php dopt('d_post_comment_b'),*CTF2019国际赛事官方版writeup

转载

有话我职说 于 2021-03-18 05:52:31 发布 164 收藏
文章标签： php dopt('d_post_comment_b')

*CTF2019 offical writeup

pwn

upxofcpp

Idea

Make an easy UAF challenge under Ubuntu 16.04

Keep it fun

Vulnerability

Unpack the original file by upx upxofcpp

Simple UAF

Key points

PIE is enabled and there is no function prepared for information leakage

But it is packed by upx! Running the upx'ed program, you will find many rwx memory regions, including the heapThis ELF file is packed by upx-3.91 installed via apt install upx-ucl on Ubuntu 16.04

If you focus on the unpacked one, you will miss the rwx memory regions

Exploit

Fake the vtable via malloc_consolidate, write your shellcode on the heap

Trigger the UAF vulnerability by remove or show, then call rax will jump on the shellcode

Details in solve.py

It has nothing to do with libc :P

Flag

*ctf{its_time_to_say_goodbye_to_ubuntu_16_04}

OOB

It's a v8 challenge, related to my recently work.

The patch added a new function to Array object. You can use it to read and write elements off by one. It's very easy to make a type confusion bug because the you can use oob function to modify an object's map, which records the shape of an object. Then read and write out of boundary, construct arbitrary read and write, hijack pc.

solve

step1

Alloc array A

A = [1.1,1.2];

The memory layout of A:

| ... |

| slot 1 | <== A's element

| slot 2 |

| the map of A | <== the pointer of struct A

| ... |

| array length |

| ... |

|the pointer of A's element|

| ... |

Alloc JSObject B

B = {

"x1":1.1,

"x2":1.1,

"x3":1.1,

"x4":1.1,

}

The memory layout of B:

| the map of B | <== the pointer of struct B

| ... |

| length |

| solt1 | <== B's element

| slot2 |

| slot3 |

| slot4 |

step2

set A's map to B's map.

we can't use oob function to get B's map value, but the distance of A's map and B's map is a const value. Therefore, load A's map, add a const value and then store it to A's map.

step3

now v8 will treat A type as B's type, read and write out of boundary.

You can alloc a ArrayBuffer beside A, and overwrite its backingstore to construct arbitrary read and write.

step4

Alloc a Builtin Array Object beside A, there is PIE information inside Array's code pointer. Leak it to bypass ASLR, and find libc address, stack address.

step5

write rop to stack and use nop slide to hijack pc.

hackme

Idea

I read a blog at https://cyseclabs.com/blog/linux-kernel-heap-spray recently, it described a very powerful way to exploit some kernel vulnerabilities, so I want to use it for my challenge.

This challenge just like a normal heap challenge, but is implemented it in kernel space.

```
static long hackme_ioctl(struct file *file, unsigned int cmd, unsigned long arg){
union {
struct {uint32_t idx; void* ptr; uint64_t size; } add;
struct {uint32_t idx; } del;
struct {uint32_t idx; void* ptr; uint64_t size; uint64_t pc; } edit;
struct {uint32_t idx; void* ptr; uint64_t size; uint64_t pc; } show;
} u;
long ret;
copy_from_user(&u,(void*)arg,sizeof(u));
switch (cmd){
case ADD:
ret = add_note(u.add.idx,u.add.ptr,u.add.size);
break;
case DEL:
ret = del_note(u.del.idx);
break;
case EDIT:
```

```
ret = edit_note(u.edit.idx,u.edit.ptr,u.edit.size,u.edit.pc);

break;

case SHOW:

ret = show_note(u.show.idx,u.show.ptr,u.show.size,u.show.pc);

break;

default:

ret = -1;

break;

}

return ret;

}
```

the bug is in add and delete functions. At delete function, It only erases the pointer but keeps the old size, and there is no lock at add function. So a basic exploit as follows:

add then delete a large note to keep a big old size at a slot.

call add_note, malloc a small note and trap pc to user space at [A] using userfault_fd.

start a new thread and edit the victim note_unit, it causes read and write out of boundary at kernel heap space.

search ptmx struct and hijack it.

```
struct note_unit{

void* ptr;

uint64_t size;

};

static long add_note(uint32_t idx,void* ptr, uint64_t size){

void *tmp;

if(pool[idx].ptr)

return -1;

tmp = kmalloc(size,GFP_KERNEL);

if(!tmp)

return -1;

pool[idx].ptr = tmp;

copy_from_user(tmp,ptr,size); <== A

pool[idx].size = size;
return 0;
}
static long del_note(uint32_t idx){
if(pool[idx].ptr){
kfree(pool[idx].ptr);
pool[idx].ptr = 0;
return 0;
}else{
return -1;
}
}
```

Build kernel guide: https://gist.github.com/chrisdone/02e165a0004be33734ac2334f215380e

heap_master

Idea

heap master combined some ideas of 0ctf2018 heapstorm 2 and hitcon2018 baby tcache together.

