More Smoked Leet Chicken

We pwn CTFs

Apr
20

PlaidCTF 2014 wheeeee writeup

- Writeups

by hellman

> Although it seems like The Plague's messaging service is secure, there are bound to be bugs in any 20th century crypto system. We've recovered a version of the block cipher The Plague implemented. Use their online encryptor tool, at 54.82.75.29:8193, to break the cipher and figure out Plague's secret plans. NOTE: When the service sends you a hex-encoded string, respond with a hex-encoded string.

We are given a block cipher and an encryption oracle. Here's block encrypting code:

```
def encrypt_block(plaintext, key):
    txt = plaintext
    l, r = (txt >> M) & ((1 << M) - 1), txt & ((1 << M) - 1)
    for x in xrange(2 ** 24):
        if x % 2 == 0:

            l1 = r
            r1 = l ^ F(r, key[0])
            l, r = l1, r1
        else:
            l1 = l
            r1 = l ^ F2(r, key[1])
            l, r = l1, r1
    return l << M | r

def F(s, k):
    return f[s ^ k]
```

```
def F2(s, k):
    return f2[s ^ k]
```

It seems like a Feistel network but actually isn't (because of `l1 = l` in odd rounds). Also we can notice interesting things: block size is very small (24 bits) and number of rounds is really huge (2^24). There exists well known attack against weak ciphers with large number of rounds — slide attack.

The idea is to find a pair of plaintexts for which `one_cipher_round(p1, key) = p2`. Thus if one round is weak, we can extract key from it and then check the key using encryption oracle: `one_cipher_round(full_cipher(p1, key), key) = full_cipher(p2, key)` (it means that ciphertexts are also 1 round "away" from each other).

For this challenge, we can't take just "first" round for slide, because odd and even rounds are different. We need to take both odd and even rounds, e.g. find 2–round slide. How can we extract keys from such slide? Unfortunately, there can be 2^12 possible keys for each such pair:

- fix input for F2 (let's call it r1)
- r1 = F1(r0) ^ l0 — here we can get value for key[0]
- r2 = l2 ^ F2(r1) — here we can get value for key[1]

How can we find such pairs? Since block size is rather small, we can select such pairs randomly and check them all. But we can generate them in a more clever way: we know that for 2 round encryption we have `CipherLeft = PlainRight`. This means we need plaintexts with one's left half equal another's right. So let's fix some half value and generate N plaintexts with left half equal to the value and N plaintexts with right half equal to the value.

How much should be N? Due to birthday paradox, it should be near `sqrt(1<<12) ~= 64` for high probability. Let's generate 100 pairs of each kind.

For each two plaintext–ciphertext pairs, where plaintexts fit the latter property (p1right = p2left), we can make additional quick check – ciphertexts must fit the same property too (c1right = c2left). If this properties hold, we can then check extracted keys by simply encrypting plaintexts and checking against known ciphertexts.

Here's the code (asking service for encryption replaced with encrypting under random key):

```
def reverse(p1, p2):
    l0, r0 = split(p1)
    l2, r2 = split(p2)

    keys = set()
    for k1 in range(1 << M):
        f1out = F1[r0 ^ k1]
        f2in = l0 ^ f1out
        f2out = l2 ^ r2
        s_xor_k2 = F2i[f2out]
        k2 = s_xor_k2 ^ f2in
        keys.add((k1, k2))
    return keys

def split(txt):
```

```
def split(txt):
    l0, r0 = (txt >> M) & ((1 << M) - 1), txt & ((1 << M) - 1)
    return l0, r0


k_rand = tuple(gen_key())
m = {}
left_blocks = defaultdict(set)
right_blocks = defaultdict(set)
for i in range(200):
    if i & 1:
        # left half = random, right half = zero
        p = random.randint(0, (1 << M) - 1) << M
    else:
        # left half = zero,   right half = random
        p = random.randint(0, (1 << M) - 1)

    c = encrypt_block(p, k_rand)
    m[p] = c
    pl, pr = split(p)
    left_blocks[pl].add(p)
    right_blocks[pr].add(p)

for val in range(1 << M):
    for p1 in right_blocks[val]:
        for p2 in left_blocks[val]:
            c1 = m[p1]
            c2 = m[p2]
            ks1 = reverse(p1, p2)
            ks2 = reverse(c1, c2)

            l0, r0 = split(c1)
            l2, r2 = split(c2)
            if r0 != l2:
                continue

            for k in ks1 & ks2:
                if encrypt_block(p1, k) == c1:
                    print "Good key", k
                    assert k == k_rand
                    quit()
```

The flag was **"Gotta love it when you can SLIDE. The flage is id_almost_rather_be_sledding"**.

Tags: 2014, cipher, crypto, ctf, plaidctf, python, slide

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

- March 2017 (3)
- October 2016 (6)
- September 2016 (3)
- May 2016 (5)
- April 2016 (2)
- March 2016 (5)
- September 2015 (2)
- May 2015 (4)
- April 2014 (4)
- March 2014 (1)
- February 2014 (4)
- January 2014 (2)
- December 2013 (1)
- October 2013 (1)
- June 2013 (6)
- April 2013 (1)
- February 2013 (2)
- November 2012 (6)
- October 2012 (7)
- May 2012 (6)
- April 2012 (2)
- March 2012 (4)
- February 2012 (17)
- January 2012 (12)
- December 2011 (6)
- October 2011 (5)
- September 2011 (10)
- August 2011 (8)
- July 2011 (3)
- June 2011 (5)
- April 2011 (10)
- March 2011 (7)

- January 2011 (2)
- December 2010 (1)
- November 2010 (1)
- October 2010 (11)
- September 2010 (11)

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

Valid XHTML 1.0 Strict Valid CSS Level 2.1

More Smoked Leet Chicken uses Graphene theme by Syahir Hakim.