菁英班作业第5课

环境

真机：华为nova7 HarmonyOS 3.0 未root

IDA pro

项目目录

Crackme2_unupx: Crackme2脱upx壳

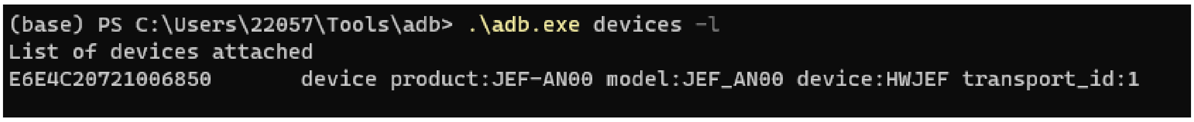
assets: 说明文档图片目录

说明文档.pdf

一、CrackMe1分析

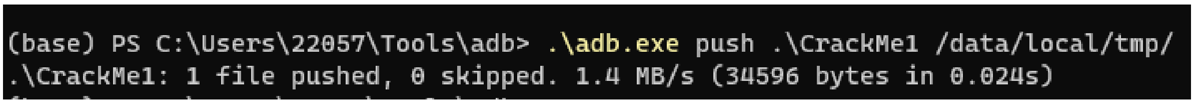
1、实验环境搭建

adb连接手机。



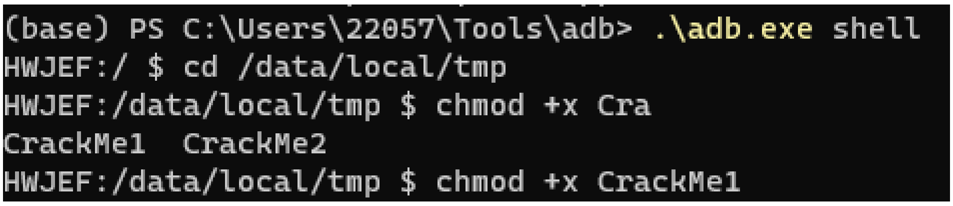
```
(base) PS C:\Users\22057\Tools\adb> .\adb.exe devices -l
List of devices attached
E6E4C20721006850       device product:JEF-AN00 model:JEF_AN00 device:HWJEF transport_id:1
```

将可执行文件push进手机。



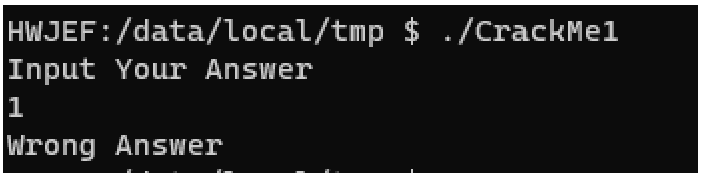
```
(base) PS C:\Users\22057\Tools\adb> .\adb.exe push .\CrackMe1 /data/local/tmp/
.\CrackMe1: 1 file pushed, 0 skipped. 1.4 MB/s (34596 bytes in 0.024s)
```

进入adb shell中，给CrackMe1可执行权限。



```
(base) PS C:\Users\22057\Tools\adb> .\adb.exe shell
HWJEF:/ $ cd /data/local/tmp
HWJEF:/data/local/tmp $ chmod +x Cra
CrackMe1  CrackMe2
HWJEF:/data/local/tmp $ chmod +x CrackMe1
```

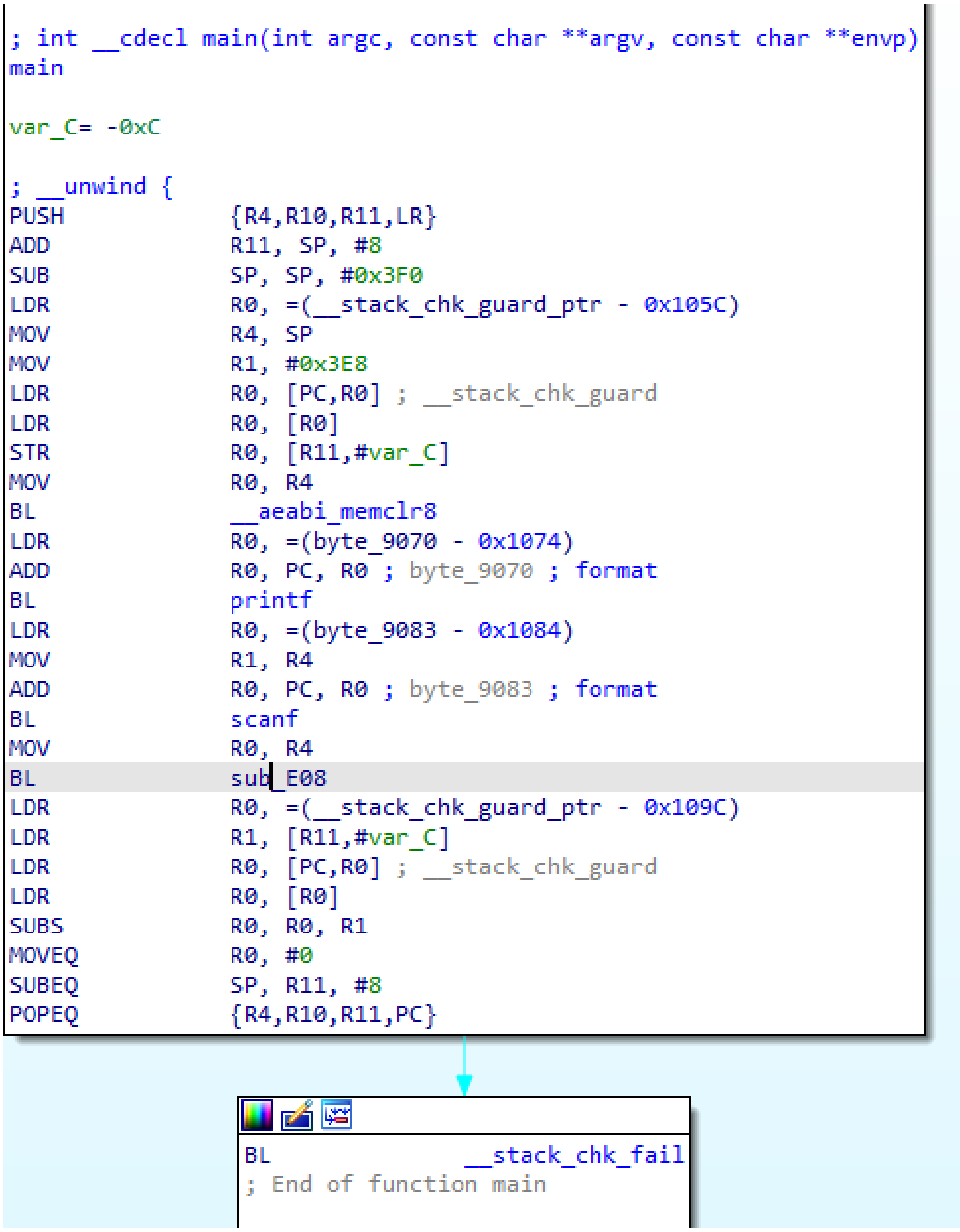
初步运行，输入1，显示错误答案。



```
HWJEF:/data/local/tmp $ ./CrackMe1
Input Your Answer
1
Wrong Answer
```

分析其可能采用字符串比较的方式进行跳转。

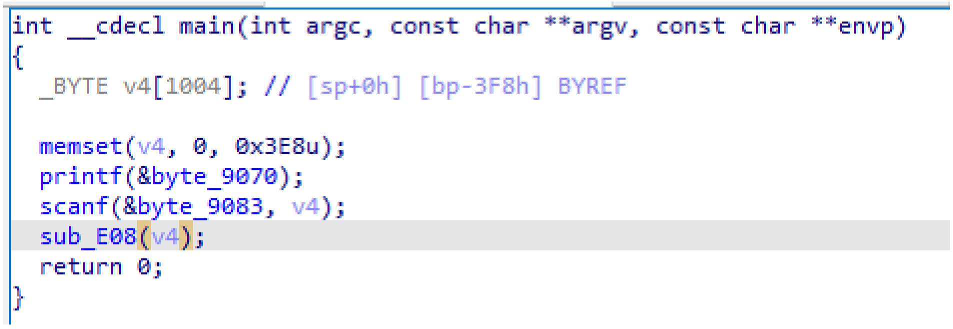
2、使用IDA进行静态分析



```
; int __cdecl main(int argc, const char **argv, const char **envp)
main

var_C= -0xC

; __unwind {
PUSH            {R4,R10,R11,LR}
ADD             R11, SP, #8
SUB             SP, SP, #0x3F0
LDR             R0, =(__stack_chk_guard_ptr - 0x105C)
MOV             R4, SP
MOV             R1, #0x3E8
LDR             R0, [PC,R0] ; __stack_chk_guard
LDR             R0, [R0]
STR             R0, [R11,#var_C]
MOV             R0, R4
BL              __aeabi_memclr8
LDR             R0, =(byte_9070 - 0x1074)
ADD             R0, PC, R0 ; byte_9070 ; format
BL              printf
LDR             R0, =(byte_9083 - 0x1084)
MOV             R1, R4
ADD             R0, PC, R0 ; byte_9083 ; format
BL              scanf
MOV             R0, R4
BL              sub_E08
LDR             R0, =(__stack_chk_guard_ptr - 0x109C)
LDR             R1, [R11,#var_C]
LDR             R0, [PC,R0] ; __stack_chk_guard
LDR             R0, [R0]
SUBS            R0, R0, R1
MOVEQ           R0, #0
SUBEQ           SP, R11, #8
POPEQ           {R4,R10,R11,PC}
```

```
BL              __stack_chk_fail
; End of function main
```

