11 Connect-only connections can use the wrong connection

Share:

TIMELINE

m42a submitted a report to curl. Jul 31st (2 years ago)

Summary:

If a connect-only easy handle is not read from or written to, its connection can time out and be closed. If a new connection is created it can be allocated at the same address, causing the easy handle to use the new connection. This new connection may not be connected to the same server as the old connection, which can allow sensitive information intended to go to the first server to instead go to the second server.

This sequence of events would be uncommon in ordinary usage, so I have attached a sample program that implements a simple caching allocator, which causes the address to be re-used deterministically.

According to git bisect, this behavior was introduced in commit 755083d.

Steps To Reproduce:

1. Compile the source code below
2. Listen on ports 1234, 1235, and 1236
3. Run the compiled program
4. Notice that the data which was supposed to be sent to port 1234 is actually sent to port 1236

Supporting Material/References:

Code 4.84 KiB Wrap lines Copy Download

```
#include <iostream>
#include <stdexcept>
#include <thread>
#include <chrono>
#include <unordered_map>

#include <string.h>

#include <curl/curl.h>

using namespace std::literals;

static void require(bool b)
{
    if (!b)
        throw std::runtime_error("Assertion failed");
}

struct alloc
{
    alloc *next_alloc;
    std::size_t size;
};

std::unordered_map<std::size_t, alloc *> cached_allocations;

void *malloc_(size_t size)
{
    auto &ptr = cached_allocations[size];
    if (ptr)
    {
        void *ret = (char *)ptr + sizeof(alloc);
        ptr = ptr->next_alloc;
        return ret;
    }
    auto new_ptr = (alloc *)calloc(1, size + sizeof(alloc));
    new_ptr->next_alloc = nullptr;
    new_ptr->size = size;
    void *ret = ((char *)new_ptr) + sizeof(alloc);
    return ret;
}

void free_(void *ptr)
{
    auto alloc_ptr = (alloc *)((char *)ptr - sizeof(alloc));
    auto &last_alloc = cached_allocations[alloc_ptr->size];
    alloc_ptr->next_alloc = last_alloc;
    last_alloc = alloc_ptr;
}

```

```
    auto new_alloc_ptr = (alloc *)realloc(alloc_ptr, size + sizeof(alloc));
    new_alloc_ptr->size = size;
    return (char *)new_alloc_ptr + sizeof(alloc);
}

char *strdup_(const char *str)
{
    auto size = strlen(str) + 1;
    auto new_str = (char *)malloc(size);
    return strcpy(new_str, str);
}

void *calloc_(size_t nmemb, size_t size)
{
    auto full_size = nmemb*size;
    return malloc_(full_size);
}


int main()
{
    curl_global_init_mem(CURL_GLOBAL_DEFAULT, &malloc_, &free_, &realloc_, &strdup_, &calloc_);

    auto multi = curl_multi_init();
    require(multi);

    auto easy1234 = curl_easy_init();
    require(easy1234);
    auto eret = curl_easy_setopt(easy1234, CURLOPT_URL, "http://127.0.0.1:1234/");
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1234, CURLOPT_CONNECT_ONLY, 1);
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1234, CURLOPT_VERBOSE, 1L);
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1234, CURLOPT_MAXAGE_CONN, 1L);
    require(eret == CURLE_OK);
    auto mret = curl_multi_add_handle(multi, easy1234);
    require(mret == CURLM_OK);

    // Create connection to port 1234
    while (true)
    {
        int running;
        mret = curl_multi_socket_action(multi, CURL_SOCKET_TIMEOUT, 0, &running);
        require(mret == CURLM_OK);
        int remaining;
        if (auto info = curl_multi_info_read(multi, &remaining))
        {
            require(info->msg == CURLMSG_DONE);
            require(info->easy_handle == easy1234);
            require(info->data.result == CURLE_OK);
            break;
        }
    }

    // Allow connection to port 1234 to age out
    std::this_thread::sleep_for(2s);

    auto easy1235 = curl_easy_init();
    require(easy1235);
    eret = curl_easy_setopt(easy1235, CURLOPT_URL, "http://127.0.0.1:1235/");
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1235, CURLOPT_CONNECT_ONLY, 1);
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1235, CURLOPT_VERBOSE, 1L);
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1235, CURLOPT_MAXAGE_CONN, 1L);
    require(eret == CURLE_OK);
    mret = curl_multi_add_handle(multi, easy1235);
    require(mret == CURLM_OK);

    // Create connection to port 1235, then close connection to port 1234 as it is too old
    while (true)
    {
        int running;
        mret = curl_multi_socket_action(multi, CURL_SOCKET_TIMEOUT, 0, &running);
        require(mret == CURLM_OK);
        int remaining;
```

```
            require(info->easy_handle == easy1235);
            require(info->data.result == CURLE_OK);
            break;
        }
    }

    auto easy1236 = curl_easy_init();
    require(easy1236);
    eret = curl_easy_setopt(easy1236, CURLOPT_URL, "http://127.0.0.1:1236/");
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1236, CURLOPT_CONNECT_ONLY, 1);
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1236, CURLOPT_VERBOSE, 1L);
    require(eret == CURLE_OK);
    eret = curl_easy_setopt(easy1236, CURLOPT_MAXAGE_CONN, 1L);
    require(eret == CURLE_OK);
    mret = curl_multi_add_handle(multi, easy1236);
    require(mret == CURLM_OK);

    // Create connection to port 1236, which re-uses the memory of the previous connection to port 1234
    while (true)
    {
        int running;
        mret = curl_multi_socket_action(multi, CURL_SOCKET_TIMEOUT, 0, &running);
        require(mret == CURLM_OK);
        int remaining;
        if (auto info = curl_multi_info_read(multi, &remaining))
        {
            require(info->msg == CURLMSG_DONE);
            require(info->easy_handle == easy1236);
            require(info->data.result == CURLE_OK);
            break;
        }
    }

    char c = 'a';
    size_t n;
    // Attempts to send data to port 1234, but actually uses the connection to port 1236
    eret = curl_easy_send(easy1234, &c, 1, &n);
    require(eret == CURLE_OK);

    mret = curl_multi_remove_handle(multi, easy1236);
    require(mret == CURLM_OK);
    mret = curl_multi_remove_handle(multi, easy1235);
    require(mret == CURLM_OK);
    mret = curl_multi_remove_handle(multi, easy1234);
    require(mret == CURLM_OK);
    mret = curl_multi_cleanup(multi);
    require(mret == CURLM_OK);
}
```

