Subscribe to RSS feed

Search Search

More Smoked Leet Chicken

We pwn CTFs

- Home
- rfCTF 2011
- Hack.lu 2010 CTF write-ups
- Leet More 2010 write-ups

« Codegate 2014 Quals – Angry Doraemon (pwn 250)

PlaidCTF 2014 RSA writeup »

Mar
02

Boston Key Party CTF – Differential Power (Crypto 400)

- Writeups

by hellman

> we hooked up a power meter to this encryption box. we don't know the key. that's what we want to know. you can encrypt any string of 8 characters on the service http://54.218.22.41:6969/string_to_encrypt

encrypt.asm
chall source (released after ctf)

This was cool challenge about differential power analysis. Actually, it was rather simplified and we simply got number of flipped bits during each instruction.

The code was rather simple:

```
add $t1, $zero, $zero# clear out $t1 ; 00004820
addi $t1, $t1, 0x9e# TEA magic is 0x9e3779b7 ; 2129009E
sll $t1, $t1, 8# shift out making room in the bottom 4; 00094a00
addi $t1, $t1, 0x37 ; 21290037
sll $t1, $t1, 8 ; 00094a00
addi $t1, $t1, 0x79 ; 21290079
sll $t1, $t1, 8 ; 00094a00
addi $t1, $t1, 0xb9 # now $t1 holds the magic 0x9e3779b9 ; 212900b9
add $t2, $zero, $zero# $t2 is the counter ; 00005020
add $t0, $zero, $zero# $t0 is the sum ; 00004020
lw $t8, $zero, 8# k0 mem[8-23] = k ; 8c180008
lw $s7, $zero, 12# k1 ; 8C17000C
lw $s6, $zero, 16# k2 ; 8C160010
lw $t3, $zero, 20# k3 now our keys are in registers ;  8c0b0014
```

```
14  lw $t7, $zero, 0# v0 mem[0-7] = v ; 8c0f0000
15  lw $t6, $zero, 4# v1, our plaintext is in the registers ; 8c0e0004
16  loop: add $t0, $t0, $t1# sum+=delta ; 01094020
17  sll $s4, $t6, 4# (v1 << 4) ; 000ea100
18  add $s4, $s4, $t8# +k0  part 1 is in s4 ; 0298a020
19  add $s3, $t6, $t0# (v1 + sum) part 2 is in s3 ; 01c89820
20  srl $s2, $t6, 5# (v1 >> 5) ; 000e9142
21  add $s2, $s2, $s7# +k1, now do the xors part 3 in s2 ; 02579020
22  xor $s1, $s2, $s3# xor 2 and 3 parts ; 02728826
23  xor $s1, $s1, $s4# xor 1(2,3) ; 2348826
24  add $t7, $t7, $s1# done with line 2 of the tea loop ; 01f17820
25  sll $s4, $t7, 4# (v0 << 4) ; 000fa100
26  add $s4, $s4, $s6# +k2 part 1 in s4 ; 0296a020
27  add $s3, $t7, $t0# (v0 + sum) part 2 in s3  ; 01e89820
28  srl $s2, $t7, 5# (v0 >> 5) ; 000f9142
29  add $s2, $s2, $t3# +k3 part 2 in s2 ; 024b9020
30  xor $s1, $s2, $s3# xor 2 and 3 parts ; 2728826
31  xor $s1, $s1, $s4# xor 1(2,3) ; 2348826
32  add $t6, $t6, $s1# done with line 2! ; 01d17020
33  addi $s0, $zero, 32# for compare ; 20100020
34  addi $t2, $t2, 1# the counter ; 214a0001
35  bne $t2, $s0, 17# bne loop, now save back to the memory ; 15500010
```

Easy to see it's TEA cipher. We need to get 4 32bit keys (16 bytes overall).

If we sent a string to a web service, it responded with an array of values. Each value corresponded to a number of bits flipped during an instruction execution.

How can we extract key from the data?

Easy, we can make use of such instructions:

```
18  add $s4, $s4, $t8# +k0  part 1 is in s4 ; 0298a020
...
21  add $s2, $s2, $s7# +k1, now do the xors part 3 in s2 ; 02579020
...
```

We can make a guess for **k0**, **s4** is known, and we can check then number of flipped bits. We can repeat this a couple of times to narrow down the key space. Also we should use other instructions, like `22 xor $s1, $s2, $s3# xor 2 and 3 parts ; 02728826`, because that **add**'s depend only on some parts of plaintext (because of shifts) and therefore will not yield the whole information about the key.

