

原 RTMPdump (libRTMP) 源代码分析 4：连接第一步——握手 (Hand Shake)

2013年10月22日 22:44:27 阅读数：11311

=====

RTMPdump(libRTMP) 源代码分析系列文章：

[RTMPdump 源代码分析 1：main\(\)函数](#)

[RTMPDump \(libRTMP\) 源代码分析2：解析RTMP地址——RTMP_ParseURL\(\)](#)

[RTMPdump \(libRTMP\) 源代码分析3：AMF编码](#)

[RTMPdump \(libRTMP\) 源代码分析4：连接第一步——握手 \(HandShake\)](#)

[RTMPdump \(libRTMP\) 源代码分析5：建立一个流媒体连接 \(NetConnection部分\)](#)

[RTMPdump \(libRTMP\) 源代码分析6：建立一个流媒体连接 \(NetStream部分 1\)](#)

[RTMPdump \(libRTMP\) 源代码分析7：建立一个流媒体连接 \(NetStream部分 2\)](#)

[RTMPdump \(libRTMP\) 源代码分析8：发送消息 \(Message\)](#)

[RTMPdump \(libRTMP\) 源代码分析9：接收消息 \(Message\) \(接收视音频数据\)](#)

[RTMPdump \(libRTMP\) 源代码分析10：处理各种消息 \(Message\)](#)

=====

函数调用结构图

RTMPDump (libRTMP)的整体的函数调用结构图如下图所示。


[单击查看大图](#)



详细分析

在这里分析一下RTMPdump (libRTMP) 连接到支持RTMP协议的服务器的第一步：握手 (Hand Shake)。

RTMP连接的过程曾经分析过：[RTMP流媒体播放过程](#)

在这里不再细说，分析一下位于handshake.h文件里面实现握手 (HandShake) 功能的函数：

注意：handshake.h里面代码量很大，但是很多代码都是为了处理RTMP的加密版协议的，例如rtmps；因此在这里就不做过多分析了，我们只考虑普通的RTMP协议。

```
[cpp]  
1.  static int
2.  HandShake(RTMP * r, int FP9HandShake)
3.  {
4.      int i, offalg = 0;
5.      int dhposClient = 0;
6.      int digestPosClient = 0;
7.      int encrypted = r->Link.protocol & RTMP_FEATURE_ENC;
8.
9.      RC4_handle keyIn = 0;
10.     RC4_handle keyOut = 0;
11.
12.     int32_t *ip;
13.     uint32_t uptime;
14.
15.     uint8_t clientbuf[RTMP_SIG_SIZE + 4], *clientsig=clientbuf+4;
16.     uint8_t serversig[RTMP_SIG_SIZE], client2[RTMP_SIG_SIZE], *reply;
17.     uint8_t type;
18.     getoff *getdh = NULL, *getdig = NULL;
19.
20.     if (encrypted || r->Link.SWFSize)
21.         FP9HandShake = TRUE;
22.     else
23.         //普通的
```

```

24.     FP9HandShake = FALSE;
25.
26.     r->Link.rc4keyIn = r->Link.rc4keyOut = 0;
27.
28.     if (encrypted)
29.     {
30.         clientsig[-1] = 0x06; /* 0x08 is RTMPE as well */
31.         offalg = 1;
32.     }
33.     else
34.         //0x03代表RTMP协议的版本（客户端要求的）
35.         //数组竟然能有“-1”下标
36.         //C0中的字段(1B)
37.         clientsig[-1] = 0x03;
38.
39.     uptime = htonl(RTMP_GetTime());
40.     //void *memcpy(void *dest, const void *src, int n);
41.     //由src指向地址为起始地址的连续n个字节的数据复制到以dest指向地址为起始地址的空间内
42.     //把uptime的前4字节（其实一共就4字节）数据拷贝到clientsig指向的地址中
43.     //C1中的字段(4B)
44.     memcpy(clientsig, &uptime, 4);
45.
46.     if (FP9HandShake)
47.     {
48.         /* set version to at least 9.0.115.0 */
49.         if (encrypted)
50.         {
51.             clientsig[4] = 128;
52.             clientsig[6] = 3;
53.         }
54.         else
55.         {
56.             clientsig[4] = 10;
57.             clientsig[6] = 45;
58.         }
59.         clientsig[5] = 0;
60.         clientsig[7] = 2;
61.
62.         RTMP_Log(RTMP_LOGDEBUG, "%s: Client type: %02X", __FUNCTION__, clientsig[-1]);
63.         getdig = digoff[offalg];
64.         getdh = dhoff[offalg];
65.     }
66.     else
67.     {
68.         //void *memset(void *s, int ch, size_t n);将s中前n个字节替换为ch并返回s;
69.         //将clientsig[4]开始的4个字节替换为0
70.         //这是C1的字段
71.         memset(&clientsig[4], 0, 4);
72.     }
73.
74.     /* generate random data */
75. #ifdef _DEBUG
76.     //将clientsig+8开始的1528个字节替换为0（这是一种简单的方法）
77.     //这是C1中的random字段
78.     memset(clientsig+8, 0, RTMP_SIG_SIZE-8);
79. #else
80.     //实际中使用rand()循环生成1528字节的伪随机数
81.     ip = (int32_t *) (clientsig+8);
82.     for (i = 2; i < RTMP_SIG_SIZE/4; i++)
83.         *ip++ = rand();
84. #endif
85.
86.     /* set handshake digest */
87.     if (FP9HandShake)
88.     {
89.         if (encrypted)
90.         {
91.             /* generate Diffie-Hellmann parameters */
92.             r->Link.dh = DHInit(1024);
93.             if (!r->Link.dh)
94.             {
95.                 RTMP_Log(RTMP_LOGERROR, "%s: Couldn't initialize Diffie-Hellmann!",
96.                     __FUNCTION__);
97.                 return FALSE;
98.             }
99.
100.             dhposClient = getdh(clientsig, RTMP_SIG_SIZE);
101.             RTMP_Log(RTMP_LOGDEBUG, "%s: DH pubkey position: %d", __FUNCTION__, dhposClient);
102.
103.             if (!DHGenerateKey((DH *) r->Link.dh))
104.             {
105.                 RTMP_Log(RTMP_LOGERROR, "%s: Couldn't generate Diffie-Hellmann public key!",
106.                     __FUNCTION__);
107.                 return FALSE;
108.             }
109.
110.             if (!DHGetPublicKey((DH *) r->Link.dh, &clientsig[dhposClient], 128))
111.             {
112.                 RTMP_Log(RTMP_LOGERROR, "%s: Couldn't write public key!", __FUNCTION__);
113.                 return FALSE;
114.             }

