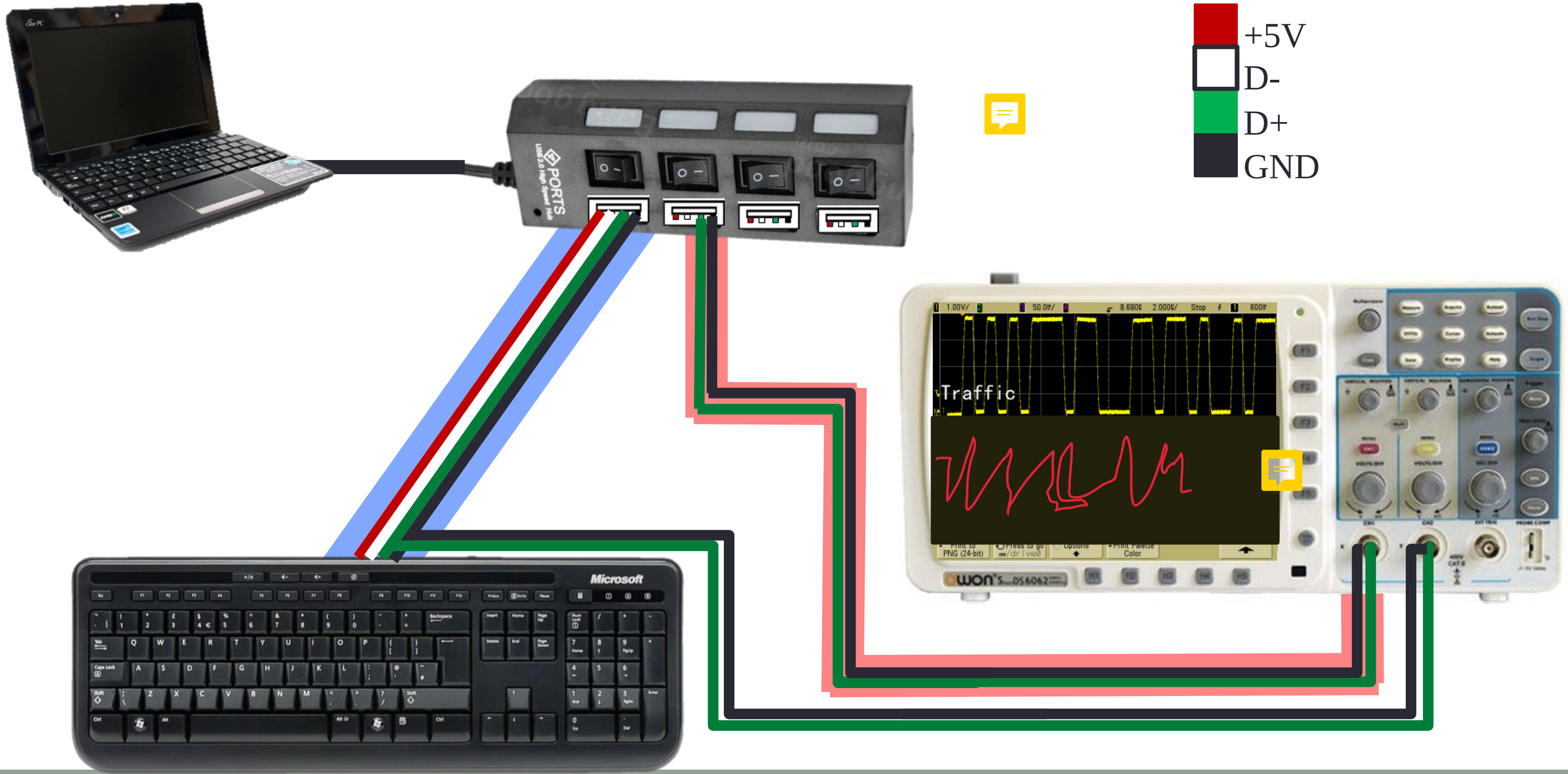
Off-path Device Snooping

Confidentiality of downstream traffic is compromised
How about upstream?



