otp

- 生成プログラムotp.binを逆アセンブルしてみると、sha256関連の関数が含まれていた。
- よって、sha256がOTPの生成に大きな役割を果していると考えられるので、METASMを用いてフックして実行してみた。

切換行號

```
require 'metasm'

process = Metasm::OS.current.create_process('./otp.bin -g token65.bin')
debugger = process.debugger
debugger.bpx(0x40613a, false) do
  puts "SHA256_hash:"
  addr = debugger[:rdi]
  size = debugger[:rsi]
  @addr2 = debugger[:rdx]
  p ['%x'%addr,size]
  p debugger[addr,size].unpack("H*")
end
debugger.bpx(0x4061ae, false) do
  puts "result: %s" % debugger[@addr2,32].unpack("H*")
end
debugger.run_forever
```

- すると、OTPは次のような形式となっていることが分かった。

```
16文字：ユーザー名+secretの上位7byteをそれぞれ1byte毎にsha256sumしてxorしたもの
16文字：sequenceを16進数表記にしたもの
32文字：sequence + secretのsha256sumの先頭32文字
```

- 解析した生成部をRubyプログラムに直すと次のようになる。

切換行號

```
require 'digest/sha2'
require 'facets'
def gen(user, secret, seq)
  s = "\0" * (user.size + 8)
  d = "\0" * 8
  user.chars.each.with_index do |c,i|
    s[i] = c
    digest = Digest::SHA256.digest(s)
    d ^= digest
    s[i] = "\0"
  end
  8.times do |i|
    next if i == 0
    s[user.size + i] = if i == 0
                         0.chr
                       else
                         ((secret >> (i*8)) & 255).chr
                       end
    digest = Digest::SHA256.digest(s)
    d ^= digest
    s[user.size + i] = "\0"
  end
  f_hash = d.unpack("H*")[0]

  sequence = [seq].pack("q").unpack("H*")[0]

  s_hash = Digest::SHA256.hexdigest([seq].pack("q") + [secret].pack("q"))

  f_hash + sequence + s_hash[0,32]
end

p gen('nomeaning',2459565876494672537,37)
```

- adminのsecretを特定する必要がある。
- まず先頭の16文字のsignatureから、7byteを半分全列挙しつつ全探索することで特定できる。
- 次のようなプログラムで特定した。

切換行號

```ruby
require 'digest/sha2'
require 'facets'
target = ["9ae684ca583214d3"].pack("H*") # 目的とするHash
"admin".chars.each.with_index do |c,i| # ユーザー名によるHashを打ち消す
  s = "\0" * 13
  s[i] = c
  target ^= Digest::SHA256.digest(s)
end

# 1792 vars equation
matrix = Array.new(128){Array.new(1793)}
(0...7).each do |pos|
  (0...256).each do |char|
    s = "\0" * 13
    s[pos + 6] = char.chr
    digest = Digest::SHA256.digest(s)[0,8]
    # 256bit
    print '%d' % digest.unpack("H*")[0].to_i(16)
    print ' '
  end
  puts
end

puts target.unpack("H*")[0].to_i(16)
```

切換行號

```cpp
#include <bitset>
#include <algorithm>
#include <iostream>
#include <vector>
#include <cstdint>
using namespace std;
unsigned long long a[7][256], target;
unsigned long long list[256*256*256];
unsigned long long result;
int sel[7];
int from, to;
int search(int pos, unsigned long long cur) {
  if(pos == 7) {
    if(binary_search(list, list+256*256*256, cur)) {
      cout << "-----" << endl;
      cout << cur << endl;
      for(int i = 0; i < 7; i++) {
        cout << sel[i] << endl;
      }
      result =  cur;
      cout << "-----" << endl;
    }
    return 1;
  }
  for(int i = 0; i < 256; i++) {
    if(pos == 3 && !(from <= i && i < to))continue;
    sel[pos] = i;
    search(pos + 1, cur ^ a[pos][i]);
    if(pos == 3) cout << "search:" << i << endl;
  }
  return 1;
}

int main(int argc, char **argv) {
  from = atoi(argv[1]);
  to = atoi(argv[2]);
  cerr << from << "," << to << endl;
  for(int i = 0; i < 7; i++) {
```

```
    for(int j = 0; j < 256; j++) {
      cin >> a[i][j];
    }
  }
  cin >> target;
  // 3つを選択する
  for(int i = 0; i < 256; i++) {
    for(int j = 0; j < 256; j++) {
      for(int k = 0; k < 256; k++) {
        list[i*256*256+j*256+k] = a[0][i] ^ a[1][j] ^ a[2][k];
      }
    }
  }
  sort(list, list+256*256*256);
  search(3, target);
  for(int i = 0; i < 256; i++) {
    for(int j = 0; j < 256; j++) {
      for(int k = 0; k < 256; k++) {
        if(result == (a[0][i] ^ a[1][j] ^ a[2][k])) {
          cout << i << endl << j << endl << k << endl;
        }
      }
    }
  }
}
```

```
$ ruby solve.rb | ./search 0 256
...
-----
9807512490306708300
0
0
0
140
153
187
73
-----
search:140
...
193
14
67
```

- 残りの1byteは後ろのハッシュとマッチするものを探索すれば良い。次のプログラムでsecretを求めてさらに次のOTPを表示した。

切換行號

```
secret = [193,14,67,140,153,187,73].reverse.pack("C*").unpack("H*")[0].to_i(16)*256
while true
  break if gen('admin',secret,1337) ==
'9ae684ca583214d339050000000000000fd635dded0bbb40e162da79fba55ae32'
  secret += 1
end
puts gen('admin', secret, 1338)
```

```
$ ruby gen.rb
9ae684ca583214d33a0500000000000030949b105cb796db1a0488099b684373
```

このOTPでログインしたところ、フラグ31C3_a7e3683344e954efd8a58a2a3da7fbe8が得られた。

CTF/Writeup/31C3 CTF/otp (上次是 nomeaning 在 2014-12-30 14:15:11 編輯的)