It maped a random segment at init. You can edit it and free the pointer in the segment arbitrarily, but there is no show function. And malloc function seems very interesting, the value which malloc returned is never used again.

```
void add(){
printf("size: ");
uint64_t size = read_num();
if( size < 0 ){
puts("Invaild input");
return;
}
malloc(size);
}
```

solve

0ctf2018 heapstorm2 introduced a new method to write a heap pointer at arbitrary address using largebin attack when malloc. The bug code is in https://code.woboq.org/userspace/glibc/malloc/malloc.c.html#3858

```
if ((unsigned long) (size)
< (unsigned long) chunksize_nomask (bck->bk))
```

{

fwd = bck;

bck = bck->bk;

victim->fd_nextsize = fwd->fd;

victim->bk_nextsize = fwd->fd->bk_nextsize;

fwd->fd->bk_nextsize = victim->bk_nextsize->fd_nextsize = victim; <== no check

}

Unlike heapstorm2, there is no show function and I chroot the path so you must excute your own shellcode to read flag (no more one gadget), thats means you must leak some address information before expolit it, so I think it's obvious that you should play with FILE struct.

At babytache, Angelboy overwrites the stdout flags and IO_write buffer to leak libc. It's a very fresh way to bypass ASLR.

Here is expolit steps:

step 1

free large chunk to write main_arena unsortbin address to the new segment.

step 2

partial overwrite libc address, point it to stdout flag, use largebin attack(House of Storm) to overwrite stdout flag. if fp->flag & 0xa00 == 1 and fp->flag & 0x1000 == 1 then it will leak something when f->write_base != f->write_ptr

step4

partial overwrite libc address, point it to stdout write buffer, use largebin attack again to overwrite stdout the last byte of _IO_write_base to '\x00'.

Now program will print libc address and our segment address(we have write it to fp->flag) before print menu.

step5

overwrite _dl_open_hook just like step 3 & 4, it's a pointer of a function pointer, write our segment address in it.

step6

trigger libc malloc or free error, program will load _dl_open_hook to a register and call [reg]. In this libc, the register is rbx. And I find some gadgets in glibc as follows:

0x7FD7D: mov rdi, [rbx+48h]

mov rsi, r13

call qword ptr [rbx+40h]

Perfect! we control rbx and rdi now! Then jump to setcontext and control all registers.

girlfriend

Note

I long for love, but also like single life. But it's a little difficult to find a girlfriend by playing CTF.

idea

we compile glibc-2.29 which tcache will check double free. But it is also easy to bypass by filling 7 chuncks on it. Now you will double free by fastbin. You can malloc trunck near free_hook, and revise it to system, free a trunck ptr which point to "/bin/sh\x00" will get the shell. By the way, libc is easy to leak by malloc and delete a large chunck.

solve

check hack.py

quicksort

note

In the problem diion, "I'm very quick", I mean the algrithom, not myself.

idea

Tell me the array size and numbers one by one, and I'll sort it by quicksort. The vulnerability is that we read number by gets() which is eary to find. The canary is open but you can bypass it by fill sth in __stk_check_fil func on got table, like ret. We can overflow the variable first, and be careful with the got and stack variable

solve

hack.py

shellcode

A easy shellcode

idea

input string must in the string, char *s = "ZZJ loves shell_code,and here is a gift:\x0f\05 enjoy it!\n"; and will take over the execution flow. '\x0f\x05' is system, 'Z', '_' is pop rdi and pop rdx. You just need to write to read. But I made a a stupid mistake that I check the input with pointer, which is very easy to bypass by '\x00'....this problem is not that baby.

pop rdx

pop rdx

pop rdx

pop rdx

pop rdi

pop rdi

syscall

and here is a normal shellcode.

call here

.ascii "/bin/sh"

.byte 0

here:

pop rdi

xor rsi,rsi

xor rdx,rdx

mov rax,0x3b

syscall

reverse

matr1x

Idea

zzj wanna design a magic cube challenge, but it's too easy.

Then I added some code obfuscation and changed his check funtion.

the code confusion module is a modified version from https://yurichev.com/blog/58/, blogger implemented it on Window PE, I make it works on x86 platform.

There is five kinds of obfuscation.

PUSH JMP INSTEAD OF CALL

call func to

call . + $5

pop eax

add eax,0x0a

push eax

jmp func

POP JMP INSTEAD OF RET

ret to

pop ebx

jmp ebx

plz check the source code for other methods.