对main函数进行反编译



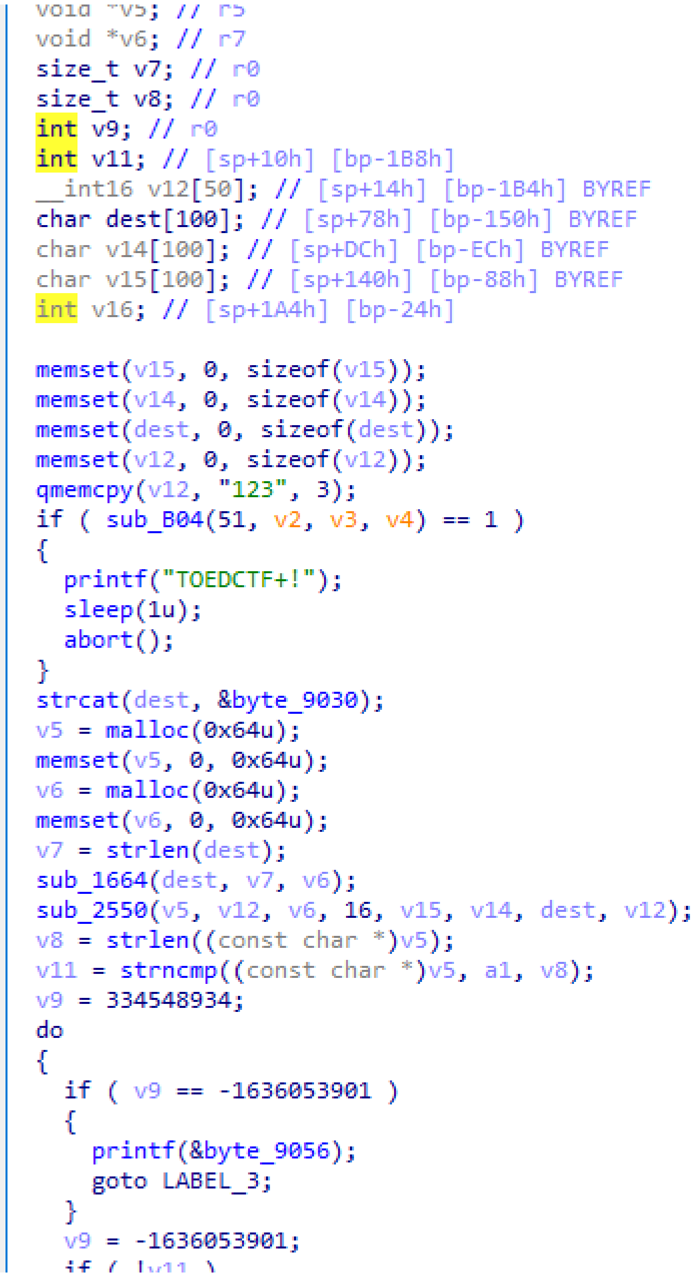
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  _BYTE v4[1004]; // [sp+0h] [bp-3F8h] BYREF

  memset(v4, 0, 0x3E8u);
  printf(&byte_9070);
  scanf(&byte_9083, v4);
  sub_E08(v4);
  return 0;
}
```

发现其先输出提示字符串"Input Your Answer"。

再读入字符串，保存在v4中。

将v4传入函数sub_E08。

进入函数sub_E08



```
void *v5; // r5
void *v6; // r7
size_t v7; // r0
size_t v8; // r0
int v9; // r0
int v11; // [sp+10h] [bp-1B8h]
__int16 v12[50]; // [sp+14h] [bp-1B4h] BYREF
char dest[100]; // [sp+78h] [bp-150h] BYREF
char v14[100]; // [sp+DCh] [bp-ECh] BYREF
char v15[100]; // [sp+140h] [bp-88h] BYREF
int v16; // [sp+1A4h] [bp-24h]

memset(v15, 0, sizeof(v15));
memset(v14, 0, sizeof(v14));
memset(dest, 0, sizeof(dest));
memset(v12, 0, sizeof(v12));
qmemcpy(v12, "123", 3);
if ( sub_B04(51, v2, v3, v4) == 1 )
{
  printf("TOEDCTF+!");
  sleep(1u);
  abort();
}
strcat(dest, &byte_9030);
v5 = malloc(0x64u);
memset(v5, 0, 0x64u);
v6 = malloc(0x64u);
memset(v6, 0, 0x64u);
v7 = strlen(dest);
sub_1664(dest, v7, v6);
sub_2550(v5, v12, v6, 16, v15, v14, dest, v12);
v8 = strlen((const char *)v5);
v11 = strncmp((const char *)v5, a1, v8);
v9 = 334548934;
do
{
  if ( v9 == -1636053901 )
  {
    printf(&byte_9056);
    goto LABEL_3;
  }
  v9 = -1636053901;
  if ( !v11 )
```

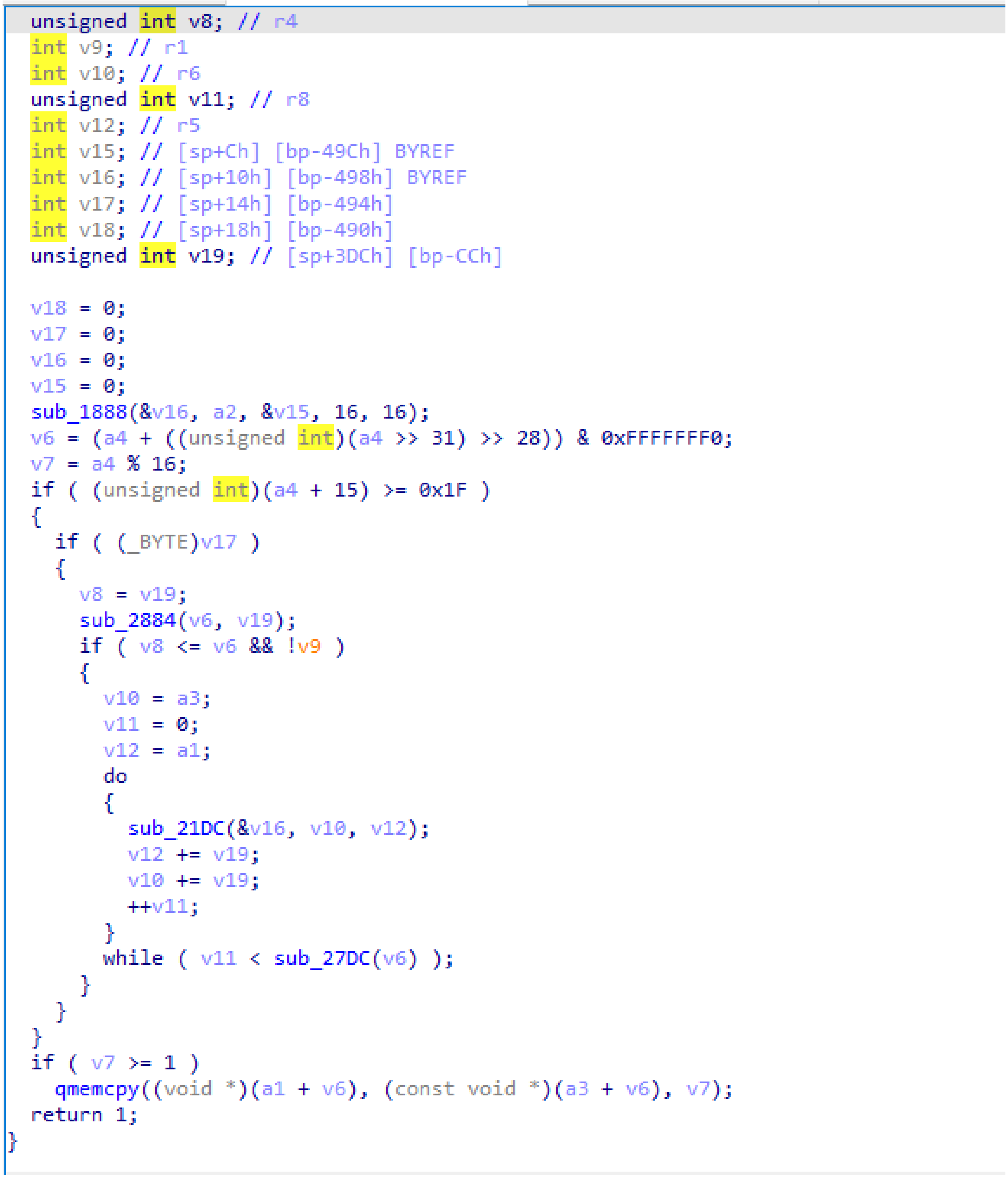
前面一部分内容暂且不管，发现函数调用strncmp

v11 = strncmp((const char *)v5, a1, v8);

其中的a1参数为传入的输入字符串地址，v8是v5的字符串长度。判断正确字符串保存在v5中。

而v5通过函数sub_2550得出。

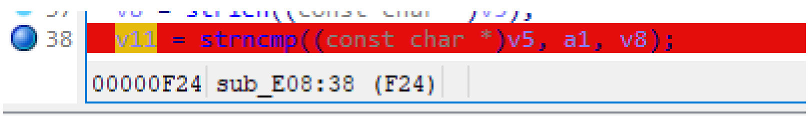
进入函数sub_2550