```

```

115.     }
116.
117.     digestPosClient = getdig(clientsig, RTMP_SIG_SIZE); /* reuse this value in verification */
118.     RTMP_Log(RTMP_LOGDEBUG, "%s: Client digest offset: %d", __FUNCTION__,
119.     digestPosClient);
120.
121.     CalculateDigest(digestPosClient, clientsig, GenuineFPKey, 30,
122.     &clientsig[digestPosClient]);
123.
124.     RTMP_Log(RTMP_LOGDEBUG, "%s: Initial client digest: ", __FUNCTION__);
125.     RTMP_LogHex(RTMP_LOGDEBUG, clientsig + digestPosClient,
126.     SHA256_DIGEST_LENGTH);
127. }
128.
129. #ifdef _DEBUG
130.     RTMP_Log(RTMP_LOGDEBUG, "Clientsig: ");
131.     RTMP_LogHex(RTMP_LOGDEBUG, clientsig, RTMP_SIG_SIZE);
132. #endif
133.     //发送数据报C0+C1
134.     //从clientsig-1开始发, 长度1536+1, 两个包含并
135.     //握手-----
136.     r->dlg->AppendCInfo("建立连接: 第1次连接。发送握手数据C0+C1");
137.     //-----
138.     if (!WriteN(r, (char *)clientsig-1, RTMP_SIG_SIZE + 1))
139.         return FALSE;
140.     //读取数据报, 长度1, 存入type
141.     //是服务器的S0, 表示服务器使用的RTMP版本
142.     if (ReadN(r, (char *)&type, 1) != 1) /* 0x03 or 0x06 */
143.         return FALSE;
144.     //握手-----
145.     r->dlg->AppendCInfo("建立连接: 第1次连接。接收握手数据S0");
146.     //-----
147.     RTMP_Log(RTMP_LOGDEBUG, "%s: Type Answer : %02X", __FUNCTION__, type);
148.     //客户端要求的版本和服务器提供的版本不同
149.     if (type != clientsig[-1])
150.         RTMP_Log(RTMP_LOGWARNING, "%s: Type mismatch: client sent %d, server answered %d",
151.         __FUNCTION__, clientsig[-1], type);
152.     //握手-----
153.     r->dlg->AppendCInfo("建立连接: 第1次连接。成功接收握手数据S0, 服务器和客户端版本相同");
154.     //-----
155.     //客户端和服务端随机序列长度是否相同
156.     //握手-----
157.     r->dlg->AppendCInfo("建立连接: 第1次连接。接收握手数据S1");
158.     //-----
159.     if (ReadN(r, (char *)serversig, RTMP_SIG_SIZE) != RTMP_SIG_SIZE)
160.         return FALSE;
161.
162.     /* decode server response */
163.     //把serversig的前四个字节赋值给uptime
164.     memcpy(&uptime, serversig, 4);
165.     //大端转小端
166.     uptime = ntohl(uptime);
167.
168.     RTMP_Log(RTMP_LOGDEBUG, "%s: Server Uptime : %d", __FUNCTION__, uptime);
169.     RTMP_Log(RTMP_LOGDEBUG, "%s: FMS Version : %d.%d.%d", __FUNCTION__, serversig[4],
170.     serversig[5], serversig[6], serversig[7]);
171.
172.     if (FP9HandShake && type == 3 && !serversig[4])
173.         FP9HandShake = FALSE;
174.
175. #ifdef _DEBUG
176.     RTMP_Log(RTMP_LOGDEBUG, "Server signature:");
177.     RTMP_LogHex(RTMP_LOGDEBUG, serversig, RTMP_SIG_SIZE);
178. #endif
179.
180.     if (FP9HandShake)
181.     {
182.         uint8_t digestResp[SHA256_DIGEST_LENGTH];
183.         uint8_t *signatureResp = NULL;
184.
185.         /* we have to use this signature now to find the correct algorithms for getting the digest and DH positions */
186.         int digestPosServer = getdig(serversig, RTMP_SIG_SIZE);
187.
188.         if (!VerifyDigest(digestPosServer, serversig, GenuineFMSKey, 36))
189.         {
190.             RTMP_Log(RTMP_LOGWARNING, "Trying different position for server digest!");
191.             offalg ^= 1;
192.             getdig = digoff[offalg];
193.             getdh = dhoff[offalg];
194.             digestPosServer = getdig(serversig, RTMP_SIG_SIZE);
195.
196.             if (!VerifyDigest(digestPosServer, serversig, GenuineFMSKey, 36))
197.             {
198.                 RTMP_Log(RTMP_LOGERROR, "Couldn't verify the server digest"); /* continuing anyway will probably fail */
199.                 return FALSE;
200.             }
201.         }
202.
203.         /* generate SWFVerification token (SHA256 HMAC hash of decompressed SWF, key are the last 32 bytes of the server handshake) */
204.         if (r->Link.SWFSize)
205.