NOISE AROUND OPI

NOISE AFTER CALL

LOAD

yy

I learned Bison program on my Compiling Principle course. It provides a programable way to DIY your own parser, so I make a flag parser.

You must input like *CTF{word_word_word}, or it will give you syntax error.

And there is a encrypt function, it seems very complex and important.

set a break on encrypt, and input a flag like *CTF{word_word_word}, you will find that program calls this function three times. And at the start of encrypt, input data xor data from an array, at the end of encrypt, transformed data is stored in this array again. If you know crypto, you will recognize that it's a part of AES with CBC. The key is static and program only stored the transformed table of key. There is two ways to solve this AES, one is very hard and another is very easy.

Easy way

reverse the transformed table to get origin key. Then use AES library to decrypt origin data.

Hard way

reverse the last part of AES because it's a linear transform. I use z3 to help me, and just run the encrypt function backwards.

by the way, the origin data is not flag, but it's just a box transfrom, so here is my solutions:

```
from z3 import *

from pwn import *

with open("./table","r") as f:

table = map(ord,f.read())

def _hash(x,y,z):

return ( (27 * ((x^y) >> 7)) ^ ((x^y)<<1) ^ x ^ z ) & 0xff

def _hash_z3(x,y,z):

return ( (27 * LShR(x^y,7)) ^ ((x^y)<<1) ^ x ^ z ) & 0xff

def hash(a,b,c,d):

tmp = a^b^c^d

return _hash(a,b,tmp),_hash(b,c,tmp),_hash(c,d,tmp),_hash(d,a,tmp)

def dehash(x,y,z,w):

a=BitVec('a',8)

b=BitVec('b',8)

c=BitVec('c',8)

d=BitVec('d',8)

tmp = a^b^c^d

s=Solver()
```

```
s.add(_hash_z3(a,b,tmp) == x,\

_hash_z3(b,c,tmp) == y,\

_hash_z3(c,d,tmp) == z,\

_hash_z3(d,a,tmp) == w)

s.check()

return int(s.model()[a].as_string()),\

int(s.model()[b].as_string()),\

int(s.model()[c].as_string()),\

int(s.model()[d].as_string())

def AES_part2(_key,_flag):

assert len(_flag) == 0x10

flag = map(ord,_flag)

key = map(ord,_key)

flag = [ p^q for p,q in zip(flag,key[:0x10]) ]

for i in range(0x10,0xa0,0x10):

p = [table[x] for x in flag]

flag[0:4] = hash(p[0],p[5],p[10],p[15])

flag[4:8] = hash(p[4],p[9],p[14],p[3])

flag[8:12] = hash(p[8],p[13],p[2],p[7])

flag[12:16] = hash(p[12],p[1],p[6],p[11])

flag = [p^q for p,q in zip(flag,key[i:i+0x10])]

ret = ['']*0x10

ret[0] = table[flag[0]]

ret[1] = table[flag[5]]

ret[2] = table[flag[10]]

ret[3] = table[flag[15]]

ret[4] = table[flag[4]]

ret[5] = table[flag[9]]

ret[6] = table[flag[14]]

ret[7] = table[flag[3]]

ret[8] = table[flag[8]]
```

```
ret[9] = table[flag[13]]

ret[10] = table[flag[2]]

ret[11] = table[flag[7]]

ret[12] = table[flag[12]]

ret[13] = table[flag[1]]

ret[14] = table[flag[6]]

ret[15] = table[flag[11]]

flag = [p^q for p,q in zip(ret,key[0xa0:0xb0])]

return "".join(map(chr,flag))

def deAES_part2(_key,_flag):

key = map(ord,_key)

flag = map(ord,_flag)

flag = [p^q for p,q in zip(flag,key[0xa0:0xb0])]

ret = ['']*0x10

ret[0] = table.index(flag[0])

ret[5] = table.index(flag[1])

ret[10] = table.index(flag[2])

ret[15] = table.index(flag[3])

ret[4] = table.index(flag[4])

ret[9] = table.index(flag[5])

ret[14] = table.index(flag[6])

ret[3] = table.index(flag[7])

ret[8] = table.index(flag[8])

ret[13] = table.index(flag[9])

ret[2] = table.index(flag[10])

ret[7] = table.index(flag[11])

ret[12] = table.index(flag[12])

ret[1] = table.index(flag[13])

ret[6] = table.index(flag[14])

ret[11] = table.index(flag[15])

flag = ret
```

```
for i in range(0x90,0,-0x10):

flag = [p^q for p,q in zip(flag,key[i:i+0x10])]

p = [''] *0x10

tmp = dehash(flag[0],flag[1],flag[2],flag[3])

p[0],p[5],p[10],p[15] = dehash(*flag[0:4])

p[4],p[9],p[14],p[3] = dehash(*flag[4:8])

p[8],p[13],p[2],p[7] = dehash(*flag[8:12])

p[12],p[1],p[6],p[11] = dehash(*flag[12:16])

flag = [ table.index(x) for x in p]

flag = [p^q for p,q in zip(flag,key[:0x10])]

return "".join(map(chr,flag))

with open("./out","r") as f:

data = f.read()

init_key = data[:0xb0]

result = data[-0x10:]

cmp=[

0,0,0x50cf402af81446ae,0x1212068c04fed331,0xc3d961e826c7fa23,0x3df0c71a70453ca9,

0xac376eab16bcbedf,0x78625df7949c8b14,0x5ad331b21d9816fc,0xa37bca9a86603adc,

0x09d2ffd9b2f1d5b5,0x021956c03dd777d4,0xe377a2e86c429bb6,0x862aa9914032ac99,

0x9b415cc33c47faf3,0x9e5a30d4d00705e8,0x44b6adfba39b528d,0x48fe77229c83723f,

0xffed4e00128486fe,0xca121f84231944ac

]

ct = "\xfe\x86\x84\x12\x00\x4e\xed\xff\xac\x44\x19\x23\x84\x1f\x12\xca"

#pt = "\x82"*16

#print pt.encode("hex")

#pt = "aaaaaaaaaaaasdfg"

#ct = AES_part2(init_key,pt)

#print ct.encode("hex")

box="\x82\x05\x86\x8a\x0b\x11\x96\x1d\x27\xa9\x2b\xb1\xf3\x5e\x37\x38\xc2\x47\x4e\x4f\xd6\x58\xde\xe2\xe5\xe

pool="abcdefghijklmnopqrstuvwxyz0123456789"

for cnt in range(2,len(cmp),2):
```