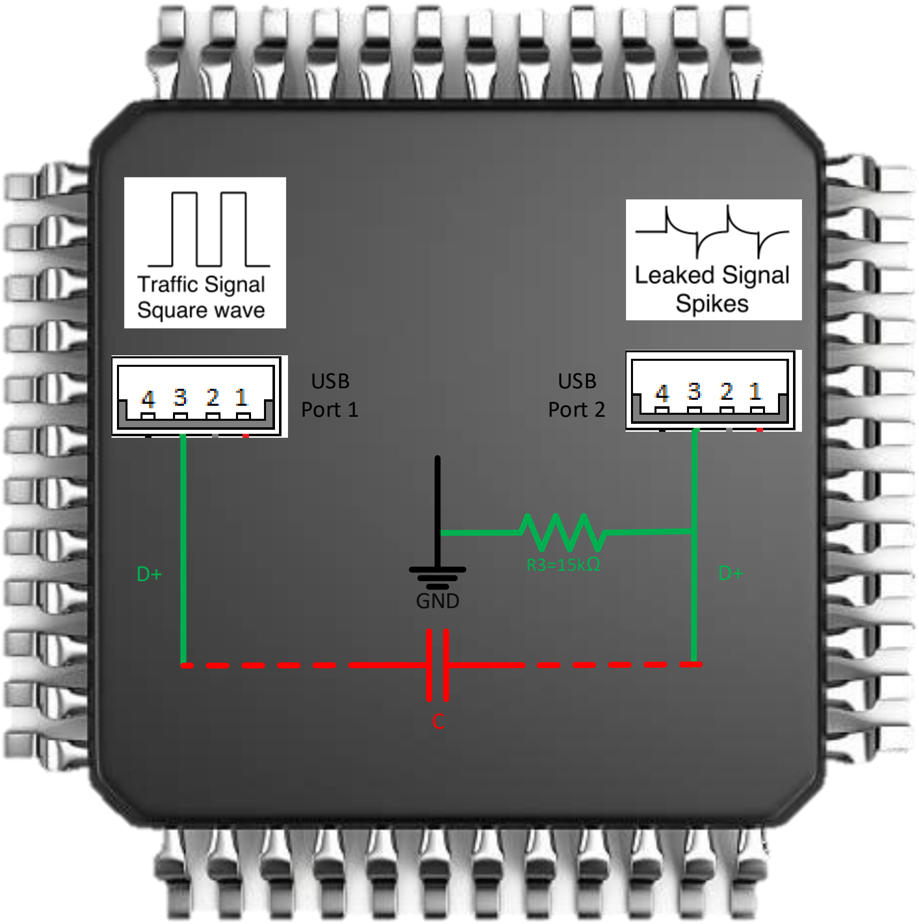
Attack model – upstream confidentiality

Observing Crosstalk Leakage

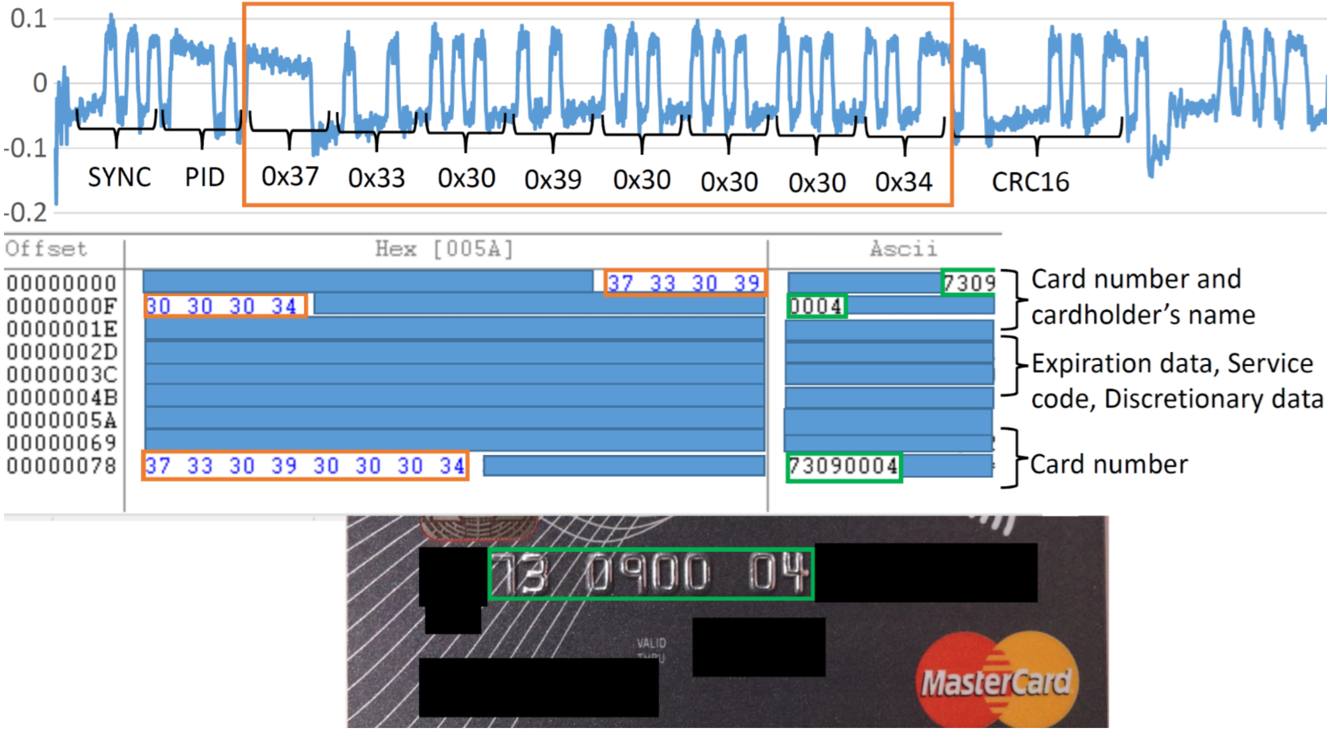


Leakage Mechanism

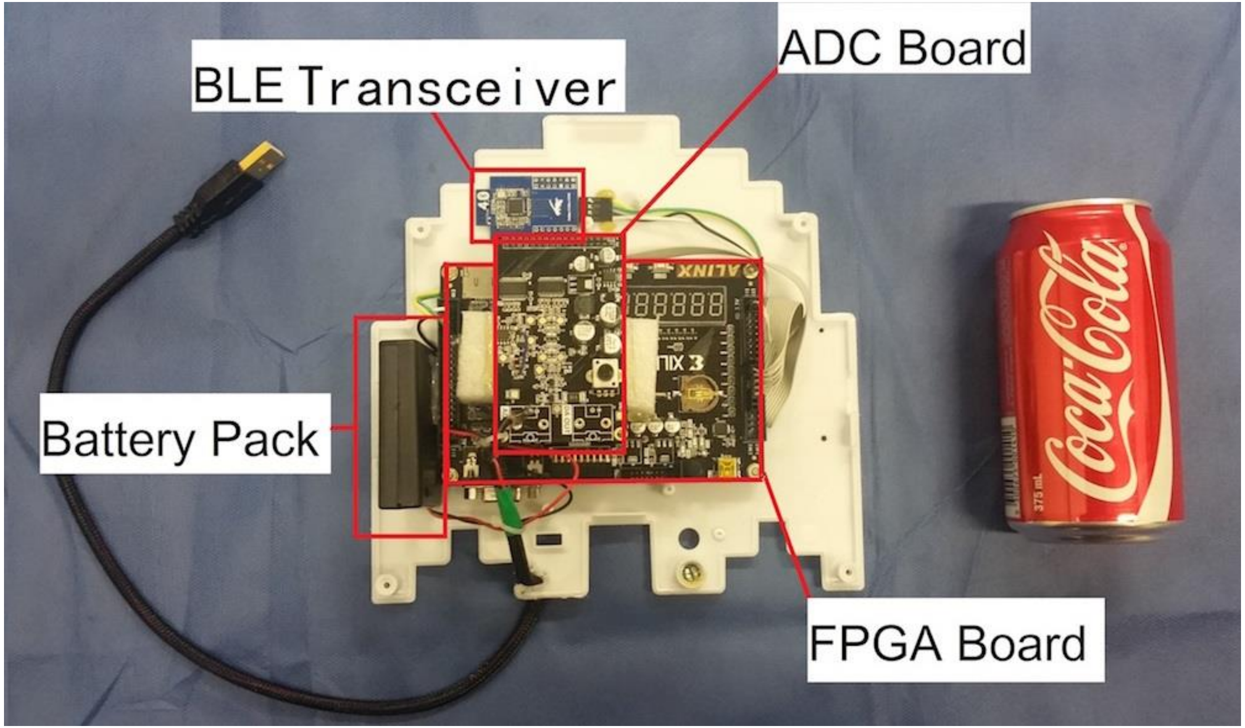
- USB hubs typically consist of one main chip
- Two USB logic blocks should be isolated from each other
- Due to manufacturing imperfections parasitic capacitance is present between different USB ports on the chip
- Fluctuations on one port are visible from other ports



