CS 598 - Computer Security in the Physical World:

USB Security

Professor Adam Bates

Fall 2016

Course Updates

- Due today: Summaries for USBFILTER / GoodUSB
- Moving forward, summaries for papers are due midnight before class.
- First Project Deadline: Project Ideas, September 6th

Presenting a Paper

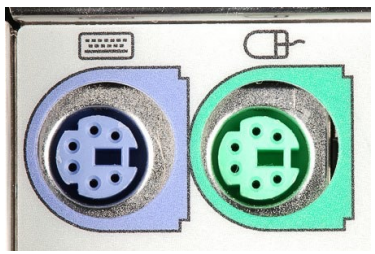
- Requires the technical preparation necessary for writing a summary, but also much more!
- Audience engagement is vital
    - Construct a narrative
    - Engage the audience
    - Identify an insight
    - Argue a point
    - Extend an argument
- Relate what you've learned, and what strikes you about the work: be engaged with the content

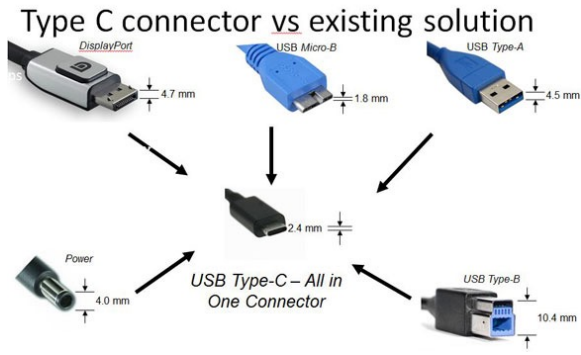
Presentation Advice

- Keep your points simple and repeat key insights
- Know the jargon that you will be using
- Present a narrative - tell a story
- Pace the talk so that you're not rushing or dragging
- Think about the goals of your presentation
  - Leave audience with the high points in their head
- Practice and prepare!
- Read http://pages.cs.wisc.edu/~markhill/conference-talk.html

Behold, USB….

- Universal Serial Bus
  - USB 1.0/2.0/3.0/3.1/Type-C
- Speed
  - 10 gigabits per second
- Ubiquitous





When USB goes bad…

When USB goes bad…

When USB goes bad…

BadUSB - On Accessories that Turn Evil by Karsten Nohl + Jakob Lell

McAfee®

Norton™ by Symantec

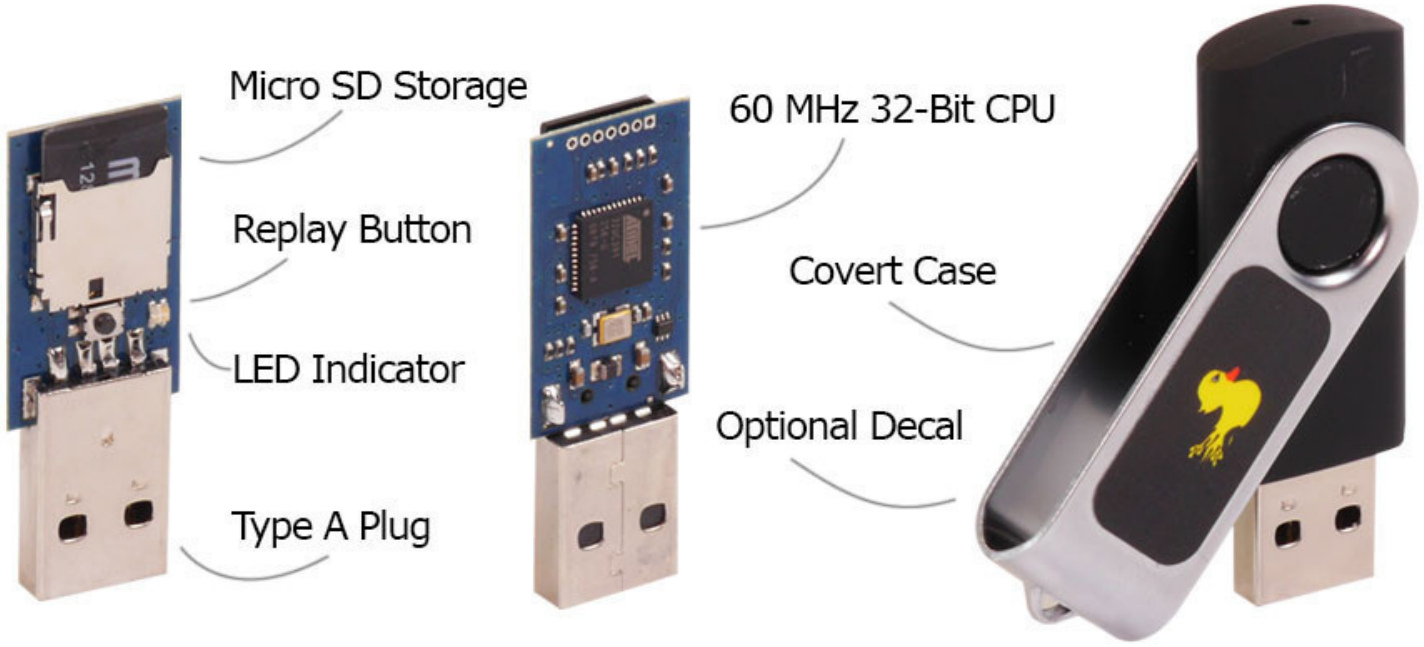
When USB goes bad…

BadUSB - On Accessories that Turn Evil by Karsten Nohl + Jakob Lell

When USB goes bad…

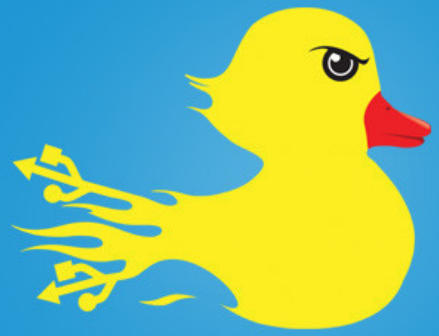
BadUSB - On Accessories that Turn Evil by Karsten Nohl + Jakob Lell

Penetration Tools



USB RUBBER DUCKY

THE MOST LETHAL DUCK EVER TO GRACE AN UNSUSPECTING USB PORT



Write

payloads with a **simple scripting language** or online payload generator including

- WiFi AP with disabled firewall
- Reverse Shell binary injection
- Powershell wget & execute
- Retrieve SAM and SYSTEM
- Create Wireless Association

Encode

the Ducky Script using the cross-platform open-source duck encoder, or download a pre-encoded binary from the online payload generator.

