Fwd: CVE-2022-2347 - Unchecked Download Size and Direction in U-Boot USB DFU

From: "Eduardo' Vela\" <Nava>" <evn () google com>
Date: Fri, 8 Jul 2022 10:11:58 +0200

```
---------- Forwarded message ---------
From: Eduardo' Vela" <Nava> <evn () google com>
Date: Fri, 8 Jul 2022, 10:07
Subject: CVE-2022-2347 - Unchecked Download Size and Direction in U-Boot
USB DFU
To: <sultan.qasimkhan () nccgroup com>, 3pvd <3pvd () google com>, <
u-boot () lists denx de>


```
Vendor: DENX Software Engineering
Vendor URL: https://www.denx.de/wiki/U-Boot
Versions affected: v2012.10-rc1 to <version of fix>
Systems Affected: All systems with CONFIG_DFU_OVER_USB or CONFIG_SPL_DFU
enabled
Author: <Sultan Qasim Khan> <sultan.qasimkhan () nccgroup com>
Advisory URL / CVE Identifier: CVE-2022-2347
Risk: 7.7 AV:P/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N
```


    Summary

    U-Boot is a popular and feature-rich bootloader for embedded systems.
It includes optional support for the USB Device Firmware Update (DFU)
protocol, which can be used by devices to download new firmware, or upload
their current firmware.

    The U-Boot DFU implementation does not bound the length field in USB
DFU download setup packets, and it does not verify that the transfer
direction corresponds to the specified command. Consequently, if a physical
attacker crafts a USB DFU download setup packet with a `wLength `greater
than 4096 bytes, they can write beyond the heap-allocated request buffer.
It is also possible to read its content (and beyond it) if the direction
bit for the setup packet indicates a device to host direction.
```

Location

```
    drivers/usb/gadget/f_dfu.c
```

Functions `dfu_handle, state_dfu_idle, state_dfu_dnload_idle, handle_dnload`

Impact

Data beyond the heap-allocated `req->buf `buffer may be corrupted or read by a connected USB host when a device running U-Boot is in DFU mode. This may enable a malicious host to gain code execution on the device running U-Boot, or read sensitive data from the device.

Details

USB DFU setup packets are handled by the `dfu_handle `function. The DFU command is specified by the `ctrl->bRequest `field, and the transfer direction for the data phase is specified by the direction bit `ctrl->bRequestType & USB_DIR_IN`. The `dfu_handle `function calls state-specific handlers such as `state_dfu_idle `or `state_dfu_dnload_idle`, and uses the value returned by the state handler as the length for the data phase of the transfer. The buffer that will be written to or read from in the data phase of the transfer is `req->buf`, which is heap allocated as 4096 (`USB_BUFSIZ`) bytes in `composite_bind `of [drivers/usb/gadget/composite.c](https://source.denx.de/u-boot/u-boot/-/blob/4df50f89f5769732c6cce67f956371140680ff5d/drivers/usb/gadget/composite.c#L1396).

The request structure that is set up is then queued with the USB controller driver via a call to `usb_ep_queue`. There are several USB controllers supported by U-Boot, such as the popular Designware DWC2 whose support is implemented in <code>[drivers/usb/gadget/dwc2_udc_otg.c](https://source.denx.de/u-boot/u-boot/-/blob/master/drivers/usb/gadget/dwc2_udc_otg.c) </code>and <code>[drivers/usb/gadget/dwc2_udc_otg_xfer_dma.c](https://source.denx.de/u-boot/u-boot/-/blob/master/drivers/usb/gadget/dwc2_udc_otg_xfer_dma.c)</code>. These drivers are unaware of the allocated size for the request buffer (<code>req->buf</code>), and assume the supplied length field (<code>req->length</code>) is safe for the allocated buffer.

```
    static int
    dfu_handle(struct usb_function *f, const struct usb_ctrlrequest *ctrl)
    {
    struct usb_gadget *gadget = f->config->cdev->gadget;
    struct usb_request *req = f->config->cdev->req;
    struct f_dfu *f_dfu = f->config->cdev->req->context;
    …


    if (req_type == USB_TYPE_STANDARD) {
        …
    } else /* DFU specific request */
        value = dfu_state[f_dfu->dfu_state] (f_dfu, ctrl, gadget, req);

    if (value >= 0) {
        req->length = value;
        req->zero = value < len;
    value = usb_ep_queue(gadget->ep0, req, 0);
        if (value < 0) {
            debug("ep_queue --> %d\n", value);
            req->status = 0;
        }
    }

    return value;
    }
```

