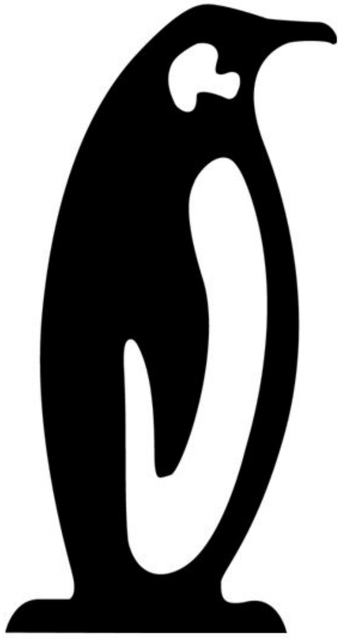


Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

HID-BPF in the kernel, 2 years later

Benjamin Tissoires & Peter Hutterer
bentiss@kernel.org
peter.hutterer@redhat.com

LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024

Introduction (sort of)

This is a follow up talk from LPC 2022, in Dublin

- Please refer to that previous talk for details about HID and why HID-BPF was required

This is a follow up talk from LPC 2022, in Dublin

- Please refer to that previous talk for details about HID and why HID-BPF was required
- It was a great talk, I assure you that

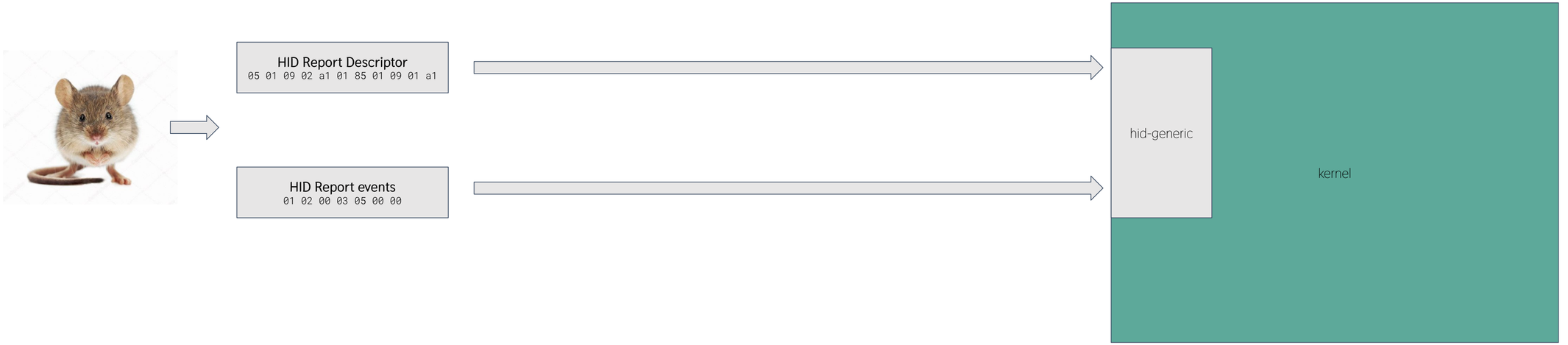
This is a follow up talk from LPC 2022, in Dublin

- Please refer to that previous talk for details about HID and why HID-BPF was required
- It was a great talk, I assure you that
- Really, I mean it