Carry multiple payloads, each on its own micro SD card.

Load

the micro SD card into the ducky then place inside the generic USB drive enclosure for covert deployment.

Deploy

the ducky on any target Windows, Mac and Linux machine and watch as your payload executes in mere seconds.

What went wrong?

USB Interfaces represent a set of unrestricted host permissions.

- OS knows *nothing* about the device
  - *but loads drivers to make the device happy anyway!*
- User knows *something* about the device…
  - *E.g., from the appearance of the device*
  - ***but no one is asking them about it!***

Today we're going to explore two papers that work within existing constraints to try and mitigate the threat of BadUSB…

1. GoodUSB
2. USBFILTER

GoodUSB

- Encodes user's expectations to authorize USB activity
  - *Let the user determine what the device should do*
- Tracks device's claimed identity
  - *Let the OS recognize if the device appears to have been plugged in before*
- Profiles suspicious devices in virtualized honeypot
  - *Let the user see what the device*

Design Challenges

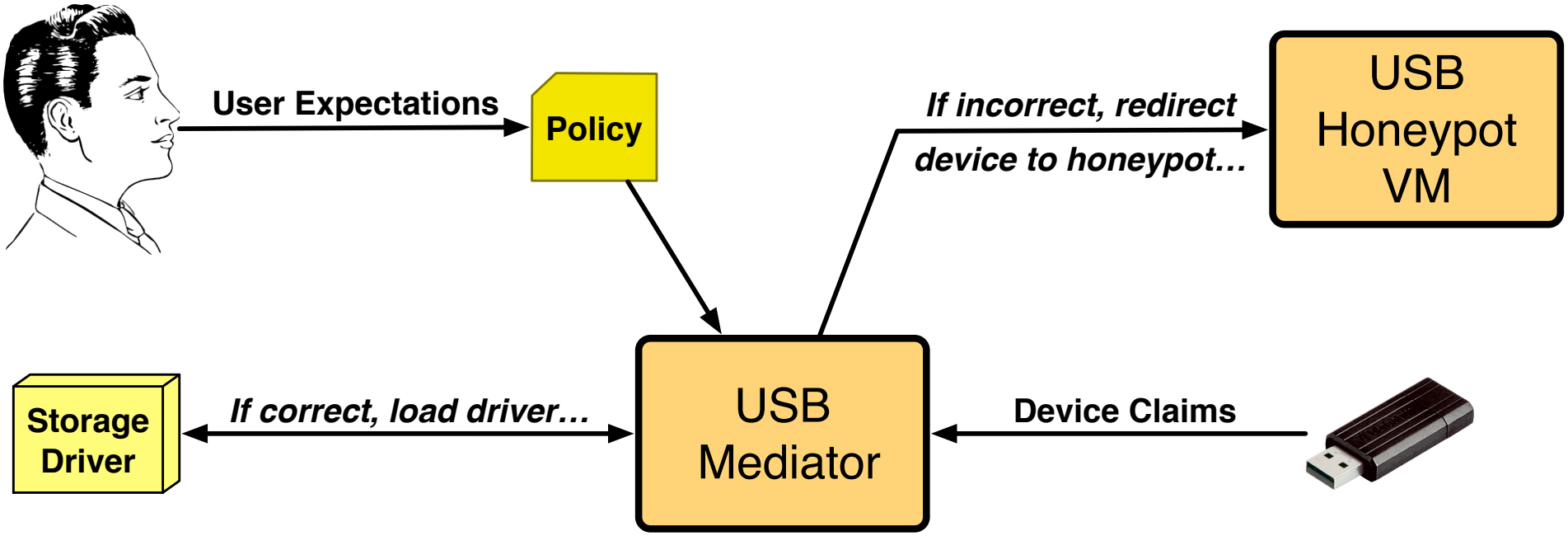
1. How can we mediate USB Enumerations?

2. How can we verify the identity of USB devices?

3. If a device is suspicious, what can we do about it?

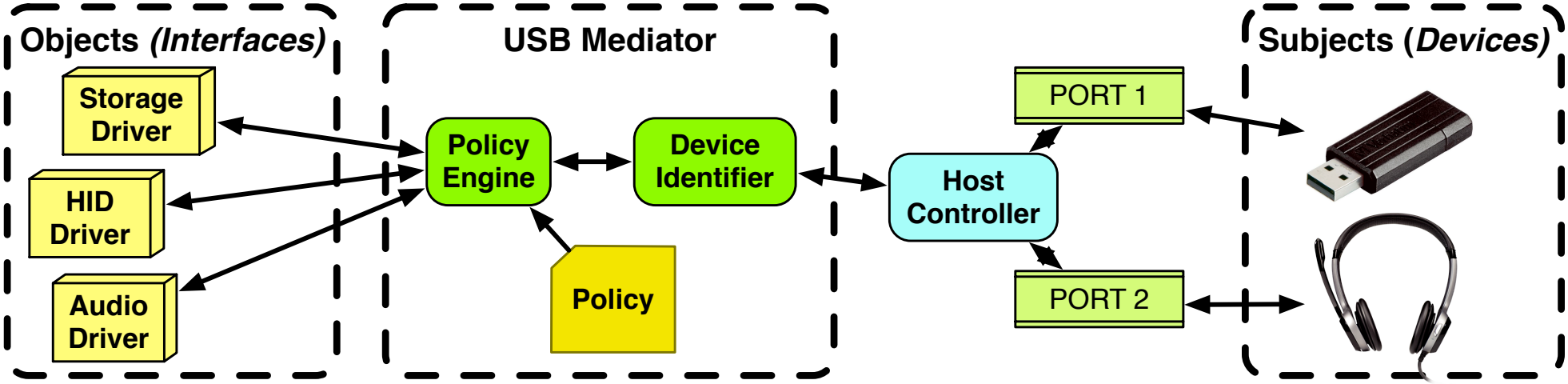
USB Mediator

- GoodUSB sits between the user and the device
  - Enforcing policies
  - Redirecting devices to honeypot