The DFU state handlers which support the download command (`state_dfu_idle` and `state_dfu_dnload_idle`) return the value returned by `handle_dnload` when `ctrl->bRequest `is `USB_REQ_DFU_DNLOAD`. No checking of the transfer direction is performed; DFU download requests are assumed to always be OUT transfers (host to device). However, a malicious or compromised host could issue a download request setup packet with the `USB_DIR_IN `bit set (device to host). A DFU download request with the `USB_DIR_IN `bit set would cause data in req->buf to be sent to the host, rather than filling the buffer with data received from the host.

The `handle_dnload `function simply returns the length argument passed to it without any bounds checking. Both state handlers that call `handle_dnload `pass it the `wLength `field of the setup packet without any bounds checks. Consequently, a malicious host that sends a DFU setup packet with a length longer than 4096 bytes would result in a read or write beyond `req->buf`. The DFU functional descriptor does declare a maximum `wTransferSize `of `DFU_USB_BUFSIZ `(4096 bytes), and compliant hosts would abide by not sending setup packets specifying lengths longer than this. However, a malicious or non-compliant host may send a DFU setup packet for a transfer longer than this.

```
    static int handle_dnload(struct usb_gadget *gadget, u16 len)
    {
    struct usb_composite_dev *cdev = get_gadget_data(gad get);
    struct usb_request *req = cdev->req;
    struct f_dfu *f_dfu = req->context;

    if (len == 0)
        f_dfu->dfu_state = DFU_STATE_dfuMANIFEST_SYNC;

    req->complete = dnload_request_complete;
    return len;
    }
```

```
    static int state_dfu_idle(struct f_dfu *f_dfu,
```

~~~
     const struct usb_ctrlrequest *ctrl,
     struct usb_gadget *gadget,
     struct usb_request *req)
    {
    u16 w_value = le16_to_cpu(ctrl->wValue);
    u16 len = le16_to_cpu(ctrl->wLength);
    int value = 0;

    switch (ctrl->bRequest) {
    case USB_REQ_DFU_DNLOAD:
        if (len == 0) {
            f_dfu->dfu_state = DFU_STATE_dfuERROR;
            value = RET_STALL;
            break;
        }
        f_dfu->dfu_state = DFU_STATE_dfuDNLOAD_SYNC;
        f_dfu->blk_seq_num = w_value;
        value = handle_dnload(gadget, len);
        break;
    …
    }

    return value;
    }
```
~~~

    **Recommendation **

    Limit USB transfer lengths to a maximum of `DFU_USB_BUFSIZ `before adding them to the endpoint transfer queue in `dfu_handle`. In every DFU setup packet handler, also verify that the direction bit `ctrl->bRequestType & USB_DIR_IN `matches the request type (such as upload or download).

    **Vendor Communication **

1. Feb 27 2022 - Initial email to security () denx de (this was the wrong email)
2. April 30 2022 - Follow up (60 days)
3. June 3 2022 - Email to wd () denx de (bounced but provided alternative contacts)
4. June 7 2022 - Discussion to post to the public mailing list
5. July 8 2022 - Public disclosure

    **Written by: **Sultan Qasim Khan from NCC Group
https://www.nccgroup.com/

By Date   By Thread

Current thread:

- **Fwd: CVE-2022-2347 - Unchecked Download Size and Direction in U-Boot USB DFU** *Eduardo' Vela" <Nava> (Jul 08)*

Nmap Security Scanner

Ref Guide

Install Guide

Docs

Download

Nmap OEM

Npcap packet capture

User's Guide

API docs

Download

Npcap OEM

Security Lists

Nmap Announce

Nmap Dev

Full Disclosure

Open Source Security

BreachExchange

Security Tools

Vuln scanners

Password audit

Web scanners

Wireless

Exploitation

About

About/Contact

Privacy

Advertising

Nmap Public Source License