4 CVE-2021-22924: Bad connection reuse due to flawed path name checks

Share:

TIMELINE

nyymi submitted a report to curl. Jun 10th (2 ye

Summary:

`Curl_ssl_config_matches` attempts to compare whether two SSL connections have identical SSL security options or not. The idea is to avoid reusing a connectio that uses less secure, or completely different security options such as capath, cainfo or certificate/issuer pinning.

Unfortunately this function has several flaws in it:

1. It completely fails to take into account "BLOB" type certificate values, such as set by `CURLOPT_CAINFO_BLOB` and `CURLOPT_ISSUERCERT_BLOB`. If the application ca made to initiate connection to a user specified location (where these BLOB options are not used) before the "more secure" connection using these options is r the attacker can point the application to connect to the same address and port, effectively poisoning the connection cache with a connection that has been established with different cainfo or issuecert settings. This leads to attacker being able to neutralize these options and make libcurl ignore them for the connections for which they're set. I have no obvious CWE number for this one, but CWE-664 `Improper Control of a Resource Through its Lifetime` might fit.
2. `CURLOPT_ISSUERCERT` value is not matched. Similar to above.
3. Similarly, the function has an implementation flaw where path names use case-insensitive comparison for capath, cainfo and pinned public key paths. This can to a situation where if the attacker can specify the capath, cainfo or pinned public key name that have a different path capitalization. Again, if the attacker can specify some of these values for the connection that is performed before the later supposedly secure connection is made, the attacker is able to make the furt connection use incorrect capath, cainfo or pinned public key. This is CWE-41 `Improper Resolution of Path Equivalence`.
4. Finally, the pinned public key fingerprint set by `CURLOPT_PINNEDPUBLICKEY` `sha256//` is incorrectly compared as case-insenstive value. If the attacker is able to create a otherwise valid certificate that has a fingerprint that has the same fingerprint string but with different capitalization (very difficult to pull off in practice and the application could be tricked to use this value for `CURLOPT_PINNEDPUBLICKEY` and create a connection, later connection could be confused to think that tl pinned public key is the same one.

Exploiting any of these issues requires a situation where the attacker can coax the application to create a TLS connection to the same host and port that will be performed by the application itself later on (for example some backend connection or other high security connection the attacker wishes to man in the middle). In these situations the existing connection with less security guarantees may be reused, allowing man in the middle attacks against the later supposedly secure connection, resulting in loss of confidentiality and integrity. Since this requires an active attack it can't be thought to have direct availability impact. In most cases where this would result in exploitation would be scenarios where there would be a privilege barrier between the user providing the connection target addresses (lo priority) and the libcurl using application performing the actual connections (higher priority). It can also be exploitable in a scenario where the attacker will try to m the middle connections performed by other users of the same service (lateral attack towards users at the same privilege level).

Exploiting the first two issues is plausible in a situation where the attacker can obtain a valid certificate for the host, but from issuer that doesn't match what the application pinning will check for. If the app uses the blob variants to set up pinning and the attacker is able to obtain a certificate for the specific host from for exa Let's Encrypt, the "pin stripping" attack would be plausible.

Exploiting the 3rd issue is be possible in a situation where the attacker can write to a location that has the same path but with a different capitalization. One examp such situation would be an application that uses a `/tmp`, `/dev/shm` or similar sticky world writable location to store the capath/cainfo/pinned public key file. The attacker would then be able to use the same location but with different file name capitalization to fool the application to reuse the existing connection for later connections that actually would use a different capath, cainfo or pinned public key. This attack requires that the attacker can provide the options for capath, cainfo the public cert pinning somehow (the application would need to enable this as part of its normal functionality).

Steps To Reproduce:

This proof of concept demonstrates the 3rd issue with the curl tool:

1. `cp /etc/ssl/certs/ca-certificates.crt ca.crt`
2. `touch CA.crt`
3. `curl --capath /dev/null --cacert $PWD/ca.crt https://curl.se --next --capath /dev/null --cacert $PWD/CA.crt https://curl.se`

If `Curl_ssl_config_matches` comparison is implemented correctly the 2nd connection should fail.