xor = deAES_part2(init_key,p64(cmp[cnt])+p64(cmp[cnt+1]))

pt= "".join(map(chr,[ord(x)^ord(y) for x,y in zip(xor,p64(cmp[cnt-2])+p64(cmp[cnt-1]))]))

s=""

for p in pt:

if p in box:

s+=pool[box.find(p)]

print s

# print AES_part2(init_key,pt2).encode("hex")

the output is

yycl

funct10nl

1scl

h4rdl

andcl

n0cl

n33dl

tocl

r3v3rs3l

flag is *CTF{yy_funct10n_1s_h4rd_and_n0_n33d_to_r3v3rs3}

fanoGo

Idea

Implents fano with golang. The code will read your ascii input, convert it to strings of '0' and '1' and then decode with fano. fano code book is generated by the corpus.txt. If the decode result is equal with the given string(essay), you will get the flag. Both the corpus and essay is from Winston, Churchill. But I'm sorry that forget to remove the encode function, so many teams just patch the decode by encode, then input the essay........

Solve

You could run it and find the codebook in memory or reverse it from main_main. Don't reverse the functions in package like "fmt_Println", just look up the go manual. Impletes the binary-string function use biu I have write a encode funtions in fano.go, and there is a commentted snippet to encode the essay string , send the key and you will get the flag.

cipher_result := F.Encode(essay)

fmt.Println(cipher_result)

f1, err1 := os.Create("key.txt")

```
if err1 != nil {
fmt.Print(err1)
}
defer f1.Close()
n, _ := f1.WriteString(cipher_result)
fmt.Printf("write %d bytes\n", n)
func Str2Bytes(cipher string) string{
for len(cipher)%8 != 0{
cipher += "0"
}
Bytes := ""
for p:=0; p
var a byte
biu.ReadBinaryString(cipher[p:p+8], &a)
Bytes+=string(a)
}
// for _,x :=range Bytes{
// fmt.Println(byte(x))
// }
return Bytes
}
func (f *Fano) Encode(plain string) string {
cipher := ""
for i:=0; i< len(plain); i++{
cipher += f.codebook[uint8(plain[i])].code
}
return Str2Bytes(cipher)
}
```