```

```

206.     const char swfVerify[] = { 0x01, 0x01 };
207.     char *vend = r->Link.SWFVerificationResponse+sizeof(r->Link.SWFVerificationResponse);
208.
209.     memcpy(r->Link.SWFVerificationResponse, swfVerify, 2);
210.     AMF_EncodeInt32(&r->Link.SWFVerificationResponse[2], vend, r->Link.SWFSize);
211.     AMF_EncodeInt32(&r->Link.SWFVerificationResponse[6], vend, r->Link.SWFSize);
212.     HMACsha256(r->Link.SWFHash, SHA256_DIGEST_LENGTH,
213.         &serversig[RTMP_SIG_SIZE - SHA256_DIGEST_LENGTH],
214.         SHA256_DIGEST_LENGTH,
215.         (uint8_t *)&r->Link.SWFVerificationResponse[10]);
216. }
217.
218. /* do Diffie-Hellmann Key exchange for encrypted RTMP */
219. if (encrypted)
220. {
221.     /* compute secret key */
222.     uint8_t secretKey[128] = { 0 };
223.     int len, dhposServer;
224.
225.     dhposServer = getdh(serversig, RTMP_SIG_SIZE);
226.     RTMP_Log(RTMP_LOGDEBUG, "%s: Server DH public key offset: %d", __FUNCTION__,
227.         dhposServer);
228.     len = DHComputeSharedSecretKey((DH *)r->Link.dh, &serversig[dhposServer],
229.         128, secretKey);
230.     if (len < 0)
231.     {
232.         RTMP_Log(RTMP_LOGDEBUG, "%s: Wrong secret key position!", __FUNCTION__);
233.         return FALSE;
234.     }
235.
236.     RTMP_Log(RTMP_LOGDEBUG, "%s: Secret key: ", __FUNCTION__);
237.     RTMP_LogHex(RTMP_LOGDEBUG, secretKey, 128);
238.
239.     InitRC4Encryption(secretKey,
240.         (uint8_t *) & serversig[dhposServer],
241.         (uint8_t *) & clientsig[dhposClient],
242.         &keyIn, &keyOut);
243. }
244.
245.
246.     reply = client2;
247. #ifdef _DEBUG
248.     memset(reply, 0xff, RTMP_SIG_SIZE);
249. #else
250.     ip = (int32_t *)reply;
251.     for (i = 0; i < RTMP_SIG_SIZE/4; i++)
252.         *ip++ = rand();
253. #endif
254.     /* calculate response now */
255.     signatureResp = reply+RTMP_SIG_SIZE-SHA256_DIGEST_LENGTH;
256.
257.     HMACsha256(&serversig[digestPosServer], SHA256_DIGEST_LENGTH,
258.         GenuineFPKey, sizeof(GenuineFPKey), digestResp);
259.     HMACsha256(reply, RTMP_SIG_SIZE - SHA256_DIGEST_LENGTH, digestResp,
260.         SHA256_DIGEST_LENGTH, signatureResp);
261.
262.     /* some info output */
263.     RTMP_Log(RTMP_LOGDEBUG,
264.         "%s: Calculated digest key from secure key and server digest: ",
265.         __FUNCTION__);
266.     RTMP_LogHex(RTMP_LOGDEBUG, digestResp, SHA256_DIGEST_LENGTH);
267.
268. #ifdef FP10
269.     if (type == 8 )
270.     {
271.         uint8_t *dptr = digestResp;
272.         uint8_t *sig = signatureResp;
273.         /* encrypt signatureResp */
274.         for (i=0; i<SHA256_DIGEST_LENGTH; i+=8)
275.             rtmp8_sig(sig+i, sig+i, dptr[i] % 15);
276.     }
277. #if 0
278.     else if (type == 9))
279.     {
280.         uint8_t *dptr = digestResp;
281.         uint8_t *sig = signatureResp;
282.         /* encrypt signatureResp */
283.         for (i=0; i<SHA256_DIGEST_LENGTH; i+=8)
284.             rtmp9_sig(sig+i, sig+i, dptr[i] % 15);
285.     }
286. #endif
287. #endif
288.     RTMP_Log(RTMP_LOGDEBUG, "%s: Client signature calculated:", __FUNCTION__);
289.     RTMP_LogHex(RTMP_LOGDEBUG, signatureResp, SHA256_DIGEST_LENGTH);
290. }
291. else
292. {
293.     //直接赋值
294.     reply = serversig;
295. #if 0
296.     uintime = htonl(RTMP_GetTime());