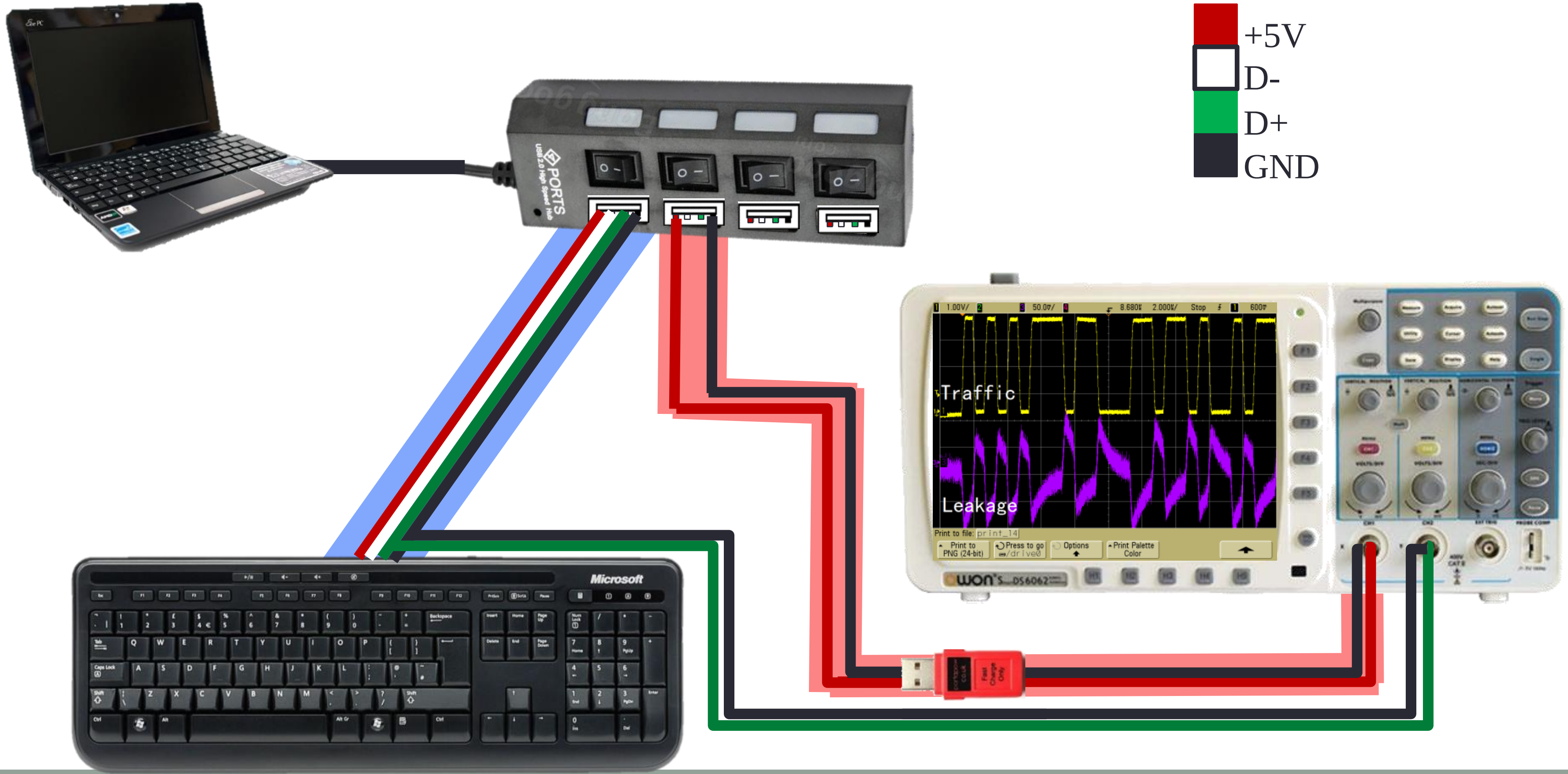
Snooping interesting data



Keystroke exfiltrator

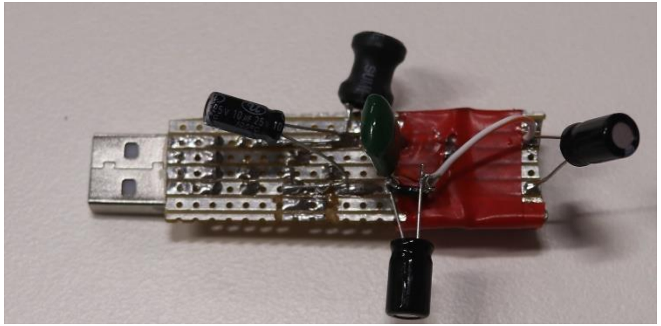


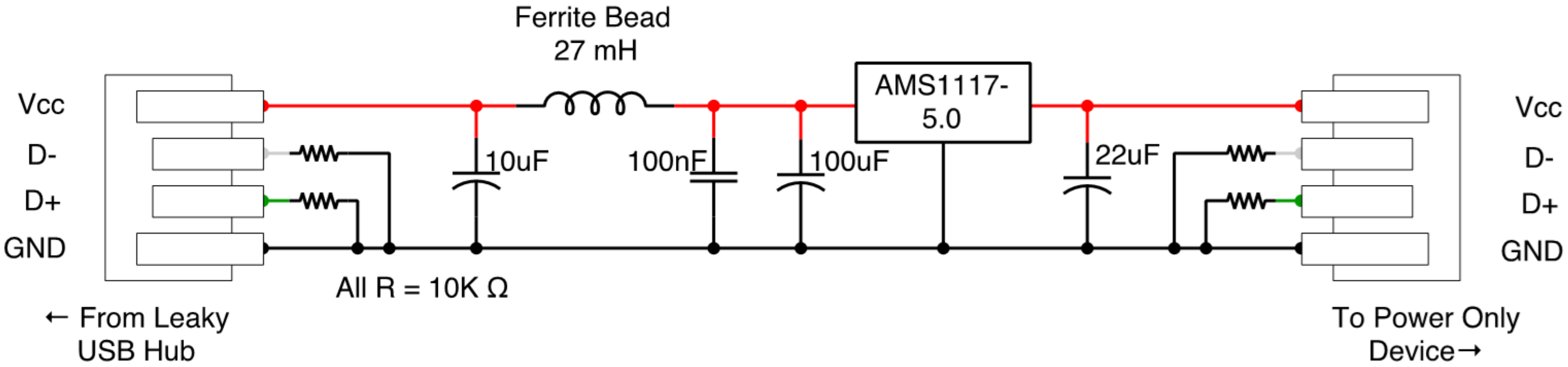
Also works just through power line



Countermeasures

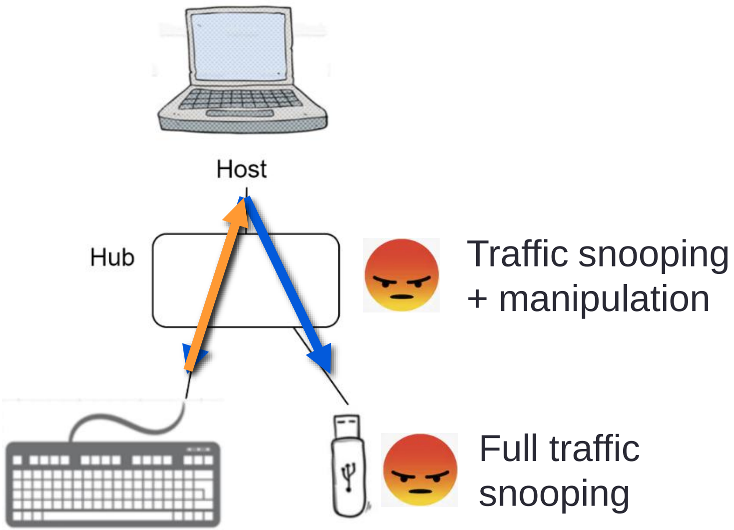
- Optical decoupling
- Improved USB condom

