•[e7bc5d2c0cf4480348f5504196561297]•

. Task Description .

. Introduction .

Looking into the file using a hexeditor point to ELF file x64. Then I start a "kali-x64 vm" machine to test the program output and required params. Executing the program without arguments give a "no" with 1 parameter "na" with 2 a "bad" and with 3 or more parameters a "stahp".

Lets go to open it using Ida64 version. And locate this strings.

. Hands to work .

. First look .

After load the file into IDA, I open the string windows and search for all these strings "no","na","bad" and "stahp".

```
.rodata:00000000004F3B44 aStahp          db 'stahp',0            ; DATA XREF: sub_41C77D+1ACB#o
.rodata:00000000004F3FE9 aNo             db 'no',0               ; DATA XREF: sub_452079+153D#o
.rodata:00000000004F3FEC aNa             db 'na',0               ; DATA XREF: sub_452079+4C55#o
.rodata:00000000004F3BF2 aBad            db 'bad',0              ; DATA XREF: sub_435E20+12EC#o
```

And inspect the code in these addresses can determine the correct number of parameters.

"no" calling:

```
.text:000000000045359B                cmp     [rbp+var_B4], 4
.text:00000000004535A2                setle   al
.text:00000000004535A5                test    al, al
.text:00000000004535A7                jnz     loc_453388
.text:00000000004535AD                cmp     [rbp+var_A84], 1
.text:00000000004535B4                jnz     short loc_4535CA
.text:00000000004535B6                mov     edi, offset aNo ; "no"
.text:00000000004535BB                call    sub_45EBE0
.text:00000000004535C0                mov     edi, 34h
.text:00000000004535C5                call    sub_45E790
```

"na" calling:

```
.text:0000000000456CB7                cmp     [rbp+var_1E0], 2
.text:0000000000456CBE                setle   al
.text:0000000000456CC1                test    al, al
.text:0000000000456CC3                jnz     short loc_456CA3
.text:0000000000456CC5                cmp     [rbp+var_A84], 2
.text:0000000000456CCC                jnz     short loc_456CE2
.text:0000000000456CCE                mov     edi, offset aNa ; "na"
.text:0000000000456CD3                call    sub_45EBE0
.text:0000000000456CD8                mov     edi, 1A7h
.text:0000000000456CDD                call    sub_45E790
```

"stahp" calling:

```
.text:000000000041E23F
.text:000000000041E23F loc_41E23F:                             ; CODE XREF: sub_41C77D+1A97#j
.text:000000000041E23F                 cmp     [rbp+var_354], 3
.text:000000000041E246                 jle     short loc_41E25C
.text:000000000041E248                 mov     edi, offset aStahp ; "stahp"
.text:000000000041E24D                 call    sub_45EBE0
.text:000000000041E252                 mov     edi, 0Eh
.text:000000000041E257                 call    sub_45E790
```

This 3 chunks are enough to determine the correct number of params in this case its 2. Ok now lets go to determine where read the first param. But first I take in count what the code are too dirty and many instruction do nothing. Then before start the analisys I code a .idc to clean all this "trash" and can follow better the flow into dead code.