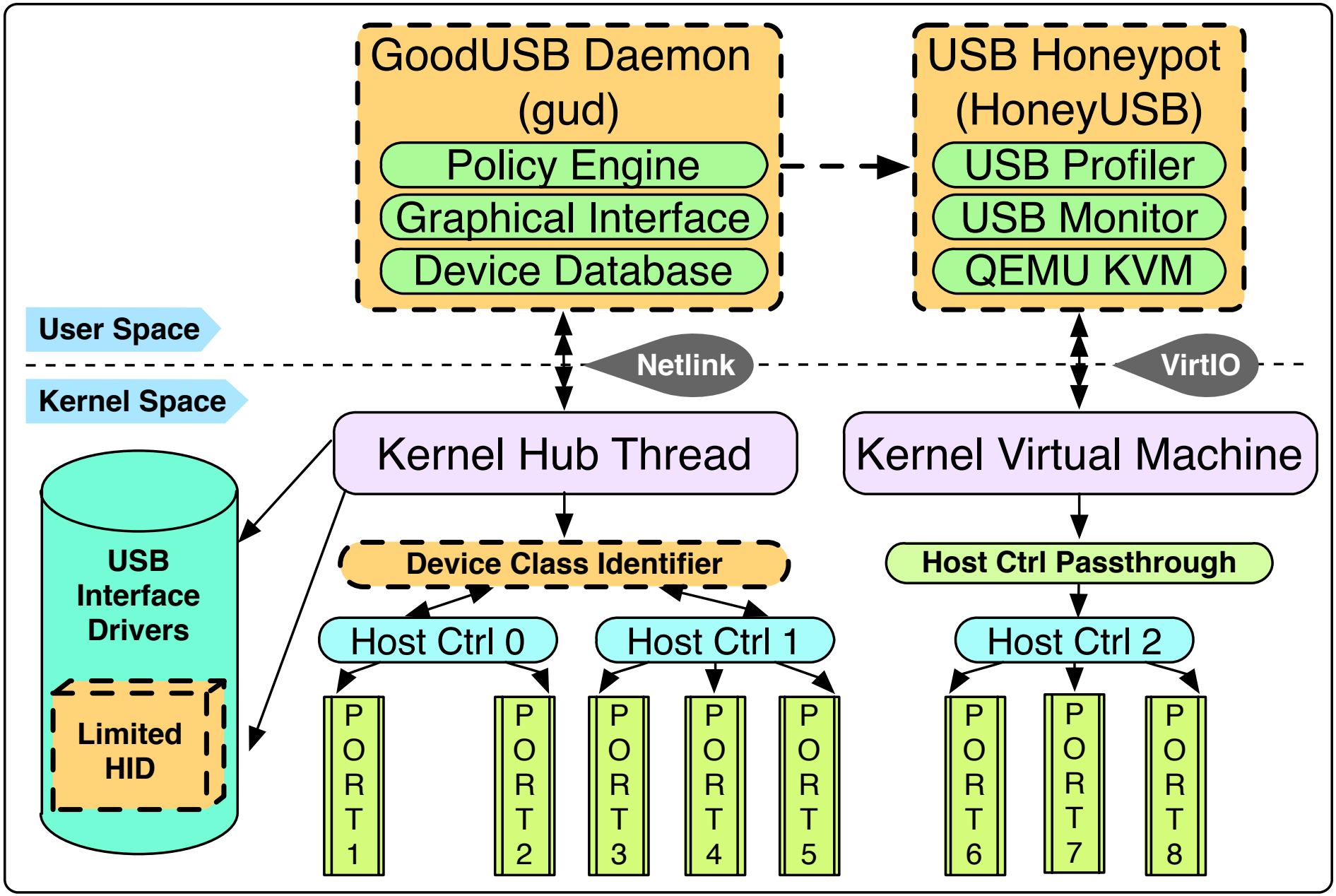


USB Mediator

- GoodUSB is implemented in the Linux kernel
  - Identifying the device firmware
  - Enforcing polices at the driver level