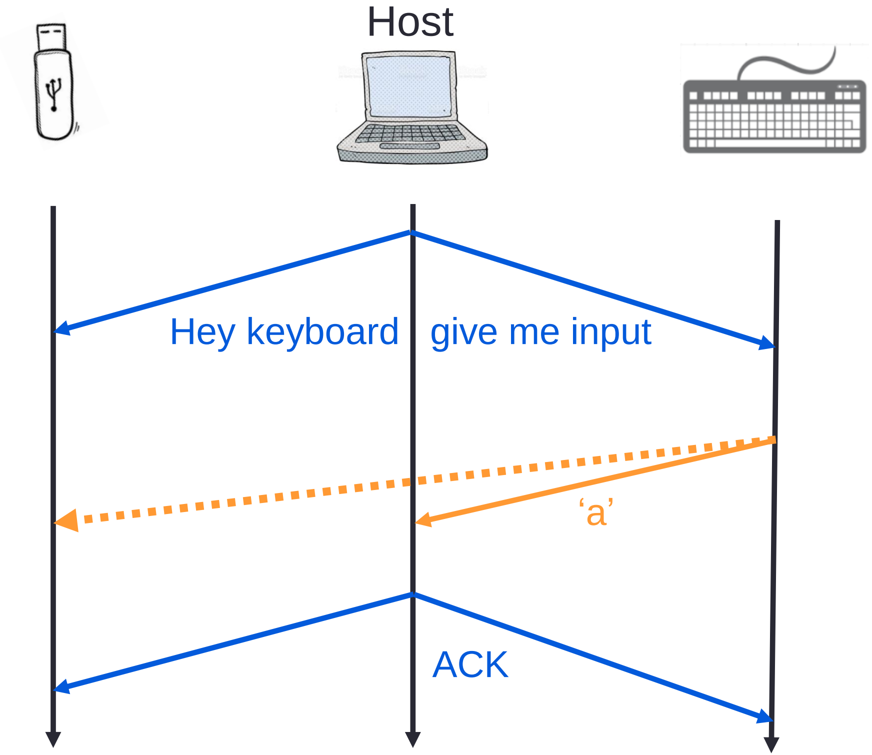




Confidentiality is completely compromised

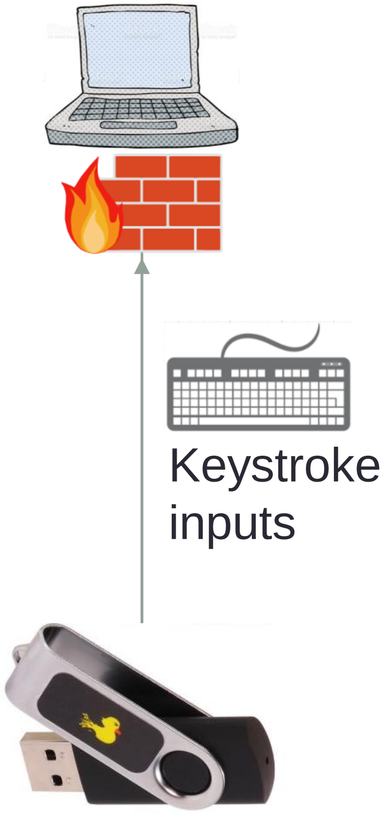
How about integrity & availability?





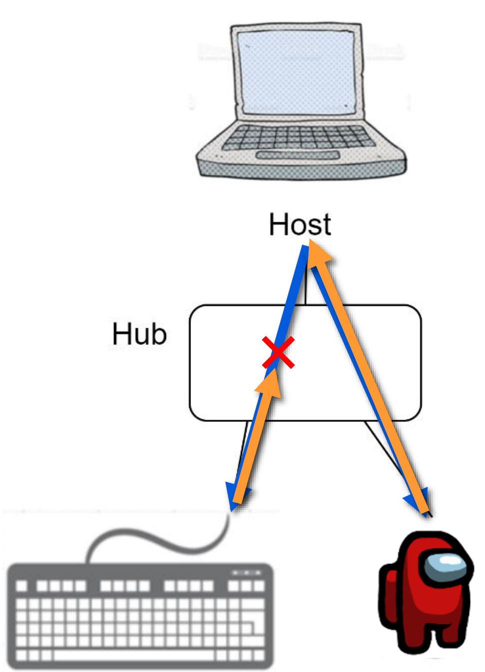
Masquerading Attacks

- Innocuous devices with unexpected function
- Typically emulating keyboards
- Leverages:
  - Default trust
  - User-understanding gap
- Firewall approach in software is the go-to defence

