



链滴

蘑菇街采集无水印资源 (mw-sign 值计算)

作者: [724555508](#)

原文链接: <https://ld246.com/article/1594022847457>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)



一. 需求描述

通过“蘑菇街”App分享出来的链接获取无水印视频。**该内容只供学习交流，不可用于非法用途。**

二. 分析接口

示例地址：<https://g.mogu.com/1f2MGp2l>

直接请求移动端的，这里就不示例web了，接口太多

The screenshot displays a mobile browser interface with a video player on the left and a network inspection tool on the right. The video player shows a woman holding a white fur collar. The network tool shows a list of resources and a detailed view of a request to <https://g.mogu.com/1f2MGp2l> with a 302 status code. The 'Location' header in the response is highlighted with a red box.

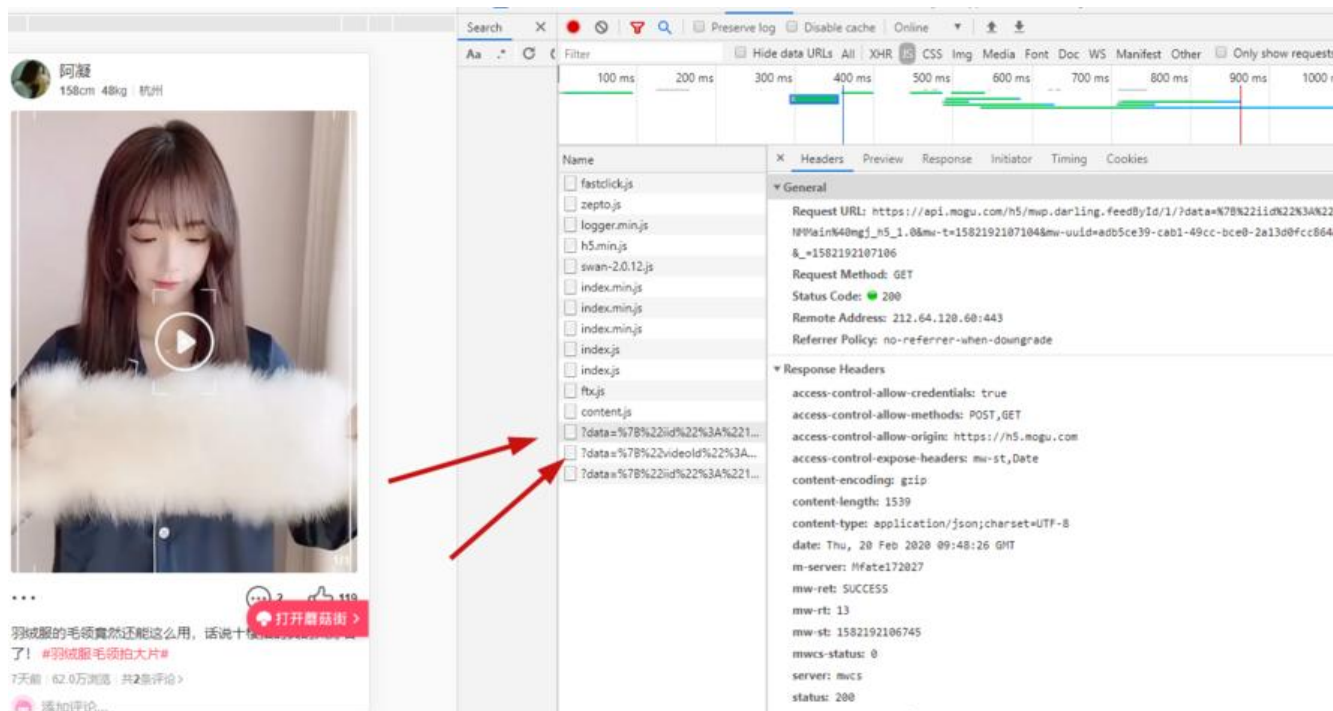
```
Request URL: https://g.mogu.com/1f2MGp2l
Request Method: GET
Status Code: 302
Remote Address: 212.64.117.182:443
Referer Policy: no-referrer-when-downgrade

Response Headers
content-language: zh-CN
content-length: 0
date: Thu, 20 Feb 2020 09:48:26 GMT
location: https://hs.mogu.com/brand-content/content.html?id=127yx5c&uid=1145i7c&uname=阿凝&type=208x_fullscreen=true&x_navbgalpha=0&acm=3.mce.1.19_1zapv0w.137314..xip6srR20kftV.sd_130-gi_xip6srR20kgk5-t_xip6srR201257392&ptp=1.Xz9sPa7W.0.0.3KJ2Yx1X
server: muccs
server: JuanniuX/15.12.29
status: 302
z-proxy: mfatel61108
z-server: mfatel63004

Request Headers
:authority: g.mogu.com
:method: GET
:path: /1f2MGp2l
:schema: https
```