System Overview



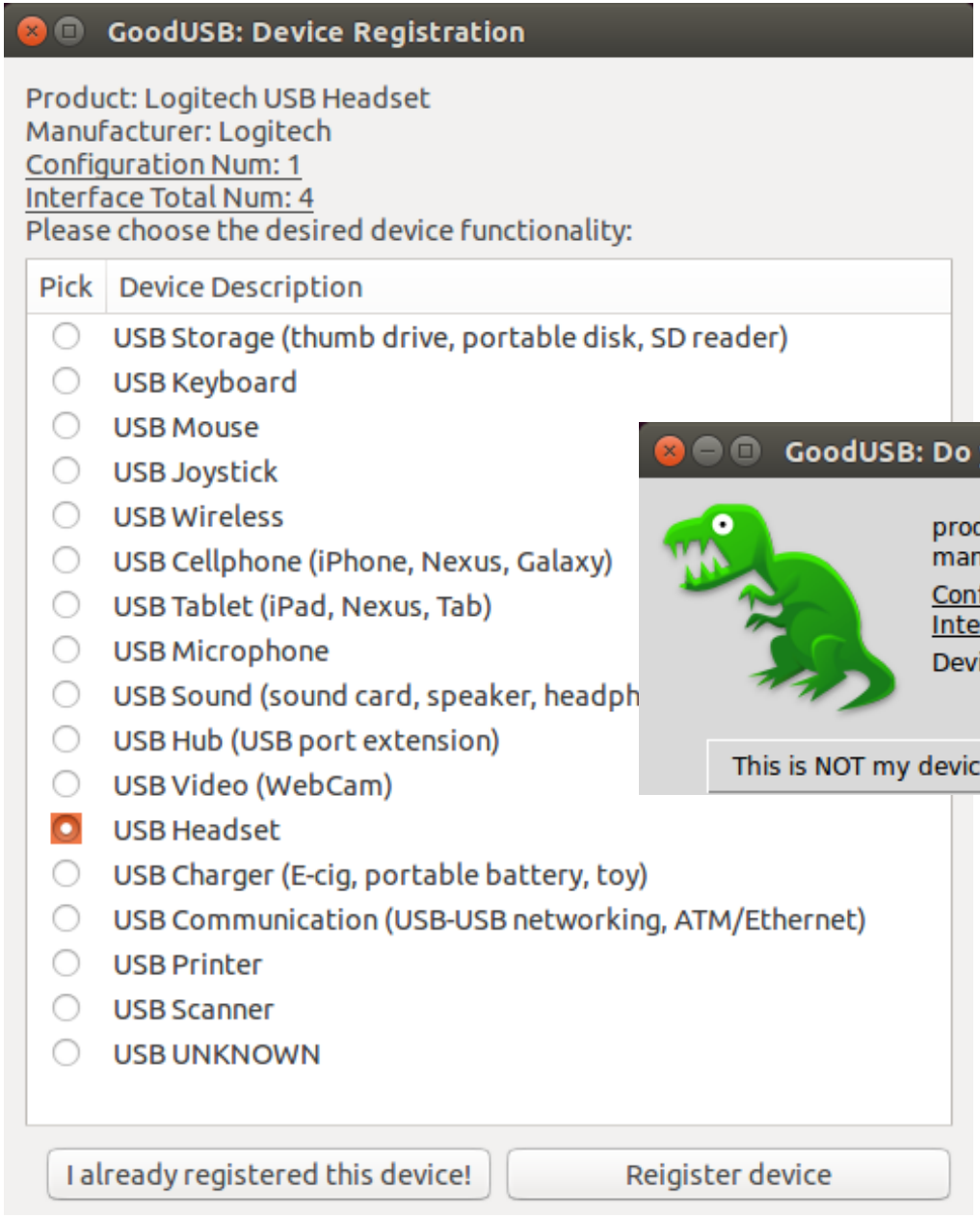
Kernel Enhancements

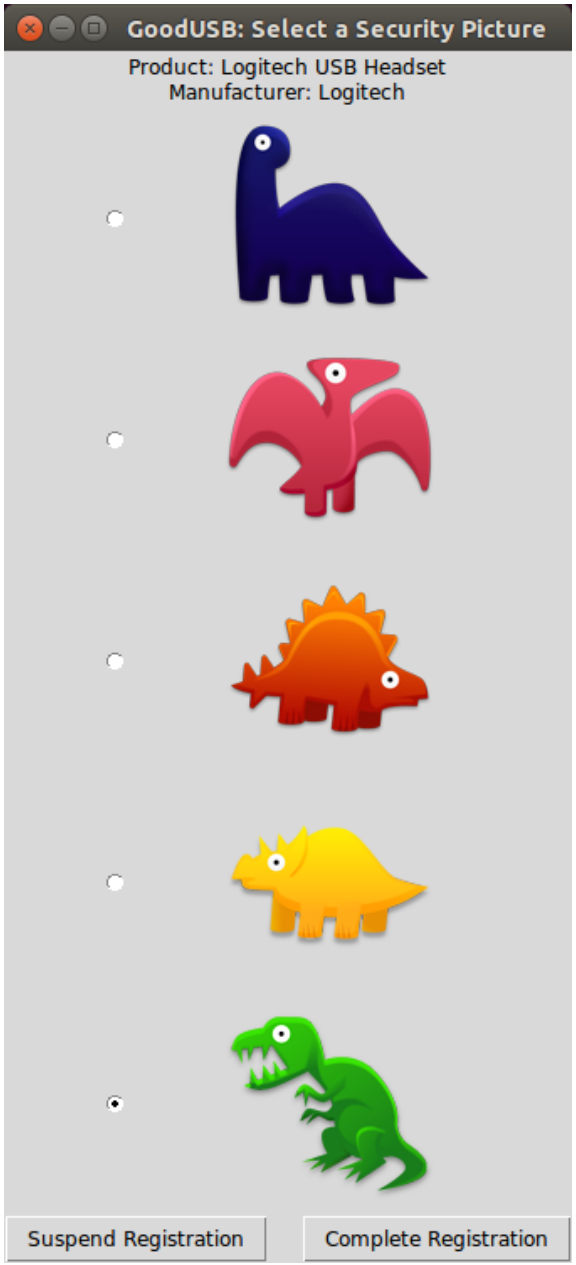
- Device Class Identifier
  - *SHA1 (USB descriptors)*
- Kernel Hub Thread Instrumentation
  - *Suspend the driver binding*
- Netlink Socket
  - *Communicate with the user-space*

Policy Mapping

Flash Drive	Storage	
Headset	Audio	HID
Smartphone	Storage	
Charger		

Identifying USB Devices





Profiling USB Devices

- QEMU KVM
  - *USB device pass-thru vs. USB host controller pass-thru*
- USB Monitor
  - *A udev rule to start USB device profiling*
- USB Profiler
  - *Generate a comprehensive USB device report for inspection*
  - *usbmon, lsusb, usbhid-dump, usb-devices, tcpdump*

Profiling USB Devices

```
usbpro HID analyzer started:
============================
_F2___x_t_erm_ENTER
_p_w_d_ENTER
_i_d_ENTER
_c_a_t_SPACE/etc/passwd_ENTER
_
============================
usbpro HID analyzer done
```

Evaluation

- USB Headset
- USB Rubber Ducky
- Teensy 3.1
- Smartphones

Evaluation

	Overhead in Microseconds
1st Enumeration	7 (5.2%)
Subsequent Enumerations	7 (5.0%)
Honeypot Redirection	262.1 (N/A)