"IDC Script":

```
#include <idc.idc>
static Nopea(ini,fin)
{
         auto x;
         for (x=ini;x<fin;x++)
         {
           PatchByte(x,0x90);
         }
}



static Basura()
{
  //l4a();
  //l4();
  l5();
  l6();
  l7();
  //l8();
  //l9();
  //l10();
  //l11();
}

////////////////////////////////////////////////////////////////////////////////////////////////////


//.text:000000000045D9FC C7 45 F8 00 00 00 00                               mov     [rbp+var_8], 0
//.text:000000000045DA03 83 7D F8 00                                        cmp     [rbp+var_8], 0
//.text:000000000045DA07 0F 85 75 02 00 00                                  jnz     loc_45DC82

//.text:0000000000452C03 C7 45 F4 00 00 00 00                               mov     [rbp+var_C], 0
//.text:0000000000452C0A 83 7D F4 00                                        cmp     [rbp+var_C], 0
//.text:0000000000452C0E 74 0E                                              jz      short loc_452C1E
static l3()
{
    auto chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"c7 45");
    if (ad>0)
    {
       chk1=Byte(ad+7);
       chk2=Byte(ad+11);
       if (chk1==0x83 )
       {
          if (chk2==0x74)
         {
            ini=ad;
           fin=ad+13;
           Message("Encontrado mov cmp jz corto en:%x l:%x\n",ini,fin);
            Nopea(ini,fin-2);
            PatchByte(fin-2,0xeb);

           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
           MakeCode(fin+2);
```

```
          }
/*        if (chk2==0x0f && Byte(ad+13)==0x75)
         {
            ini=ad;
           fin=ad+17;
           Message("Encontrado mov cmp jnz largo en:%x l:%x\n",ini,fin);
         }
          if (chk2==0x0f && Byte(ad+13)==0x74)
         {
            ini=ad;
           fin=ad+17;
           Message("Encontrado mov cmp jz largo en:%x l:%x\n",ini,fin);
         }*/


       }
    }
    address=ad+1;
 }while(ad>0);
}


//text:0000000000453232 48 C7 85 B8 FA FF FF C2 80 72 00                    mov     [rbp+var_548], 7280C2h
//.text:000000000045323D 48 8B 85 B8 FA FF FF                               mov     rax, [rbp+var_548]
//.text:0000000000453244 0F B6 00                                           movzx   eax, byte ptr [rax]
//.text:0000000000453247 88 05 7E 68 2D 00                                  mov     cs:byte_729ACB, al
static l2()
{
    auto chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"0F B6 00");
    if (ad>0)
    {
       chk1=Byte(ad-18);
       chk2=Byte(ad-7);
       if (chk1==0x48 && chk2==0x48)
       {
          ini=ad-18;
         fin=ad+9;
         Message("Encontrado cmp setle test : %x l:%x\n",ini,fin);
          Nopea(ini,fin);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}

//.text:000000000040E1D5 83 45 E0 01                                        add     [rbp+var_20], 1
//.text:000000000040E1D9 83 7D E0 02                                        cmp     [rbp+var_20], 2
//.text:000000000040E1DD 0F 9E C0                                           setle   al
//.text:000000000040E1E0 84 C0                                              test    al, al
//.text:000000000040E1E2 75 BC                                              jnz     short loc_40E1A0
/*
static l1()
{
    auto chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"83 45");
    if (ad>0)
    {
       chk1=Byte(ad+4);
       chk2=Byte(ad+8);
       if (chk1==0x83 && chk2==0x0f)
       {
              ini=ad;
         fin=ad+19;
         Message("Encontrado cmp setle test : %x l:%x\n",ini,fin);
                Nopea(ini,fin);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}
*/

//.text:0000000000454B32 C6 45 EB 44                                        mov     [rbp+var_15], 44h
//.text:0000000000454B36 80 7D EB 44                                        cmp     [rbp+var_15], 44h
//.text:0000000000454B3A 75 60                                              jnz     short loc_454B9C
```

```
//.text:00000000004432CC C6 45 DD 44                                        mov     [rbp+var_23], 44h
//.text:00000000004432D0 80 7D DD 44                                        cmp     [rbp+var_23], 44h
//.text:00000000004432D4 74 37                                              jz      short loc_44330D


//.text:0000000000442BE2 C6 45 DC 44                                        mov     [rbp+var_24], 44h
//.text:0000000000442BE6 80 7D DC 44                                        cmp     [rbp+var_24], 44h
//.text:0000000000442BEA 0F 85 84 05 00 00                                  jnz     loc_443174
static l0()
{
    auto chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"C6 45");
    if (ad>0)
    {
//      Message("Encontrado en: %x \n",ad);
      chk1=Byte(ad+4);
      chk2=Byte(ad+8);
      if (chk1==0x80 && chk2==0x0f && Byte(ad+9)==0x85)
       {
         ini=ad;
         fin=ad+14;
         Message("limpiando 0f 85 : %x l:%x\n",ini,fin);
         Nopea(ini,fin);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);

       }
      if (chk1==0x80 && chk2==0x75)
       {
                ini=ad;
         fin=ad+10;
         Message("limpiando 75 corto: %x l:%x\n",ini,fin);
         Nopea(ini,fin);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);

       }
       if (chk1==0x80 && chk2==0x74)
       {
         ini=ad;
         fin=ad+8;
         Message("limpiando 74 corto: %x l:%x\n",ini,fin);
         Nopea(ini,fin);
                PatchByte(fin,0xeb);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);
         MakeCode(fin+2);

       }

    }
    address=ad+1;
  }while(ad>0);

}
//.text:00000000004558BE 83 45 C4 01                                        add     [rbp+var_3C], 1
//.text:00000000004558C2 83 7D C4 02                                        cmp     [rbp+var_3C], 2
//.text:00000000004558C6 0F 9E C0                                           setle   al
//.text:00000000004558C9 84 C0                                              test    al, al
//.text:00000000004558CB 0F 85 B1 C8 FF FF                                  jnz     loc_452182
static l4()
{
    auto chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"0F 9E C0 ");
    if (ad>0)
    {
       chk1=Byte(ad-8);
       chk2=Byte(ad+5);
       if (chk1==0x83 && chk2==0x0f)
       {
         ini=ad-8;
         fin=ad+11;
         Message("Encontrado cmp setle test : %x l:%x\n",ini,fin);
         Nopea(ini,fin);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}
```

```
//.text:0000000000455E78 83 85 8C FE FF FF 01                                  add     [rbp+var_174], 1
//.text:0000000000455E7F 83 BD 8C FE FF FF 01                                  cmp     [rbp+var_174], 1
//.text:0000000000455E86 0F 9E C0                                              setle   al
//.text:0000000000455E89 84 C0                                                 test    al, al
//.text:0000000000455E8B 0F 85 A8 FA FF FF                                     jnz     loc_455939
static l5()
{
    auto chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"0F 9E C0 ");
    if (ad>0)
    {
       chk1=Byte(ad-13);
       chk2=Byte(ad+5);
       if (chk1==0x85 && chk2==0x0f)
       {
         ini=ad-14;
         fin=ad+11;
         Message("Encontrado cmp setle test : %x l:%x\n",ini,fin);
         Nopea(ini,fin);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}

static Saltos()
{
  l0();
  l2();
  l3();
  l4();
  l5();
}
static limpia()
{
         sub_ecx_corto();
         sub_edi_corto();
         and_eax();
         xor_eax();
         xor_ofs_eax();
         add_eax_eax();
         add_eax_edx();
         mov_cmp_jnz_largo();

         movzx();

         imul_eax_ofs();


}
//.text:0000000000447930 8B 45 F4                                              mov     eax, [rbp+var_C]
//.text:0000000000447933 0F AF 45 FC                                           imul    eax, [rbp+var_4]
//.text:0000000000447937 89 45 F4                                              mov     [rbp+var_C], eax
static imul_eax_ofs()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"0F AF");
    if (ad>0)
    {
       chk1=Byte(ad-3);
       chk2=Byte(ad+4);
            chk3=Byte(ad+5);

       if (chk1==0x8b && chk2==0x89 && chk3==0x45 )
       {
           ini=ad-3;
           fin=ad+7;
           Message("Encontrado imul_eax_ofs_corto en:%x l:%x\n",ini,fin);
           Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);

       }
            if (chk1==0x8b && chk2==0x89 && chk3==0x85 )
       {
           ini=ad-3;
           fin=ad+10;
           Message("Encontrado imul_eax_ofs_largo en:%x l:%x\n",ini,fin);
           Nopea(ini,fin);
```

```
          MakeUnkn(ini,fin-ini);
          MakeCode(ini);

       }

    }
    address=ad+1;
 }while(ad>0);
}

//.text:0000000000411D46 48 C7 45 88 BF 80 72 00                          mov     qword ptr [rbp-78h],
//.text:0000000000411D4E 48 8B 45 88                                      mov     rax, [rbp-78h]
//.text:0000000000411D52 0F B6 00                                         movzx   eax, byte ptr [rax]
//.text:0000000000411D55 88 05 CD 7B 31 00                                mov     cs:byte_729928, al
static movzx()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"0F B6 00");
    if (ad>0)
    {
       chk1=Byte(ad-12);
       chk2=Byte(ad+3);

       if (chk1==0x48 && chk2==0x88  )
       {
           ini=ad-12;
           fin=ad+9;
           Message("Encontrado movxz en:%x l:%x\n",ini,fin);
           Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);

       }
    }
    address=ad+1;
 }while(ad>0);
}


//.text:0000000000442BE2 C6 45 DC 44                                      mov     [rbp+var_24], 44h
//.text:0000000000442BE6 80 7D DC 44                                      cmp     [rbp+var_24], 44h
//.text:0000000000442BEA 0F 85 84 05 00 00                                jnz     loc_443174
static mov_cmp_jnz_largo()
{
    auto chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"C6 45");
    if (ad>0)
    {
//      Message("Encontrado en: %x \n",ad);
      chk1=Byte(ad+4);
      chk2=Byte(ad+8);
      if (chk1==0x80 && chk2==0x0f && Byte(ad+9)==0x85)
       {
         ini=ad;
         fin=ad+14;
         Message("limpiando 0f 85 : %x l:%x\n",ini,fin);
         Nopea(ini,fin);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);

       }
      if (chk1==0x80 && chk2==0x75)
       {
