NAME

curl_easy_ssls_export − export SSL sessions

SYNOPSIS

```
#include <curl/curl.h>

typedef CURLcode curl_ssls_export_function(CURL *handle,
                                           void *userptr,
                                           const char *session_key,
                                           const unsigned char *shmac,
                                           size_t shmac_len,
                                           const unsigned char *sdata,
                                           size_t sdata_len,
                                           curl_off_t valid_until,
                                           int ietf_tls_id,
                                           const char *alpn,
                                           size_t earlydata_max);

CURLcode curl_easy_ssls_export(CURL *handle,
                               curl_ssls_export_function *export_fn,
                               void *userptr);
```

DESCRIPTION

This function iterates over all SSL session tickets that belong to the easy handle and invokes the **export_fn** callback on each of them, as long as the callback returns **CURLE_OK**.

The callback may then store this information and use *curl_easy_ssls_import(3)* in another libcurl instance to add SSL session tickets again. Reuse of SSL session tickets may result in faster handshakes and some connections might be able to send request data in the initial packets (0−RTT).

From all the parameters passed to the **export_fn** only two need to be persisted: either **session_key** or **shamc** and always **sdata**. All other parameters are informative, e.g. allow the callback to act only on specific session tickets.

Note that SSL sessions that involve a client certificate or SRP username/password are not exported.

Export Function Parameter

Session Key

This is a printable, null−terminated string that starts with **hostname:port** the session ticket is originating from and also contains all relevant SSL parameters used in the connection. The key also carries the name and version number of the TLS backend used.

It is recommended to only persist **session_key** when it can be protected from outside access. Since the hostname appears in plain text, it would allow any third party to see how curl has been used for.

Salted Hash

A binary blob of **shmac_len** bytes that contains a random salt and a cryptographic hash of the salt and **session_key**. The salt is generated for every session individually. Storing **shmac** is recommended when placing session tickets in a file, for example.

A third party may brute−force known hostnames, but cannot "grep" for them.

Session Data

A binary blob of **sdata_len** bytes, **sdata** contains all relevant SSL session ticket information for a later import − apart from **session_key** and **shmac**.

valid_until
: Seconds since EPOCH (1970−01−01) until the session ticket is considered valid.

TLS Version
: The IETF assigned number for the TLS version the session ticket originates from. This is **0x0304** for TLSv1.3, **0x0303** for 1.2, etc. Session tickets from version 1.3 have better security properties, so an export might store only those.

ALPN
: The ALPN protocol that had been negotiated with the host. This may be **NULL** if negotiation gave no result or had not been attempted.

Early Data
: The maximum amount of bytes the server supports to receive in early data (0−RTT). This is 0 unless the server explicitly indicates support.

PROTOCOLS

This functionality affects all TLS based protocols: HTTPS, FTPS, IMAPS, POP3S, SMTPS etc.

This option works only with the following TLS backends: GnuTLS, OpenSSL, mbedTLS and wolfSSL

EXAMPLE

```
CURLcode my_export_cb(CURL *handle,
                      void *userptr,
                      const char *session_key,
                      const unsigned char *shmac,
                      size_t shmac_len,
                      const unsigned char *sdata,
                      size_t sdata_len,
                      curl_off_t valid_until,
                      int ietf_tls_id,
                      const char *alpn,
                      size_t earlydata_max)
{
 /* persist sdata */
 return CURLE_OK;
}

int main(void)
{
 CURLSHcode sh;
 CURLSH *share = curl_share_init();
 CURLcode result;
 CURL *curl;

 sh = curl_share_setopt(share, CURLSHOPT_SHARE, CURL_LOCK_DATA_SSL_SESSION);
 if(sh)
  printf("Error: %s\n", curl_share_strerror(sh));

 curl = curl_easy_init();
 if(curl) {
  curl_easy_setopt(curl, CURLOPT_SHARE, share);

  /* run a transfer, all TLS sessions received will be added
   * to the share. */
  curl_easy_setopt(curl, CURLOPT_URL, "https://example.com/");
  curl_easy_perform(curl);
```

```
  /* export the TLS sessions collected in the share */
  result = curl_easy_ssls_export(curl, my_export_cb, NULL);

  /* always cleanup */
  curl_easy_cleanup(curl);
 }
 curl_share_cleanup(share);
}
```

AVAILABILITY

Added in curl 8.12.0

RETURN VALUE

This function returns a CURLcode indicating success or error.

CURLE_OK (0) means everything was OK, non−zero means an error occurred, see *libcurl−errors(3)*. If *CURLOPT_ERRORBUFFER(3)* was set with *curl_easy_setopt(3)* there can be an error message stored in the error buffer when non−zero is returned.

SEE ALSO

CURLOPT_SHARE(3), **curl_easy_ssls_import**(3), **curl_share_setopt**(3)