```
  unsigned int v8; // r4
  int v9; // r1
  int v10; // r6
  unsigned int v11; // r8
  int v12; // r5
  int v15; // [sp+Ch] [bp-49Ch] BYREF
  int v16; // [sp+10h] [bp-498h] BYREF
  int v17; // [sp+14h] [bp-494h]
  int v18; // [sp+18h] [bp-490h]
  unsigned int v19; // [sp+3DCh] [bp-CCh]

  v18 = 0;
  v17 = 0;
  v16 = 0;
  v15 = 0;
  sub_1888(&v16, a2, &v15, 16, 16);
  v6 = (a4 + ((unsigned int)(a4 >> 31) >> 28)) & 0xFFFFFFF0;
  v7 = a4 % 16;
  if ( (unsigned int)(a4 + 15) >= 0x1F )
  {
    if ( (_BYTE)v17 )
    {
      v8 = v19;
      sub_2884(v6, v19);
      if ( v8 <= v6 && !v9 )
      {
        v10 = a3;
        v11 = 0;
        v12 = a1;
        do
        {
          sub_21DC(&v16, v10, v12);
          v12 += v19;
          v10 += v19;
          ++v11;
        }
        while ( v11 < sub_27DC(v6) );
      }
    }
  }
  if ( v7 >= 1 )
    qmemcpy((void *)(a1 + v6), (const void *)(a3 + v6), v7);
  return 1;
}
```

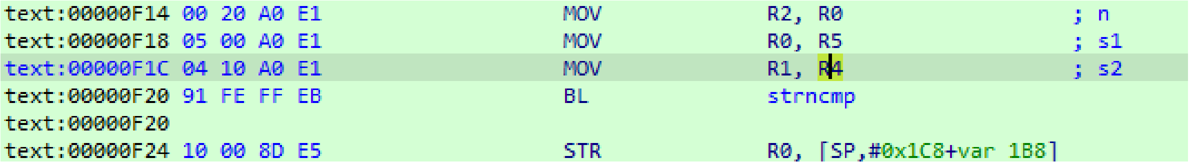
发现其函数实现较为复杂，故放弃静态分析正确字符串，转向动态分析。

记录其关键比较函数strncmp地址及其参数，在此位置设置断点



```
v6 = strlen((const char *)v5);
v11 = strncmp((const char *)v5, a1, v8);
00000F24 sub_E08:38 (F24)
```

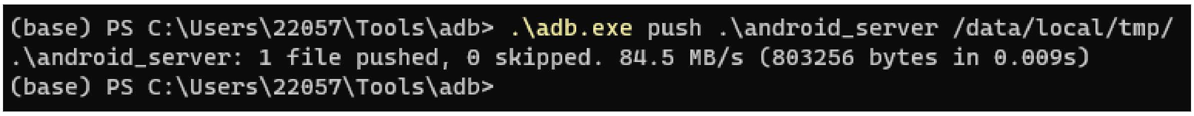
其参数位于R0,R1寄存器内



```
text:00000F14 00 20 A0 E1                 MOV             R2, R0                  ; n
text:00000F18 05 00 A0 E1                 MOV             R0, R5                  ; s1
text:00000F1C 04 10 A0 E1                 MOV             R1, R4                  ; s2
text:00000F20 91 FE FF EB                 BL              strncmp
text:00000F20
text:00000F24 10 00 8D E5                 STR             R0, [SP,#0x1C8+var_1B8]
```

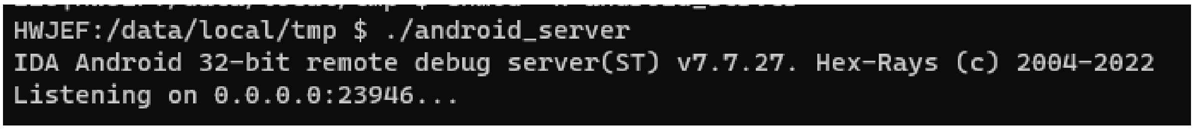
3、IDA动态调试环境搭建

将IDA自带的android_server传入手机中



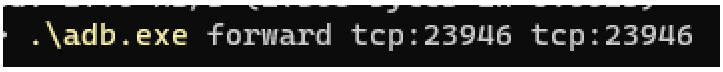
```
(base) PS C:\Users\22057\Tools\adb> .\adb.exe push .\android_server /data/local/tmp/
.\android_server: 1 file pushed, 0 skipped. 84.5 MB/s (803256 bytes in 0.009s)
(base) PS C:\Users\22057\Tools\adb>
```

启动android_server



```
HWJEF:/data/local/tmp $ ./android_server
IDA Android 32-bit remote debug server(ST) v7.7.27. Hex-Rays (c) 2004-2022
Listening on 0.0.0.0:23946...
```

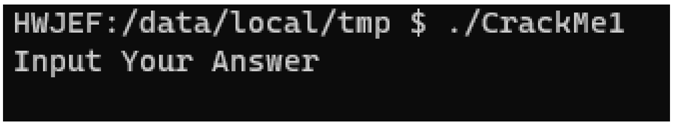
进行端口映射



```
.\adb.exe forward tcp:23946 tcp:23946
```

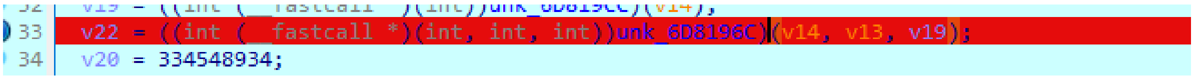
4、IDA动态调试

运行CrackMe1



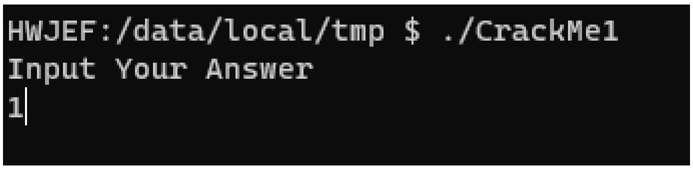
```
HWJEF:/data/local/tmp $ ./CrackMe1
Input Your Answer
```

在strncmp处打下断点



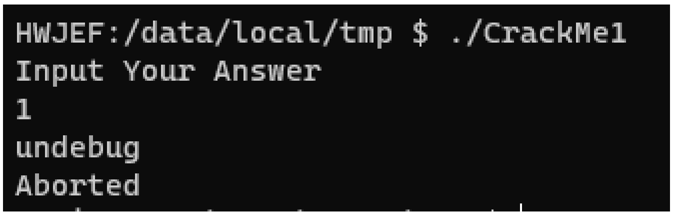
```
v22 = ((int (__fastcall *)(int, int, int))unk_6D8196C)(v14, v13, v19);
v20 = 334548934;
```

输入测试数据1，继续运行



