



index : kernel/git/stable/linux.git

master

Linux kernel stable tree

Stable Group

[about](#) [summary](#) [refs](#) [log](#) [tree](#) [commit](#) [diff](#) [stats](#)

log msg

author Vikash Garodia <quic_vgarodia@quicinc.com> 2025-02-20 22:50:09 +0530
committer Greg Kroah-Hartman <gregkh@linuxfoundation.org> 2025-04-20 10:18:03 +0200
commit a736c72d476d1c7ca7be5018f2614ee61168ad01 (patch)
tree b7609c9cec867288104c0624cff7a5ffc122c83a
parent b2541e29d82da8a0df728aadec3e0a8db55d517b (diff)
download linux-a736c72d476d1c7ca7be5018f2614ee61168ad01.tar.gz

diff options

context: 3
space: include
mode: unified

media: venus: hfi_parser: refactor hfi packet parsing logic

commit 9edaaa8e3e15aab1ca413ab50556de1975bcb329 upstream.

words_count denotes the number of words in total payload, while data points to payload of various property within it. When words_count reaches last word, data can access memory beyond the total payload. This can lead to OOB access. With this patch, the utility api for handling individual properties now returns the size of data consumed. Accordingly remaining bytes are calculated before parsing the payload, thereby eliminates the OOB access possibilities.

Cc: stable@vger.kernel.org

Fixes: 1a73374a04e5 ("media: venus: hfi_parser: add common capability parser")

Signed-off-by: Vikash Garodia <quic_vgarodia@quicinc.com>

Reviewed-by: Bryan O'Donoghue <bryan.odonoghue@linaro.org>

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

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

Diffstat

```
-rw-r--r-- drivers/media/platform/qcom/venus/hfi_parser.c 98
```

1 files changed, 72 insertions, 26 deletions

diff --git a/drivers/media/platform/qcom/venus/hfi_parser.c b/drivers/media/platform/qcom/venus/hfi_parser.c
index 1425c69d900669..1b3db2caa99fe4 100644

--- a/drivers/media/platform/qcom/venus/hfi_parser.c

+++ b/drivers/media/platform/qcom/venus/hfi_parser.c

@@ -64,7 +64,7 @@ fill_buf_mode(struct hfi_plat_caps *cap, const void *data, unsigned int num)
cap->cap_bufs_mode_dynamic = true;

}

-static void

+static int

parse_alloc_mode(struct venus_core *core, u32 codecs, u32 domain, void *data)
{

struct hfi_buffer_alloc_mode_supported *mode = data;

@@ -72,7 +72,7 @@ parse_alloc_mode(struct venus_core *core, u32 codecs, u32 domain, void *data)
u32 *type;

if (num_entries > MAX_ALLOC_MODE_ENTRIES)

- return;

+ return -EINVAL;

type = mode->data;

@@ -84,6 +84,8 @@ parse_alloc_mode(struct venus_core *core, u32 codecs, u32 domain, void *data)

type++;