Impact

This could cause sensitive data intended for one server to be transmitted to a different server.

bagder curl staff posted a comment. Jul 31st (2 years ago)

Thanks for your report, we will look into it!

bagder curl staff posted a comment. Aug 1st (2 years ago)

Is the `curl_global_init_mem` needed for this problem to appear? Does the problem not appear if we just use `curl_global_init` and uses the default memory functions?

m42a posted a comment. Aug 1st (2 years ago)

The problem can happen with any memory allocator that reuses freed memory. The custom allocator causes it to happen deterministically with this test case, but it should be possible to create a test case tailored to any specific memory allocator if you know its memory re-use strategy.

bagder curl staff posted a comment. Updated Aug 1st (2 years ago)

Sorry, but if the problem relies on your provided functions, isn't this then just proving that the problem is related to those functions and not necessarily in curl itself?

I have not tried to proof-read your code but for example your calloc() seems broken (it doesn't always return cleared memory), which then hardly is curl's fault.

m42a posted a comment. Aug 1st (2 years ago)

I have attached a program that reproduces the issue (on my system) without intercepting the allocation functions. Note that this relies on the system allocator, and thus may fail to reproduce for you. I'm running Arch Linux, using the system libraries, and compiling with GCC 10.1.0 with no additional compiler flags.

1 attachment:
F932910: curl-malloc.cpp

m42a posted a comment. Aug 1st (2 years ago)

Kind of. It's a single-threaded variant of an ABA problem where the easy handle identifies a connection by memory address and just assumes that any connection at that address is the original connection. So if the memory was just freed it would simply fail to find the connection, but the fact that a second connection was allocated at the address that the first connection used to be causes it to think that it has a connection when in fact that connection is unrelated.

bagder curl staff posted a comment. Aug 1st (2 years ago)

What libcurl version are you using when you see the problem?

I tested both git master and 7.71.1 just now and `require()` on line 124 triggers (see below). I use glibc 2.31 and g++ 10.2.0 on debian unstable.