Proposed Fix:

In Curl_ssl_config_matches:

- Add "blob" binary matching for `CURLOPT_CAINFO_BLOB` and `CURLOPT_ISSUERCERT_BLOB`
- Add case-sensitive matching for `CURLOPT_ISSUERCERT` value
- Use case-sensitive matching for paths and public key cert signature(s)
- Ensure that there are no other SSL parameters that are improperly compared or omitted from the equivalence check

Impact

TLS man in the middle

nyymi posted a comment. Jun 10th (2 ye

Depending on the application specifics the impact can be limited to being able to bypass certificate pinning.

bagder curl staff posted a comment. Jun 11th (2 ye

Thank you for your report!

We will take some time and investigate your reports and get back to you with details and possible follow-up questions as soon as we can!

nyymi posted a comment. Updated Jun 11th (2 ye

```
     blobcmp(data->cert_blob, needle->cert_blob) &&
     blobcmp(data->ca_info_blob, needle->ca_info_blob) &&
```

So point 1 is slightly wrong: the function does actually check for `CURLOPT_CAINFO_BLOB` (`ca_info_blob`). `CURLOPT_ISSUERCERT_BLOB` (`issuercert_blob`) is not check however.

nyymi posted a comment. Jun 11th (2 ye

- issue 1 was crated by commit https://github.com/curl/curl/commit/cac5374298b3e79405bbdabe38941227c73a4c96 for version 7.71.0
- issue 2 is much much older. Couldn't readily figure it out, but looks like at least 16 years old or so (the code has moved around a lot, at least).
- Issue 3 was created by commit https://github.com/curl/curl/commit/cb4e2be7c6d42ca0780f8e0a747cecf9ba45f151 for version 7.52.0
- Issue 4 I believe existed since public key pinning was added and the option didn't get checked at all before. It seems this problem was reported initially at https://curl.se/mail/lib-2019-09/0061.html and commit https://github.com/curl/curl/commit/3c5f9ba899ace6a0a406e421c4c1f6e626a95d05 attempted t it, but did the compare also in case-insensitive way

nyymi posted a comment. Jun 11th (2 ye

`CURLOPT_CAINFO_BLOB` being checked after all renders issue 1 far less severe than it initially looked. This means that the certificate here needs to otherwise valid. S issues 1 and 2 can only lead to certificate pinning (`CURLOPT_ISSUERCERT` / `CURLOPT_ISSUERCERT_BLOB`) bypass.

nyymi posted a comment. Updated Jun 11th (2 ye

Issue 3 can lead to actual TLS man in the middle, but for this to happen the application code really needs to be quite specific:

- the application needs to allow setting `CURLOPT_CAPATH` or `CURLOPT_CAINFO` as part of the normal functionality (attacker would need to use this functionality to capath or cainfo to enable man in the middle of their own connection that then would be reused for some other supposedly secure connection later)
- attacker must be able write to locations that match the paths used by some other security context - except for different capitalization

Not very common use pattern luckily, but not entirely out of the realms of possibility of actually being used by some application.

Issue 1 can only lead to man in the middle condition (assuming all other validation is in place) if someone is able to obtain a legitimate certificate for the site and then form a connection using this certificate (which the attacker can then obviously MiTM since they have the private key). If further connections u `CURLOPT_ISSUERCERT` or `CURLOPT_ISSUERCERT_BLOB` will then reuse this established connection allowing the now man in the middle to be maintained.

nyymi posted a comment. Jun 11th (2 ye

My understanding is that this is limited to HTTP/2 and pipelining.

Anyway, here's a PoC using pycurl:

Code 932 Bytes Wrap lines Copy Dow

```
#!/usr/bin/env python3

import pycurl

def main():
    # simulated attacker controlled connection with crafted CA.crt file
    c1 = pycurl.Curl()
    c1.setopt(pycurl.URL, "https://curl.se")
    c1.setopt(pycurl.CAINFO, '/tmp/CA.crt')

     # simulated connection in another security context that the attacker tries to MiTM
     c2 = pycurl.Curl()
     c2.setopt(pycurl.URL, "https://curl.se")
     c2.setopt(pycurl.CAINFO, '/tmp/ca.crt')

     m = pycurl.CurlMulti()
     m.add_handle(c1)

     while True:
         ret, num_handles = m.perform()
         if ret != pycurl.E_CALL_MULTI_PERFORM: break
     while num_handles:
         ret = m.select(1.0)
         if ret == -1: break
         while True:
             ret, num_handles = m.perform()
             if ret != pycurl.E_CALL_MULTI_PERFORM: break

         numq, oklist, errlist = m.info_read(2)
         if c1 in oklist:
             m.add_handle(c2)

 if __name__ == "__main__":
     main()
```