```
HWJEF:/data/local/tmp $ ./CrackMe1
Input Your Answer
1
```

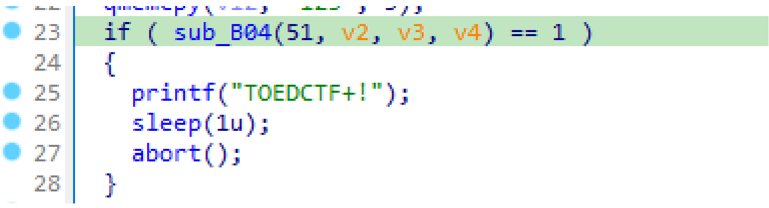
产生中断，错误，判断其可能存在反调试策略。



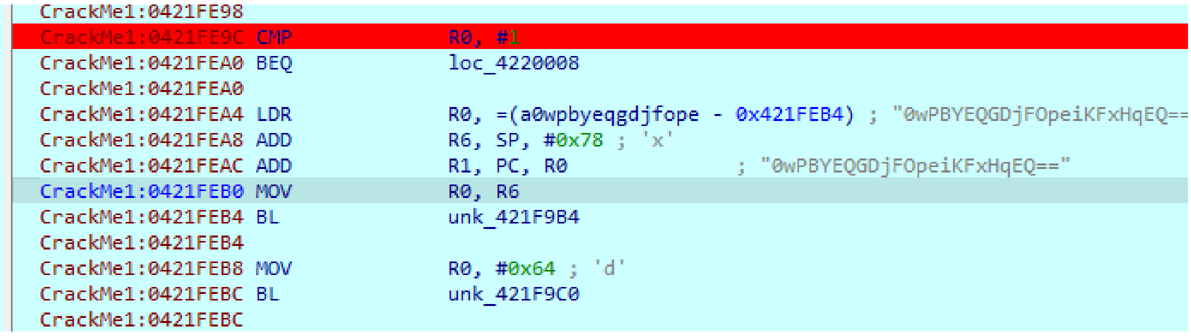
```
HWJEF:/data/local/tmp $ ./CrackMe1
Input Your Answer
1
undebug
Aborted
```

5、绕过反调试策略

静态分析发现此处存在printf与abort函数，在此处打下断点尝试

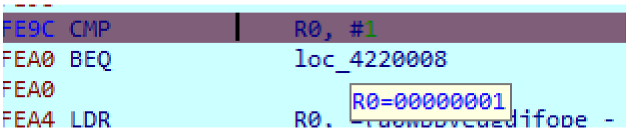


```
if ( sub_B04(51, v2, v3, v4) == 1 )
{
  printf("TOEDCTF+!");
  sleep(1u);
  abort();
}
```



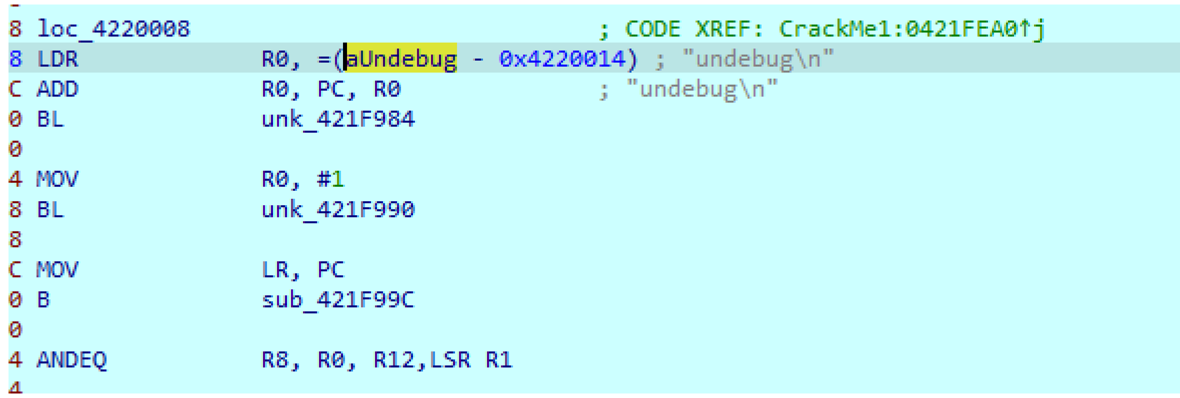
```
CrackMe1:0421FE98
CrackMe1:0421FE9C CMP                R0, #1
CrackMe1:0421FEA0 BEQ                loc_4220008
CrackMe1:0421FEA0
CrackMe1:0421FEA4 LDR                R0, =(a0wpbyeqgdjfope - 0x421FEB4) ; "0wPBYEQGDjFOpeiKFxHqEQ==
CrackMe1:0421FEA8 ADD                R6, SP, #0x78 ; 'x'
CrackMe1:0421FEAC ADD                R1, PC, R0                 ; "0wPBYEQGDjFOpeiKFxHqEQ=="
CrackMe1:0421FEB0 MOV                R0, R6
CrackMe1:0421FEB4 BL                 unk_421F9B4
CrackMe1:0421FEB4
CrackMe1:0421FEB8 MOV                R0, #0x64 ; 'd'
CrackMe1:0421FEBC BL                 unk_421F9C0
CrackMe1:0421FEBC
```

运行到此处，发现R0值为1



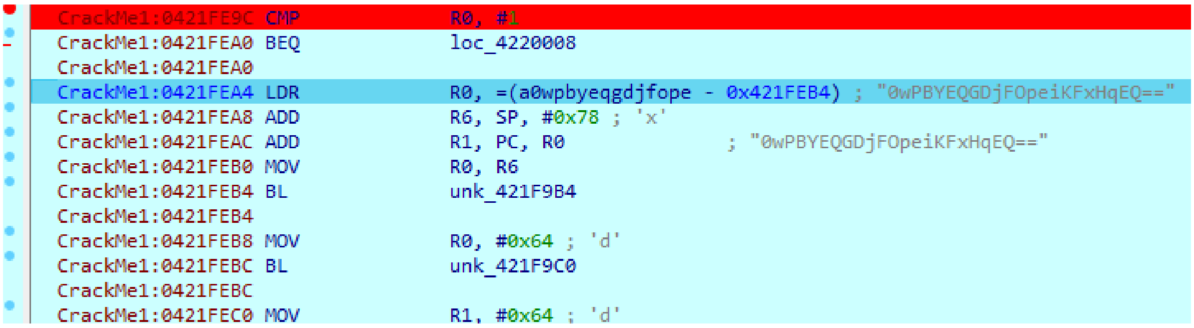
```
FE9C CMP                R0, #1
FEA0 BEQ                loc_4220008
FEA0                      R0=00000001
FEA4 LDR                R0, =(a0wpbyeqgdjfope -
```

二者相等，则跳转至目的地址，输出undebug。



```
8 loc_4220008                              ; CODE XREF: CrackMe1:0421FEA0↑j
8 LDR              R0, =(aUndebug - 0x4220014) ; "undebug\n"
C ADD              R0, PC, R0              ; "undebug\n"
0 BL               unk_421F984
0
4 MOV              R0, #1
8 BL               unk_421F990
8
C MOV              LR, PC
0 B                sub_421F99C
0
4 ANDEQ            R8, R0, R12,LSR R1
4
```

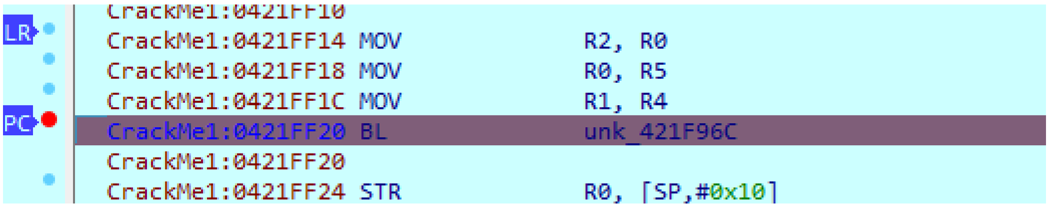
此处修改R0的值为0，让其不进行跳转。