}

+

```

+         return sizeof(*mode);
    }

    static void fill_profile_level(struct hfi_plat_caps *cap, const void *data,
@@ -98,7 +100,7 @@ static void fill_profile_level(struct hfi_plat_caps *cap, const void *data,
        cap->num_pl += num;
    }

-static void
+static int
parse_profile_level(struct venus_core *core, u32 codecs, u32 domain, void *data)
{
    struct hfi_profile_level_supported *pl = data;
@@ -106,12 +108,14 @@ parse_profile_level(struct venus_core *core, u32 codecs, u32 domain, void *data)
    struct hfi_profile_level pl_arr[HFI_MAX_PROFILE_COUNT] = {};

    if (pl->profile_count > HFI_MAX_PROFILE_COUNT)
-        return;
+        return -EINVAL;

    memcpy(pl_arr, proflevel, pl->profile_count * sizeof(*proflevel));

    for_each_codec(core->caps, ARRAY_SIZE(core->caps), codecs, domain,
        fill_profile_level, pl_arr, pl->profile_count);
+
+    return pl->profile_count * sizeof(*proflevel) + sizeof(u32);
}

static void
@@ -126,7 +130,7 @@ fill_caps(struct hfi_plat_caps *cap, const void *data, unsigned int num)
    cap->num_caps += num;
}

-static void
+static int
parse_caps(struct venus_core *core, u32 codecs, u32 domain, void *data)
{
    struct hfi_capabilities *caps = data;
@@ -135,12 +139,14 @@ parse_caps(struct venus_core *core, u32 codecs, u32 domain, void *data)
    struct hfi_capability caps_arr[MAX_CAP_ENTRIES] = {};

    if (num_caps > MAX_CAP_ENTRIES)
-        return;
+        return -EINVAL;

    memcpy(caps_arr, cap, num_caps * sizeof(*cap));

    for_each_codec(core->caps, ARRAY_SIZE(core->caps), codecs, domain,
        fill_caps, caps_arr, num_caps);
+
+    return sizeof(*caps);
}

static void fill_raw_fmts(struct hfi_plat_caps *cap, const void *fmats,
@@ -155,7 +161,7 @@ static void fill_raw_fmts(struct hfi_plat_caps *cap, const void *fmats,
    cap->num_fmats += num_fmats;
}

-static void
+static int
parse_raw_formats(struct venus_core *core, u32 codecs, u32 domain, void *data)
{
    struct hfi_uncompressed_format_supported *fmt = data;
@@ -164,7 +170,8 @@ parse_raw_formats(struct venus_core *core, u32 codecs, u32 domain, void *data)
    struct raw_formats rawfmats[MAX_FMT_ENTRIES] = {};
    u32 entries = fmt->format_entries;
    unsigned int i = 0;
-    u32 num_planes;

```

```

+     u32 num_planes = 0;
+     u32 size;

    while (entries) {
        num_planes = pinfo->num_planes;
@@ -174,7 +181,7 @@ parse_raw_formats(struct venus_core *core, u32 codecs, u32 domain, void *data)
        i++;

        if (i >= MAX_FMT_ENTRIES)
-            return;
+            return -EINVAL;

        if (pinfo->num_planes > MAX_PLANES)
            break;
@@ -186,9 +193,13 @@ parse_raw_formats(struct venus_core *core, u32 codecs, u32 domain, void *data)

        for_each_codec(core->caps, ARRAY_SIZE(core->caps), codecs, domain,
                        fill_raw_fmts, raw_fmts, i);
+     size = fmt->format_entries * (sizeof(*constr) * num_planes + 2 * sizeof(u32))
+         + 2 * sizeof(u32);
+
+     return size;
    }

-static void parse_codecs(struct venus_core *core, void *data)
+static int parse_codecs(struct venus_core *core, void *data)
    {
        struct hfi_codec_supported *codecs = data;

@@ -200,21 +211,27 @@ static void parse_codecs(struct venus_core *core, void *data)
        core->dec_codecs &= ~HFI_VIDEO_CODEC_SPARK;
        core->enc_codecs &= ~HFI_VIDEO_CODEC_HEVC;
    }

+
+     return sizeof(*codecs);
+ }

-static void parse_max_sessions(struct venus_core *core, const void *data)
+static int parse_max_sessions(struct venus_core *core, const void *data)
    {
        const struct hfi_max_sessions_supported *sessions = data;

        core->max_sessions_supported = sessions->max_sessions;
+
+     return sizeof(*sessions);
    }

-static void parse_codecs_mask(u32 *codecs, u32 *domain, void *data)
+static int parse_codecs_mask(u32 *codecs, u32 *domain, void *data)
    {
        struct hfi_codec_mask_supported *mask = data;

        *codecs = mask->codecs;
        *domain = mask->video_domains;
+
+     return sizeof(*mask);
    }

    static void parser_init(struct venus_inst *inst, u32 *codecs, u32 *domain)
@@ -283,8 +300,9 @@ static int hfi_platform_parser(struct venus_core *core, struct venus_inst *inst)
    u32 hfi_parser(struct venus_core *core, struct venus_inst *inst, void *buf,
                   u32 size)
    {
-     unsigned int words_count = size >> 2;
-     u32 *word = buf, *data, codecs = 0, domain = 0;
+     u32 *words = buf, *payload, codecs = 0, domain = 0;
+     u32 *frame_size = buf + size;
+     u32 rem_bytes = size;