web

mywebsql

https://github.com/eddietcc/CVEnotes/blob/master/MyWebSQL/RCE/readme.md crypto

babyprng

You are provided with a pseudorandom number generator with high bias and supposed to extract uniform randomness from the sequence. So just implement von Neumann extractor with the opcodes and duplicate your outputs for the length check.

```
from pwn import *

import string

from hashlib import sha256

context.log_level='debug'

def dopow():

chal = c.recvline()

post = chal[12:28]

tar = chal[33:-1]

c.recvuntil(':')

found = iters.bruteforce(lambda x:sha256(x+post).hexdigest()==tar, string.ascii_letters+string.digits, 4)

c.sendline(found)

#c = remote('127.0.0.1', 10002)

c = remote('34.92.185.118', 10002)

dopow()

code = '\x02\x02\x05\01\x35\x02'+'\x00'*10+'\x05\x41'

c.sendlineafter('(hex): ', code.encode('hex'))

c.interactive()
```

notfeal

This challenge is about differential cryptanalysis for FEAL-4 cipher. You can find a good tutorial at http://theamazingking.com/crypto-feal.php. But the FEAL-4 implementation here is non-standard so you need to understand the process of cryptanalysis and modify the differential path accordingly.

```
import os

import string

from hashlib import sha256

from pwn import *

context.log_level='debug'

def dopow():

chal = c.recvline()
```

```
post = chal[12:28]

tar = chal[33:-1]

c.recvuntil(':')

found = iters.bruteforce(lambda x:sha256(x+post).hexdigest()==tar, string.ascii_letters+string.digits, 4)

c.sendline(found)

def doxor(ss,dd):

res = ''

for i in range(8):

res += chr(ord(ss[i])^dd[i])

return res

def tonum(x):

return (u32(x[:4])<<32)+u32(x[4:])

def doout(stats):

print '{',

for i in range(3):

print '{',

for j in range(6):

print stats[i*6+j],

if j!=5:

print ',',

print '}',

if i!=2:

print ',',

print '}'

#c = remote('127.0.0.1', 10001)

c = remote('34.92.185.118', 10001)

dopow()

diffs = [[0,0,0,0,0x80,0x80,0,0], [0x80,0x80,0,0,0x80,0x80,0,0], [0,0,0,2,0,0,0,2]]

statsp0 = []

statsc0 = []

statsp1 = []
```

```
statsc1 = []

for diff in diffs:

for i in range(6):

pt0 = os.urandom(8)

pt1 = doxor(pt0, diff)

c.sendlineafter('(hex): ', pt0.encode('hex'))

ct0 = c.recvline()[:16]

ct0 = ct0.decode('hex')

c.sendlineafter('(hex): ', pt1.encode('hex'))

ct1 = c.recvline()[:16]

ct1 = ct1.decode('hex')

statsp0.append(tonum(pt0))

statsc0.append(tonum(ct0))

statsp1.append(tonum(pt1))

statsc1.append(tonum(ct1))

c.sendlineafter('(hex): ', 'go')

c.recvuntil('flag:\n')

cflag = c.recvline()

doout(statsp0)

doout(statsc0)

doout(statsp1)

doout(statsc1)

print cflag

// this file has been modified and you can recover subkey for each round interactively

//Differential Cryptanalysis of FEAL-4

//Uses a chosen-plaintext attack to fully recover the key

//For use with tutorial at http://theamazingking.com/crypto-feal.php

#include

#include

#include

#define MAX_CHOSEN_PAIRS 10000
```

```
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

int winner = 0;

int loser = 0;

unsigned long subkey[6];

unsigned long statickey[6] = {0x6bb508a,0xd72eaec0,0x71d88eee,0xa5da0171,0x3139398f,0x2ccdf0f0};

unsigned char rotl2(unsigned char a) {return ((a << 2) | (a >> 6));}

unsigned long leftHalf(unsigned long long a) {return (a >> 32LL);}

unsigned long rightHalf(unsigned long long a) {return a;}

unsigned char sepByte(unsigned long a, unsigned char index) {return a >> (8 * index);}

unsigned long combineBytes(unsigned char b3, unsigned char b2, unsigned char b1, unsigned char b0)

{

return b3 << 24L | (b2 << 16L) | (b1 << 8L) | b0;

}

unsigned long long combineHalves(unsigned long leftHalf, unsigned long rightHalf)

{