```
CrackMe1:0421FE9C CMP                R0, #1
CrackMe1:0421FEA0 BEQ                loc_4220008
CrackMe1:0421FEA0
CrackMe1:0421FEA4 LDR                R0, =(a0wpbyeqgdjfope - 0x421FEB4) ; "0wPBYEQGDjFOpeiKFxHqEQ=="
CrackMe1:0421FEA8 ADD                R6, SP, #0x78 ; 'x'
CrackMe1:0421FEAC ADD                R1, PC, R0                 ; "0wPBYEQGDjFOpeiKFxHqEQ=="
CrackMe1:0421FEB0 MOV                R0, R6
CrackMe1:0421FEB4 BL                 unk_421F9B4
CrackMe1:0421FEB4
CrackMe1:0421FEB8 MOV                R0, #0x64 ; 'd'
CrackMe1:0421FEBC BL                 unk_421F9C0
CrackMe1:0421FEBC
CrackMe1:0421FEC0 MOV                R1, #0x64 ; 'd'
```

6、获取最终结果

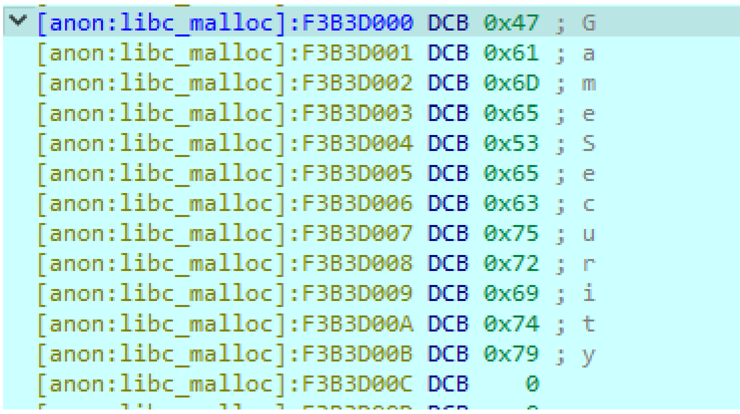
运行至strncmp函数处。



```
   CrackMe1:0421FF10
LR CrackMe1:0421FF14 MOV                R2, R0
   CrackMe1:0421FF18 MOV                R0, R5
   CrackMe1:0421FF1C MOV                R1, R4
PC CrackMe1:0421FF20 BL                 unk_421F96C
   CrackMe1:0421FF20
   CrackMe1:0421FF24 STR                R0, [SP,#0x10]
```

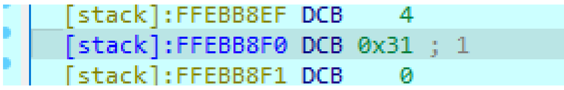
查看R0，R1寄存器内容。

R0：为正确字符串GameSecurity



```
[anon:libc_malloc]:F3B3D000 DCB 0x47 ; G
[anon:libc_malloc]:F3B3D001 DCB 0x61 ; a
[anon:libc_malloc]:F3B3D002 DCB 0x6D ; m
[anon:libc_malloc]:F3B3D003 DCB 0x65 ; e
[anon:libc_malloc]:F3B3D004 DCB 0x53 ; S
[anon:libc_malloc]:F3B3D005 DCB 0x65 ; e
[anon:libc_malloc]:F3B3D006 DCB 0x63 ; c
[anon:libc_malloc]:F3B3D007 DCB 0x75 ; u
[anon:libc_malloc]:F3B3D008 DCB 0x72 ; r
[anon:libc_malloc]:F3B3D009 DCB 0x69 ; i
[anon:libc_malloc]:F3B3D00A DCB 0x74 ; t
[anon:libc_malloc]:F3B3D00B DCB 0x79 ; y
[anon:libc_malloc]:F3B3D00C DCB    0
```

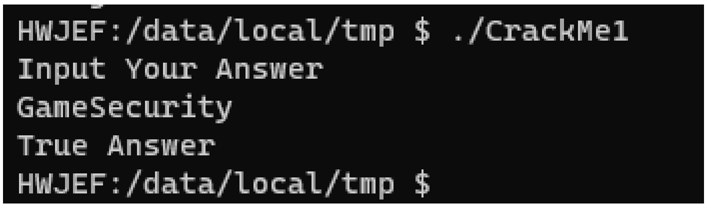
R1：为测试内容1



```
[stack]:FFEBB8EF DCB    4
[stack]:FFEBB8F0 DCB 0x31 ; 1
[stack]:FFEBB8F1 DCB    0
```

最终结果应为GameSecruity

7、测试获取的结果



```
HWJEF:/data/local/tmp $ ./CrackMe1
Input Your Answer
GameSecurity
True Answer
HWJEF:/data/local/tmp $
```

结果正确

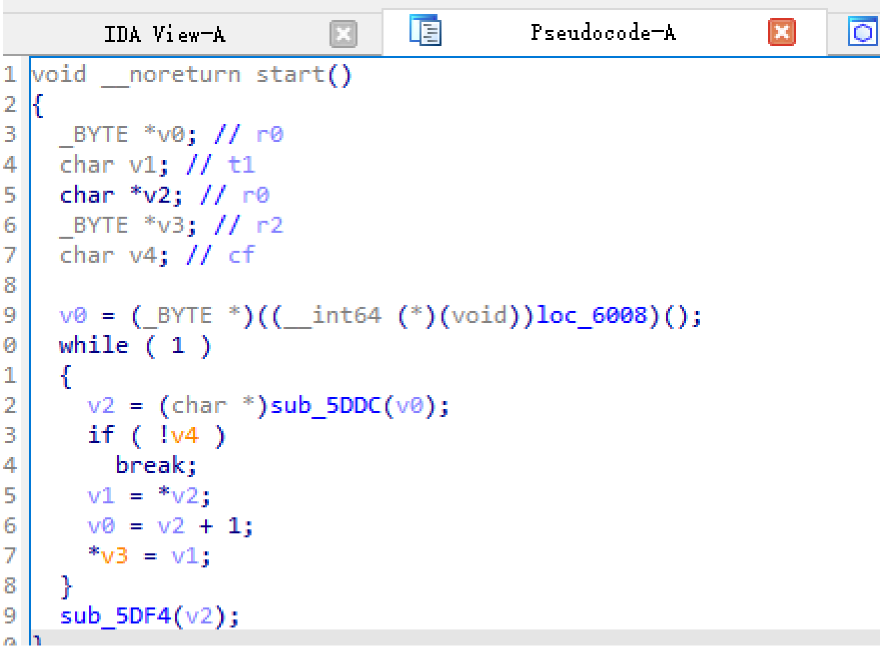
二、CrackMe2分析

1、实验环境搭建

略

2、使用IDA进行静态分析

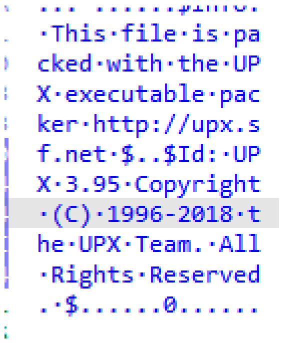
对start函数进行反编译



```
void __noreturn start()
{
  _BYTE *v0; // r0
  char v1; // t1
  char *v2; // r0
  _BYTE *v3; // r2
  char v4; // cf

  v0 = (_BYTE *)((__int64 (*)(void))loc_6008)();
  while ( 1 )
  {
    v2 = (char *)sub_5DDC(v0);
    if ( !v4 )
      break;
    v1 = *v2;
    v0 = v2 + 1;
    *v3 = v1;
  }
  sub_5DF4(v2);
}
```

均为数据加载代码，判断其为加壳应用。

通过查看信息，判断其为upx加壳



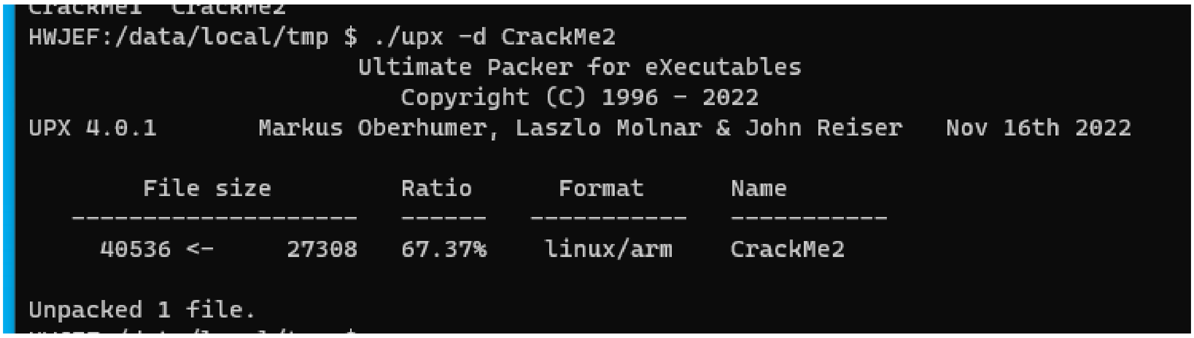
```
·This·file·is·pa
cked·with·the·UP
X·executable·pac
ker·http://upx.s
f.net·$..$Id:·UP
X·3.95·Copyright
·(C)·1996-2018·t
he·UPX·Team.·All
·Rights·Reserved
.·$......@......
```

3、upx脱壳

使用upx工具进行脱壳