Discussion Questions

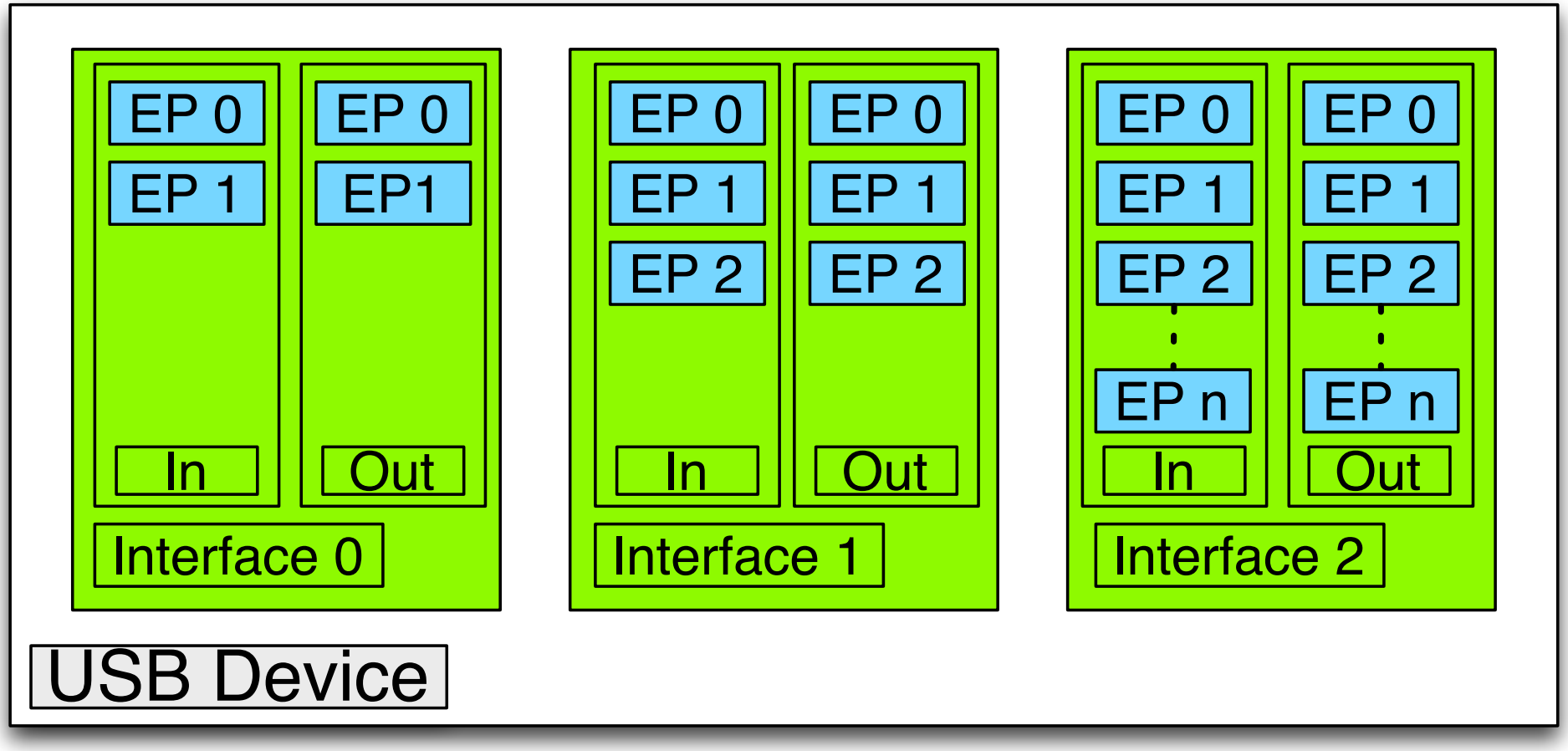
- What are the key contributions of the paper?
- What are the limitations to this approach?
- Why does GoodUSB ask for the user's opinion?
- How effectively was this idea developed?
- Criticisms / limitations of the paper?
- Do you buy this idea? Would this work well in a realistic workplace to protect against social engineers?

Rule #1 of Conference Presentations….

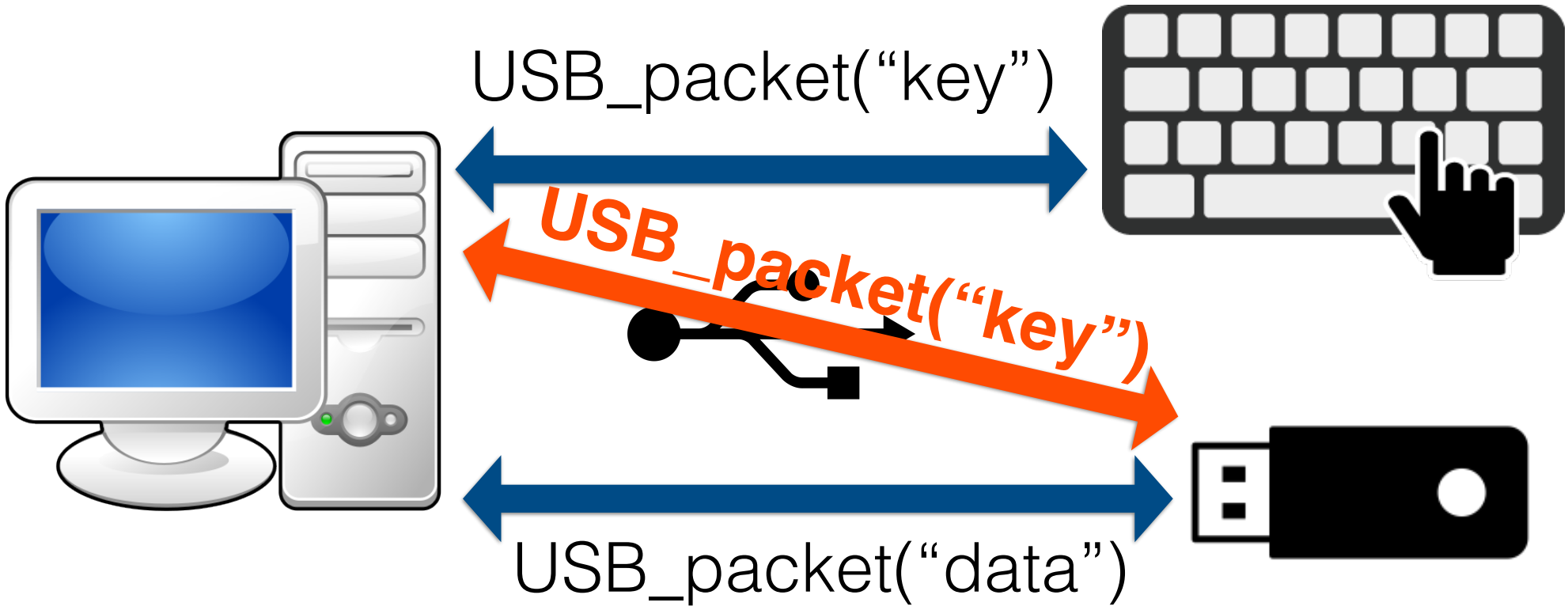
Rule #1 of Conference Presentations….

