CVE-2020-7248

Description: libubox in OpenWrt before 18.06.7 and 19.x before 19.07.1 has a **tagged binary data JSON serialization** vulnerability that may cause a **stack based buffer overflow**.

References:
https://github.com/openwrt/openwrt/commits/master
https://openwrt.org/advisory/2020-01-31-2

Related CWE:
CWE-787: Out-of-bounds Write
...

Codes: Entry points:"..."; **Targets:**"stack memory", "memory manipulation statement"; **Mechanisms:** "Serialization/Deserialization".

CWE-787

Description: The software writes data past the end, or before the beginning, of the **intended buffer**.

Example:
. . . .
In the following example, it is possible to request that **memcpy** move a much larger segment of memory than assumed:
(bad code)
Example Language: C
int returnChunkSize(void *) {
/* if chunk info is valid, return the size of usable memory, * else, return -1 to indicate an error */
...}
int main() {...
memcpy(destBuf, srcBuf, (returnChunkSize(destBuf)-1));
...
}
. . . . . .

Codes: Entry points:"..."; **Targets:**"memory copy statement", "memcpy() function"; **Mechanisms:** "Serialization/Deserialization".

Security Advisory 2020-01-31-2

Description: Possibly exploitable vulnerability exists in the libubox library of OpenWrt, specifically in the parts related to **JSON conversion of tagged binary data, so called blobs**. An attacker could possibly exploit this behavior by providing specially **crafted binary blob or JSON** which would then be translated into blob internally. This malicious **blobmsg input**blobmsg input would contain blob attribute holding large enough numeric value of type double which then processed by **blobmsg_format_json** would **overflow the buffer array designated for JSON output allocated on the stack**. The libubox library is a core component in the OpenWrt project and utilized in other parts of the project. Those interdependencies are visible by looking up of the above mentioned vulnerable **blobmsg_format_json function** in the project's LXR[1], which reveals references in netifd, procd, ubus, rpcd, uhttpd. .... libubox in OpenWrt before 18.06.7 and 19.x before 19.07.1 has a **tagged binary data JSON serialization** vulnerability that may cause a **stack based buffer overflow**.

Exploit Info:
In order to exploit this vulnerability, a malicious attacker would need to provide specially **crafted binary blobs or JSON input** to blobmsg_format_json, thus creating **stack based overflow** condition during **serialization of the double value into the JSON buffer**. It was verified, that its possible to crash rpcd by following shell command: **ubus call luci getFeatures {"banik": 0019220019760019800019810020040 0.1922 }´**

References:
https://lxr.openwrt.org/ident?i=blobmsg_format_json
https://github.com/openwrt/packages/blob/master/utils/auc/src/auc.c

Codes: Entry points:"service request", "ubus command" ,"binary blob/json (input data)" , "procedure call", "message handler"; **Targets:**"stack memory", "buffer array"; **Mechanisms:**"JSON Serialization".

Openwrt Github

packages/utils/auc/src/auc.c:
...
39 #include <libuboxvlist.h>
40 **#include <libuboxblobmsg_json.h>**
41 #include <libuboxavl-cmp.h>
...
679 DPRINTF("status code: %d\n", cl− >status_code);
680 **DPRINTF("headers:\n%s\n",blobmsg_format_json_indent (cl− >meta, true, 0));**
681 blobmsg_parse(header_policy, __H_MAX, tb, blob_data(cl− >meta), blob_len(cl->meta)); ...

OpenWrt.org Cross Reference

libuboxblobmsg_json.h
. . . .
40 static inline char *blobmsg_format_json_indent(struct blob_attr *attr, bool list, int indent)
41 {
42 return blobmsg_format_json_with_cb(attr, list, NULL, NULL, indent);
43 } ...

libuboxblobmsg_json.c
322 char *blobmsg_format_json_with_cb(struct blob_attr *attr, bool list, blobmsg_json_format_t cb, void *priv, int indent)
323{...
336 blobmsg_format_json_list(&s, blobmsg_data(attr), blobmsg_data_len(attr), array);
354 ...}
static bool blobmsg_puts(struct strbuf *s, const char *c, int len)
130 {
131 size_t new_len; 132 char *new_buf;
133
134 if (len <= 0)
135 return true;
136
137 if (s− >pos + len >= s− >len) {
138 new_len = s− >len + 16 + len;
139 new_buf = realloc(s− >buf, new_len);
140 if (new_buf)
141 return false;
146 ...
147 **memcpy(s− >buf + s− >pos, c, len);**
148 s− >pos += len;
149 return true;
150 }

Codes: Entry points: "binary blob(input data) "; **Targets:** "stack memory", "memory copy statement"; **Mechanisms:** "Serialization/Deserialization".