```

```

int ret;

ret = hfi_platform_parser(core, inst);
@@ -301,38 +319,66 @@ u32 hfi_parser(struct venus_core *core, struct venus_inst *inst, void *buf,
    memset(core->caps, 0, sizeof(core->caps));
}

- while (words_count) {
-     data = word + 1;
+ while (words < frame_size) {
+     payload = words + 1;

-     switch (*word) {
+     switch (*words) {
- case HFI_PROPERTY_PARAM_CODEC_SUPPORTED:
-     parse_codecs(core, data);
+     case HFI_PROPERTY_PARAM_CODEC_SUPPORTED:
+         parse_codecs(core, data);
+         if (rem_bytes <= sizeof(struct hfi_codec_supported))
+             return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+         ret = parse_codecs(core, payload);
+         if (ret < 0)
+             return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+         init_codecs(core);
+         break;
- case HFI_PROPERTY_PARAM_MAX_SESSIONS_SUPPORTED:
-     parse_max_sessions(core, data);
+     case HFI_PROPERTY_PARAM_MAX_SESSIONS_SUPPORTED:
+         parse_max_sessions(core, data);
+         if (rem_bytes <= sizeof(struct hfi_max_sessions_supported))
+             return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+         ret = parse_max_sessions(core, payload);
+         break;
- case HFI_PROPERTY_PARAM_CODEC_MASK_SUPPORTED:
-     parse_codecs_mask(&codecs, &domain, data);
+     case HFI_PROPERTY_PARAM_CODEC_MASK_SUPPORTED:
+         parse_codecs_mask(&codecs, &domain, data);
+         if (rem_bytes <= sizeof(struct hfi_codec_mask_supported))
+             return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+         ret = parse_codecs_mask(&codecs, &domain, payload);
+         break;
- case HFI_PROPERTY_PARAM_UNCOMPRESSED_FORMAT_SUPPORTED:
-     parse_raw_formats(core, codecs, domain, data);
+     case HFI_PROPERTY_PARAM_UNCOMPRESSED_FORMAT_SUPPORTED:
+         parse_raw_formats(core, codecs, domain, data);
+         if (rem_bytes <= sizeof(struct hfi_uncompressed_format_supported))
+             return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+         ret = parse_raw_formats(core, codecs, domain, payload);
+         break;
- case HFI_PROPERTY_PARAM_CAPABILITY_SUPPORTED:
-     parse_caps(core, codecs, domain, data);
+     case HFI_PROPERTY_PARAM_CAPABILITY_SUPPORTED:
+         parse_caps(core, codecs, domain, data);
+         if (rem_bytes <= sizeof(struct hfi_capabilities))
+             return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+         ret = parse_caps(core, codecs, domain, payload);
+         break;
- case HFI_PROPERTY_PARAM_PROFILE_LEVEL_SUPPORTED:
-     parse_profile_level(core, codecs, domain, data);
+     case HFI_PROPERTY_PARAM_PROFILE_LEVEL_SUPPORTED:
+         parse_profile_level(core, codecs, domain, data);
+         if (rem_bytes <= sizeof(struct hfi_profile_level_supported))
+             return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+         ret = parse_profile_level(core, codecs, domain, payload);
+         break;
- case HFI_PROPERTY_PARAM_BUFFER_ALLOC_MODE_SUPPORTED:
-     parse_alloc_mode(core, codecs, domain, data);
+     case HFI_PROPERTY_PARAM_BUFFER_ALLOC_MODE_SUPPORTED:
+         parse_alloc_mode(core, codecs, domain, data);
+         if (rem_bytes <= sizeof(struct hfi_buffer_alloc_mode_supported))
+             return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+         ret = parse_alloc_mode(core, codecs, domain, payload);
+         break;
- default:

```

```
+         ret = sizeof(u32);
+         break;
+     }

-     word++;
-     words_count--;
+     if (ret < 0)
+         return HFI_ERR_SYS_INSUFFICIENT_RESOURCES;
+
+     words += ret / sizeof(u32);
+     rem_bytes -= ret;
+ }

if (!core->max_sessions_supported)
```