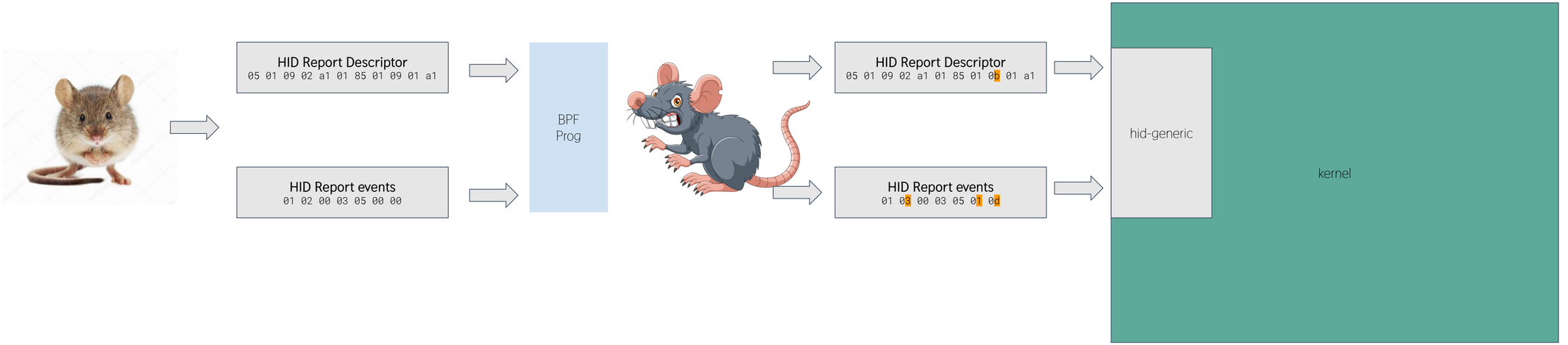
This is a follow up talk from LPC 2022, in Dublin

- Please refer to that previous talk for details about HID and why HID-BPF was required
- It was a great talk, I assure you that
- Really, I mean it
- But for those who were not there

HID-BPF overview



HID-BPF overview



HID-BPF - a minimal program

```
HID_BPF_CONFIG(
	HID_DEVICE(BUS_USB, HID_GROUP_GENERIC, VID_HOLTEK, PID_G10)
);


SEC(HID_BPF_RDESC_FIXUP)
int BPF_PROG(hid_fix_rdesc, struct hid_bpf_ctx *hctx)
{
	__u8 *data = hid_bpf_get_data(hctx, 0 /* offset */, 4096 /* size */);
	if (data)
		data[50] = 0x01;                                    // Change report descriptor
	return 0;
}

SEC(HID_BPF_DEVICE_EVENT)
int BPF_PROG(disable_button, struct hid_bpf_ctx *hctx)
{
	__u8 *data = hid_bpf_get_data(hctx, 0 /* offset */, 9 /* size */);
	if (data)
		data[1] &= (1 << 3);                                // Disable bit 3 in second byte of input report
	return 0;
}
```

So it got merged (story time)

- HID-BPF got merged in v6.3 (Sun Apr 23, 2023)
- Not much happened until November 2023...
- But then...

But then

https://www.davidrevoy.com/article995/how-a-kernel-update-broke-my-stylus-need-help

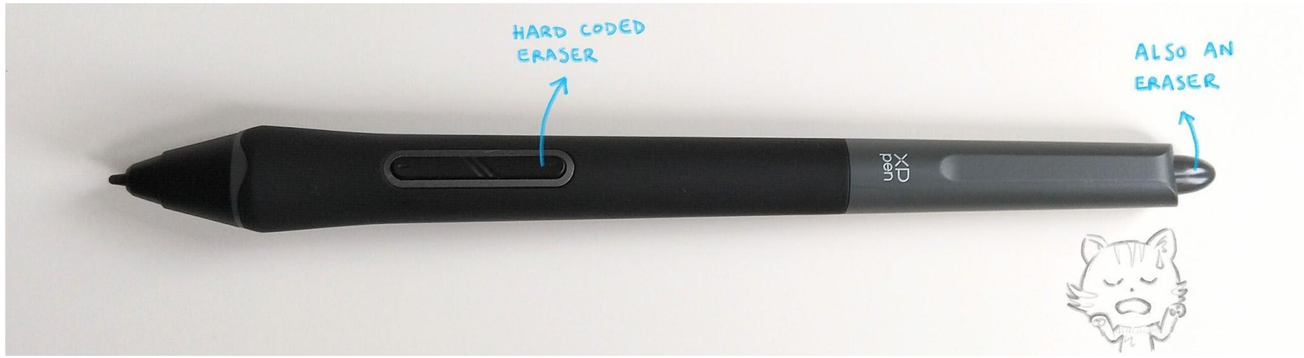
TL;DR:

- ~~Bad~~ Innovative design choice from Microsoft:

TL;DR:

- ~~Bad~~ Innovative design choice from Microsoft:

- Even ~~worse~~ innovative-ier design from XP-Pen:



Solved in... 2 days

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: 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

IT WORKS!