Never miss out on a good branding opportunity!

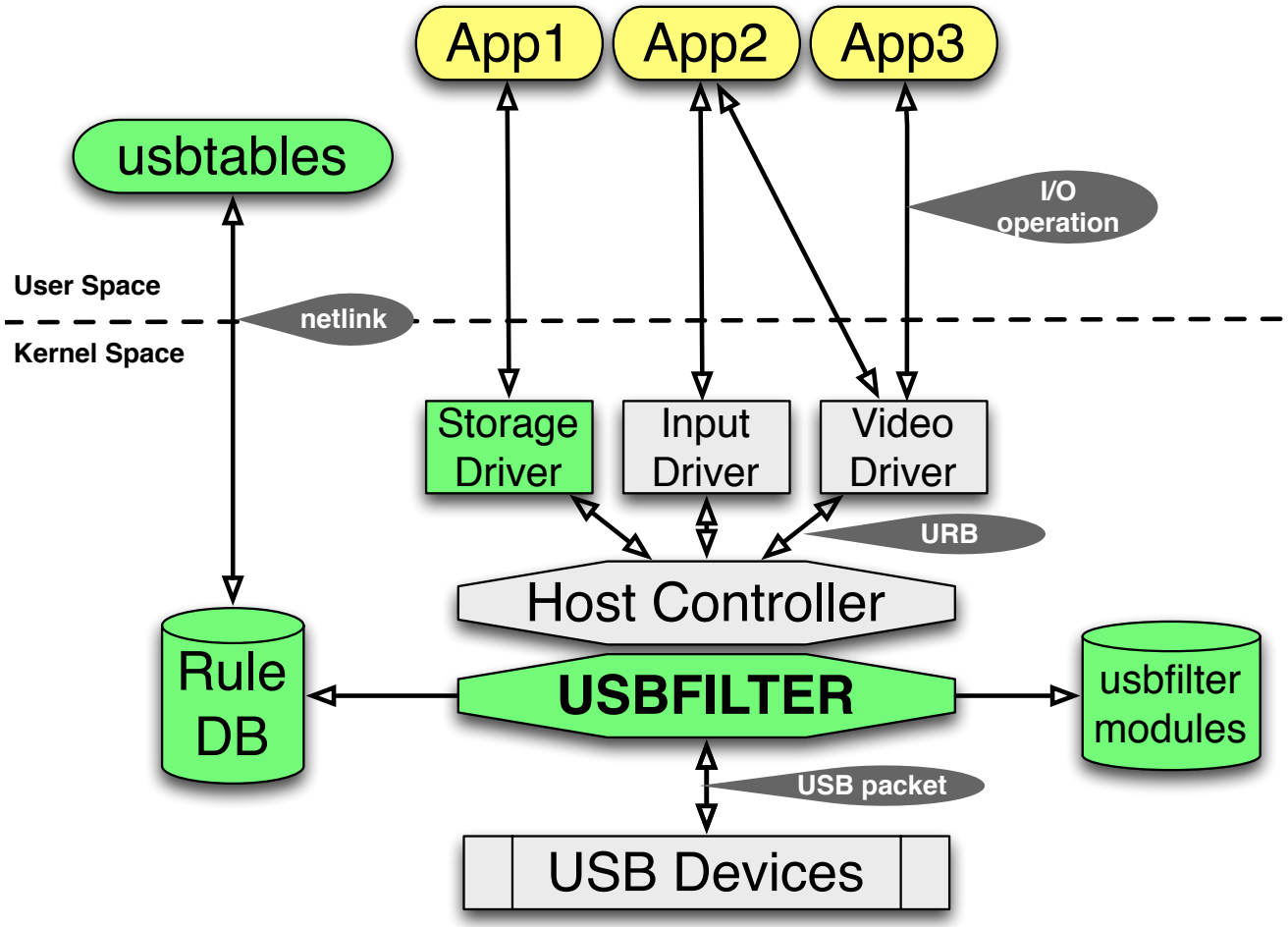
USB Device


EP 0
EP 1
In
EP 0
EP1
Out
Interface 0
EP 0
EP 1
EP 2
In
EP 0
EP 1
EP 2
Out
Interface 1
EP 0
EP 1
EP 2
EP n
In
EP 0
EP 1
EP 2
EP n
Out
Interface 2
USB Device

USB Packets



USBFILTER

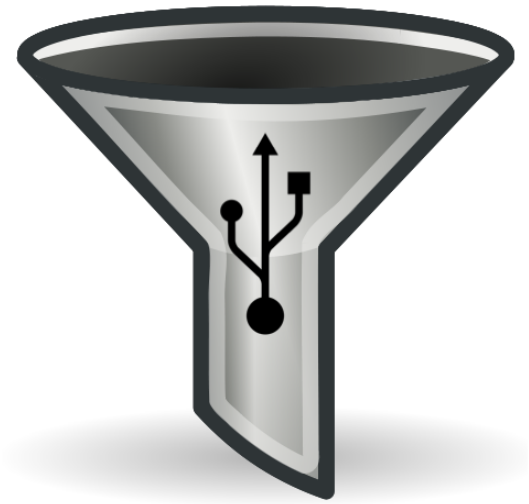

App1
App2
App3
usbtables
I/O operation
User Space
netlink
Kernel Space
Storage Driver
Input Driver
Video Driver
URB
Host Controller
Rule DB
USBFILTER
usbfilter modules
USB packet
USB Devices