return (((unsigned long long)(leftHalf)) << 32LL) | (((unsigned long long)(rightHalf)) & 0xFFFFFFFFLL);

}

unsigned char gBox(unsigned char a, unsigned char b, unsigned char mode)

{

return rotl2(a + b + mode);

}

unsigned long fBox(unsigned long plain)

{

unsigned char x0 = sepByte(plain, 0);

unsigned char x1 = sepByte(plain, 1);

unsigned char x2 = sepByte(plain, 2);

unsigned char x3 = sepByte(plain, 3);

unsigned char t0 = (x2 ^ x3);

unsigned char y1 = gBox(x0 ^ x1, t0, 1);

unsigned char y0 = gBox(x0, y1, 0);

unsigned char y2 = gBox(t0, y1, 0);
unsigned char y3 = gBox(x3, y2, 1);

return combineBytes(y0, y1, y2, y3);

//return combineBytes(y3, y2, y1, y0);

}

unsigned long long encrypt(unsigned long long plain)

{

unsigned long left = leftHalf(plain);

unsigned long right = rightHalf(plain);

//printf("%x,%x\n",left,right);

left = left ^ subkey[4];

right = right ^ subkey[5];

right = right ^ left;

//printf("%x,%x\n",left,right);

unsigned long round2Right = left;

unsigned long round2Left = right ^ fBox(left ^ subkey[0]);

//printf("after k0: %x,%x\n",round2Left,round2Right);

unsigned long round3Right = round2Left;

unsigned long round3Left = round2Right ^ fBox(round2Left ^ subkey[1]);

//printf("after k1: %x,%x\n",round3Left,round3Right);

unsigned long round4Right = round3Left;

unsigned long round4Left = round3Right ^ fBox(round3Left ^ subkey[2]);

//printf("after k2: %x,%x\n",round4Left,round4Right);

left = round4Right ^ fBox(round4Left ^ subkey[3]);

right = round4Left;

//printf("after k3: %x,%x\n",left,right);

unsigned long cipherRight = left ^ right;

unsigned long cipherLeft = right;

//printf("%x,%x\n",cipherLeft,cipherRight);

return combineHalves(cipherLeft, cipherRight);

}

void generateSubkeys(int seed)
```

```
{

srand(seed);

int c;

for(c = 0; c < 6; c++)

//subkey[c] = (rand() << 16L) | (rand() & 0xFFFFL);

subkey[c] = statickey[c];

}

int numPlain;

unsigned long long plain0[MAX_CHOSEN_PAIRS];

unsigned long long cipher0[MAX_CHOSEN_PAIRS];

unsigned long long plain1[MAX_CHOSEN_PAIRS];

unsigned long long cipher1[MAX_CHOSEN_PAIRS];

unsigned long long statsp0[3][6] = { { 7048188517316564675 , 12825066424297617397 ,
13982275348835645359 , 7997352591980253550 , 9943703837341437757 , 5368176796867583232 }, {
9982887355749123894 , 12785352781874485157 , 18182536123490373230 , 5668066430831395550 ,
1137695181956960488 , 12891026720139082070 }, { 10322810123817239080 , 5646801720180827774 ,
15389916003603960434 , 5464895006114500050 , 5548532938388242886 , 4542632487350202421 } };

unsigned long long statsc0[3][6] = { { 9533448304884396727 , 8340911223642448591 ,
486015852071413092 , 14732768206283729966 , 12927250386057848896 , 3498771934970192171 }, {
8077475163787409928 , 360703764514960024 , 7002065340444720468 , 3836488262879817996 ,
4116651873321838340 , 14724546609183381907 }, { 14913951476290284680 , 2798676141719979792 ,
13740334194824131372 , 18286444664320791649 , 10713566717792549958 , 16772546883815774487 } };

unsigned long long statsp1[3][6] = { { 7048188517316597315 , 12825066424297650037 ,
13982275348835612463 , 7997352591980286446 , 9943703837341405117 , 5368176796867616128 }, {
9983028642993326006 , 12785211494630348581 , 18182677410734575342 , 5666653558390022974 ,
1137553894712758376 , 12891166907871656406 }, { 8881658242723136040 , 4205649839757813340 ,
15534031191713370738 , 4023743125691486610 , 5692648126497653190 , 4398517299307900981 } };

unsigned long long statsc1[3][6] = { { 11841643330673051197 , 14299002726156088838 ,
1892929039183045184 , 12424629587526100290 , 6011876236192051371 , 5807051352085183119 }, {