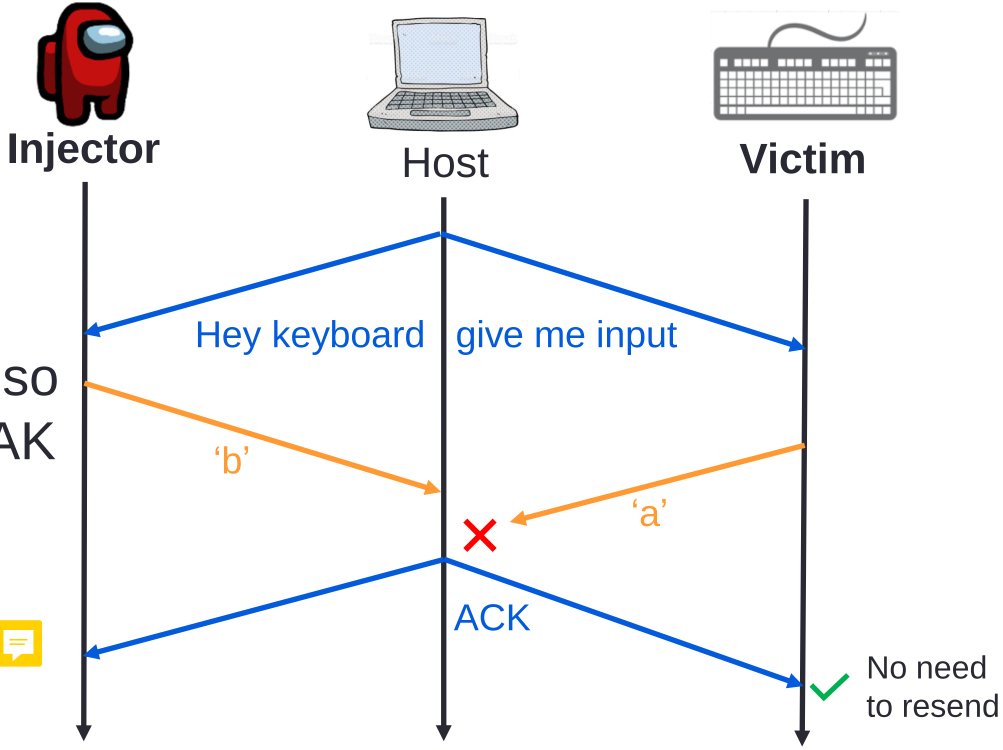


Off-Path Injection Attacks

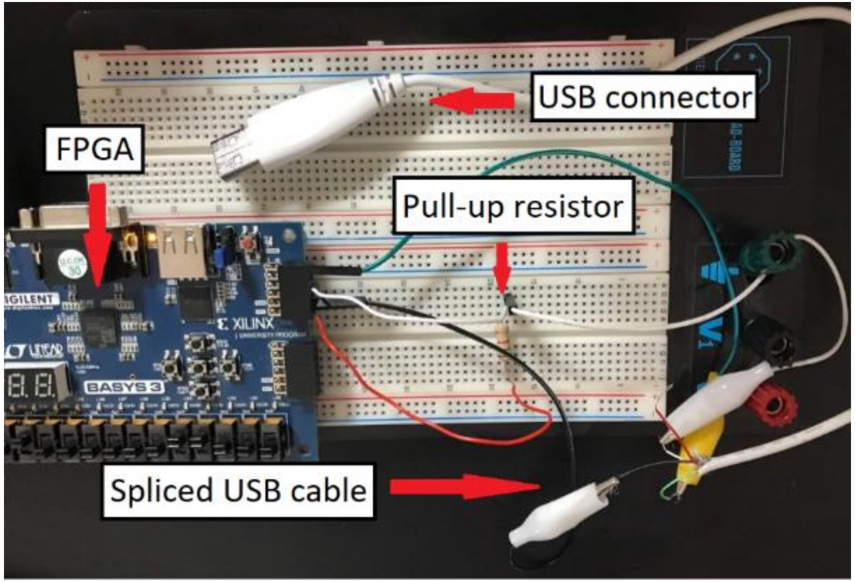
Integrity & Availability are also compromised



Can also just NAK



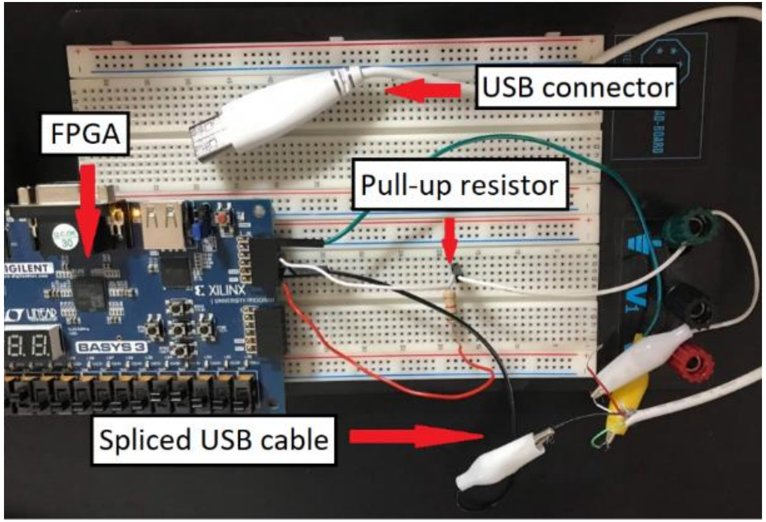
Keystroke Injection Demo !



Injection platforms

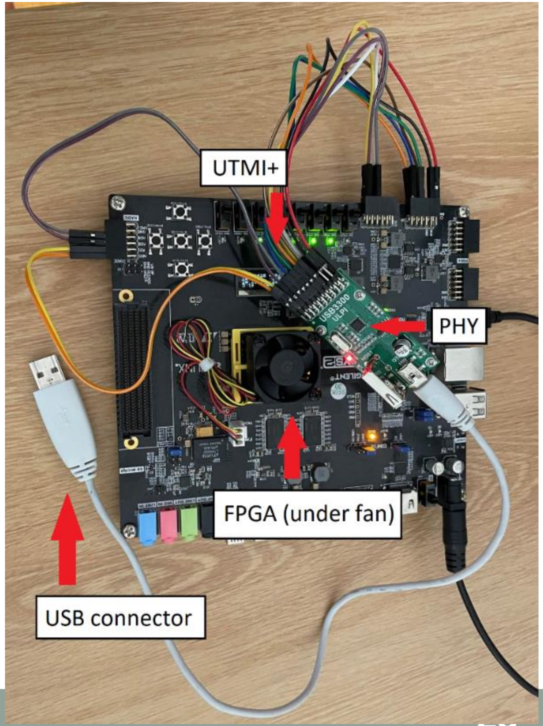
USB 1.1 (LS/FS)

Configured to connect as a mouse and inject keystroke traffic



USB 2.0 (HS)

Connects as serial device and injects mass-storage file contents, hijacking OS boot