Shortly after

- Peter had to fix something by himself:

> "holy crap, this was the most enjoyable experience I ever had working with the kernel. hid-bpf is amazing, doubly so with udev-hid-bpf. amazing job you've done there"

This wasn't possible without all of the hard work from the BPF core team and all the users and developers of BPF.

Thank you!

Timeline

Since its first inclusion

HID-BPF Kernel Timeline

- v6.3: Started with tracing
- v6.10: bpf progs in the kernel
- v6.10: new kfuncs
- v6.11: struct_ops re-implementation
- v6.13: hid-generic control
- ...
- ~v6.14: trying to take over the world

v6.10: Current fixes are now integrated in the kernel

- located in **drivers/hid/bpf/progs**
- not built by default
- not loaded by the kernel
- loaded by **udev-hid-bpf** (we'll discuss that later)

The objective is to have an "upstream" for those and have "normal" development cycle

v6.10 & v6.11: Sleepable contexts

Why?

- In David's case, I papered over one sub-case where I would have need to take a decision later

How?

- v6.10: **bpf_wq** in the kernel
  - currently no delayed_wq, implemented through a **bpf_timer + bpf_wq**

- v6.11: Full rewrite of HID-BPF by using **BPF struct_ops**
  - truly amazing, simpler, and much more powerful
    - you get a small verifier in your own subsystem!
  - no more preloading of BPF program at boot (i.e. no overhead when not in used)

Results:

- Surface Dial fun: enable/disable haptic feedback with long press
- Logitech HID++ in bpf to easier solve some issues
  - (high res scrolling)

v6.10: New kfuncs!

- **hid_bpf_input_report** and **hid_bpf_try_input_report**
  - inject one input report from the current BPF execution
- **hid_bpf_hw_output_report**
  - call an output report on the device and wait for it to finish

Results:

- We now have roughly the same HID API from BPF than in the HID subsystem
- control even more devices!
  - X-Keys button pads
  - Logitech
  - etc...

v6.11: New hooks!

- **hid_hw_raw_request**
  - called when doing an ioctl on an hidraw device
- **hid_hw_output_report**
  - called when doing a write on an hidraw device

- for both of them:
  - "Source" argument and infinite loop prevention

Results:

- **HID Firewall** is now possible
  - HIDIOCREVOKE potentially gives any app a hidraw fd - what if that app decides to flash the firmware?

- Emulate HID LampArray on compatible devices
  - (my Corsair keyboard requires a key sniffer for that ATM)

v6.13: hid-generic assignment

- Some sort of control of the kernel
  - force hid-generic to prevent another driver to revert BPF changes

v6.13+: future objectives

- Also ensure that we can make use of HID-BPF for any request to the device:
  - whether it comes from userspace or from the kernel

udev-hid-bpf

aka “modprobe for HID BPF programs”

https://gitlab.freedesktop.org/libevdev/udev-hid-bpf/

udev-hid-bpf: what is it?

- udev-hid-bpf is a **collection** of HID-BPF programs
- udev-hid-bpf is a **binary executable** like modprobe/insmod
- udev-hid-bpf is **scaffolding** for udev rules/hwdb

udev-hid-bpf: the BPF programs

- udev-hid-bpf is a **collection** of HID-BPF programs - like drivers/hid/hid-*.c

```
$ tree src/bpf
src/bpf
├── stable
│   ├── 0010-FR-TEC__Raptor-Mach-2.bpf.c
│   ├── 0010-IOGEAR__Kaliber-MMOmentum.bpf.c        ← in upstream kernel
│   ├── 0010-XPPen__Artist24.bpf.c
│   └── …
├── testing
│   ├── 0010-Mistel__MD770.bpf.c
│   ├── 0010-Rapoo__M50-Plus-Silent.bpf.c           ← newly added
│   └── …
├── userhacks
│   ├── 0010-Logitech-MX-Master-3B-middle-button.bpf.c
│   ├── 0010-mouse_invert_y.bpf.c                   ← have fun...
│   ├── 0010-QuinHeng__PCsensor-FootSwitch.bpf.c
│   └── …
```