              ini=ad;
         fin=ad+10;
         Message("limpiando 75 corto: %x l:%x\n",ini,fin);
         Nopea(ini,fin);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);

       }
       if (chk1==0x80 && chk2==0x74)
       {
         ini=ad;
         fin=ad+8;
         Message("limpiando 74 corto: %x l:%x\n",ini,fin);
         Nopea(ini,fin);
              PatchByte(fin,0xeb);
         MakeUnkn(ini,fin-ini);
         MakeCode(ini);
         MakeCode(fin+2);
```

```
       }

    }
    address=ad+1;
  }while(ad>0);

}



//.text:000000000045AC5B 8B 45 F8               mov     eax, [rbp+var_8]
//.text:000000000045AC5E 01 C0                  add     eax, eax
//.text:000000000045AC60 89 45 FC               mov     [rbp+var_4], eax
static add_eax_eax()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"01 c0");
    if (ad>0)
    {
       chk1=Byte(ad-3);
       chk2=Byte(ad+2);
       chk3=Byte(ad+3);

       if (chk1==0x8b && chk2==0x89 && chk2==0x45 )
       {
           ini=ad-3;
           fin=ad+5;
           Message("Encontrado add_eax_eax en:%x l:%x\n",ini,fin);
           Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}



//.text:0000000000452649 8B 45 F4                                         mov     eax, [rbp+var_C]
//.text:000000000045264C 8B 55 F8                                         mov     edx, [rbp+var_8]
//.text:000000000045264F 01 D0                                            add     eax, edx
//.text:0000000000452651 89 45 FC                                         mov     [rbp+var_4], eax
static add_eax_edx()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"01 d0");
    if (ad>0)
    {
       chk1=Byte(ad-3);
       chk2=Byte(ad-6);
       chk3=Byte(ad+2);
       chk4=Byte(ad+3);
       if (chk1==0x8b && chk2==0x8b && chk3==0x89  && chk4==0x45)
       {
           ini=ad-6;
           fin=ad+5;
           Message("Encontrado add_eax_edx en:%x l:%x\n",ini,fin);
           Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}

//.text:0000000000452654 8B 45 F8                                         mov     eax, [rbp+var_8]
//.text:0000000000452657 31 45 FC                                         xor     [rbp+var_4], eax
//.text:000000000045265A C7 45 F8 00 00 00 00                             mov     [rbp+var_8], 0

//.text:0000000000453D63 8B 45 F4                                         mov     eax, [rbp+var_C]
//.text:0000000000453D66 31 45 F8                                         xor     [rbp+var_8], eax
//.text:0000000000453D69 C7 85 10 FF FF FF 00 00 00 00                    mov     [rbp+var_F0], 0

static xor_ofs_eax()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
```

```
do
{

   ad=FindBinary(address,SEARCH_DOWN,"31 45");
   if (ad>0)
   {
      chk1=Byte(ad-3);
           chk3=Byte(ad+3);
      chk4=Byte(ad+4);
      if (chk1==0x8b && chk3==0xc7 )
      {
          ini=ad-3;
          if (chk4==0x45)
          {
            fin=ad+10;
                           Message("Encontrado xor_ofs_eax corto en:%x l:%x\n",ini,fin);
                           Nopea(ini,fin);
            MakeUnkn(ini,fin-ini);
            MakeCode(ini);
                    }
          if (chk4==0x85)
          {
                           fin=ad+13;
                           Message("Encontrado xor_ofs_eax largo en:%x l:%x\n",ini,fin);
                           Nopea(ini,fin);
            MakeUnkn(ini,fin-ini);
            MakeCode(ini);
                    }

      }
   }
   address=ad+1;
}while(ad>0);
}



//.text:00000000004370B7 8B 45 F8                                         mov     eax, [rbp+var_8]
//.text:00000000004370BA 8B 55 EC                                         mov     edx, [rbp+var_14]
//.text:00000000004370BD 31 D0                                            xor     eax, edx
//.text:00000000004370BF 89 45 F4                                         mov     [rbp+var_C], eax

//.text:000000000040473E 8B 45 FC                                         mov     eax, [rbp+var_4]
//.text:0000000000404741 8B 55 E4                                         mov     edx, [rbp+var_1C]
//.text:0000000000404744 31 D0                                            xor     eax, edx
//.text:0000000000404746 89 85 48 FF FF FF                                mov     [rbp+var_B8], eax
static xor_eax()
{
   auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
   address=ScreenEA();
   ad=0;
do
{

   ad=FindBinary(address,SEARCH_DOWN,"31 d0");
   if (ad>0)
   {
      chk1=Byte(ad-3);
      chk2=Byte(ad-6);
      chk3=Byte(ad+2);
           chk4=Byte(ad+3);
      if (chk1==0x8b && chk2==0x8b && chk3==0x89 && chk4==0x45 )
      {
               ini=ad-6;
          fin=ad+5;
          Message("Encontrado xor_eax_corto en:%x l:%x\n",ini,fin);
               Nopea(ini,fin);
          MakeUnkn(ini,fin-ini);
          MakeCode(ini);

      }
      if (chk1==0x8b && chk2==0x8b && chk3==0x89 && chk4==0x85 )
      {
               ini=ad-6;
          fin=ad+8;
          Message("Encontrado xor_eax_largo en:%x l:%x\n",ini,fin);
               Nopea(ini,fin);
          MakeUnkn(ini,fin-ini);
          MakeCode(ini);

      }
   }
   address=ad+1;
}while(ad>0);
}



//.text:00000000004370A1 8B 45 FC                                         mov     eax, [rbp+var_4]
//.text:00000000004370A4 8B 55 E8                                         mov     edx, [rbp+var_18]
//.text:00000000004370A7 21 D0                                            and     eax, edx
```

```
//.text:00000000004370A9 89 45 F8                                         mov     [rbp+var_8], eax

//.text:000000000040527B 8B 45 EC                                         mov     eax, [rbp+var_14]
//.text:000000000040527E 8B 55 F4                                         mov     edx, [rbp+var_C]
//.text:0000000000405281 21 D0                                            and     eax, edx
//.text:0000000000405283 89 85 48 FF FF FF                                mov     [rbp+var_B8], eax

static and_eax()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"21 D0");
    if (ad>0)
    {
       chk1=Byte(ad-3);
       chk2=Byte(ad-6);
       chk3=Byte(ad+2);
       chk4=Byte(ad+3);
       if (chk1==0x8b && chk2==0x8b && chk3==0x89 && chk4==0x45 )
       {
                ini=ad-6;
           fin=ad+5;
           Message("Encontrado and_eax_corto en:%x l:%x\n",ini,fin);
           Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
       }
            if (chk1==0x8b && chk2==0x8b && chk3==0x89 && chk4==0x85 )
       {
                ini=ad-6;
           fin=ad+8;
           Message("Encontrado and_eax_largo en:%x l:%x\n",ini,fin);
           Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}



//.text:000000000043BBA2 8B 45 E8                                         mov     eax, [rbp+var_18]
//.text:000000000043BBA5 8B 55 E4                                         mov     edx, [rbp+var_1C]
//.text:000000000043BBA8 89 D1                                            mov     ecx, edx
//.text:000000000043BBAA 29 C1                                            sub     ecx, eax
//.text:000000000043BBAC 89 C8                                            mov     eax, ecx
//.text:000000000043BBAE 89 45 98                                         mov     [rbp+var_68], eax
static sub_ecx_corto()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"29 C1");
    if (ad>0)
    {
       chk1=Byte(ad-2);
       chk2=Byte(ad-5);
       chk3=Byte(ad-8);

       chk4=Byte(ad+2);
       chk5=Byte(ad+4);

       if (chk1==0x89 && chk2==0x8b && chk3==0x8b && chk4=0x89 && chk5==0x89)
       {
                ini=ad-8;
           fin=ad+7;
           Message("Encontrado sub_ecx_corto  en:%x l:%x\n",ini,fin);
                  Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}
//.text:0000000000436BCF 8B 45 F4                                         mov     eax, [rbp+var_C]
//.text:0000000000436BD2 8B 55 EC                                         mov     edx, [rbp+var_14]
//.text:0000000000436BD5 89 D7                                            mov     edi, edx
//.text:0000000000436BD7 29 C7                                            sub     edi, eax
//.text:0000000000436BD9 89 F8                                            mov     eax, edi
//.text:0000000000436BDB 89 45 E4                                         mov     [rbp+var_1C], eax
```

```
static sub_edi_corto()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"29 C7");
    if (ad>0)
    {
       chk1=Byte(ad-2);
       chk2=Byte(ad-5);
       chk3=Byte(ad-8);

       chk4=Byte(ad+2);
       chk5=Byte(ad+4);

       if (chk1==0x89 && chk2==0x8b && chk3==0x8b && chk4=0x89 && chk5==0x89)
       {
            ini=ad-8;
           fin=ad+7;
           Message("Encontrado sub_edi_corto  en:%x l:%x\n",ini,fin);
                Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}


//.text:0000000000436E63 8B 45 E8                                         mov     eax, [rbp+var_18]
//.text:0000000000436E66 0F AF 45 E8                                      imul    eax, [rbp+var_18]
//.text:0000000000436E6A 89 45 E4                                         mov     [rbp+var_1C], eax

//.text:0000000000405180 8B 45 F8                                         mov     eax, [rbp+var_8]
//.text:0000000000405183 0F AF 45 EC                                      imul    eax, [rbp+var_14]
//.text:0000000000405187 89 85 48 FF FF FF                                mov     [rbp+var_B8], eax

static MALimul_eax_ofs()
{
    auto chk4,chk5,chk3,chk2,ini,fin,va,x,len,address,v1,v2,ad,chk1,destino,salto;
    address=ScreenEA();
    ad=0;
 do
 {

    ad=FindBinary(address,SEARCH_DOWN,"0f af");
    if (ad>0)
    {
       chk1=Byte(ad-3);
       chk2=Byte(ad+4);
            chk3=Byte(ad+5);
       if (chk1==0x8b && chk2==0x89 && chk3==0x45 )
       {
                ini=ad-3;
           fin=ad+7;
           Message("Encontrado imul_eax_ofs corto en:%x l:%x\n",ini,fin);
                Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
       }
            if (chk1==0x8b && chk2==0x89 && chk3==0x85 )
       {
                ini=ad-3;
           fin=ad+10;
           Message("Encontrado imul_eax_ofs largo en:%x l:%x\n",ini,fin);
                Nopea(ini,fin);
           MakeUnkn(ini,fin-ini);
           MakeCode(ini);
       }
    }
    address=ad+1;
 }while(ad>0);
}
```