```

```

297.         memcpy(reply+4, &uptime, 4);
298.     #endif
299. }
300.
301. #ifdef _DEBUG
302.     RTMP_Log(RTMP_LOGDEBUG, "%s: Sending handshake response: ",
303.         __FUNCTION__);
304.     RTMP_LogHex(RTMP_LOGDEBUG, reply, RTMP_SIG_SIZE);
305. #endif
306.     //把reply中的1536字节数据发送出去
307.     //对应C2
308.     //握手-----
309.     r->dlg->AppendCInfo("建立连接：第1次连接。发送握手数据C2");
310.     //-----
311.     if (!WriteN(r, (char *)reply, RTMP_SIG_SIZE))
312.         return FALSE;
313.
314.     /* 2nd part of handshake */
315.     //读取1536字节数据到serversig
316.     //握手-----
317.     r->dlg->AppendCInfo("建立连接：第1次连接。读取握手数据S2");
318.     //-----
319.     if (ReadN(r, (char *)serversig, RTMP_SIG_SIZE) != RTMP_SIG_SIZE)
320.         return FALSE;
321.
322. #ifdef _DEBUG
323.     RTMP_Log(RTMP_LOGDEBUG, "%s: 2nd handshake: ", __FUNCTION__);
324.     RTMP_LogHex(RTMP_LOGDEBUG, serversig, RTMP_SIG_SIZE);
325. #endif
326.
327.     if (FP9HandShake)
328.     {
329.         uint8_t signature[SHA256_DIGEST_LENGTH];
330.         uint8_t digest[SHA256_DIGEST_LENGTH];
331.
332.         if (serversig[4] == 0 && serversig[5] == 0 && serversig[6] == 0
333.             && serversig[7] == 0)
334.         {
335.             RTMP_Log(RTMP_LOGDEBUG,
336.                 "%s: Wait, did the server just refuse signed authentication?",
337.                 __FUNCTION__);
338.         }
339.         RTMP_Log(RTMP_LOGDEBUG, "%s: Server sent signature:", __FUNCTION__);
340.         RTMP_LogHex(RTMP_LOGDEBUG, &serversig[RTMP_SIG_SIZE - SHA256_DIGEST_LENGTH],
341.             SHA256_DIGEST_LENGTH);
342.
343.         /* verify server response */
344.         HMACsha256(&clientsig[digestPosClient], SHA256_DIGEST_LENGTH,
345.             GenuineFMSKey, sizeof(GenuineFMSKey), digest);
346.         HMACsha256(serversig, RTMP_SIG_SIZE - SHA256_DIGEST_LENGTH, digest,
347.             SHA256_DIGEST_LENGTH, signature);
348.
349.         /* show some information */
350.         RTMP_Log(RTMP_LOGDEBUG, "%s: Digest key: ", __FUNCTION__);
351.         RTMP_LogHex(RTMP_LOGDEBUG, digest, SHA256_DIGEST_LENGTH);
352.
353. #ifdef FP10
354.         if (type == 8 )
355.         {
356.             uint8_t *dptr = digest;
357.             uint8_t *sig = signature;
358.             /* encrypt signature */
359.             for (i=0; i<SHA256_DIGEST_LENGTH; i+=8)
360.                 rtmpe8_sig(sig+i, sig+i, dptr[i] % 15);
361.         }
362. #if 0
363.         else if (type == 9)
364.         {
365.             uint8_t *dptr = digest;
366.             uint8_t *sig = signature;
367.             /* encrypt signatureResp */
368.             for (i=0; i<SHA256_DIGEST_LENGTH; i+=8)
369.                 rtmpe9_sig(sig+i, sig+i, dptr[i] % 15);
370.         }
371. #endif
372. #endif
373.         RTMP_Log(RTMP_LOGDEBUG, "%s: Signature calculated:", __FUNCTION__);
374.         RTMP_LogHex(RTMP_LOGDEBUG, signature, SHA256_DIGEST_LENGTH);
375.         if (memcmp
376.             (signature, &serversig[RTMP_SIG_SIZE - SHA256_DIGEST_LENGTH],
377.                 SHA256_DIGEST_LENGTH) != 0)
378.         {
379.             RTMP_Log(RTMP_LOGWARNING, "%s: Server not genuine Adobe!", __FUNCTION__);
380.             return FALSE;
381.         }
382.         else
383.         {
384.             RTMP_Log(RTMP_LOGDEBUG, "%s: Genuine Adobe Flash Media Server", __FUNCTION__);
385.         }
386.
387.         if (encrypted)