Making guesses for 32bit number is not such fast. There are many ways from here: we can suppose that key is printable and narrow supposed charset more; use more effective way of guessing — putting a plaintexts with a few bits set and thus check whether there was a carry at some position. Also we can get number of "1" bits in keys from **lw** instructions:

```
10  lw $t8, $zero, 8# k0 mem[8-23] = k ; 8c180008
```

Anyway, this is rather messy and need some accuracy. Let's feed that to z3. Something like this:

```
from z3 import *

S = Solver()
k0, k1, k2, k3 = BitVecs("k0 k1 k2 k3", 32)
```

```
S.add(k0 & 0x80808080 == 0)
S.add(k1 & 0x80808080 == 0)
S.add(k2 & 0x80808080 == 0)
S.add(k3 & 0x80808080 == 0)
S.add(bitsum(k0) == k0bitsum)
S.add(bitsum(k1) == k1bitsum)
S.add(bitsum(k2) == k2bitsum)
S.add(bitsum(k3) == k3bitsum)
```

And then just translate instructions to code and add bitsum checks:

```
for s, data in known.items():
    v0 = unpack(">I", s[:4])[0]
    v1 = unpack(">I", s[4:8])[0]

    s0 = s1 = s2 = s3 = s4 = s5 = s6 = s7 = s8 = 0
    t0 = t1 = t2 = t3 = t4 = t5 = t6 = t7 = t8 = 0

    # 0  add $t1, $zero, $zero# clear out $t1 ; 00004820
    # 1  addi $t1, $t1, 0x9e# TEA magic is 0x9e3779b7 ; 2129009E
    # 2  sll $t1, $t1, 8# shift out making room in the bottom 4; 00094a00
    # 3  addi $t1, $t1, 0x37 ; 21290037
    # 4  sll $t1, $t1, 8 ; 00094a00

    # 5  addi $t1, $t1, 0x79 ; 21290079
    # 6  sll $t1, $t1, 8 ; 00094a00
    # 7  addi $t1, $t1, 0xb9 # now $t1 holds the magic 0x9e3779b9 ; 212900b9
    t1 = 0x9e3779b9

    # 8  add $t2, $zero, $zero# $t2 is the counter ; 00005020
    t2 = 0

    # 9  add $t0, $zero, $zero# $t0 is the sum ; 00004020
    t0 = 0

    ...

    # 16  loop: add $t0, $t0, $t1# sum+=delta ; 01094020
    t0 = (t0 + t1) & 0xffffffff

    # 17  sll $s4, $t6, 4# (v1 << 4) ; 000ea100
    s4 = (t6 << 4) & 0xffffffff

    # 18  add $s4, $s4, $t8# +k0  part 1 is in s4 ; 0298a020
    s4b = (s4 + t8) & 0xffffffff
    S.add(bitsum(s4b ^ s4) == data[18])
    s4 = s4b

    # 19  add $s3, $t6, $t0# (v1 + sum) part 2 is in s3 ; 01c89820
    s3b = (t6 + t0) & 0xffffffff
    S.add(bitsum(s3b ^ s3) == data[19])
    s3 = s3b

    ...
```

[Full script](#):

```
$ time py sat.py
10 collected
sat
[k2 = 1769241186,
 k1 = 1684368738,
 k3 = 1915756833,
```

```
 k0 = 1718380912]
0x666c6970 flip
0x64656d62 demb
0x69747a62 itzb
0x72302121 r0!!

real    0m35.580s
user    0m35.224s
sys     0m0.079s
```

So, the flag: “**flipdembitzbr0!!**“.

Tags: 2014, bkp, crypto, ctf, dpa, python, tea, z3

1 comment

1. 

   ***bowknotbowknot* says:**

   March 20, 2016 at 19:40 (UTC 3)

   Reply

   features for sport; music plus information.The nuvi is easily portable, compact as well as software is prime quality.It will help improves well being, through excitement and informative assistance, and is obtainable for many users despite age their brand.The main disadvantage is the possibility that there is significantly apparatus needed that features for sport; music plus information.The nuvi is easily portable, compact as well as software is prime quality.It will help improves well being, through excitement and informative assistance, and is obtainable for many users despite age their brand.The main disadvantage is the possibility that there is significantly apparatus needed that may be costly.Available requirements will be an ipod touch Nano, the Physical activities Kit and then the Nike + shoes or boots.Although different shoes can be acquired that have got a pouch with the sensor, these have a different point of view so never achieve precisely the same results.The other point worthy of noting is which the iPod Sporting events kit requires an individual to enjoy a good perception of computers to achieve the full advantages.This document is within GNU FDL license and can also be distributed which has no previous authorization with the author.Risk author’s identify and many of the URLs (links) mentioned while in the article along with biography need to be kept..

Leave a Reply

Your email address will not be published.

Message:

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <s> <strike> <strong>`

Name:

Email:

Website:

Submit Comment

Submit Comment

Archives

Meta

- Register
- Log in
- Entries RSS
- Comments RSS
- WordPress.org

Tags

2010 2011 2012 2013 2014 2016 aes aslr binary bruteforce c++ codegate crt crypto ctf defcon exploit exploitation formatstring gits hack.lu hacklu hash ictf leetmore libnum nuit du hack nx pctf plaid plaidctf ppp python quals reverse reversing rop rsa sage shellcode vm web writeup x64 xor