1. use `openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365` to create key + cert pair for CN `curl.se`
2. `cp cert.pem /tmp/CA.crt`
3. `cp /etc/ssl/certs/ca-certificates.crt /tmp/ca.crt`
4. use the key.pem and cert.pem to host a fake curl.se HTTP2 enabled site
5. point curl.se to this fake server (for example via `/etc/hosts`)
6. launch the python PoC app

bagder curl staff posted a comment. Updated Jun 14th (2 ye

~~I'm sorry but I don't understand the PoC.~~

~~If you can modify the `CAINFO` file name used by the application or the contents of the specified file then you can indeed make the application accept the attacker' site. How is that related to wrong connection reuse?~~

bagder curl staff posted a comment. Jun 14th (2 ye

ah sorry, I was blind for the case differences. Now I understand.

bagder curl staff posted a comment. Jun 14th (2 ye

Doing correct file name comparisons is difficult. On Windows they're also case insensitive. On mac they're *often* insensitive, and on Linux they're *rarely* insensitive in the two latter it'll depend on the target file system.

Ideas on how this could be fixed?

bagder curl staff changed the status to Triaged. Jun 14th (2 ye

Confirmed security issue

nyymi posted a comment. Updated Jun 14th (2 ye

> Ideas on how this could be fixed?

Indeed this is a difficult problem: You even have systems that can have both kind of filesystems mounted at the same time: mac os can have both case sensitive ar case insensitive mounts at the same time. This means that you can't easily just decide on case sensitive or case sensitive comparison based on OS type.

There's fpathconf/pathconf(..., _PC_CASE_SENSITIVE) one some platforms at least (including macos). First I thought about using this functionality to see if both : and dest are case-insensitive and then using case-insensitive comparison if so, else case sensitive. But then there's the matter of that some file systems are usin UTF-8 file names... So regular case-insensitive string comparison might not work.

I guess the best solution is to:

- If non-unixy platform we know always is case insensitive use case insensitive comparison (windows/dos/others?), else
- Perform case sensitive comparison. If string are the same, return "same path" status, else
- `stat()` both paths
- if both `stat()` fail: if errno are the same the paths can be considered the same else return "different paths" status
- if one of the `stat()` fails and the other succeeds, consider the paths different
- if `st_dev` and `st_ino` are the same values consider the paths the same else return "different paths"

bagder curl staff posted a comment. Jun 18th (about 1 y

I think we should just switch to case sensitive for every path comparison and document it. The downside of a mismatch is just worse connection reuse and a performance penalty. It is still likely to be rare. The cost of doing a "perfect" compare seems to not really balance it up.

nyymi posted a comment. Jun 18th (about 1 y

Sounds good. It indeed is not a problem to have a false negative here, whereas false positive can be a real problem.

bagder curl staff posted a comment. Jun 21st (about 1 y

[PROPOSED PATCH] Unfortunately I had to touch 6 files so I'm attaching my v1 here.

1 attachment:
F1346346: 0001-vtls-fix-connection-reuse-checks-for-issuer-cert-and.patch

bagder curl staff posted a comment. Jun 21st (about 1 y

The patch doesn't update the docs. It seems better to keep that separate to keep it as small as possible.

bagder curl staff posted a comment. Jun 21st (about 1 y

First advisory draft (also attached)

Bad connection reuse due to case insensitive path name checks

Project curl Security Advisory, July 21st 2021 -
Permalink

VULNERABILITY

libcurl keeps previously used connections in a connection pool for subsequent transfers to reuse, if one of them matches the setup.

Due to errors in the logic, the config matching function did not take 'issuer cert' into account and it compared the involved paths *case insensitively*, which could lead to libcurl reusing wrong connections.