```
upx -d CrackMe2
```

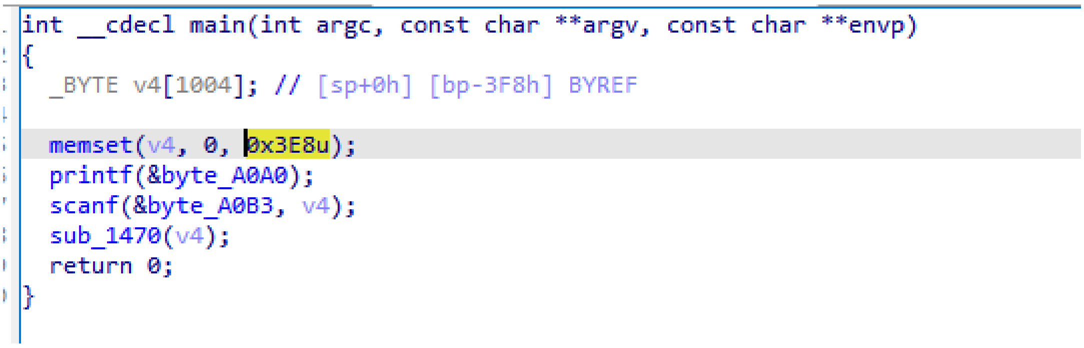
脱壳成功



```
CrackMe1  CrackMe2
HWJEF:/data/local/tmp $ ./upx -d CrackMe2
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2022
UPX 4.0.1       Markus Oberhumer, Laszlo Molnar & John Reiser   Nov 16th 2022

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     40536 <-     27308   67.37%    linux/arm    CrackMe2

Unpacked 1 file.
```

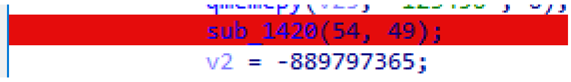
4、使用IDA进行静态分析脱壳后可执行文件

main函数与CrackMe1差不多



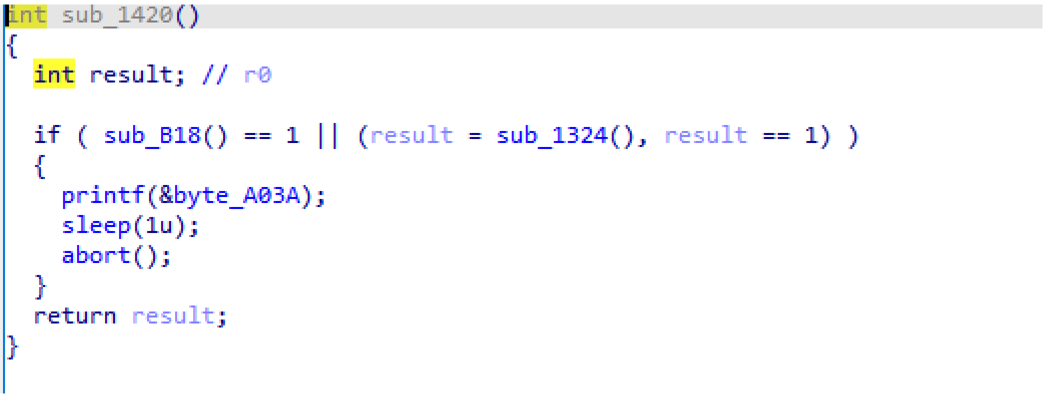
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  _BYTE v4[1004]; // [sp+0h] [bp-3F8h] BYREF

  memset(v4, 0, 0x3E8u);
  printf(&byte_A0A0);
  scanf(&byte_A0B3, v4);
  sub_1470(v4);
  return 0;
}
```

sub_1420函数



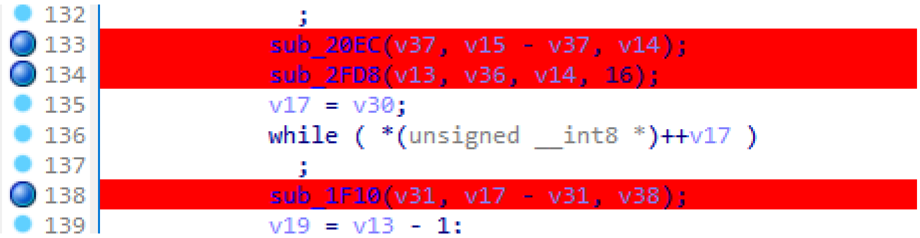
```
sub_1420(54, 49);
v2 = -889797365;
```

进入后发现可能与反调试有关



```
int sub_1420()
{
  int result; // r0

  if ( sub_B18() == 1 || (result = sub_1324(), result == 1) )
  {
    printf(&byte_A03A);
    sleep(1u);
    abort();
  }
  return result;
}
```

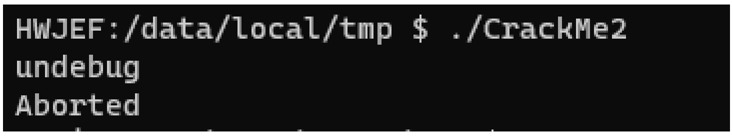
以下三个函数可能与字符串比较有关，动态调试时在此打下断点



```
              ;
            sub_20EC(v37, v15 - v37, v14);
            sub_2FD8(v13, v36, v14, 16);
            v17 = v30;
            while ( *(unsigned __int8 *)++v17 )
              ;
            sub_1F10(v31, v17 - v31, v38);
            v19 = v13 - 1;
```

5、IDA动态调试

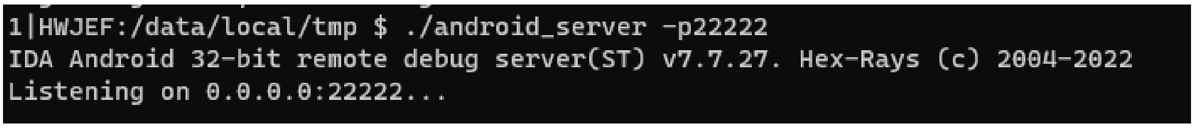
IDA尚未附加便检测到调试退出



```
HWJEF:/data/local/tmp $ ./CrackMe2
undebug
Aborted
```

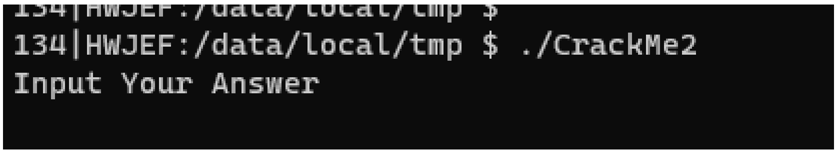
判断其存在检测android_server端口代码

修改android_server端口为22222



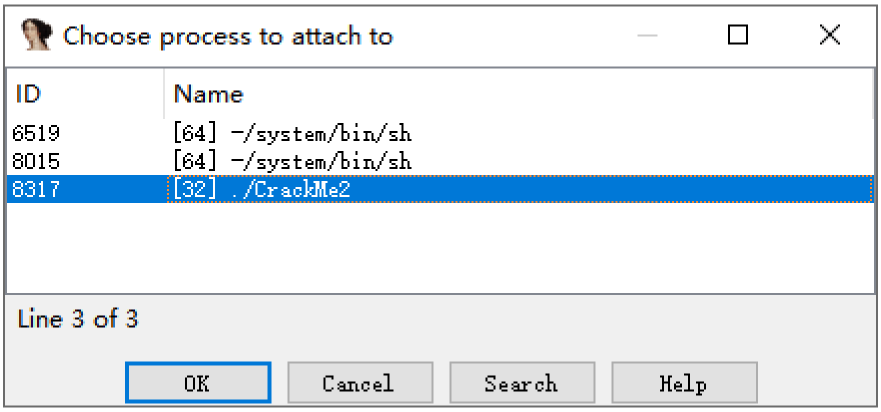
```
1|HWJEF:/data/local/tmp $ ./android_server -p22222
IDA Android 32-bit remote debug server(ST) v7.7.27. Hex-Rays (c) 2004-2022
Listening on 0.0.0.0:22222...
```

正常启动说明其绕过了检测



```
134|HWJEF:/data/local/tmp $
134|HWJEF:/data/local/tmp $ ./CrackMe2
Input Your Answer
```