Code 167 Bytes Wrap lines Copy Download

```
1  // Attempts to send data to port 1234, but actually uses the connection to port 1236
2  eret = curl_easy_send(easy1234, &c, 1, &n);
3  require(eret == CURLE_OK); <= this one
```

m42a posted a comment. Aug 1st (2 years ago)

I'm also using glibc 2.31 and curl 7.71.1.

m42a posted a comment. Aug 1st (2 years ago)

Here is a program that uses a separate jemalloc arena for curl; you may find it easier to reproduce with this. I'm using jemalloc 5.2.1.

1 attachment:
F932957: curl-malloc.cpp

bagder curl staff posted a comment. Aug 1st (2 years ago)

Ah yes. Thanks for providing the different versions, they're very helpful. The jemalloc recipe does indeed reproduce the problem for me.

The issue

The problem is that when `data->state.lastconnect` is set to point to a connectdata struct, the pointer is not updated if a connection is removed. Then the pointed-to connection might be removed and if then a new connection is created before the pointer is used (and it gets allocated on the same memory address), the `lastconnect` pointer can end up pointing to the next connection instead of the original one.

Limitations

I believe this flaw is limited to affecting curl_easy_send/curl_easy_recv and CURLINFO_LASTSOCKET/CURLINFO_ACTIVESOCKET, not any of the transfer APIs.

Security issue?

It's a very limited bug that is triggered in rare circumstances that really need the planets to be properly aligned, but then that's probably also why this can become a real security issue: a program can be written to use those functions and never see the problem happen until way later in a situation that could be problematic.

I'll take another round of thinking, and I hope other security team members do as well, but I think it smells like a security problem.

bagder curl staff posted a comment. Aug 3rd (2 years ago)

@m42a: while we look deeper at this, do you have a fix or intend to work on one?

bagder curl staff posted a comment. Aug 3rd (2 years ago)

It struck me we should rather change the logic to use the `connection_id` to find the connection with instead, as that's an increasing number for every new connection... I'm trying out a patch locally now.

bagder curl staff posted a comment. Aug 3rd (2 years ago)

Here's a first take on a patch. Seems to no longer reproduce the problem in my tests.

1 attachment:
F934582: 0001-Curl_easy-remember-last-connection-by-id-not-referen.patch

bagder curl staff posted a comment. Aug 3rd (2 years ago)

Here's my first attempt at writing an advisory for this issue.

---

libcurl: wrong connect-only connection

Project curl Security Advisory, August 19th 2020 -
Permalink

VULNERABILITY

An application that performs multiple requests with libcurl's multi API and sets the `CURLOPT_CONNECT_ONLY` option, might in rare circumstances experience that when subsequently using the setup connect-only transfer, libcurl will pick and use the wrong connection - and instead pick another one the application has created since then.

`CURLOPT_CONNECT_ONLY` is the option to tell libcurl to not perform an actual transfer, only connect. When that operation is completed, libcurl remembers which connection it used for that transfer and "easy handle". It remembers the connection using a pointer to the internal `connectdata` struct in memory.

If more transfers is then done with the same multi handle before the connect-only connection is used, leading to the initial connect-only

connection.

If after those operations, the application then wants to use the original transfer's connect-only setup to for example use `curl_easy_send()` to send raw data over that connection, libcurl could **erroneously** find an existing connection still being alive at the address it remembered since before even though this is now a new and different connection.

The application could then accidentally send data over that connection which wasn't at all intended for that recipient, entirely unknowingly.

We are not aware of any exploit of this flaw.

INFO

This bug has existed at least since commit
c43127414d, first shipped in
curl 7.29.0.

This flaw cannot trigger for users of the curl tool but only for applications using libcurl and the `CURLOPT_CONNECT_ONLY` option.

The flaw only happens if the exact same memory address is re-used again for the new connection as for the original connect-only connection.

The Common Vulnerabilities and Exposures (CVE) project has assigned the name CVE-2020-XXXX to this issue.

CWE-825: Expired Pointer Dereference

Severity: Low

AFFECTED VERSIONS

- Affected versions: libcurl 7.29.0 to and including 7.71.1
- Not affected versions: libcurl < 7.29.0 and libcurl >= 7.72.0

THE SOLUTION

A ...

RECOMMENDATIONS

We suggest you take one of the following actions immediately, in order of preference:

A - Upgrade curl to version 7.72.0

B - Apply the patch on your curl version and rebuild

C - Do not use `CURLOPT_CONNECT_ONLY`

TIMELINE

This issue was first reported to the curl project on July 31, 2020.

This advisory was posted on August 19th 2020.

CREDITS

This issue was reported by Marc Aldorasi. Patched by Daniel Stenberg.