File paths are, or can be, case sensitive on many systems but not all, and can even vary depending on used file systems.

The comparison also didn't include the 'issuer cert' which a transfer can set to qualify how to verify the server certificate.

We are not aware of any exploit of this flaw.

INFO

The Common Vulnerabilities and Exposures (CVE) project has assigned the name CVE-2021-VVVVV to this issue.

CWE-295: Improper Certificate Validation

Severity: Medium

AFFECTED VERSIONS

- Affected versions: curl 7.10.4 to and including 7.77.0
- Not affected versions: curl < 7.10.4 and curl >= 7.78.0

Also note that libcurl is used by many applications, and not always advertised as such.

THE SOLUTION

The SSL configs are compared appropriately.

A fix for CVE-2021-VVVVV

RECOMMENDATIONS

A - Upgrade curl to version 7.78.0

B - Apply the patch to your local version

TIMELINE

This issue was reported to the curl project on June 11, 2021.

This advisory was posted on July 21, 2021.

CREDITS

This issue was reported by Harry Sintonen. Patched by Daniel Stenberg.

Thanks a lot!

1 attachment:
F1346351: CVE-2021-VVVVV.md

nyymi posted a comment. Jun 21st (about 1 y

Both proposed patch and advisory are looking good to me.

bagder curl staff updated CVE reference to **CVE-2021-22924**. Jun 28th (about 1 y

Jun 28th (about 1 year ago)
bagder curl staff changed the report title from **Connection reuse improper SSL parameter equivalence check** to **CVE-2021-22924: Bad connection reuse due to flawed path name checks**.

url rewarded nyymi with a **$1,200** bounty. Jun 30th (about 1 y

The curl security team has decided to reward hacker @nyymi with the amount of 1200 USD for finding and reporting this issue. Many thanks for your great work!

bagder curl staff posted a comment. Jul 17th (about 1 y

Seth Arnold commented (on the distros@ list):

Hello Daniel, all, I gave this patch a very quick review, and have a few comments: first, some fields are moving from one struct to another, that might cause problem if memory for either struct might be allocated by callers 'outside' of the curl project. Just grepping around it looked a bit like both structs are only ever included in other structs and not likely to be allocated / deallocated on their own, so I don't think this is a problem.

Second, safe_strcmp() -- I expected it to be constant-time comparison based on the name, and I worry someone else may use it that way without looking first. Also, it will give inconsistent results if called like: safe_strcmp(foo, NULL) and safe_strcmp(NULL, foo), which would be a problem if it were used as a strcmp() in a sorting routine.

Do with this as you like, including politely nodding if appropriate. :)

Thanks

nyymi posted a comment. Updated Jul 17th (about 1 y

IMO the first point is a non-issue. Any external code that depends on internal libcurl structures is heavily broken anyway. As far as I understand nothing outside of libcurl can ever allocate any of the structures used in libcurl, it all goes thru the API, and the libcurl API compatibility remains intact.

The comment about `safe_strcmp` has a point thought, the name of the function is a bit misleading. However, since it's not exported to outside world I think the ri manageable. However, it could be renamed to something like `nullsafe_strcmp`, and perhaps the sorting consistency fixed in regard of `non-NULL, NULL` vs `NULL, NULL` case. It would not cause any significant overhead. ~~However, that is totally unrelated to this security patch and can be done out of this ticket.~~

Edit -- Oh wait the function was added by this patch... ok right, so those changes should probably be incorporated.

nyymi posted a comment. Updated Jul 17th (about 1 y

Thinking about it, the function being called `*_strcmp` is the misleading bit, since it doesn't behave like a `strcmp` class function. It could just be renamed to somet else that doesn't include `strcmp` in the name.

bagder curl staff posted a comment. Jul 17th (about 1 y

I'll rename it to just `safecmp` (also, my rebase just now required a minor hands-on).

bagder curl staff closed the report and changed the status to **Resolved**. Jul 21st (about 1 y

bagder curl staff requested to disclose this report. Jul 21st (about 1 y

nyymi agreed to disclose this report. Jul 21st (about 1 y

This report has been disclosed. Jul 21st (about 1 y