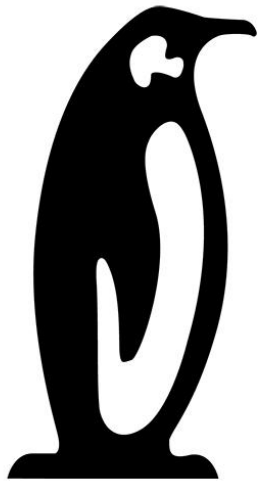


Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022

HID-BPF

Benjamin Tissoires
Red Hat

benjamin.tissoires@{redhat|gmail}.com

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

Foreword

- still a WIP, but getting closer (v10 is the latest, targetting v6.2)
- API mostly designed but still missing a few bits

HID-BPF == HID+BPF

Agenda

- HID
- ~~BPF~~
- HID-BPF: why?
- HID-BPF: what?
- HID-BPF: how?

HID, a Plug & Play protocol

Linux
Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

HID?

- Human Interface Devices
- Win 95 era protocol for handling plug and play USB devices (mice, keyboards)
  - now Bluetooth, BLE, I2C, Intel/AMD Sensors, (SPI in-progress)
- Most devices nowadays are working with generic drivers

HID?

- Human Interface Devices
- Win 95 era protocol for handling plug and play USB devices (mice, keyboards)
  - now Bluetooth, BLE, I2C, Intel/AMD Sensors, (SPI in-progress)
- Most devices nowadays are working with generic drivers

For that, they rely on HID report descriptors.

HID report descriptor

- describes the device protocol in a "simple" language (no loops, conditionals, etc...)
- static for each device (in flash)

```
 1   0x05, 0x01,       // Usage Page (Generic Desktop)
 2   0x09, 0x02,       // Usage (Mouse)
 3   0xa1, 0x01,       // Collection (Application)            <-- Application(Mouse)
 4   0x09, 0x01,       //  Usage (Pointer)
 5   0xa1, 0x00,       //  Collection (Physical)              <-- Physical(Pointer)
 6   0x05, 0x09,       //   Usage Page (Button)
 7   0x15, 0x00, 0x25, 0x01, 0x19, 0x01, 0x29, 0x05,  //   Logical Min/Max and Usage Min/Max
 8   0x75, 0x01,       //   Report Size (1)                    <- each usage is 1 bit
 9   0x95, 0x05,       //   Report Count (5)                   <- we got 5 of them
10   0x81, 0x02,       //   *Input* (Data,Var,Abs)      <--- 5 bits for 5 buttons
11   0x95, 0x03,       //   Report Count (3)
12   0x81, 0x01,       //   *Input* (Cnst,Arr,Abs)      <--- 3 bits of padding
13   0x05, 0x01,       //   Usage Page (Generic Desktop)
14   0x16, 0x01, 0x80, 0x26, 0xff, 0x7f, //   Logical Min/Max
15   0x09, 0x30,       //   Usage (X)
16   0x09, 0x31,       //   Usage (Y)
17   0x75, 0x10,       //   Report Size (16)
18   0x95, 0x02,       //   Report Count (2)
19   0x81, 0x06,       //   *Input* (Data,Var,Rel)      <--- X,Y of 16 bits
20   0x15, 0x81, 0x25, 0x7f,   //   Logical Min/Max (-127,127)
21   0x09, 0x38,       //   Usage (Wheel)
22   0x75, 0x08,       //   Report Size (8)
23   0x95, 0x01,       //   Report Count (1)
24   0x81, 0x06,       //   *Input* (Data,Var,Rel)      <--- Wheel of 8 bits
25   0x05, 0x0c,       //   Usage Page (Consumer Devices)
26   0x0a, 0x38, 0x02, //   Usage (AC Pan)
27   0x95, 0x01,       //   Report Count (1)
28   0x81, 0x06,       //   *Input* (Data,Var,Rel)      <--- AC Pan of 8 bits
```

Documentation

- Device Class Definition
- HID Usage Tables

Device Class Definition

https://www.usb.org/document-library/device-class-definition-hid-111

- last update: May 27, 2001
- there are the equivalent files for I2C, Bluetooth, BLE, SPI
- defines generic protocol that every HID device must speak
  - operational model
  - descriptors (USB + HID report descriptor)
  - parser of report descriptors
  - requests
  - report protocol

Device Class Definition

https://www.usb.org/document-library/device-class-definition-hid-111

- last update: May 27, 2001
- there are the equivalent files for I2C, Bluetooth, BLE, SPI
- defines generic protocol that every HID device must speak
  - operational model
  - descriptors (USB + HID report descriptor)
  - parser of report descriptors
  - requests
  - report protocol

The protocol is somewhat stable.

