Google CTF 2018

DM Collision Writeup

shediyo @ PseudoRandom

From the challenge code I see that I have to find a collision and a preimage for 0 for the compression fuction.

The compression function is a Davis-Mayer compression from 16 to 8 bytes through interpretation of the 16-byte block as a key K of 8 bytes and plaintext P of 8 bytes and the output block is $DM(K||P) = DES'_K(P) \oplus P$, where DES' is a modified DES cipher for which the exact implementation is also given. ($\oplus$ is the xor operation, || is concetanation)

<u>Finding a collision</u>

To find a collision one needs to find plaintexts P, P' and keys K, K' so that:

$$DM(K||P) = DM(K'||P') \Rightarrow DES'_K(P) \oplus P = DES'_{K'}(P') \oplus P' \Rightarrow$$

$$P \oplus P' = DES'_K(P) \oplus DES'_{K'}(P')$$

In the case $P = P'$ the goal is simplified:

$$DES'_K(P) \oplus DES'_{K'}(P) = 0 \implies DES'_K(P) = DES'_{K'}(P)$$

That is, finding two different keys $K \neq K'$ for which the DES' encryption gives the same output for the same plaintext P.

Looking at the key scheduling algorithm:

```
  # Only 56 bits are used. A bit in each byte is reserved for parity checks.
  C = [key[PC1_C[i] - 1] for i in range(28)]
  D = [key[PC1_D[i] - 1] for i in range(28)]
```

Since only 56 bits of the key are used, but the input key is 64 bits – we can swap one of the unused bits in any key K and get a key K' for which we get the same result.

For example:

$$P = 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, ,0xff$$

$$K = 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, ,0xff$$

$$K' = 0xfe, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, ,0xff$$

<u>Finding a 0-preimage</u>

To find a 0-preimage one needs to find a plaintext P and a key K so that:

$$DM(K||P) = 0 \Rightarrow DES_K(P) \oplus P = 0 \Rightarrow DES'_K(P) = P$$

DES is a Feistel-Network (FN) cipher, and the input of 64 bits is split after a permutation to 2 blocks of 32 bits, left and right - (L, R).

In round i – for input (L, R), the output is $(R, L \oplus F_{K_i}(R))$, where K_i is the round key for round i.

Choosing $L = R$ in round i the transition is $(R, R) \to (R, R \oplus F_{K_i}(R))$.

The goal can now be changed for finding a 32-bit input R and a 64-bit key K for which $F_{K_i}(R) = 0$ for each round i, that's because the FN will be doing no effect $(R, R) \to (R, R) \to \cdots \to (R, R)$.

Looking again at the key scheduling algorithm, I see that the round keys are just rotations of 2 blocks of 28 bits:

```
for ri in range(16):
  C = LeftShift(C, KS_SHIFTS[ri])
  D = LeftShift(D, KS_SHIFTS[ri])

  CD = Concat(C, D)
  ki = [CD[PC2[i] - 1] for i in range(48)]
  yield ki
```

Choosing C as 0^{28} or 1^{28} and D as 0^{28} or 1^{28} we get the same round key for each round. (notation - 0^x is a concatenation of x zeroes)

So, for 4 special keys (after permutation) $0^{56}, 1^{56}, 0^{28}||1^{28}, 1^{28}||0^{28}$ the goal is even simpler - finding R for which $F_K(R) = 0$.

Now I look at the cipher function:

```
# Confusion step.
res = Xor(Expand(inp), key)
sbox_out = []
for si in range(48 // 6):
  sbox_inp = res[6 * si:6 * si + 6]
  sbox = SBOXES[si]
  row = (int(sbox_inp[0]) << 1) + int(sbox_inp[-1])
  col = int(''.join([str(b) for b in sbox_inp[1:5]]), 2)

  bits = bin(sbox[row][col])[2:]
  bits = '0' * (4 - len(bits)) + bits
  sbox_out += [int(b) for b in bits]

# Diffusion step.
res = sbox_out
res = [res[P[i] - 1] for i in range(32)]
return res
```

The cipher function in 3 steps:

1. The input is expanded from 32 bits to an expanded input of 48 bits via $Expand(R) \oplus K$
2. The expanded input passes as eight 6-bit blocks through sboxes which are from 6 to 4 bits, outputting 32 bits (eight 4-bit blocks).
3. The output is bitwise-permuted.

If in step 2 I'll have a 0 output from each sbox, in step 3 the permutation will have no effect and I'll still have a 0 output from the cipher.

I saw that there are 4 preimages of 0 for each sbox, and so $4^8 = 2^{16}$ preimages of 0 in step 2 (as expanded input).

To succeed finding a cipher preimage I have a constraint from step 1 - the sboxes preimage must be an expansion of a 32-bit input, that is $Preimage \oplus K = Expand(R)$ for some R, otherwise I won't get a valid input for the cipher function.

My strategy of checking whether I have such an input is recursively finding the preimages for each sbox, and then passing the known bits as constraints to the following recursive solution of the next sbox, until I get a full legitimate input.

Implemented in the following way:

```
def rec_preimage_cipher(stage, first_val, second_val, val_32, val_1,
key=(0,0)):
  sbox_inp = [-1, -1, -1, -1, -1, -1]
  sbox_inp[0], sbox_inp[1] = first_val, second_val

  for i in range(2 ** 4):
    sbox_inp[2], sbox_inp[3] = (i % 2), ((i // 2) % 2)
    sbox_inp[4] = val_32 if stage == 7 else ((i // 4) % 2)
    sbox_inp[5] = val_1 if stage == 7 else ((i // 8) % 2)
    keyed_sbox_inp = [0] * 6
    for j in range(6):
      keyed_sbox_inp[j] = sbox_inp[j] ^ key[stage // 4]
    sbox = SBOXES[stage]
    row = (int(keyed_sbox_inp[0]) << 1) + int(keyed_sbox_inp[-1])
    col = int(''.join([str(b) for b in keyed_sbox_inp[1:5]]), 2)

    if sbox[row][col] == 0:
      if stage == 7:
        return sbox_inp
      mid_res = rec_preimage_cipher(stage + 1, sbox_inp[4],
sbox_inp[5], val_32, val_1, key)
      if mid_res is not None:
        return sbox_inp + mid_res

  return None

def preimage_cipher():
  chosen_input, final_key = None, None
  for key in [(0,0), (0,1), (1,0), (1,1)]:
    for t in [(0,0), (0,1), (1,0), (1,1)]:
      res = rec_preimage_cipher(0, t[0], t[1], t[0], t[1], key)
      if res is not None:
        chosen_input = res
        final_key = key
        break
```

I got one good preimage input and key:

$$Input = 0x7b, 0xf6, 0x29, 0x85, 0x7b, 0xf6, 0x29, 0x85$$

$$Key = 0xff, 0xff, 0xff, 0xff, 0, 0, 0, 0$$

Both of them are after permutations, inversing the permutations I got:

$$P = DES_K(P) = 0xcf, 0xf0, 0x33, 0xcc, 0xf0, 0xfc, 0xf0, 0x33$$

$$K = 0xe1, 0xe1, 0xe1, 0xe1, 0xf0, 0xf0, 0xf0, 0xf0$$

After sending all the results to the server I got the flag:

CTF{7h3r35 4 f1r3 574r71n6 1n my h34r7 r34ch1n6 4 f3v3r p17ch 4nd 175 br1n61n6 m3 0u7 7h3 d4rk}