```

```

388.     {
389.         char buff[RTMP_SIG_SIZE];
390.         /* set keys for encryption from now on */
391.         r->Link.rc4keyIn = keyIn;
392.         r->Link.rc4keyOut = keyOut;
393.
394.
395.         /* update the keystreams */
396.         if (r->Link.rc4keyIn)
397.         {
398.             RC4_encrypt((RC4_KEY *)r->Link.rc4keyIn, RTMP_SIG_SIZE, (uint8_t *) buff);
399.         }
400.
401.         if (r->Link.rc4keyOut)
402.         {
403.             RC4_encrypt((RC4_KEY *)r->Link.rc4keyOut, RTMP_SIG_SIZE, (uint8_t *) buff);
404.         }
405.     }
406. }
407. else
408. {
409.     //int memcmp(const void *buf1, const void *buf2, unsigned int count); 当buf1=buf2时, 返回值=0
410.     //比较serversig和clientsig是否相等
411.     //握手-----
412.     r->dlg->AppendCInfo("建立连接：第1次连接。比较握手数据签名");
413.     //-----
414.     if (memcmp(serversig, clientsig, RTMP_SIG_SIZE) != 0)
415.     {
416.         //握手-----
417.         r->dlg->AppendCInfo("建立连接：第1次连接。握手数据签名不匹配!");
418.         //-----
419.         RTMP_Log(RTMP_LOGWARNING, "%s: client signature does not match!",
420.             __FUNCTION__);
421.     }
422. }
423. //握手-----
424. r->dlg->AppendCInfo("建立连接：第1次连接。握手成功");
425. //-----
426. RTMP_Log(RTMP_LOGDEBUG, "%s: Handshaking finished...", __FUNCTION__);
427. return TRUE;
428. }

```

rtmpdump源代码 (Linux) : <http://download.csdn.net/detail/leixiaohua1020/6376561>

rtmpdump源代码 (VC 2005 工程) : <http://download.csdn.net/detail/leixiaohua1020/6563163>

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/12954329>

文章标签： [RTMPdump](#) [rtmp](#) [源代码](#) [握手](#) [连接](#)

个人分类： [libRTMP](#)

所属专栏： [开源多媒体项目源代码分析](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com