CURL AND LIBCURL

HOW I MADE A HEAP OVERFLOW IN CURL

OCTOBER 11, 2023 | DANIEL STENBERG | LEAVE A COMMENT

In association with the release of curl 8.4.0, we publish a security advisory and all the details for CVE-2023-38545. This problem is the worst security problem found in curl in a long time. We set it to severity **HIGH**.

While the advisory contains all the necessary details. I figured I would use a few additional words and expand the explanations for anyone who cares to understand how this flaw works and how it happened.

Background

curl has supported SOCKS5 since August 2002.

SOCKS5 is a proxy protocol. It is a rather simple protocol for setting up network communication via a dedicated "middle man". The protocol is for example typically used when setting up communication to get done over Tor but also for accessing Internet from within organizations and companies.

SOCKS5 has two different host name resolver modes. Either the client resolves the host name *locally* and passes on the destination as a resolved address, or the client passes on the entire host name to the proxy and lets the proxy itself resolve the host *remotely*.

In early 2020 I assigned myself an old long-standing curl issue: to convert the function that connects to a SOCKS5 proxy from a blocking call into a non-blocking state machine. This is for example much noticeable when an application performs a large amount of parallel transfers that all go over SOCKS5.

On February 14 2020 I landed the main commit for this change in master. It shipped in 7.69.0 as the first release featuring this enhancement. And by extension also the first release vulnerable to CVE-2023-38545.

A less wise decision

The state machine is called repeatedly when there is more network data to work on until it is done: when the connection is established.

At the top of the function I made this:

```
bool socks5_resolve_local =
  (proxytype == CURLPROXY_SOCKS5) ? TRUE : FALSE;
```

This boolean variable holds information about whether curl should resolve the host or just pass on the name to the proxy. This assignment is done at the top and thus for every invocation while the state machine is running.

The state machine starts in the INIT state, in which the main bug for today's story time lies. The flaw is inherited from the function from before it was turned into a state-machine.

```
if(!socks5_resolve_local && hostname_len > 255) {
  socks5_resolve_local = TRUE;
}
```

SOCKS5 allows the host name field to be up to 255 bytes long, meaning a SOCKS5 proxy cannot resolve a longer host name. On finding a too long host name. the curl code makes the bad decision to instead switch over to local resolve mode. It sets the local variable for that purpose to TRUE. (This condition is a leftover from code added ages ago. I think it was downright wrong to switch mode like this, since the user asked for remote resolve curl should stick to that or fail. It is not even likely to work to just switch, even in “good” situations.)

The state machine then switches state and continues.

The issue triggers

If the state machine cannot continue because it has no more data to work with, like if the SOCKS5 server is not fast enough, it returns. It gets called again when there is data available to continue working on. Moments later.

But now, look at the local variable **socks5_resolve_local** at the top of the function again. It again gets set to a value depending on proxy mode – *not remembering the changed value because of the too long host name*. Now it again holds a value that says the proxy should resolve the name remotely. But the name is too long…

curl builds a protocol frame in a memory buffer, and it copies the destination to that buffer. Since the code wrongly thinks it should pass on the host name, even though the host name is too long to fit, the memory copy can overflow the allocated target buffer. Of course depending on the length of the host name and the size of the target buffer.

Target buffer

The allocated memory area curl uses to build the protocol frame in to send to the proxy, is the same as the regular download buffer. It is simply reused for this purpose before the transfer starts. The download buffer is 16kB by default but can also be set to use a different size at the request of the application. The curl tool sets the buffer size to 100kB. The minimum accepted size is 1024 bytes.

If the buffer size is set smaller than 65541 bytes this overflow is possible. The smaller the size, the larger the possible overflow.

Host name length

A host name in a URL has no real size limit, but libcurl's URL parser refuses to accept names longer than 65535 bytes. DNS only accepts host names up 253 bytes. So, a legitimate name that is longer than 253 bytes is unusual. A real name that is longer than 1024 is virtually unheard of.

Thus it pretty much requires a malicious actor to feed a super-long host name into this equation to trigger this flaw. To use it in an attack. The name needs to be longer than the target buffer to make the memory copy overwrite heap memory.

Host name contents

The host name field of a URL can only contain a subset of octets. A range of byte values are plain invalid and would cause the URL parser to reject it. If libcurl is built to use an IDN library, that one might also reject invalid host names. This bug can therefore only trigger if the right set of bytes are used in the host name.

Attack

An attacker that controls an HTTPS server that a libcurl using client accesses over a SOCKS5 proxy (using the proxy-resolver-mode) can make it return a crafted redirect to the application via a HTTP 30x response.

Such a 30x redirect would then contain a Location: header in the style of:

```
Location: https://aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa/
```

… where the host name is longer than 16kB and up to 64kB

If the libcurl using client has automatic redirect-following enabled, and the SOCKS5 proxy is "slow enough" to trigger the local variable bug, it will copy the crafted host name into the too small allocated buffer and into the adjacent heap memory.

A heap buffer overflow has then occurred.

The fix

curl should *not* switch mode from remote resolve to local resolve due to too long host name. It should rather return an error and starting in curl 8.4.0, it does.

We now also have a dedicated test case for this scenario.

Credits

This issue was reported, analyzed and patched by Jay Satiro.

This is the largest curl bug-bounty paid to date: **4,660 USD** (plus 1,165 USD to the curl project, as per IBB policy)

OUR GOAL IS TO WRITE BUG-FREE SOFTWARE. I'LL PAY A TEN-DOLLAR BONUS FOR EVERY BUG YOU FIND AND FIX.
S. Adams E-mail: SCOTTADAMS@AOL.COM
YAHOO!
WE'RE RICH
YES!!! YES!!! YES!!!
11/13 © 1995 United Feature Syndicate, Inc.(NYC)
I HOPE THIS DRIVES THE RIGHT BEHAVIOR.
I'M GONNA WRITE ME A NEW MINIVAN THIS AFTER-NOON!

Classic related Dilbert strip. The original URL seems to no longer be available.

Rewrite it?

Yes, this family of flaws would have been impossible if curl had been written in a memory-safe language instead of C, but porting curl to another language is not on the agenda. I am sure the news about this vulnerability will trigger a new flood of questions about and calls for that and I can sigh, roll my eyes and try to answer this again.

The only approach in that direction I consider viable and sensible is to:

1. allow, use and support more dependencies written in memory-safe languages and
2. potentially and gradually replace parts of curl piecemeal, like with the introduction of hyper.

Such development is however currently happening in a near glacial speed and shows with painful clarity the challenges involved. curl will remain written in C for the foreseeable future.

Everyone not happy about this are of course welcome to roll up their sleeves and get working.

Including the latest two CVEs reported for curl 8.4.0, the accumulated total says that **41%** of the security vulnerabilities ever found in curl would likely not have happened should we have used a memory-safe language. But also: the rust language was not even a possibility for practical use for this purpose during the time in which we introduced maybe the first 80% of the C related problems.

It burns in my soul

Reading the code now it is impossible not to see the bug. Yes, it truly aches having to accept the fact that I did this mistake without noticing and that the flaw then remained undiscovered in code for 1315 days. I apologize. I am but a human.

It could have been detected with a better set of tests. We repeatedly run several static code analyzers on the code and none of them have spotted any problems in this function.

In hindsight, shipping a heap overflow in code installed in over twenty billion instances is not an experience I would recommend.

Behind the scenes

To learn how this flaw was reported and we worked on the issue before it was made public. Go check the Hackerone report (will be made public asap).

CURL AND LIBCURL   SECURITY