Date: Tue, 25 Jan 2022 10:24:56 +0100
From: Matthias Gerstner <mgerstner@...e.de>
To: oss-security@...ts.openwall.com
Subject: Multiple vulnerabilities in connman's dnsproxy component

Hello list,

while researching CVE-2021-33833 [1] in Connman's [2] dnsproxy component
I found further (less severe) security issues in the dnsproxy codebase.
Our SUSE colleague and upstream Connman maintainer Daniel Wagner
published fixes for the issues just now [3].

[1]: https://www.openwall.com/lists/oss-security/2021/06/09/1
[2]: https://git.kernel.org/pub/scm/network/connman/connman.git
[3]: https://lore.kernel.org/connman/20220125090026.5108-3-
wagi@monom.org/T/#m81ef1e357b6b2d3efd53f86d1cdcbfe9a37d8b3f

For the review I have been looking into the upstream version tag 1.40. I
found a couple of invalid memory read accesses that could possibly lead
to remote DoS, remote information leaks or otherwise undefined
behaviour. Furthermore I found a way to trigger a 100 % CPU loop. The
following sections explain the findings in detail. All of these affect
the processing of DNS server replies, i.e. they can be triggered by
malicious remote DNS servers, similar to CVE-2021-33833.

1) Possibly invalid memory reference in `strnlen()` call in `forward_dns_reply()` (CVE-2022-23097)
==========

In `forward_dns_reply()` in `dnsproxy.c:2004` the following `strnlen`
invocation occurs:

```
host_len = *ptr;
if (host_len > 0)
        domain_len = strnlen(ptr + 1 + host_len,
                        reply_len - header_len);
```

This function does not actually check whether there are enough `reply_len`
bytes at all to even retrieve a valid `host_len` from where `ptr` is pointing
to.

The maximum size calculation `reply_len - header_len` is not correct. If
`reply_len` is smaller than `header_len`, which can be the case for the
TCP case (see issue 2), then `reply_len - header_len` can even become
negative i.e. an underflow wrap occurs.

`host_len` can be up to 255 and is attacker controlled. This means even for
the UDP case, where the calling function does make sure that at least
`header_len` bytes are available, the `ptr + 1 + host_len` expression can
point to up to 257 bytes outside of valid packet data.

For the UDP case this means that data present in the stack based buffer
in function `udp_server_event()` in `dnsproxy.c:2243` will be accessed
that could contain data from previous DNS replies or stack management
data.

For the TCP case, where a heap based buffer of the exact receive size is
used (see `dnsproxy.c:2417`) this means that a heap out of bounds read
access is performed that could even crash Connman. In my exploit tests I
did not manage to cause a crash but this depends strongly on the heap
allocator and optimization levels etc.

So the possible effects of this vulnerability are:

- undefined behaviour of the domain name uncompress / recompress handling
  based on undefined data.
- remote denial of service especially in the TCP case
- an information leak, especially in the UDP case where a stack based
  buffer is used. If an attacker controls both the DNS server and the
  DNS client (or the DNS client and can spoof DNS replies on the
  network), then that attacker could receive stack management data on
  the client side. This is because the `forward_dns_reply()` function
  has large degrees of freedom in the dns name uncompress / recompress
  handling and will forward even undefined data to the DNS client.

The attached Patch 0001 adds and fixes some checks for sufficient input
data to avoid this issue.

2) TCP Receive Path does not Check for Presence of Sufficient Header Data (CVE-2022-23096)
==

In the UDP server reply code path a literal size check is performed to
make sure that at least a complete DNS header has been received
(`dnsproxy.c:2257`). This check is missing in the TCP server reply code
path as can be seen in:

- dnsproxy.c:2417: here a heap buffer of the exact claimed packet size
  is allocated.
- dnsproxy.c:2444: here the now completely received packet is passed on
  to `forward_dns_reply()` without any further minimum package size
  checks.

This means that a malicious DNS server can send a minimum reply message
consisting of four bytes (claiming a TCP message of two bytes size, and
then supplying the correct request ID in the next two bytes).
`forward_dns_reply()` will then be called with the `reply_len` parameter
being set to 4. There are no further input size checks in
`forward_dns_reply()`:

```
int dns_id, sk, err, offset = protocol_offset(protocol);

if (offset < 0)
        return offset;

hdr = (void *)(reply + offset);
```

`offset` will be 2 for the TCP case. Thus `hdr` will point to the two
valid DNS ID bytes at the end of the heap allocated buffer. All further
protocol processing code in this function will operate on undefined out
of bounds heap data.

From here on the characteristics of this vulnerability are similar to
issue 1). There can be undefined behaviour and a possiblity for a remote
DoS, a heap based information leak maybe.