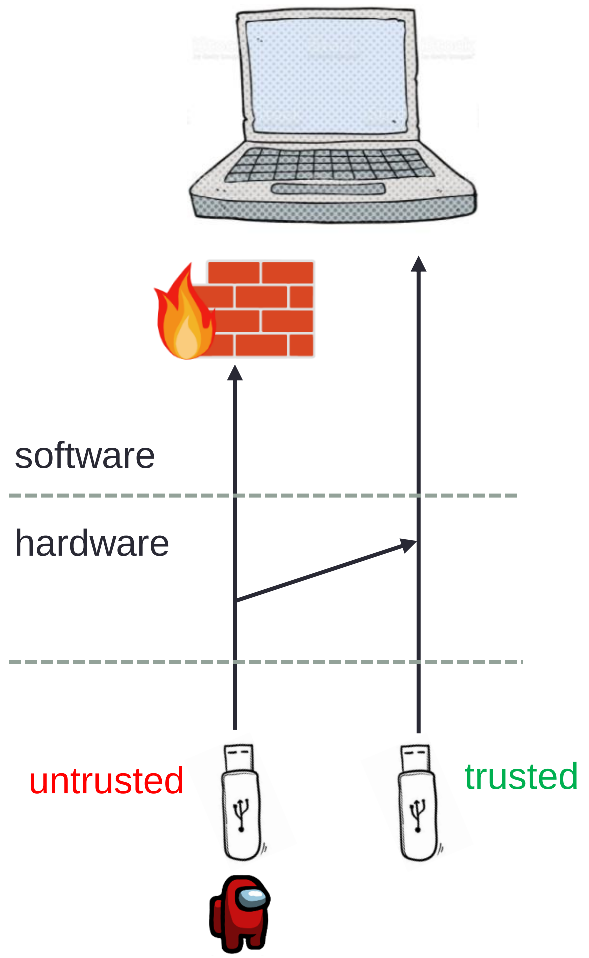


Impact

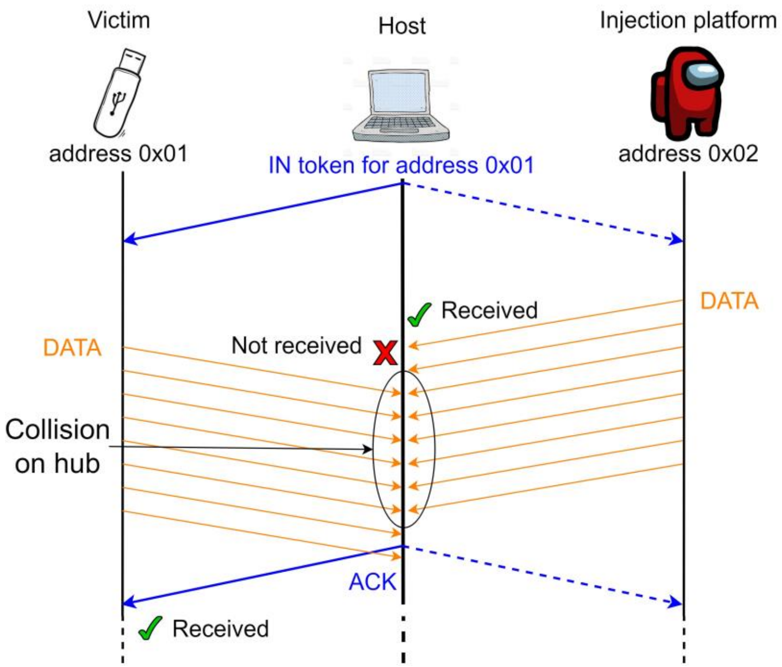
- Can exploit **any** USB interface trusted in software

<u>Policies bypassed:</u>

- `USBFilter` (*Sec. '16*)
- `GoodUSB` (*ACSAC '15*)
- USB-LOCK-RP By Advanced Systems International
- USBGuard
- Oracle VirtualBox



What really happens – collisions



USB standard allows two hub actions:

- Continue forwarding first incoming transmission (**vulnerable**)
- Stop any forwarding (**safe**)

Hubs found to be about 50/50

Countermeasures: use a safe hub

Other Physical Attacks

USB Killer

Cold boot attack

Device Profiling

By default, hosts identify devices by their self-reported *ProductID* and *VendorID* fields (sometimes *bcdDevice* too)

The USB ID Repository

The home of the `usb.ids` file

```
idVendor                 : 0x8087 (Intel)
idProduct                : 0x0AC9
bcdDevice                : 0x1729
```

Some works try to offer most robust methods:

Time-Print:

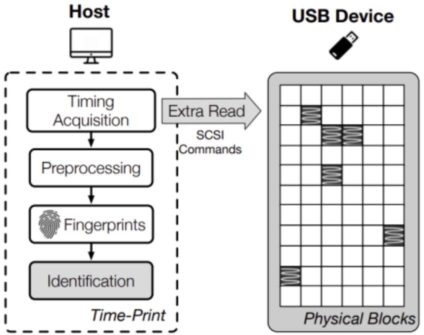
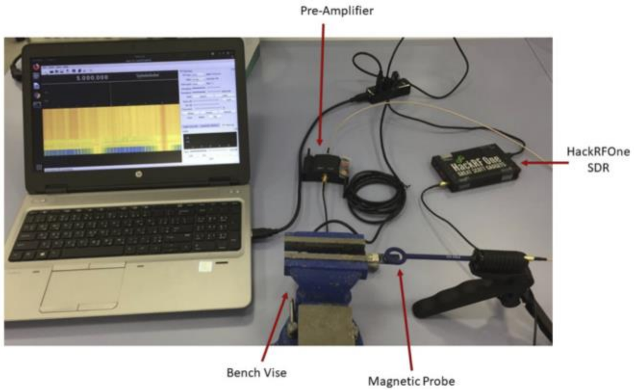


Fig. 3: The design of Time-Print.

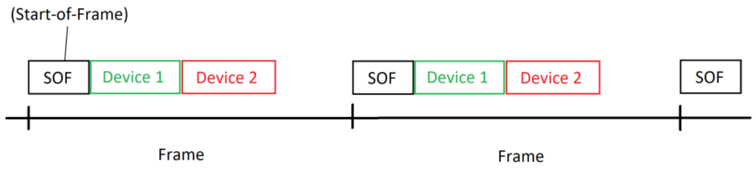
MAGNETO:

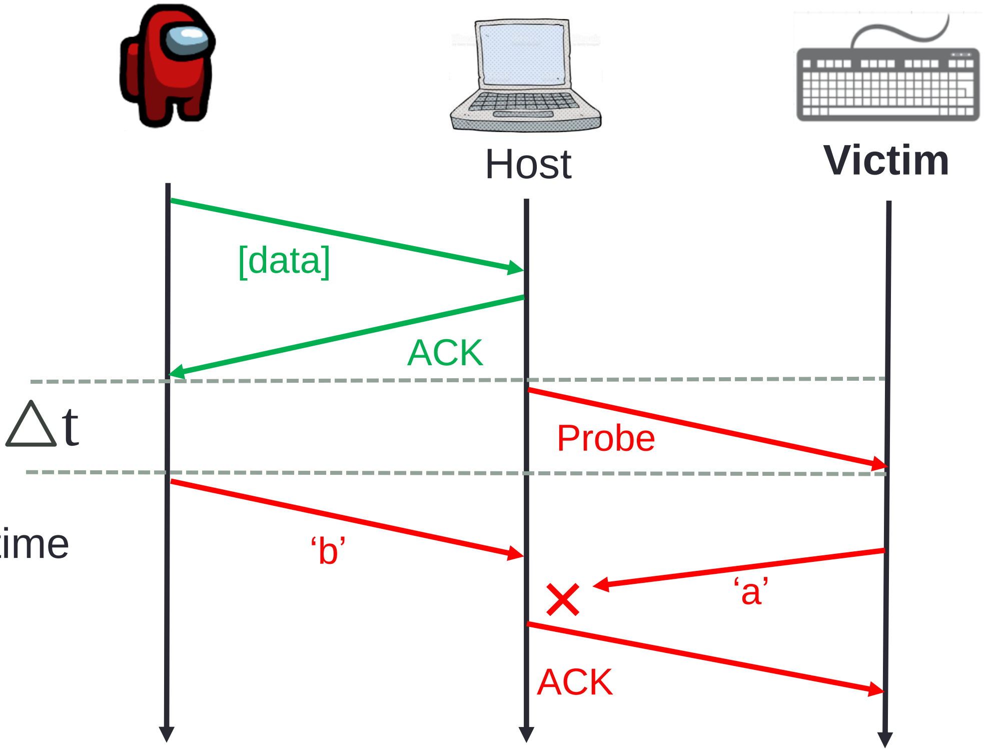


Future Work: Injection on USB 3.x

- Targeting USB 3.x
- USB 3 routes traffic, no broadcast – security by accident

Host transaction scheduling example:

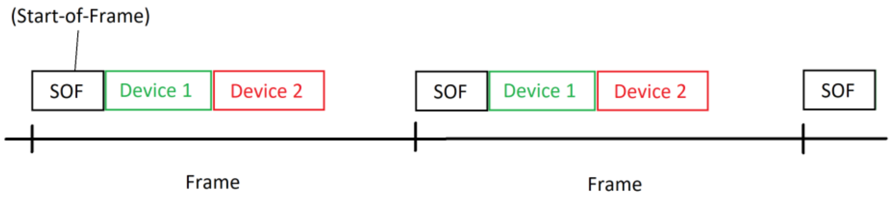


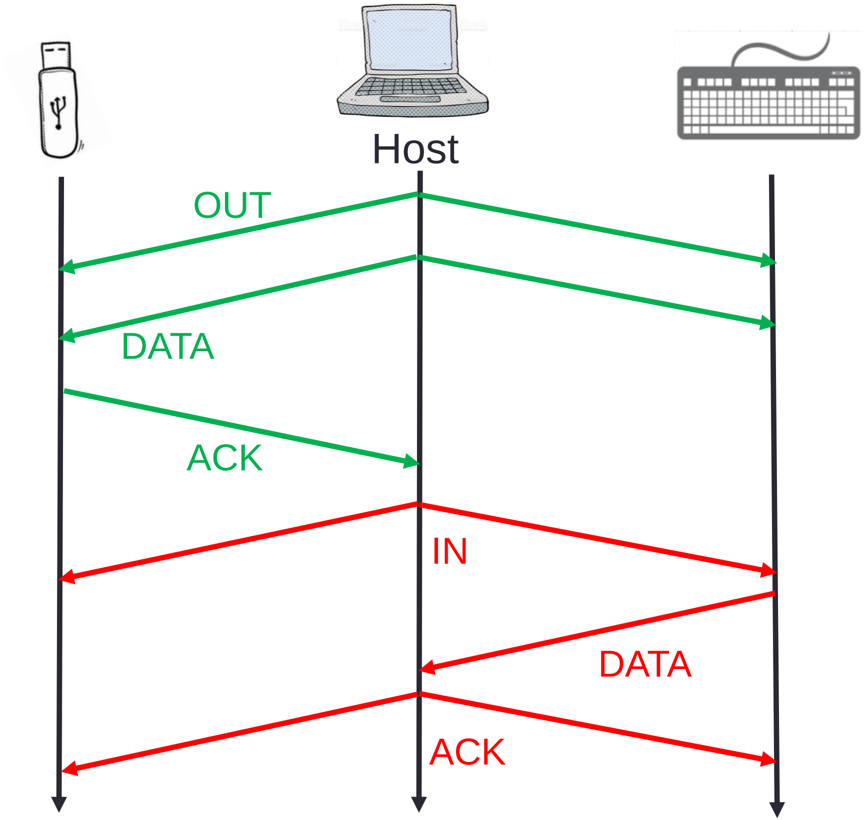


- Attacker can instead guess poll arrival time from deterministic scheduling
- Shown to work with USB 1.1 injector

Future Work: Timing side-channels

- Timing of packet arrivals & non-arrivals can tell us about the data itself
- Device 2 can monitor Device 1's data flow by measuring the time it is probed relative to the SOF
- Network traffic side-channel?





Future Work: Attacking FIDO

Server/Client (pt_c)		**Attack Library** (pt_{at})		**Token**
$ch \leftarrow \text{RP.register}(user)$				**Attacker Token** (ak_{at})
$auth_c \leftarrow \text{HMAC}(pt_c, ch)$	$\xrightarrow[auth_c]{ch}$	$auth_{at} \leftarrow \text{HMAC}(pt_{at}, ch)$	$\xrightarrow[auth_{at}]{ch}$	*token checks user presence*
$res_{at} \leftarrow \text{RP.validate}(user, sign_{at})$	$\xleftarrow{sign_{at}}$		$\xleftarrow{sign_{at}}$	$sign_{at} \xleftarrow{\$} \text{Sig.Sign}(ak_{at}, ch)$
				User Token (ak_t)
			$\xrightarrow[auth_c]{ch}$	*token checks user presence*
			$\xleftarrow{sign_c}$	$sign_c \xleftarrow{\$} \text{Sig.Sign}(ak_t, ch)$

Figure 4: Rogue key attack. Right arrows are authenticatorMakeCredential commands, and left arrows are the respective responses. RP.register and RP.validate denote requesting and validating a credential registration with the relying party. RP.register($user$) produces a challenge and RP.validate($user, sign$) produces a boolean result.

Using injection device connected alongside real 2FA token