This is a simple and dirty .idc script to clean the trash code. I made 2 main functions to do the analysis “limpia()” to clean the trash code and “saltos()” to remove the flow obfuscation. The first function can used in a debugger session and the program return same as a dirty code execution. The “saltos()” affect the flow but is a help to read the code into dead analysis.

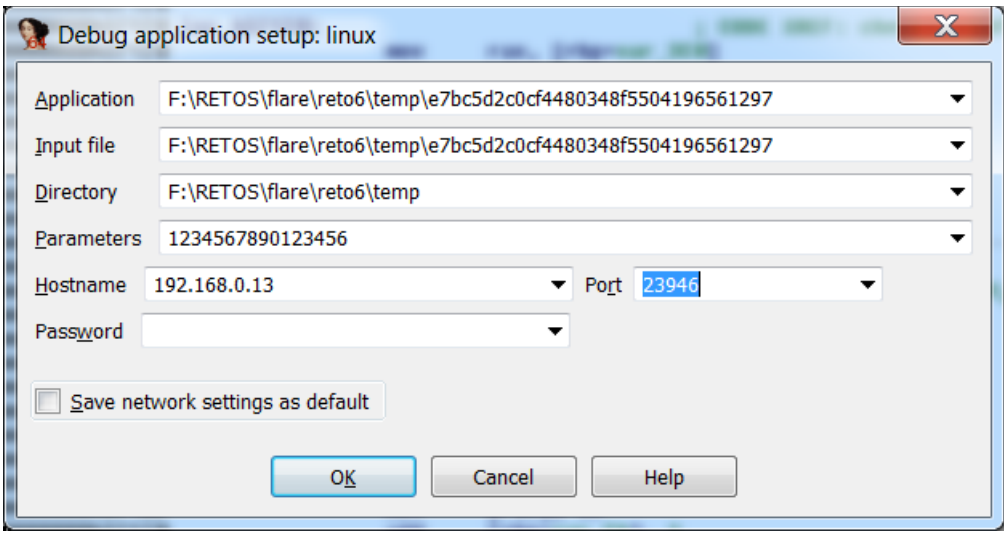
Then I load the .idc and execute “limpia();” and now I can start search into this crapy code.