1000588611665094800 , 5955897681464251364 , 8361616289863700606 , 2815539139842138997 ,
16646128912008935309 , 3222529086334277393 }, { 12518318839774872943 , 5153246371973425526 ,
710059454831319824 , 5686169572215424911 , 8960951235428587375 , 5475852056143094730 } };

void undoFinalOperation()

{

int c;

for(c = 0; c < numPlain; c++)

{
```

```
unsigned long cipherLeft0 = leftHalf(cipher0[c]);
unsigned long cipherRight0 = rightHalf(cipher0[c]) ^ cipherLeft0;
unsigned long cipherLeft1 = leftHalf(cipher1[c]);
unsigned long cipherRight1 = rightHalf(cipher1[c]) ^ cipherLeft1;
cipher0[c] = combineHalves(cipherLeft0, cipherRight0);
cipher1[c] = combineHalves(cipherLeft1, cipherRight1);
}
}
unsigned long crackLastRound(unsigned long outdiff)
{
printf(" Using output differential of 0x%08x\n", outdiff);
printf(" Cracking...");
unsigned long fakeK;
for(fakeK = 0x00000000L; fakeK < 0xFFFFFFFFL; fakeK++)
{
int score = 0;
int c;
for(c = 0; c < numPlain; c++)
{
unsigned long Y0 = leftHalf(cipher0[c]);
unsigned long Y1 = leftHalf(cipher1[c]);
unsigned long fakeInput0 = Y0 ^ fakeK;
unsigned long fakeInput1 = Y1 ^ fakeK;
unsigned long fakeOut0 = fBox(fakeInput0) ^ rightHalf(cipher0[c]);
unsigned long fakeOut1 = fBox(fakeInput1) ^ rightHalf(cipher1[c]);
uint32_t fakeDiff = fakeOut0 ^ fakeOut1;
if (fakeDiff == outdiff) score++; else break;
}
if (score == numPlain)
{
printf("found subkey : 0x%08lx\n", fakeK);
//return fakeK;

}

}

printf("failed\n");

return 0;

}

void chosenPlaintext(unsigned long long diff)

{

int c = -1;

if (diff == 0x0000000000008080LL) {

c = 0;

} else if (diff == 0x0000808000008080LL) {

c = 1;

} else if (diff == 0x0200000002000000LL) {

c = 2;

}

for (int i=0; i < numPlain; i++) {

plain0[i] = statsp0[c][i];

plain1[i] = statsp1[c][i];

cipher0[i] = statsc0[c][i];

cipher1[i] = statsc1[c][i];

}

}

void undoLastRound(unsigned long crackedSubkey)

{

int c;

for(c = 0; c < numPlain; c++)

{

unsigned long cipherLeft0 = leftHalf(cipher0[c]);

unsigned long cipherRight0 = rightHalf(cipher0[c]);

unsigned long cipherLeft1 = leftHalf(cipher1[c]);
unsigned long cipherRight1 = rightHalf(cipher1[c]);
cipherRight0 = cipherLeft0;
cipherRight1 = cipherLeft1;
cipherLeft0 = fBox(cipherLeft0 ^ crackedSubkey) ^ rightHalf(cipher0[c]);
cipherLeft1 = fBox(cipherLeft1 ^ crackedSubkey) ^ rightHalf(cipher1[c]);
cipher0[c] = combineHalves(cipherLeft0, cipherRight0);
cipher1[c] = combineHalves(cipherLeft1, cipherRight1);
}
}
void prepForCrackingK0()
{
int c;
for(c = 0; c < numPlain; c++)
{
unsigned long cipherLeft0 = leftHalf(cipher0[c]);
unsigned long cipherRight0 = rightHalf(cipher0[c]);
unsigned long cipherLeft1 = leftHalf(cipher1[c]);
unsigned long cipherRight1 = rightHalf(cipher1[c]);
unsigned long tempLeft0 = cipherLeft0;
unsigned long tempLeft1 = cipherLeft1;
cipherLeft0 = cipherRight0;
cipherLeft1 = cipherRight1;
cipherRight0 = tempLeft0;
cipherRight1 = tempLeft1;
cipher0[c] = combineHalves(cipherLeft0, cipherRight0);
cipher1[c] = combineHalves(cipherLeft1, cipherRight1);
}
}
int main()
{
/*
```

```
generateSubkeys(time(NULL));

cipher0[0] = encrypt(0x3433323138373635);

printf("---\n");

cipher1[0] = encrypt(0x3433323138373635^0x8080);

//cipher1[0] = encrypt(0x3433323138373635^0x808000008080);

//cipher1[0] = encrypt(0x3433323138373635^0x200000002000000);

printf("---\n");

printf("%llx\n", cipher0[0]) ;

printf("%llx\n", cipher1[0]) ;

numPlain = 1;

undoFinalOperation();

undoLastRound(subkey[3]);

undoLastRound(subkey[2]);

undoLastRound(subkey[1]);

printf("%llx\n", cipher0[0]) ;

printf("%llx\n", cipher1[0]) ;

return 0;

*/

printf("JK'S FEAL-4 DIFFERENTIAL CRYPTANALYSIS DEMO\n");

printf("--------------------------------------------\n");