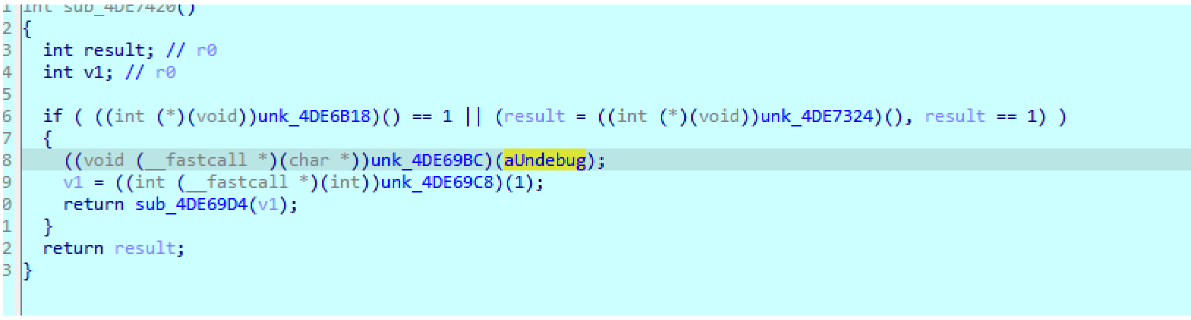
附加到CrackMe2上



ID	Name
6519	[64] -/system/bin/sh
8015	[64] -/system/bin/sh
8317	[32] ./CrackMe2

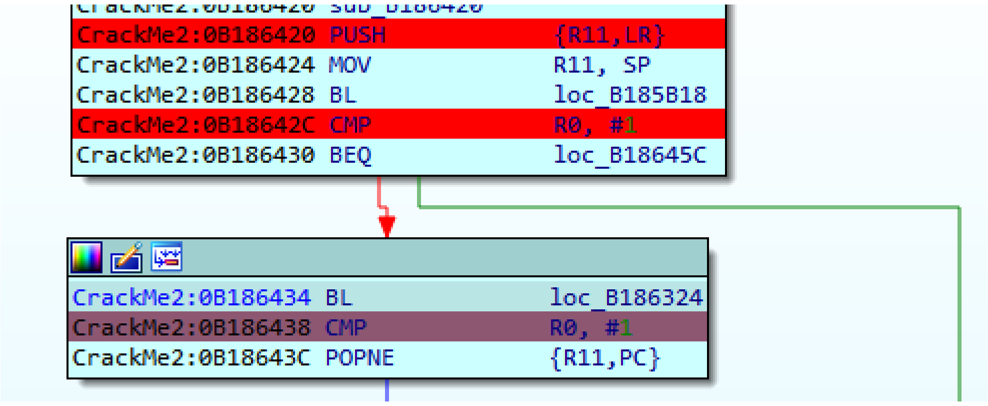
Line 3 of 3

sub1420函数中存在输出undebug信息



```
int sub_4DE7420()
{
  int result; // r0
  int v1; // r0

  if ( ((int (*)(void))unk_4DE6B18)() == 1 || (result = ((int (*)(void))unk_4DE7324)(), result == 1) )
  {
    ((void (__fastcall *)(char *))unk_4DE69BC)(aUndebug);
    v1 = ((int (__fastcall *)(int))unk_4DE69C8)(1);
    return sub_4DE69D4(v1);
  }
  return result;
}
```

判断其为反调试



```
CrackMe2:0B186420 sub_B186420
CrackMe2:0B186420 PUSH                  {R11,LR}
CrackMe2:0B186424 MOV                   R11, SP
CrackMe2:0B186428 BL                    loc_B185B18
CrackMe2:0B18642C CMP                   R0, #1
CrackMe2:0B186430 BEQ                   loc_B18645C
```

```
CrackMe2:0B186434 BL                    loc_B186324
CrackMe2:0B186438 CMP                   R0, #1
CrackMe2:0B18643C POPNE                 {R11,PC}
```

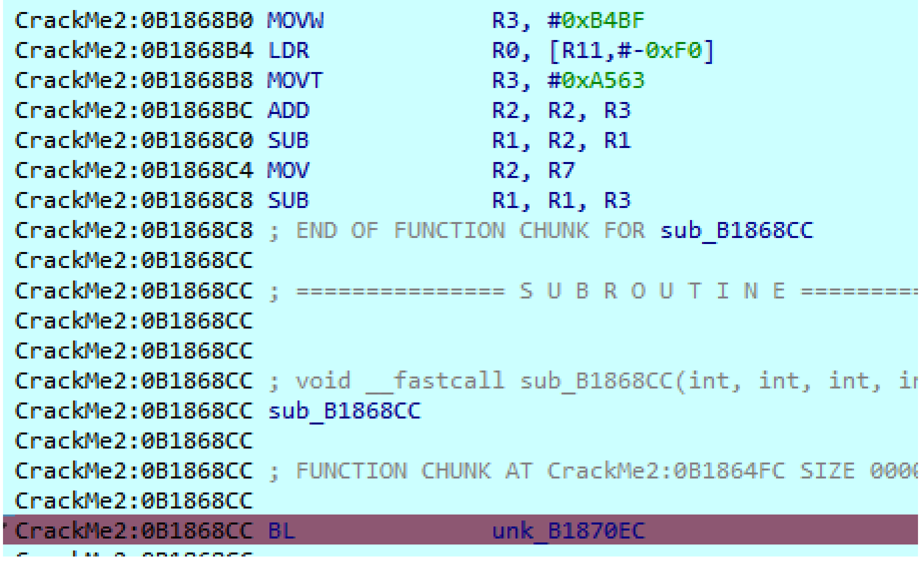
其中loc_B185B18函数为检测IDA android_server的tcp端口，loc_B186324为检测是否调试状态。

由于此前已经修改了端口，只需要修改第二次CMP R0, #1的指令即可。

将R0的值赋值为0即可绕过调试

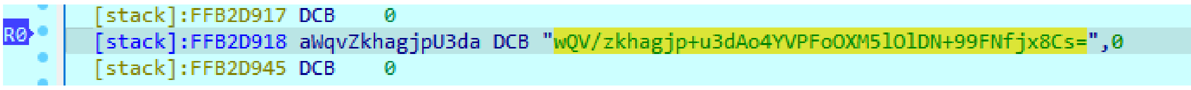
接下来对于三个可疑函数调用进行分析

函数sub20EC:



```
CrackMe2:0B1868B0 MOVW            R3, #0xB4BF
CrackMe2:0B1868B4 LDR             R0, [R11,#-0xF0]
CrackMe2:0B1868B8 MOVT            R3, #0xA563
CrackMe2:0B1868BC ADD             R2, R2, R3
CrackMe2:0B1868C0 SUB             R1, R2, R1
CrackMe2:0B1868C4 MOV             R2, R7
CrackMe2:0B1868C8 SUB             R1, R1, R3
CrackMe2:0B1868C8 ; END OF FUNCTION CHUNK FOR sub_B1868CC
CrackMe2:0B1868CC
CrackMe2:0B1868CC ; =============== S U B R O U T I N E =========
CrackMe2:0B1868CC
CrackMe2:0B1868CC
CrackMe2:0B1868CC ; void __fastcall sub_B1868CC(int, int, int, i
CrackMe2:0B1868CC sub_B1868CC
CrackMe2:0B1868CC
CrackMe2:0B1868CC ; FUNCTION CHUNK AT CrackMe2:0B1864FC SIZE 000
CrackMe2:0B1868CC
CrackMe2:0B1868CC BL              unk_B1870EC
```

R0值为

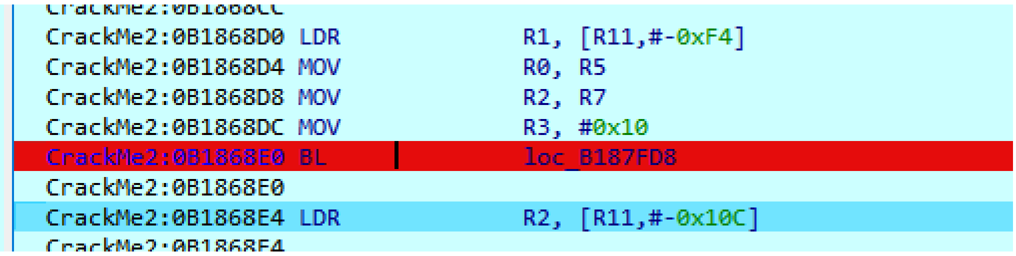


```
R0   [stack]:FFB2D917 DCB    0
     [stack]:FFB2D918 aWqvZkhagjpU3da DCB "wQV/zkhagjp+u3dAo4YVPFoOXM5lOlDN+99FNfjx8Cs=",0
     [stack]:FFB2D945 DCB    0
```