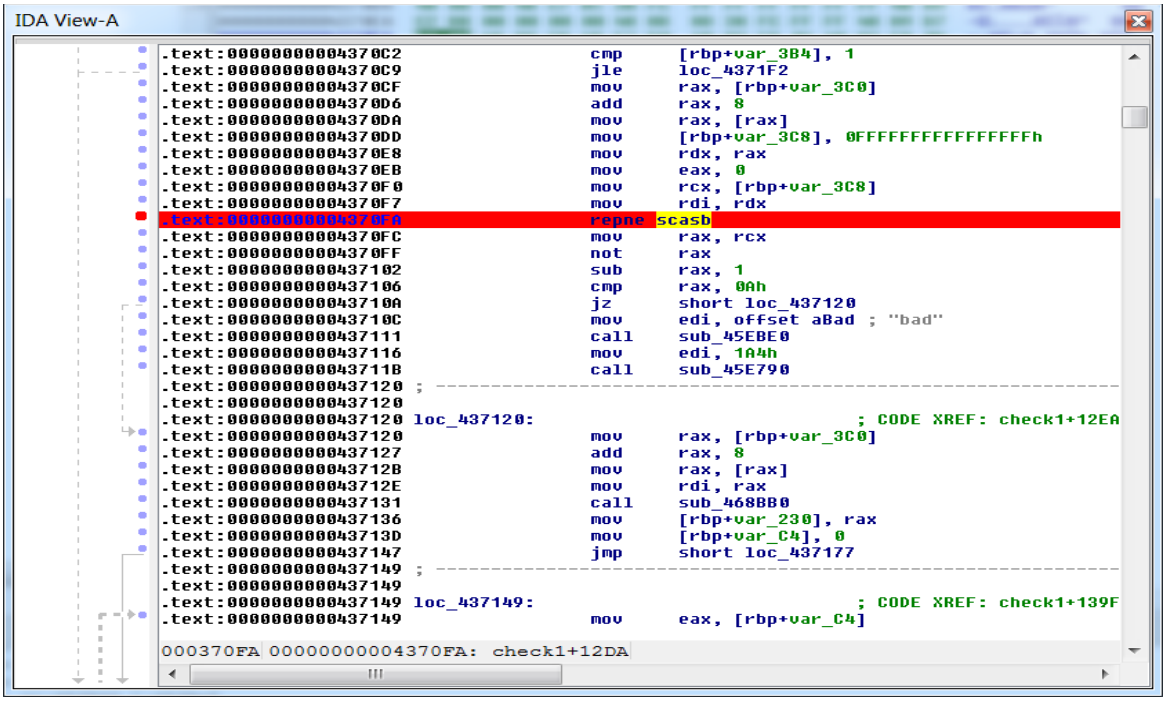
When I start the anlisys I remember what at address 0x00000000043710C they show a “bad” message and this message have relation with a string comparation.

```
.text:00000000004370C9                jle     loc_4371F2
.text:00000000004370CF                mov     rax, [rbp+var_3C0]
.text:00000000004370D6                add     rax, 8
.text:00000000004370DA                mov     rax, [rax]
.text:00000000004370DD                mov     [rbp+var_3C8], 0FFFFFFFFFFFFFFFFh
.text:00000000004370E8                mov     rdx, rax
.text:00000000004370EB                mov     eax, 0
.text:00000000004370F0                mov     rcx, [rbp+var_3C8]
.text:00000000004370F7                mov     rdi, rdx
.text:00000000004370FA                repne scasb
.text:00000000004370FC                mov     rax, rcx
.text:00000000004370FF                not     rax
.text:0000000000437102                sub     rax, 1
.text:0000000000437106                cmp     rax, 0Ah
.text:000000000043710A                jz      short loc_437120
.text:000000000043710C                mov     edi, offset aBad ; "bad"
.text:0000000000437111                call    sub_45EBE0
.text:0000000000437116                mov     edi, 1A4h
.text:000000000043711B                call    sub_45E790
```