如图，地址做了重定向，在ResponseHeader中拿到真实地址[location]

在js中找到了请求的接口



第一个接口是根据iid获取视频信息的。第二个接口是根据videoid获取视频播放路径的(其实第一个接口也有返回视频url，但是清晰度不够高，所有需要在第二个接口中取)

分析以下请求参数

param	是否必须	说明
data ntType)	是	数据json (包括id 和 cli
mw-appkey	是	100028
mw-ttid .0	是	NMMain@mgj_h5_
mw-t	是	时间戳
mw-uuid	是	从cookie中获取
mw-h5-os	是	iOS
mw-sign	是	调用js中方法z获取
callback	否	回调

其中“data”是可以手动拼出来的数据，“mw-appkey”，“mw-ttid”，“mw-h5-os”都是写死的参数值。“mw-t”为时间戳，剩下“mw-uuid”和“mw-sign”

三。获取mw-uuid

这里发现uuid跟cookie中的一致!!!

```

:path: /h5/mogu.com/brand-content/content.html?iid=127yx5c&uid=1145i7c&uname=%E9%98%8F%E5%87%9D&type=20&x_fullscreen=
82192107104&mw-uuid=adb5ce39-cab1-49cc-bce0-2a13d0fcc864&mw-h5-os=iOS&mw-sign=4797ccda2ba575efb41216495967cdf4&callback=mwpc
:scheme: https
:accept: */*
:accept-encoding: gzip, deflate, br
:accept-language: zh-CN,zh;q=0.9
cookie: __mgjuuid=adb5ce39-cab1-49cc-bce0-2a13d0fcc864; _mw_h5_token_enc=888824af6023122095e8b444cb078bd2; _mw_h5_token=b0c
=581000100000
referer: https://h5.mogu.com/brand-content/content.html?iid=127yx5c&uid=1145i7c&uname=%E9%98%8F%E5%87%9D&type=20&x_fullscreen=
e.1_19_1zapv0w.137314..xip6srR20kftY.sd_130-gi_xip6srR20kgK5-t_xip6srR20kftX-pri_9_1145i7c-idx_0&f=copy&s=NAMCpsChannel113308
1257392&ptp=1.Xz9sPa7W.0.0.3KJZYx1X
sec-fetch-dest: script
sec-fetch-mode: no-cors
sec-fetch-site: same-site
user-agent: Mozilla/5.0 (iPhone; CPU iPhone OS 13_2_3 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.0.3

```

▼ Query String Parameters view source view URL encoded

```

data: {"iid": "127yx5c", "clientType": "h5"}
mw-appkey: 100028
mw-ttid: NMain@mgj_h5_1.0
mw-t: 1582192107104
mw-uuid: adb5ce39-cab1-49cc-bce0-2a13d0fcc864
mw-h5-os: iOS
mw-sign: 4797ccda2ba575efb41216495967cdf4

```

如何获取？

通过请求 <https://list.mogujie.com> 在cookie中获取uuid (一开始我也找不到，别人说从这儿获取 hahaha)

四. 获取mw-sign

```

Name      x Headers Preview Response Initiator Timing
fastclick.js
zepto.js
logger.min.js
h5.min.js
swan-2.0.12.js
index.min.js
index.min.js
index.min.js
index.js
index.js
ftx.js
content.js
?data=%7B%22iid%22%3A%221...
?data=%7B%22videoid%22%3A...
?data=%7B%22iid%22%3A%221...