HID Usage Tables

https://www.usb.org/document-library/hid-usage-tables-13

- last update: March 1, 2022
- defines *meaning* of usages as defined in the report descriptor
  - X and Y are defined in the Generic Desktop page (0x01) as 0x30 and 0x31
- can be extended (and is) by companies
  - multitouch protocol
  - USI pens
  - HW sensors
- except for a few exceptions: an update means a new `#define` in the kernel if we care

HID

- Most devices nowadays are working with generic drivers

HID

- Most devices nowadays are working with generic drivers

Except for a few of them that need:

HID

- Most devices nowadays are working with generic drivers

Except for a few of them that need:

- a fixup in the report descriptor (45 drivers out of 82)
  - `hid-sigmamicro.c` in v5.17

HID

- Most devices nowadays are working with generic drivers

Except for a few of them that need:

- a fixup in the report descriptor (45 drivers out of 82)
  - `hid-sigmamicro.c` in v5.17
- 41 files are under 100 LoC (counted with cloc)

HID

- Most devices nowadays are working with generic drivers

Except for a few of them that need:

- a fixup in the report descriptor (45 drivers out of 82)
  - `hid-sigmamicro.c` in v5.17
- 41 files are under 100 LoC (counted with cloc)
- some driver just change the input mapping (i.e. to enable a given key)
  - `hid-razer` in v5.17

HID

- Most devices nowadays are working with generic drivers

Except for a few of them that need:

- a fixup in the report descriptor (45 drivers out of 82)
  - `hid-sigmamicro.c` in v5.17
- 41 files are under 100 LoC (counted with cloc)
- some driver just change the input mapping (i.e. to enable a given key)
  - `hid-razer` in v5.17

After attending a few Kernel Recipes editions in Paris: "Can eBPF help?"

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

HID+BPF

Use BPF in HID drivers to have user-space drivers fixes in the kernel

HID-BPF: base principles

- works only on **arrays of bytes** and talks HID
  - no access to input, or any other subsystems (LEDs, force feedback, ...)
- any *smart* processing needs to be done in userspace or at programming time:
  - parse HID report descriptor
  - compute location of various fields
- targets a specific device for a given program
- enforces GPL programs
  - simple fixes should be shipped in-tree

HID-BPF: why?

- more convenient to do simple fixes and user testing
- HID firewall
- change the device based on the user context
- tracing

HID-BPF: why?

- **more convenient to do simple fixes and user testing**
- HID firewall
- change the device based on the user context
- tracing

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development…

Linux
Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development…

- identification of the issue

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development...

- identification of the issue
- new patch created + tests

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development...

- identification of the issue
- new patch created + tests
- user needs to recompile the kernel

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development...

- identification of the issue
- new patch created + tests
- user needs to recompile the kernel
- submission on the LKML

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development...

- identification of the issue
- new patch created + tests
- user needs to recompile the kernel
- submission on the LKML
- review of the patch

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development...

- identification of the issue
- new patch created + tests
- user needs to recompile the kernel
- submission on the LKML
- review of the patch
- inclusion in branch

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development…

- identification of the issue
- new patch created + tests
- user needs to recompile the kernel
- submission on the LKML
- review of the patch
- inclusion in branch
- patch goes into Linus' tree

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development…

- identification of the issue
- new patch created + tests
- user needs to recompile the kernel
- submission on the LKML
- review of the patch
- inclusion in branch
- patch goes into Linus' tree
- kernel marked stable or patch backported in stable

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development…

- identification of the issue
- new patch created + tests
- user needs to recompile the kernel
- submission on the LKML
- review of the patch
- inclusion in branch
- patch goes into Linus' tree
- kernel marked stable or patch backported in stable
- distributions take the new kernel

HID: what it means to add a new quirk?

Spoiler alert: regular kernel development...

- identification of the issue
- new patch created + tests
- user needs to recompile the kernel
- submission on the LKML
- review of the patch
- inclusion in branch
- patch goes into Linus' tree
- kernel marked stable or patch backported in stable
- distributions take the new kernel
- user can drop the custom kernel build

HID: Adding a new quirk with BPF

- identification of the issue
- new ~~patch~~ *BPF program* created + tests
- user ~~needs to recompile the kernel~~ drops the bpf program into the filesystem

HID: Adding a new quirk with BPF

- identification of the issue
- new ~~patch~~ *BPF program* created + tests
- user ~~needs to recompile the kernel~~ drops the bpf program into the filesystem

```
1   SEC("fmod_ret/hid_bpf_rdesc_fixup")
2   int BPF_PROG(rdesc_fixup, struct hid_bpf_ctx *hid_ctx)
3   {
4         __u8 *data = hid_bpf_get_data(hid_ctx, 0, 4096 /* size */);
5
6         /* Convert Input item from Const into Var */
7         data[40] = 0x02;
8
9         return 0;
10  }
```

`data` contains the report descriptor of the device.