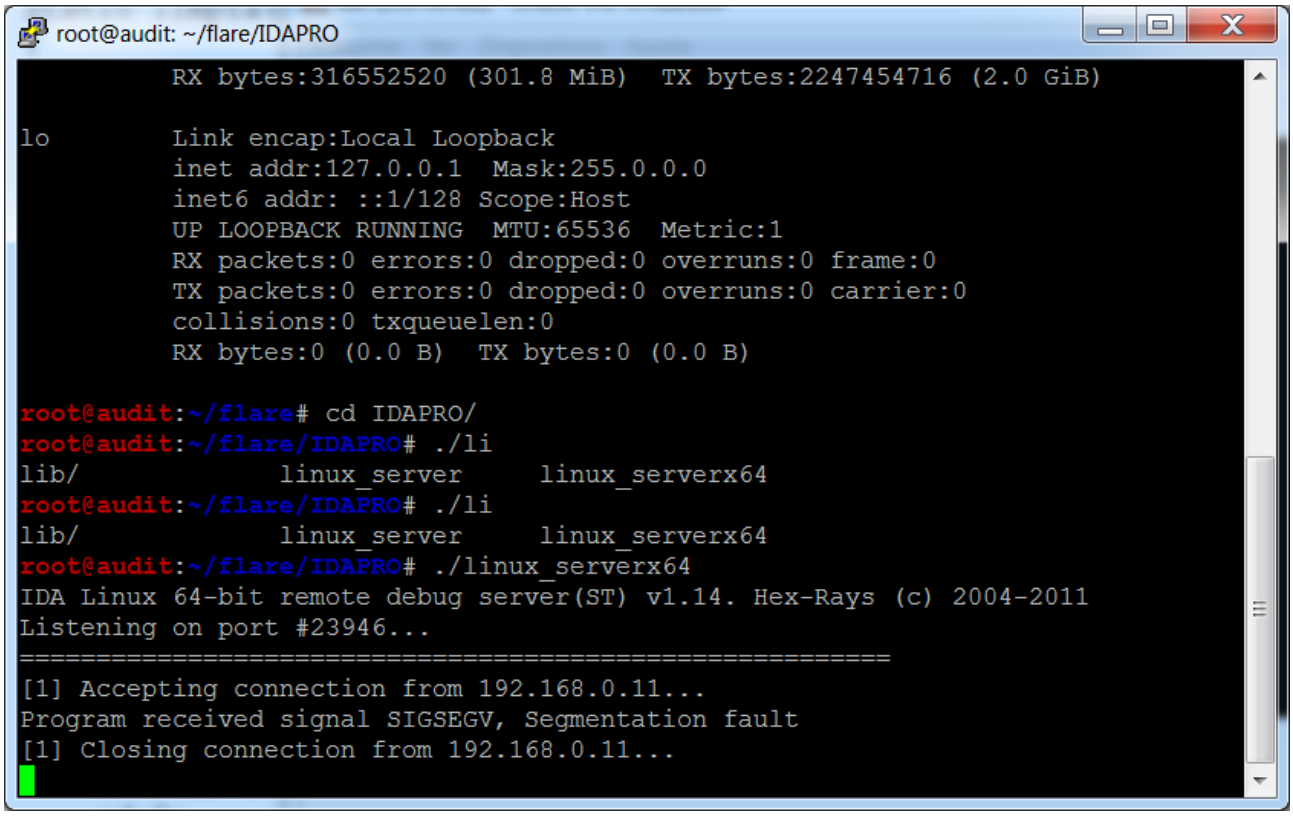
I going to the start of this function and rename to “check1” to can view from where its called. Is called from this other function what I named “mainCrapFunction” and is called from the main function. Then I assume this its the function to inspect.