udev-hid-bpf: the executable

- udev-hid-bpf is a **generic** loader for HID-BPF programs - like modprobe is to kernel modules

```
udev-hid-bpf add /sys/bus/hid/devices/0003:0F00:0BAF.3 0010-XPPen__Artist24.bpf.o
```

udev-hid-bpf: the executable

- written in Rust, so statically compiled
- we can give users a tarball with udev-hid-bpf and the compiled bpf.o files and it'll just work

```
$ tar xz udev-hid-bpf-$version && cd udev-hid-bpf-*/
$ ./udev-hid-bpf install src/bpf/stable/*-Huion__Inspiroy-2-S.bpf.o
```

- plug in device, done

udev-hid-bpf: the executable

- No skeletons of BPF objects statically compiled into udev-hid-bpf

udev-hid-bpf: the scaffolding

- No skeletons of BPF objects statically compiled into udev-hid-bpf
- BPF objects include information about what devices they apply to
  - Use of new **SEC(".hid_bpf_config")** for storing the device IDs in the BTF (ala XDP, thanks Toke!)

```
HID_BPF_CONFIG(
    HID_DEVICE(BUS_USB, HID_GROUP_GENERIC, 0x256C /* huion */, 0x66 /* INSPIROY_2_S */),
    HID_DEVICE(BUS_USB, HID_GROUP_GENERIC, 0x256C /* huion */, 0x67 /* INSPIROY_2_M */),
);
```

udev-hid-bpf: the scaffolding

- No skeletons of BPF objects statically compiled in udev-hid-bpf
- BPF objects include information about what devices they apply to
  - We extract this to generate udev rules

```
IMPORT{builtin}="hwdb --subsystem=hid --lookup-prefix=hid-bpf:"
ACTION=="add", ENV{.HID_BPF}=="1", RUN{program}+="/usr/bin/udev-hid-bpf add $sys$devpath"
```

  - and a hwdb

```
hid-bpf:hid:b0003g0001v000004D9p00000339
 HID_BPF_T_000=0009-Mistel__MD770.bpf.o
 .HID_BPF=1
```

- Distributions need to only ship the udev rules/hwdb files

A few details

udev-hid-bpf: file-based versioning
AKA "how to handle breaking kernel API changes"

- Numbered prefix for sorting: `0010-foo.bpf.o, 0020-foo.bpf.o`
  - Versions change when kernel-incompatible changes are introduced into the BPF
- udev-hid-bpf loads in reverse order until it succeeds
  - The loaded program is the one that uses the most modern kernel features on the current kernel

udev-hid-bpf: passing udev properties as variables

- BPF program can request udev properties via special variables **UDEV_PROP_***

```
/* Filled in by udev-hid-bpf */
char UDEV_PROP_HUION_FIRMWARE_ID[64];
char UDEV_PROP_HUION_MAGIC_BYTES[64];

SEC("syscall")
int probe(struct hid_bpf_probe_args *ctx)
{
	int resolution = magic_bytes_to_u16(UDEV_PROP_HUION_MAGIC_BYTES + 3);
	…
}
```

udev-hid-bpf: pytest the BPF objects

- Once the BPF is in the kernel we can rely on custom selftests

- But in udev-hid-bpf, we can unit test them with pytest
  - Recompile each bpf as shared library with a small test wrapper
  - Throw some Python ctypes at it
  - Write some tests:

```
@pytest.mark.parametrize("y", [1, -1, 10, -256])
def test_event_userhacks_invert(y):
    bpf = Bpf.load("10-mouse_invert_y")

    # this device has reports of size 9
    values = (0, 0, 0, y, 0, 0, 0, 0, 0)
    report = struct.pack("<3bh5b", *values)

    values = bpf.hid_bpf_device_event(report=report)
    values = struct.unpack("<3bh5b", values)
    y_out = values[3]
    assert y_out == -y
```

udev-hid-bpf: where to find it?

- https://gitlab.freedesktop.org/libevdev/udev-hid-bpf/
- In Fedora 40+
- and RHEL 10+ (or whatever it is called)

LINUX
PLUMBERS
CONFERENCE Vienna, Austria / Sept. 18-20, 2024