Design Goals

- Complete mediation
- Tamperproof
- Verifiability

Reference Monitor

- Granularity
- Extensibility



Rule Constructions

Proces	pid,ppid,pgid,uid,euid,gid,egid,comm
Device	bus#,dev#,port#,if#,devpath,manufacturer,
Packet	type,direction,endpoint,address
LUM	name

Rule Consistency

- General conflict

$$general\_conflict(R_a, R_b) \leftarrow$$
$$\forall C_i \ni \mathscr{C} :$$
$$(\exists C_i^a \ni R_a \wedge \exists C_i^b \ni R_b \wedge value(C_i^a) \neq value(C_i^b)) \vee$$
$$(\exists C_i^a \ni R_a \wedge \nexists C_i^b \ni R_b) \vee$$
$$(\nexists C_i^a \ni R_a \wedge \nexists C_i^b \ni R_b).$$

- Weak conflict

$$weak\_conflict(R_a, R_b) \leftarrow$$
$$general\_conflict(R_a, R_b) \wedge action(R_a) = action(R_b).$$

- Strong conflict

$$strong\_conflict(R_a, R_b) \leftarrow$$
$$general\_conflict(R_a, R_b) \wedge action(R_a) \neq action(R_b).$$

Linux USBFILTER Modules (LUM)

- User-defined extension for USBFILTER
  - <linux/usbfilter.h>
- Rule construction unit
  - writing new rules with LUM
- Looking into the USB packet
  - SCSI commands, IP packets, HID packets, and etc.

LUM: Detect SCSI Write

```
20 int lbsw_filter_urb(struct urb *urb)
21 {
22     char opcode;
23
24     /* Has to be an OUT packet */
25     if (usb_pipein(urb->pipe))
26         return 0;
27
28     /* Make sure the packet is large enough */
29     if (urb->transfer_buffer_length <= LUM_SCSI_CMD_IDX)
30         return 0;
31
32     /* Make sure the packet is not empty */
33     if (!urb->transfer_buffer)
34         return 0;
35
36     /* Get the SCSI cmd opcode */
37     opcode = ((char *)urb->transfer_buffer)[LUM_SCSI_CMD_IDX];
38
39     /* Current only handle WRITE_10 for Kingston */
40     switch (opcode) {
41     case WRITE_10:
42         return 1;
43     default:
44         break;
45     }
46
47     return 0;
48 }
```

Overview

- USBFILTER - 27 kernel source files
  - 4 new files, 23 modified files
  - Across USB, SCSI, Block, and Networking subsystems
- USBTABLES
  - Internal Prolog engine
  - 21 rule constructions

Ex: Stop BadUSB Attacks

For my keyboard/mouse:

```
usbtables -a mymouse -v busnum=1,devnum=4,portnum=2,
      devpath=1.2,product="USB Optical Mouse",
      manufacturer=PixArt -k types=1 -t allow
usbtables -a mykeyboard -v busnum=1,devnum=3,
      portnum=1,devpath=1.1,
      product="Dell USB Entry Keyboard",
      manufacturer=DELL -k types=1 -t allow
usbtables -a noducky -k types=1 -t drop
```

Ex: Pin Webcam to Skype

For Logitech webcam C310:

```
usbtables -a skype -o uid=1001,comm=skype -v
      serial=B4482A20 -t allow
usbtables -a nowebcam -v serial=B4482A20 -t drop
```

Ex: Stop Data Exfil

For any USB storage devices:

```
usbtables -a nodataexfil4
        -l name=block_scsi_write -t drop
```

Ex: Disable Mic in SW

For Logitech USB headset:

```
usbtables -a logitech-headset -v ifnum=2,product=
      "Logitech USB Headset",manufacturer=Logitech -k
      direction=1 -t drop
```

Ex: Secure phone charge

For Nexus 4:

```
usbtables -a n4-charger -v product="Nexus 4" -t drop
```

For any phone:

```
usbtables -a charger -v busnum=1,portnum=4 -t drop
```

Scalability

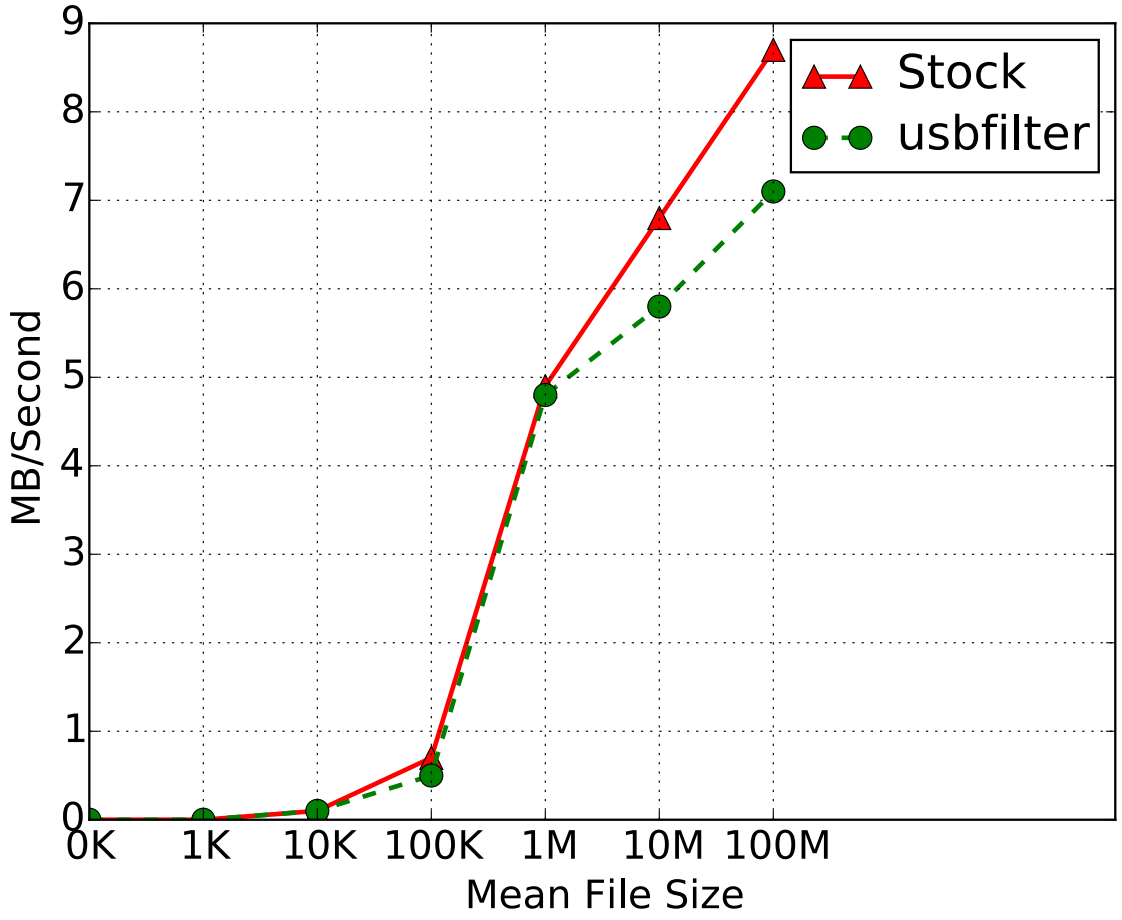
USBTABLES:

Adding a new rule	Avg (ms)
20 Base Rules	5.9
100 Base Rules	5.9

USBFILTER:

Packet filtering	Avg (µs)
20 Base Rules	2.6
100 Base Rules	9.7

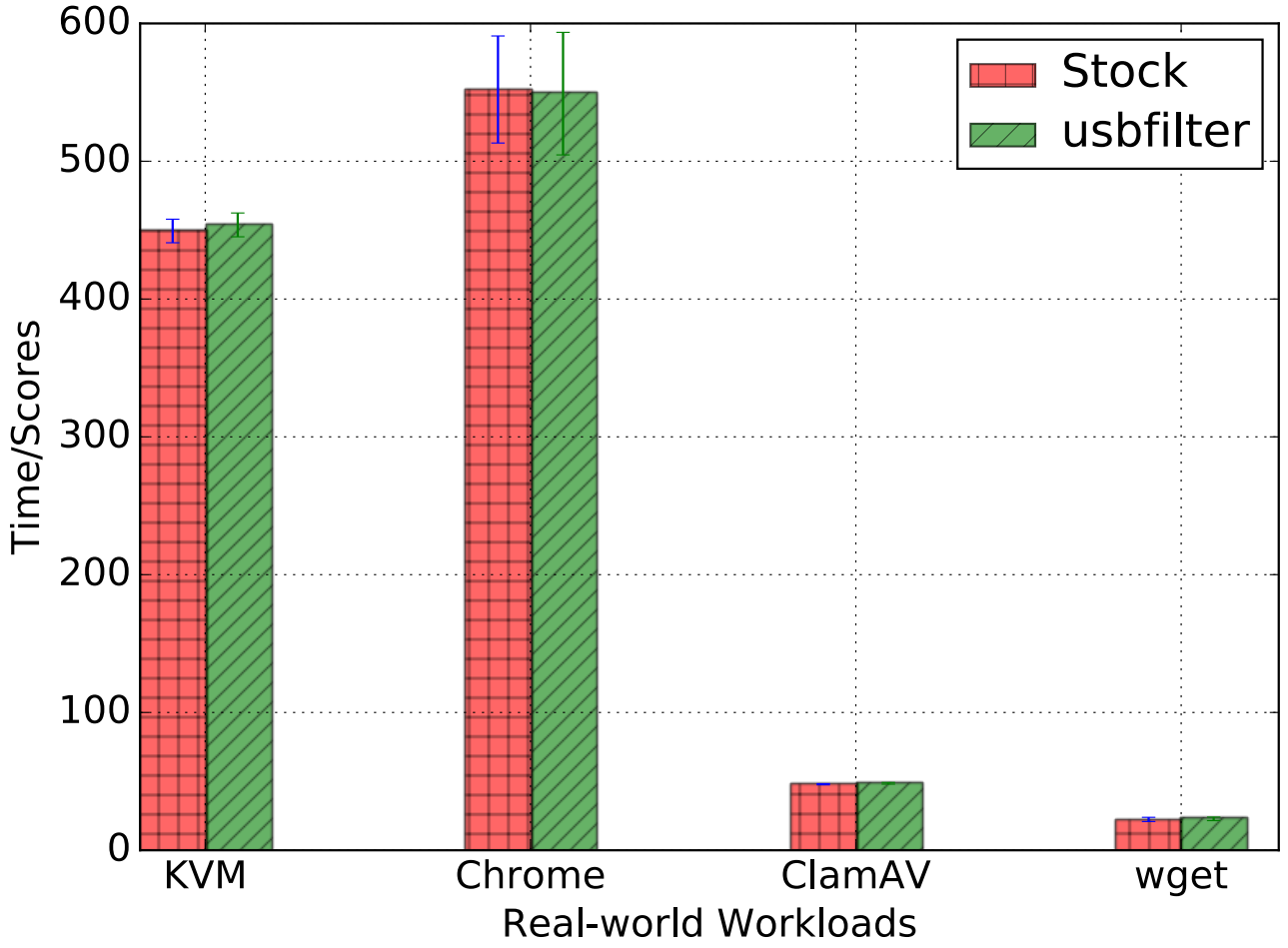
Throughput


MB/Second
9
8
7
6
5
4
3
2
1
0
0K
1K
10K
100K
1M
10M
100M
Mean File Size
Stock
usbfilter

Latency

Latency (µs)	1 KB	10 KB	100	1 MB	10 MB	100
Stock	97.6	98.1	99.2	105.5	741.7	5177.7
USBFILTER	97.7	98.2	99.6	106.3	851.5	6088.4
Overhead	0.1%	0.1%	0.4%	0.8%	14.8%	17.6%

Workload Performance



Discussion Questions

- What are the key contributions of the paper?
- What are the limitations to this approach?
- Criticisms / limitations of the paper?
- Do you buy this idea? Would this work well in a realistic workplace?
- Difference between ACSAC paper and USENIX Security paper?
- Funny paper titles — good or bad idea?