Well, coming back to the code into “check1” function they are comparing for 16 chars into the param 1. At this point I can execute the program with a breakpoint and get the param 1. Configure ida to start debugging and set a bpx at this address “0x4370fa”:





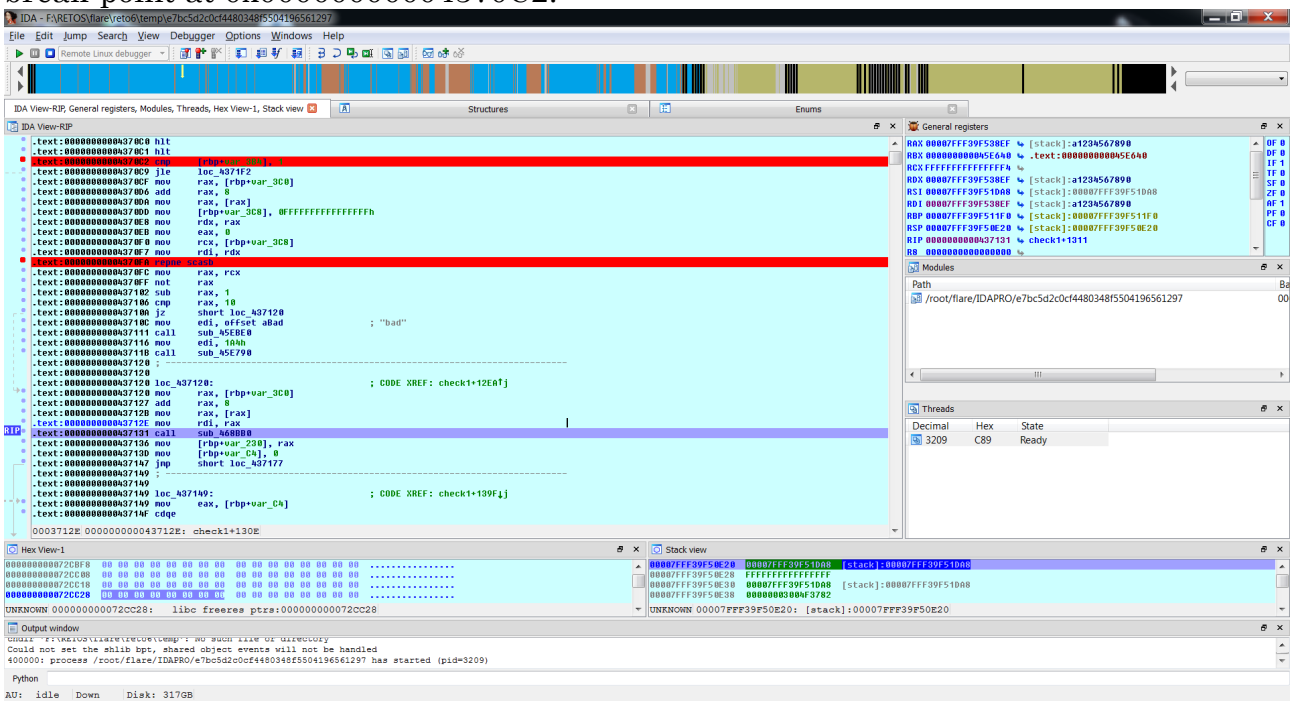
Then press “F9” to start process and wait to reach the breakpoint, humm some extrange hapend and never reach the breakpoint but in the output window can see a SIGSEGV error:



But if I execute the program in a shell with the same params the program work without problems, this suggest a some anti-debug trick. Then I going to locate this "Program received signal SIGSEGV, Segmentation fault" string and analyze from where its called.

```
.text:000000000041F20C mov     eax, 0
.text:000000000041F211 call    sub_4742B0
.text:000000000041F216 shr     rax, 3Fh
.text:000000000041F21A test    al, al
.text:000000000041F21C jz      short loc_41F232
.text:000000000041F21E mov     edi, offset aProgramReceive     ; "Program received signal SIGSEGV,
Segmen"...
.text:000000000041F223 call    sub_45EBE0
.text:000000000041F228 mov     edi, 2329h
.text:000000000041F22D call    sub_45E790
.text:000000000041F232 ; ---------------------------------------------------------------------------
.
```

At this code can see how a function is called, if get a result of 0 the "sigsev" message don't happen. Put a break-point at 0x000000000041F21C and execute the program into debugger, when the break-point are reached I change the result of ZF to take the jump and skip the end of the program. And now I land into the previous break-point at 0x00000000004370C2:



At address 0x0000000000437106 they check the length of parameter 1 to match with 10 :

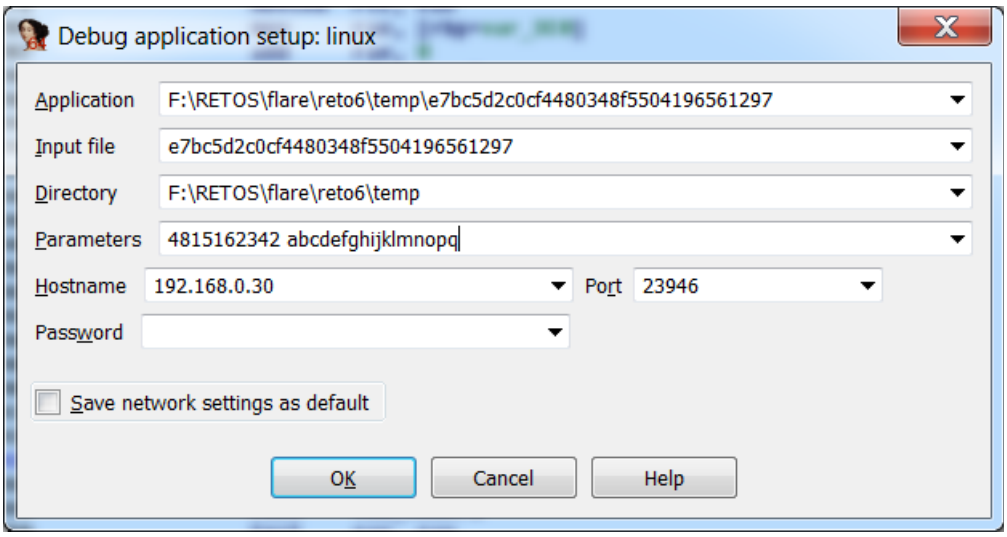
```
.text:00000000004370FA                repne scasb
.text:00000000004370FC                mov     rax, rcx
.text:00000000004370FF                not     rax
.text:0000000000437102                sub     rax, 1
.text:0000000000437106                cmp     rax, 10
.text:000000000043710A                jz      short loc_437120
.text:000000000043710C                mov     edi, offset aBad ; "bad"
.text:0000000000437111                call    sub_45EBE0
.text:0000000000437116                mov     edi, 1A4h
.text:000000000043711B                call    sub_45E790
```