Thanks a lot!

1 attachment:
F934634: CVE-2020-XXXX.md

m42a posted a comment. Aug 3rd (2 years ago)

The problem with that is that connection ids are only unique within a single multi handle, so this issue can still occur with multiple multi handles; see the attached source file. You could easily avoid this by just clearing that id in `curl_multi_remove_handle`.

My thought for a fix was to transfer ownership of the connection to the easy handle once its fully connected. The connection can't be re-used, so there's no utility to having it live in the cache once its connected. This would also address this review comment, since we wouldn't need to look for the connection if we had direct ownership of it, and there would be similar improvements in `curl_easy_recv` and `curl_easy_send`. That might be too big of a change for something we want people to be able to backport though.

1 attachment:
F934643: curl-reuse.cpp

bagder curl staff posted a comment. Aug 3rd (2 years ago)

> You could easily avoid this by just clearing that id in curl_multi_remove_handle.

Right, when the easy handle is removed from the multi it should be cleared. I'll amend my patch. It should probably also be cleared in add_handle.

> My thought for a fix was to transfer ownership of the connection to the easy handle once its fully connected.

That would be a rather big change though as we have no such connection handling now, and "normal" connections should not be moved there so it would need to be specific for connect-only ones.

patch v2

bagder curl staff posted a comment. Aug 3rd (2 years ago)
Simple mistake. v3!

1 attachment:
F934672: v3-0001-Curl_easy-remember-last-connection-by-id-not-refe.patch

bagder curl staff updated CVE reference to CVE-2020-8231. Aug 6th (2 years ago)

bagder curl staff posted a comment. Aug 6th (2 years ago)
This is CVE-2020-8231

curl rewarded m42a with a $500 bounty. Aug 7th (2 years ago)
The curl security team has decided to reward hacker @m42a with the amount of 500 USD for finding and reporting this issue. Many thanks for your great work!

bagder curl staff posted a comment. Aug 12th (2 years ago)
I've posted the advisory as a "pre-notification" to distros@openwall.org and Seth Arnold of Canonical raised this concern. I'll reply to him, then paste my reply here as well. That email list is closed for the public.

I gave the patch a very cursory read and came to the conclusion that "the fix" amounts to this hunk:

Code 291 Bytes Wrap lines Copy Download

```
1  -  if(data->state.lastconnect) {
2  +  if(data->state.lastconnect_id !=3D -1) {
3       /* Mark any connect-only connection for closure */
4       Curl_conncache_foreach(data, data->state.conn_cache,
5                              data, &close_connect_only);
6  +    data->state.lastconnect_id =3D -1;
7     }
```

This appears to be the spot where NULLing a pointer may have been missing, is this correct?

I have to admit some uncertainty with the approach -- if a pointer may have been stale, I'm not sure why an integer hanging off the struct is more likely to be correct.

This patch also changes the abi around the struct connfind objects; does this object cross application boundaries? Will this change require recompiled applications?

Thanks

bagder curl staff posted a comment. Aug 12th (2 years ago)
(This is how I replied to Seth's email about his concerns)

Thanks for checking out the patch!

> This appears to be the spot where NULLing a pointer may have been missing, is this correct?

Yes and no. Clearing the pointer there would not mitigate the full problem, as the API doesn't mandate that the user removes the easy handle from the multi handle when the connection is used and that assignment/clearing is in the curl_multi_remove_handle() function.

> I have to admit some uncertainty with the approach -- if a pointer may have been stale, I'm not sure why an integer hanging off the struct is more likely to be correct.

Because the id is an incrementing integer so each new connection gets a new number and if one old 'lastconnect_id' would linger around using a number identifying a now-closed connection, it can't be satisfied by a connection created later.

I'm not married to this specific solution and would be happy to solve it otherwise if we can come up with a cleaner or nicer approach. This one should at least fix the security issue for now.

> This patch also changes the abi around the struct connfind objects; does this object cross application boundaries? Will this change require recompiled applications?

Nothing of this change affects the ABI to the best of my knowledge. Applications don't see the connfind object; it is private.

bagder curl staff closed the report and changed the status to **Resolved**. Aug 19th (2 years ago)
Now released. The public URL for this vulnerability is here: https://curl.haxx.se/docs/CVE-2020-8231.html

m42a agreed to disclose this report. Nov 5th (2 years ago)

This report has been disclosed. Nov 5th (2 years ago)