判断其应为base64加密后内容，但解码为乱码

R1为2c，R3地址处为空值，判断其应当不是字符串比较函数

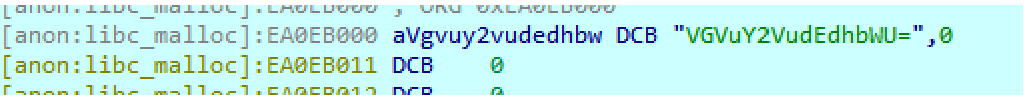
函数sub_1FD8



```
CrackMe2:0B1868D0 LDR                R1, [R11,#-0xF4]
CrackMe2:0B1868D4 MOV                R0, R5
CrackMe2:0B1868D8 MOV                R2, R7
CrackMe2:0B1868DC MOV                R3, #0x10
CrackMe2:0B1868E0 BL                 loc_B187FD8
CrackMe2:0B1868E0
CrackMe2:0B1868E4 LDR                R2, [R11,#-0x10C]
```

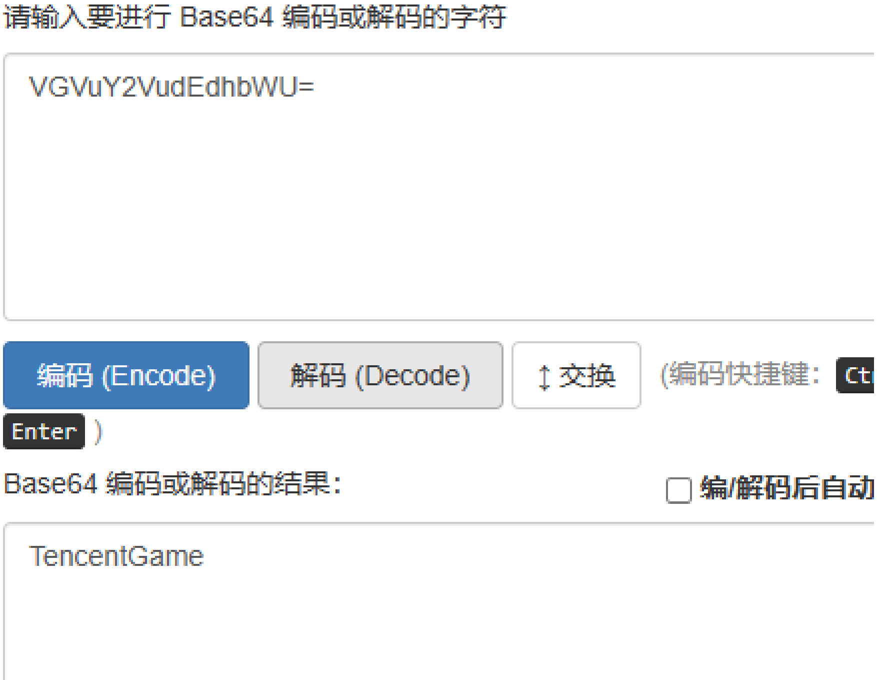
R0为R5的值

在函数执行完成后R5的位置发现一字符串



```
[anon:libc_malloc]:EA0EB000 aVgvuy2vudedhbw DCB "VGVuY2VudEdhbWU=",0
[anon:libc_malloc]:EA0EB011 DCB    0
```

判断其为base64加密后内容



请输入要进行 Base64 编码或解码的字符

VGVuY2VudEdhbWU=

Base64 编码或解码的结果:

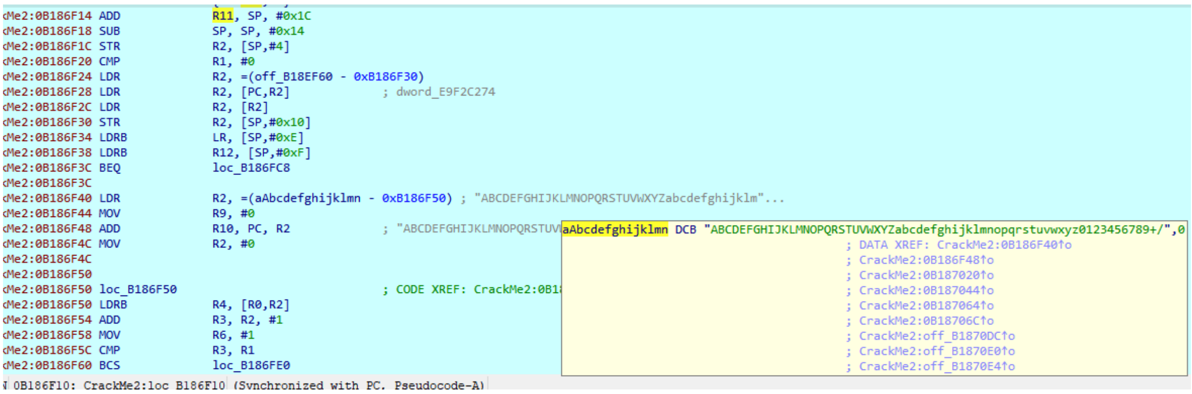
TencentGame

解码结果为TencentGame，初步猜测为结果

对于函数1F10:

其参数R0为输入字符串1，R1位字符串长度1。

在其函数内部



```
cMe2:0B186F14 ADD             R11, SP, #0x1C
cMe2:0B186F18 SUB             SP, SP, #0x14
cMe2:0B186F1C STR             R2, [SP,#4]
cMe2:0B186F20 CMP             R1, #0
cMe2:0B186F24 LDR             R2, =(off_B18EF60 - 0xB186F30)
cMe2:0B186F28 LDR             R2, [PC,R2]          ; dword_E9F2C274
cMe2:0B186F2C LDR             R2, [R2]
cMe2:0B186F30 STR             R2, [SP,#0x10]
cMe2:0B186F34 LDRB            LR, [SP,#0xE]
cMe2:0B186F38 LDRB            R12, [SP,#0xF]
cMe2:0B186F3C BEQ             loc_B186FC8
cMe2:0B186F3C
cMe2:0B186F40 LDR             R2, =(aAbcdefghijklmn - 0xB186F50) ; "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm"...
cMe2:0B186F44 MOV             R9, #0
cMe2:0B186F48 ADD             R10, PC, R2          ; "ABCDEFGHIJKLMNOPQRSTUV
cMe2:0B186F4C MOV             R2, #0
cMe2:0B186F4C
cMe2:0B186F50
cMe2:0B186F50 loc_B186F50                          ; CODE XREF: CrackMe2:0B1
cMe2:0B186F50 LDRB            R4, [R0,R2]
cMe2:0B186F54 ADD             R3, R2, #1
cMe2:0B186F58 MOV             R6, #1
cMe2:0B186F5C CMP             R3, R1
cMe2:0B186F60 BCS             loc_B186FE0
```

```
aAbcdefghijklmn DCB "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/",0
                                        ; DATA XREF: CrackMe2:0B186F40↑o
                                        ; CrackMe2:0B186F48↑o
                                        ; CrackMe2:0B187020↑o
                                        ; CrackMe2:0B187044↑o
                                        ; CrackMe2:0B187064↑o
                                        ; CrackMe2:0B18706C↑o
                                        ; CrackMe2:off_B1870DC↑o
                                        ; CrackMe2:off_B1870E0↑o
                                        ; CrackMe2:off_B1870E4↑o
```

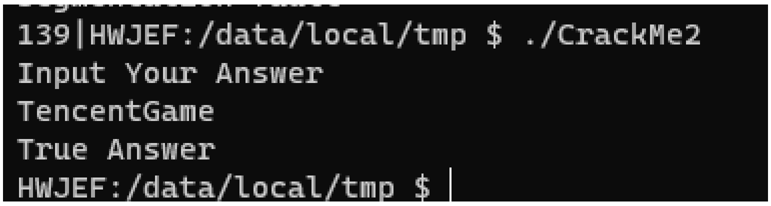
```
0B186F10: CrackMe2:loc_B186F10 (Synchronized with PC, Pseudocode-A)
```

存在ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

其应为base64加密函数

6、测试获取的结果

使用TencentGame进行测试，结果正确。



```
139|HWJEF:/data/local/tmp $ ./CrackMe2
Input Your Answer
TencentGame
True Answer
HWJEF:/data/local/tmp $ 
```