7 CVE-2022-32208: FTP-KRB bad message verification

Share:

TIMELINE

nyymi submitted a report to curl. Jun 2nd (6 months ago)

Summary:

libcurl handles `gss_unwrap` `GSS_S_BAD_SIG` error incorrectly. This enables malicious attacker to inject arbitrary FTP server responses to GSSAPI protected FTP control connection and/or make the client consume unrelated heap memory as a FTP command response.

The defective `krb5_decode` function is as follows:

Code 561 Bytes Wrap lines Copy Download

```
1  static int
2  krb5_decode(void *app_data, void *buf, int len,
3              int level UNUSED_PARAM,
4              struct connectdata *conn UNUSED_PARAM)
5  {
6   gss_ctx_id_t *context = app_data;
7   OM_uint32 maj, min;
8   gss_buffer_desc enc, dec;
9
10   (void)level;
11   (void)conn;
12
13   enc.value = buf;
14   enc.length = len;
15   maj = gss_unwrap(&min, *context, &enc, &dec, NULL, NULL);
16   if(maj != GSS_S_COMPLETE) {
17     if(len >= 4)
18       strcpy(buf, "599 ");
19     return -1;
```

```
23   len = curlx_uztosi(dec.length);
24   gss_release_buffer(&min, &dec);
25
26   return len;
27 }
```

Note how `read_data` function will set the `buf->size` to result of the decode operation as-is without considering possible `-1` return code and that size `buf->size` is of type `size_t`:

Code 128 Bytes — Wrap lines Copy Download

```
1  /* Types needed for krb5-ftp connections */
2  struct krb5buffer {
3    void *data;
4    size_t size;
5    size_t index;
6    BIT(eof_flag);
7  };
```

Code 686 Bytes — Wrap lines Copy Download

```
1  static CURLcode read_data(struct connectdata *conn,
2                            curl_socket_t fd,
3                            struct krb5buffer *buf)
4  {
5    int len;
6    CURLcode result;
7
8    result = socket_read(fd, &len, sizeof(len));
9    if(result)
10     return result;
11
12   if(len) {
13     /* only realloc if there was a length */
14     len = ntohl(len);
15     buf->data = Curl_saferealloc(buf->data, len);
16   }
17   if(!len || !buf->data)
18     return CURLE_OUT_OF_MEMORY;
19
```

```
23    buf->size = conn->mech->decode(conn->app_data, buf->data, len,
24                                   conn->data_prot, conn);
25    buf->index = 0;
26    return CURLE_OK;
27  }
```

When `gss_unwrap` returns an error the `krb5_decode` code attempts to erase the buffer by prefixing the buffer with `599 \0`. However, this doesn't take into account the case that arbitrary number of bytes can be read by `read_data` function. Hence the buffer may contain multiple lines not just one. The attacker merely needs to find a position in the FTP protocol where ftpcode `599` doesn't lead to connection termination to take over the GSSAPI protected FTP session control channel. From that point onwards the server responses can be forged by the attacker (but need to be predicted, as the attacker has no direct knowledge of the actual commands sent to the server).

It's also notable that the any `gss_unwrap` error leading to `-1` size will lead to `sec_recv` consuming unallocated heap buffer via `buffer_read` if the reading application keeps reading more data:

Code 234 Bytes Wrap lines Copy Download

```
1  static size_t
2  buffer_read(struct krb5buffer *buf, void *data, size_t len)
3  {
4    if(buf->size - buf->index < len)
5      len = buf->size - buf->index;
6    memcpy(data, (char *)buf->data + buf->index, len);
7    buf->index += len;
8    return len;
9  }
```

This can lead to disclosure of confidential information from the heap - depending on application this may reveal application secrets to the user (for example via verbose error messages). This is a local leak however, so this impact is only meaningful if the information in heap is normally hidden from the user.

Impact

- Injection of arbitrary FTP control channel server responses to supposedly GSSAPI protected FTP session.

and requirement of either man in the middle or victim connecting to malicious server.