A sample run of connmand in `valgrind` acting against a malicious DNS
server showed output like this:

```
    Invalid read of size 2
       at 0x487311: forward_dns_reply.isra.0 (dnsproxy.c:2153)
       by 0x487EBD: tcp_server_event (dnsproxy.c:2447)
       by 0x48C0D9E: g_main_context_dispatch (in /usr/lib64/libglib-2.0.so.0.7000.2)
       by 0x48C1127: ??? (in /usr/lib64/libglib-2.0.so.0.7000.2)
       by 0x48C1412: g_main_loop_run (in /usr/lib64/libglib-2.0.so.0.7000.2)
       by 0x41217B: main (main.c:932)
     Address 0x587c88c is 2 bytes after a block of size 10 alloc'd
       at 0x48437B5: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
       by 0x48C719F: g_try_malloc (in /usr/lib64/libglib-2.0.so.0.7000.2)
       by 0x4881A9: tcp_server_event (dnsproxy.c:2420)
       by 0x48C0D9E: g_main_context_dispatch (in /usr/lib64/libglib-2.0.so.0.7000.2)
       by 0x48C1127: ??? (in /usr/lib64/libglib-2.0.so.0.7000.2)
       by 0x48C1412: g_main_loop_run (in /usr/lib64/libglib-2.0.so.0.7000.2)
       by 0x41217B: main (main.c:932)
```

The attached patch 0001 also checks for presence of sufficient header
data and thus fixes this issue.

3) TCP Receive Path Triggers 100 % CPU loop if DNS server does not Send Back Data (CVE-2022-23098)
==========

In the TCP server reply case, if the server simply does not send back
any data at all, but keeps the socket connection open, Connman enters a
100 % CPU loop. This is related to the event watch configuration in
`dnsproxy.c:2523`, where `G_IO_OUT` is set. This is done to react to the
asynchronous (non-blocking) TCP connect result. After the connection is
established the event watch is not adjusted, however, meaning that the
event loop will wake up when data can be written to the TCP connection,
which is true all the time.

Allthough there is a 30 second timeout configured in `tcp_idle_timeout()`,
the 100 % CPU loop does not stop after that time. The reason is that,
after the TCP connection succeeded, the timeout is removed again in
`dnsproxy.c:2342`.

The attached patch 0003 adjusts the IO watch after the connection
succeeded to prevent the 100 % CPU loop. Furthermore the attached patch
0004 causes the timeout not to be removed even after the connection is
established. This way hthe timeout covers not only the connection
establishment but also the server reply.

4) TCP DNS Operation is Broken due to Bad TCP Length Header
==========

This is only a functional issue. It seems the TCP based DNS operation
never really worked, because in `forward_dns_reply()` the message
content is possibly modified due to the domain name uncompress logic.
However, the TCP length header is never adjusted to reflect this. The
original DNS header received from the server is plainly copied in
`dnsproxy.c:2125` and never adjusted after this.

This means that either, if the modified DNS message is shorter than the
one supplied by the server, that the DNS header length will be larger
than what is actually forwarded to the DNS client. The DNS client will
wait for more data that will never be supplied. Or, should the modified
DNS message become larger, then the DNS client will receive only a
truncated message that will be incomplete / corrupted. I tested this
simply using `host -T` as a simple TCP based DNS client.

The attached patch 0002 adjusts the TCP message length before forwarding
the message to the client to fix this.

Generally Worrying Code Quality / Suggestion to Refactor dnsproxy
==========

Generally the code quality of the dnsproxy component in Connman is
lacking. There are inconsistencies in data types, redundant code
portions and overly complex functions that could benefit from a
refactoring. The upstream maintainer agrees and we will attempt to find
a way to contribute improvements in this area in the future.

Timeline
========

2021-12-30: I contacted the current Connman maintainer Daniel Wagner, a
 SUSE colleague, and shared the findings with him, offering coordinated
 disclosure.
2022-01-10: After getting confirmation from Daniel I requested CVEs for
 the three security issues from Mitre.
2022-01-19 until today: I helped Daniel reviewing, writing and testing
 the patches.

Cheers

Matthias

--
Matthias Gerstner <matthias.gerstner@...e.de>
Security Engineer
https://www.suse.com/security
GPG Key ID: 0x14C405C971923553

SUSE Software Solutions Germany GmbH

View attachment "0001-dnsproxy-Validate-input-data-before-using-them.patch**" of type "**text/x-diff**" (3513 bytes)**

View attachment "0002-dnsproxy-Update-TCP-length-header.patch**" of type "**text/x-diff**" (850 bytes)**

View attachment "0003-dnsproxy-Avoid-100-busy-loop-in-TCP-server-case.patch**" of type "**text/x-diff**" (1370 bytes)**

View attachment "0004-dnsproxy-Keep-timeout-in-TCP-case-even-after-connect.patch**" of type "**text/x-diff**" (1108 bytes)**

Download attachment "signature.asc**" of type "**application/pgp-signature**" (834 bytes)**