```

```

b._ii = function(t, e, n, o, r, i, s) {
  var a = t + (n ^ (e | ~o)) + (r >>> 0) + s;
  return (a << i | a >>> 32 - i) + e
}

b._blocksize = 16,
b._digestsize = 16,
t.exports = function(t, e) {
  if (null == t)
    throw new Error("Illegal argument " + t);
  var n = v.wordsToBytes(b(t, e));
  return e && e.asBytes ? n : e && e.asString ? w.bytesToString(n) :
}

}, Q = ["mw-pv", "mw-sign", "mw-did"]
, $ = function(n) {
  function t() {
    return null !== n && n.apply(this, arguments) || this
  }
  return v(t, n),
  t.prototype.invoke = function(t) {
    n.prototype.invoke.call(this, t);
    var e = t.getOuterContext();
    e.statistics.onEnd("DISPATCH"),
    e.headers["mw-sign"] = z(this.buildQuery(e)),
    t.invokeNext()
  }
  t.prototype.buildQuery = function(t) {
    var e = t.headers
    , n = Object.keys(e).filter(function(t) {
      return 0 === t.indexOf("mw-") && Q.indexOf(t) < 0
    }).sort().map(function(t) {
      return e[t]
    });
    return n.push(t.api),
    n.push(t.version),
    n.push(z(t.getDataString())),
    n.push(0.instance().mState.getToken()),
    n.join("&")
  }
}

```

在js中搜索 mw-sign , 最终锁定在了这个地方, 先buildQuery , 后调用了z方法 , 在buildQuery也调用了一次z方法。

现在是已经知道了实现方式所以可以直接找到位置, 第一次分析的时候非常困难, 可以尝试debugger分析z方法的入参

第一次调用z方法, 入参为{"iid": "", "clientType": "h5"}, 有没有觉得很眼熟, 这个就是刚刚分析请参数中的data

```
8206     }, J = q
8207     , F = function(t) {
8208     return null != t && (W(t) || "function" == typeof (e = t).readFloatLE && "function" == typeof e
8209     var e
8210     );
8211     };
8212     function W(t) {
8213     return !!t.constructor && "function" == typeof t.constructor.isBuffer && t.constructor.isBuffer
8214     }
8215     var z = V(function(t) {
8216     var v, g, _, w, b;
8217     v = U,
8218     g = J.utf8,
8219     _ = F,
8220     w = J.bin,
8221     (b = function(t, e) { t = {"iid": "127yx5c", "clientType": "h5"}, e = undefined
8222     t.constructor == String ? t = e && "binary" === e.encoding ? w.stringToBytes(t) : g.stringToB
8223     for (var n = v.bytesToWords(t), o = 8 * t.length, r = 1732584193, i = -271733879, s = -1732
8224     n[u] = 16711935 & (n[u] << 8 | n[u] >>> 24) | 4278255360 & (n[u] << 24 | n[u] >>> 8);
8225     n[o >>> 5] |= 128 << o % 32,
8226     n[14 + (o + 64 >>> 9 << 4)] = o;
8227     var c = b._ff
8228     , p = b._gg
8229     , l = b._hh
8230     , h = b._ii;
8231     for (u = 0; u < n.length; u += 16) {
8232     var f = r
8233     , d = i
8234     , y = s
8235     , m = a;
8236     i = h(i = h(i = h(i = l(i = l(i = l(i = l(i = p(i = p(i = p(i = p(i = c(i = c(i =
```