`hid_bpf_rdesc_fixup()` is executed once, once the device is exported to userspace.

HID: Adding a new quirk with BPF

- identification of the issue
- new ~~patch~~ *BPF program* created + tests
- user ~~needs to recompile the kernel~~ drops the bpf program into the filesystem

User implication stops here once the BPF program is accepted.

HID: Adding a new quirk with BPF

- identification of the issue
- new ~~patch~~ *BPF program* created + tests
- user ~~needs to recompile the kernel~~ drops the bpf program into the filesystem

User implication stops here once the BPF program is accepted.

Developers continue to *include and ship* the fix in the kernel

HID-BPF: why?

- more convenient to do simple fixes and user testing
- **HID firewall**
  - Steam opens up game controllers to the world (with `uaccess`)
  - SDL is happy with that
  - What prevents a Chrome plugin to initiate a controller firmware upgrade over the network?
- change the device based on the user context
- tracing

HID-BPF: why?

- more convenient to do simple fixes and user testing
- HID firewall
  - Steam opens up game controllers to the world (with `uaccess`)
  - SDL is happy with that
  - What prevents a Chrome plugin to initiate a controller firmware upgrade over the network?
- **change the device based on the user context**
  - Microsoft Surface Dial example
- tracing

HID-BPF: why?

- more convenient to do simple fixes and user testing
- HID firewall
  - Steam opens up game controllers to the world (with `uaccess`)
  - SDL is happy with that
  - What prevents a Chrome plugin to initiate a controller firmware upgrade over the network?
- change the device based on the user context
  - Microsoft Surface Dial example
- **tracing**
  - hidraw is good, but not enough
  - we can trace external requests with eBPF

HID-BPF: what?

HID-BPF: the net-like capability

Change the incoming data flow

BPF program, compiled by clang:

```
1   SEC("fmod_ret/hid_bpf_device_event")
2   int BPF_PROG(invert_x, struct hid_bpf_ctx *hid_ctx)
3   {
4         __s16 *x = (__s16*)hid_bpf_get_data(hid_ctx, 1 /* offset */, 2 /* size */);
5
6         /* invert X coordinate */
7         *x *= -1;
8
9         return 0;
10  }
```

Yes, this is a *tracing* BPF program.

Note: this is executed *before* `hidraw` or any driver processing.

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

HID-BPF: attach our program to a device

A program is attached to a `struct hid_device` in the kernel, by using the system unique id to attach to it (to be triggered by udev):

```
struct attach_prog_args {
      int prog_fd;
      unsigned int hid;
      unsigned int flags;
      int retval;
};

SEC("syscall")
int attach_prog(struct attach_prog_args *ctx)
{
      ctx->retval = hid_bpf_attach_prog(ctx->hid,
                                        ctx->prog_fd,
                                        ctx->flags);
      return 0;
}
```

```
sudo ./hid_mouse /sys/bus/hid/devices/0018:06CB:CD7A.000A
```

HID-BPF: Load more than 1 program for `device_event`

```c
SEC("fmod_ret/hid_bpf_device_event")
int BPF_PROG(invert_x, struct hid_bpf_ctx *hid_ctx)
{
	__s16 *x = (__s16*)hid_bpf_get_data(hid_ctx, 1 /* offset */, 2 /* size */);
	/* invert X coordinate */
	*x *= -1;
	return 0;
}

SEC("fmod_ret/hid_bpf_device_event")
int BPF_PROG(invert_y, struct hid_bpf_ctx *hid_ctx)
{
	__s16 *y = (__s16*)hid_bpf_get_data(hid_ctx, 3 /* offset */, 2 /* size */);
	/* invert Y coordinate */
	*y *= -1;
	return 0;
}
```

Ordering of execution is implementation detail right now.