printf("\n");

int graphData[20];

int c;

generateSubkeys(time(NULL));

numPlain = 6;

unsigned long long inputDiff1 = 0x0000000000008080LL;

unsigned long long inputDiff2 = 0x0000808000008080LL;

unsigned long long inputDiff3 = 0x0200000002000000LL;

unsigned long outDiff = 0x02000000L;

unsigned long fullStartTime = time(NULL);

//CRACKING ROUND 4
```

```
printf("ROUND 4\n");

chosenPlaintext(inputDiff1);

undoFinalOperation();

unsigned long startTime = time(NULL);

//unsigned long crackedSubkey3 = crackLastRound(outDiff);

unsigned long crackedSubkey3 = subkey[3];

printf("%x\n", crackedSubkey3);

unsigned long endTime = time(NULL);

printf(" Time to crack round #4 = %i seconds\n", (endTime - startTime));

//CRACKING ROUND 3

printf("ROUND 3\n");

chosenPlaintext(inputDiff2);

undoFinalOperation();

undoLastRound(crackedSubkey3);

startTime = time(NULL);

//unsigned long crackedSubkey2 = crackLastRound(outDiff);

unsigned long crackedSubkey2 = subkey[2];

endTime = time(NULL);

printf(" Time to crack round #3 = %i seconds\n", (endTime - startTime));

//CRACKING ROUND 2

printf("ROUND 2\n");

chosenPlaintext(inputDiff3);

undoFinalOperation();

undoLastRound(crackedSubkey3);

undoLastRound(crackedSubkey2);

startTime = time(NULL);

//unsigned long crackedSubkey1 = crackLastRound(outDiff);

unsigned long crackedSubkey1 = subkey[1];

endTime = time(NULL);

printf(" Time to crack round #2 = %i seconds\n", (endTime - startTime));

//CRACK ROUND 1
```

```
printf("ROUND 1\n");

undoLastRound(crackedSubkey1);

unsigned long crackedSubkey0 = 0;

unsigned long crackedSubkey4 = 0;

unsigned long crackedSubkey5 = 0;

printf(" Cracking...");

startTime = time(NULL);

uint32_t guessK0;

for(guessK0 = 0; guessK0 < 0xFFFFFFFFL; guessK0++)

{

uint32_t guessK4 = 0;

uint32_t guessK5 = 0;

int c;

for(c = 0; c < numPlain; c++)

{

uint32_t plainLeft0 = leftHalf(plain0[c]);

uint32_t plainRight0 = rightHalf(plain0[c]);

uint32_t cipherLeft0 = leftHalf(cipher0[c]);

uint32_t cipherRight0 = rightHalf(cipher0[c]);

uint32_t tempy0 = fBox(cipherLeft0 ^ guessK0) ^ cipherRight0;

if (guessK4 == 0)

{

guessK4 = cipherLeft0 ^ plainLeft0;

guessK5 = tempy0 ^ cipherLeft0 ^ plainRight0;

}

else if (((cipherLeft0 ^ plainLeft0) != guessK4) || ((tempy0 ^ cipherLeft0 ^ plainRight0) != guessK5))

{

guessK4 = 0;

guessK5 = 0;

break;

}
}

if (guessK4 != 0)

{

crackedSubkey0 = guessK0;

crackedSubkey4 = guessK4;

crackedSubkey5 = guessK5;

endTime = time(NULL);

printf("found subkeys : 0x%08lx 0x%08lx 0x%08lx\n", guessK0, guessK4, guessK5);

printf(" Time to crack round #1 = %i seconds\n", (endTime - startTime));

//break;

}

}

printf("\n\n");

printf("0x%08lx - ", crackedSubkey0); if (crackedSubkey0 == subkey[0]) printf("Subkey 0 : GOOD!\n"); else
printf("Subkey 0 : BAD\n");

printf("0x%08lx - ", crackedSubkey1); if (crackedSubkey1 == subkey[1]) printf("Subkey 1 : GOOD!\n"); else
printf("Subkey 1 : BAD\n");

printf("0x%08lx - ", crackedSubkey2); if (crackedSubkey2 == subkey[2]) printf("Subkey 2 : GOOD!\n"); else
printf("Subkey 2 : BAD\n");

printf("0x%08lx - ", crackedSubkey3); if (crackedSubkey3 == subkey[3]) printf("Subkey 3 : GOOD!\n"); else
printf("Subkey 3 : BAD\n");

printf("0x%08lx - ", crackedSubkey4); if (crackedSubkey4 == subkey[4]) printf("Subkey 4 : GOOD!\n"); else
printf("Subkey 4 : BAD\n");

printf("0x%08lx - ", crackedSubkey5); if (crackedSubkey5 == subkey[5]) printf("Subkey 5 : GOOD!\n"); else
printf("Subkey 5 : BAD\n");

printf("\n");

unsigned long fullEndTime = time(NULL);

printf("Total crack time = %i seconds\n", (fullEndTime - fullStartTime));

printf("FINISHED\n");

return 0;

}
```