第二次调用z方法, 入参为

100028&IOS&1564034676371&NMMMain@mgj_pc_1.0&f65f8c1e-6286-4b4a-a917-5061e23855b&mw.p.darling.feedById&1&b9cab4ab7f543491e2c4f6c556711345&39a9ae72d3faec64f17166036f84edd_1564026637963

```
8206     }, J = q
8207     , F = function(t) {
8208     return null != t && (W(t) || "function" == typeof (e = t).readFloatLE && "function" == typeof e.slice
8209     var e
8210     );
8211     };
8212     function W(t) {
8213     return !!t.constructor && "function" == typeof t.constructor.isBuffer && t.constructor.isBuffer(t)
8214     }
8215     var z = V(function(t) {
8216     var v, g, _, w, b;
8217     v = U,
8218     g = J.utf8,
8219     _ = F,
8220     w = J.bin,
8221     (b = function(t, e) { t = "100028&IOS&1562194139385&NMMMain@mgj_h5_1.0&adb5ce39-cab1-49cc-bce0-2a13d0f
8222     t.constructor == String ? t = e && "binary" === e.encoding ? w.stringToBytes(t) : g.stringToB
8223     for (var n = v.bytesToWords(t), o = 8 * t.length, r = 1732584193, i = -271733879, s = -1732584194,
8224     n[u] = 16711935 & (n[u] << 8 | n[u] >>> 24) | 4278255360 & (n[u] << 24 | n[u] >>> 8);
8225     n[o >>> 5] |= 128 << o % 32,
8226     n[14 + (o + 64 >>> 9 << 4)] = o;
8227     var c = b._ff
8228     , p = b._gg
8229     , l = b._hh
8230     , h = b._ii;
8231     for (u = 0; u < n.length; u += 16) {
8232     var f = r
8233     , d = i
8234     , y = s
8235     , m = a;
8236     i = h(i = h(i = h(i = l(i = l(i = l(i = l(i = p(i = p(i = p(i = p(i = c(i = c(i =
```

按&截开, 最后发现每个参数代表的是

"mw-appkey", "mw-h5-os", "mw-t", "mw-ttid", "mw-uuid", 以及部分请求地址("mw.p.darling

ng.feedById/1")使用"&"拼接而成

b9cab4ab7f543491e2c4f6c556711345 这个值是第一次调用z方法的结果

39a9ae72d3faec64f157166036f84edd_1564026637963 这个是cookie中的_mwp_h5_token

如何获取mw-sign? 第二次调用z方法返回的结果就是mw-sign

五. 获取_mwp_h5_token

现在只查这个token了, 直接说怎么获取吧

拼接好参数 调用api 携带mw-uuid (mw-sign 可以不带, 因为我们这个时候并没有获取到mw-sign) 会返回以下结果

```
{"api":"mwp.darling.feedById","v":"1","ret":"FAIL_SYS_TOKEN_NEED_RENEW","token":"6aaeb8f9371b395bfc417c0948f1e2c_1582195277778","encToken":"636565ab0ff6e4d5579a246e2f7bab7","needHeaderCookie":false}
```

这里这个token就是_mwp_h5_token, encToken就是_mwp_h5_token_enc, 这两个值都有用, 为一会儿再次请求的时候需要带上cookie

拿到token之后按照第二次调用z方法执行一次, 获取出来的就是mw-sign了

六. 最后请求获取数据

拼接好所有参数再次请求一次api (必须带上cookie, 三个都要带), 即可获取到响应数据。然后在边找自己用的信息即可

七. 注意

1. 不同的接口请求地址不同, 会影响第二次z方法加密
2. uuid / token 之类的数据会失效, 具体能保持多久不知道, 目前24小时是没问题的, 可以存入缓存。