HID-BPF: `device_event`

Benefits/Use cases:

- Filter out unwanted fields in a stream
  - neutral zone of a joystick
  - spurious button clicks on old mice
- Fix the report when something should not happen
- change the device language (in conjunction with `rdesc_fixup`)

HID-BPF: changing how the device looks and talks

```c
SEC("fmod_ret/hid_bpf_rdesc_fixup")
int BPF_PROG(rdesc_fixup, struct hid_bpf_ctx *hid_ctx)
{
	__u8 *data = hid_bpf_get_data(hid_ctx, 0, 4096 /* size */);

	/* invert X and Y definitions in the event stream interpretation */
	data[39] = 0x31;
	data[41] = 0x30;

	return 0;
}
```

`data` now contains the report descriptor of the device.

(Un)attaching this program triggers a disconnect/reconnect of the device.

Only 1 program of this type per HID device.

Linux Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

HID-BPF: `rdesc_fixup`

Benefits/Use cases:

- Fix a bogus report descriptor (key not properly mapped)
- Morph a device into something else (Surface Dial into a mouse)
- Change the device language (in conjunction with `device_event`)

HID-BPF: communicate with the device

```c
struct hid_send_haptics_args {
      /* data needs to come at offset 0 so we can use ctx as an argument */
      __u8 data[10];
      unsigned int hid;
};

SEC("syscall")
int send_haptic(struct hid_send_haptics_args *args)
{
        struct hid_bpf_ctx *ctx;
        int i, ret = 0;

        ctx = hid_bpf_allocate_context(args->hid);
        if (!ctx)
                return -1; /* EPERM check */

        ret = hid_bpf_hw_request(ctx, args->data, 10, HID_FEATURE_REPORT,
                                 HID_REQ_GET_REPORT);
        args->retval = ret;

        hid_bpf_release_context(ctx);

        return 0;
}
```

HID-BPF: communicate with the device

`hid_bpf_hw_request()`

Same behavior than the in-kernel function `hid_hw_raw_request()`.

Can not be used in interrupt context.

Allows:

- query device information
- put the device into a specific mode

HID-BPF: from a testing user perspective

current WIP at https://gitlab.freedesktop.org/bentiss/udev-hid-bpf

- daemon that waits for udev events
- on plug of a device, it loads `bBBBBgGGGGvVVVVpPPPPanything.bpf.o`
  - based on the modalias (bus/group/vid/pid)
  - if there is a `probe()` syscall in the bpf object:
    - runs it to check if the program applies to the device
- on un-plug: disconnects all known HID-BPF programs attached

Written in rust, so just a `cargo build` away.

HID-BPF: shipping in the kernel

Objective is to have the same sources of BPF programs than the userspace tools.

Still to be discussed on how they are shipped/built:

- automatically create one module per source file droppped into the tree (based on the modalias in the filename)
- ship the sources in the kernel tree, but provide builds in the firmware tree
- one gigantic module that contains all of the eBPF objects to be loaded then unloaded on device events
- something else?

HID-BPF: how?

Linux
Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

Architecture - 1/2

HID-BPF is built on top of BPF, but outside of it:

- relies on `ALLOW_ERROR_INJECTION` **API** to add tracepoints
  - Introduce a tracepoint in kernel code that can be tweaked by eBPF
  - Introduced by programmer at a given place in the code

Architecture - 2/2

HID-BPF is built on top of BPF, but outside of it:

- relies on the **kfunc API** for HID-BPF custom BPF API
  - export a kernel function as eBPF dynamic API
    - no need to update libbpf
  - care needs to be taken, but eBPF takes all of the cumbersome part away:
    - argument checking
    - availability of the call
    - *versioning*

BPF changes:

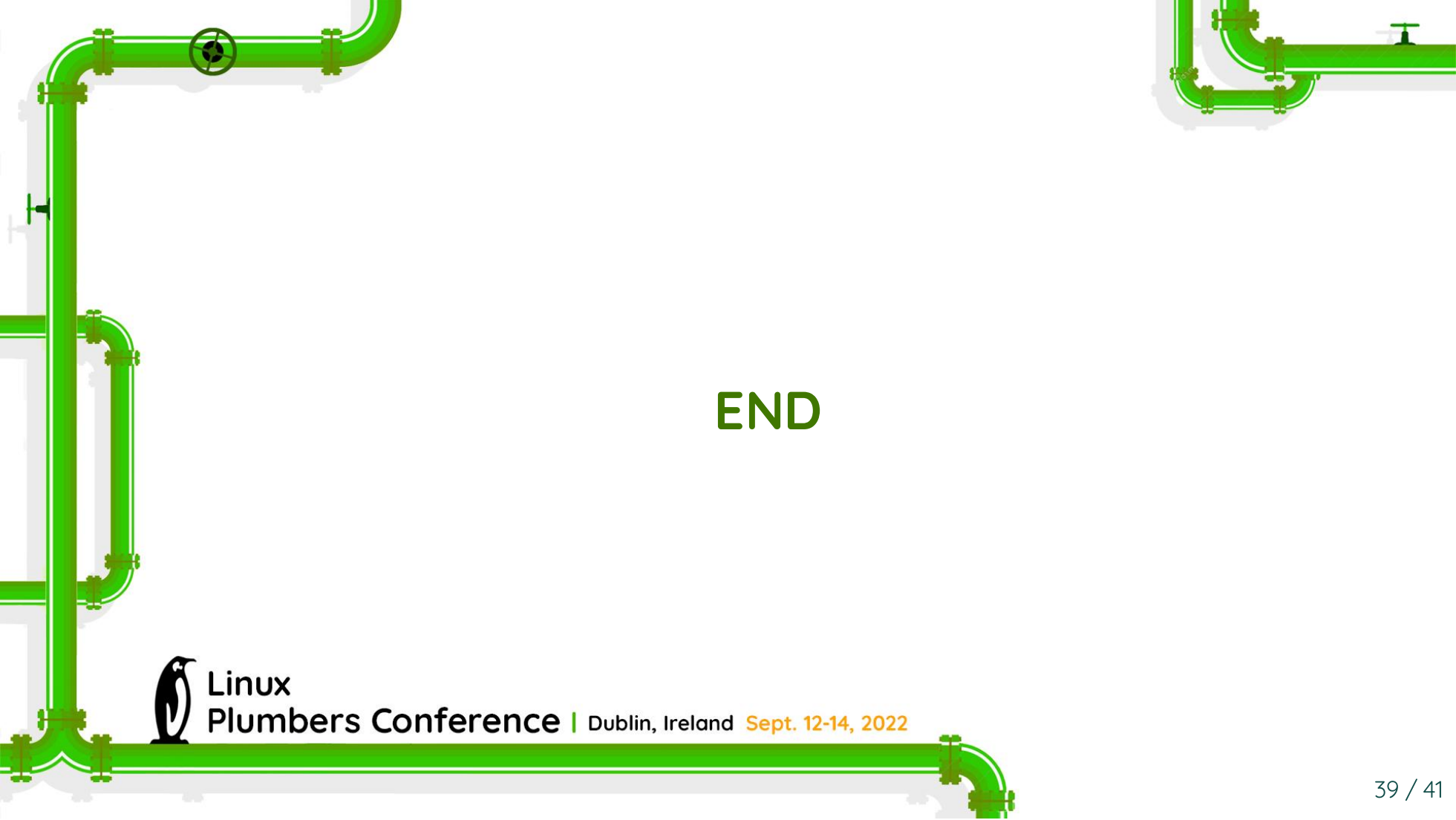
- custom implementation for attaching to a given HID device
  - handled through a preloaded eBPF program and custom maps handling
- BPF core changes:
  - Kfuncs for `SYSCALL`
  - more control of BPF maps from kernel
  - better access of ctx in `SYSCALL`
  - allow kfuncs to export a read/write or read only array of bytes

Wrap-up

Linux
Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

HID-BPF: Summary

- should simplify easy fixes in the future
- allow to add user-space defined behavior depending on the context
- can add traces in the events
- will allow to live-fix devices without having to update the kernel
- no more custom kernel API (sysfs, module parameters)
- will **not** replace in-kernel drivers for devices broken at boot time (keyboards) or for devices that need an actual driver (hid-rmi.ko)

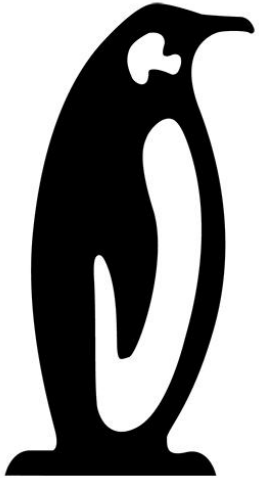


END

Linux
Plumbers Conference | Dublin, Ireland Sept. 12-14, 2022

HID-BPF: Summary

- should simplify easy fixes in the future
- allow to add user-space defined behavior depending on the context
- can add traces in the events
- will allow to live-fix devices without having to update the kernel
- no more custom kernel API (sysfs, module parameters)
- will **not** replace in-kernel drivers for devices broken at boot time (keyboards) or for devices that need an actual driver (hid-rmi.ko)



Linux Plumbers Conference

Dublin, Ireland September 12-14, 2022