And some lines bottom can see this code:

```
.text:0000000000437149 loc_437149:                             ; CODE XREF: check1+139F#j
.text:0000000000437149                 mov     eax, [rbp+var_C4]
.text:000000000043714F                 cdqe
.text:0000000000437151                 add     rax, [rbp+var_230]
.text:0000000000437158                 mov     edx, [rbp+var_C4]
.text:000000000043715E                 movsxd  rdx, edx
.text:0000000000437161                 add     rdx, [rbp+var_230]
.text:0000000000437168                 movzx   edx, byte ptr [rdx]
.text:000000000043716B                 xor     edx, 56h
.text:000000000043716E                 mov     [rax], dl
.text:0000000000437170                 add     [rbp+var_C4], 1
.text:0000000000437177
.text:0000000000437177 loc_437177:                             ; CODE XREF: check1+1327#j
.text:0000000000437177                 mov     eax, [rbp+var_C4]
.text:000000000043717D                 movsxd  rsi, eax
.text:0000000000437180                 mov     rax, [rbp+var_3C0]
.text:0000000000437187                 add     rax, 8
.text:000000000043718B                 mov     rax, [rax]
.text:000000000043718E                 mov     [rbp+var_3C8], 0FFFFFFFFFFFFFFFFh
.text:0000000000437199                 mov     rdx, rax
.text:000000000043719C                 mov     eax, 0
.text:00000000004371A1                 mov     rcx, [rbp+var_3C8]
.text:00000000004371A8                 mov     rdi, rdx
.text:00000000004371AB                 repne scasb
.text:00000000004371AD                 mov     rax, rcx
.text:00000000004371B0                 not     rax
.text:00000000004371B3                 sub     rax, 1
.text:00000000004371B7                 cmp     rsi, rax
.text:00000000004371BA                 setb    al
.text:00000000004371BD                 test    al, al
.text:00000000004371BF                 jnz     short loc_437149
.text:00000000004371C1                 mov     rax, [rbp+var_230]
.text:00000000004371C8                 mov     edx, 0Ah
.text:00000000004371CD                 mov     esi, offset aBngcgDebd ; "bngcg`debd"
```

This code do a XOR 0x65 with each character of the parameter 1, and then compare the result with the string “bngcg`debd”. Its clear, doing the same operation with the string I get the real parameter 1: “4815162342”.

Now configure IDA to pass this parameter:



Set the break-point at 0x000000000041F21C and 0x00000000004370C2, change the value of RAX to 0 after debugger check and leave the program run to reach the next break-point.

At this point I can check for the correct comparation of parameter 1 and if I leave the program follow their execution they reach at same point if the program was executed in the shell without debugger.

Is time to follow the code to know where is the flag … I set only a break-point at debugger check function to bypass this check.

And another at 0x000000000045AB3F (return address of "check1f()" function) and if the next function is called the program seem freeze, then lets go to find where the program enter in this "freeze" state. Step into the function "sub_442A8F" now named as "MAIN_PAUSE". After some debugging I can determine the address into this function where is called the "pause" this happen at 00000000004436FB. Then I set a break-point at this address and skip this execution. A trick its patch this call at runtime:

Shift+F2: to open idc command dialog.
Enter at dialog :"PatchDword(rip,0x90909090); PatchByte(rip+4,0x90);"

The obfuscation code do the execution of this call few times, for this reason I do the patch.

Then I set another break-point at return of "MAIN_PAUSE" function and leave the code to be executed until reach the last break-point.

Now its time to trace to until get this address 000000000044BB2B:

```
 text:000000000044BB01 48 89 C2                    mov     rdx, rax
.text:000000000044BB04 BE D8 03 00 00              mov     esi, 3D8h
.text:000000000044BB09 BF 00 99 72 00              mov     edi, offset byte_729900
.text:000000000044BB0E E8 51 56 FB FF              call    sub_401164
.text:000000000044BB13 48 8D 95 C0 FB FF FF        lea     rdx, [rbp+var_440]
.text:000000000044BB1A 48 8B 85 B0 FB FF FF        mov     rax, [rbp+var_450]
.text:000000000044BB21 48 83 C0 10                 add     rax, 10h
.text:000000000044BB25 48 8B 00                    mov     rax, [rax]
.text:000000000044BB28 48 89 C7                    mov     rdi, rax
.text:000000000044BB2B FF D2                       call    rdx
.text:000000000044BB2D
```

At this address I can locate a "call rdx" and following this call I arrive to the last part of the challenger:

```
 stack]:00007FFF754DED80 48 89 F8                    mov     rax, rdi
[stack]:00007FFF754DED83 E8 00 00 00 00              call    $+5
[stack]:00007FFF754DED88 48 8B 1C 24                 mov     rbx, [rsp]
[stack]:00007FFF754DED8C 48 83 C3 0A                 add     rbx, 0Ah
[stack]:00007FFF754DED90 EB 0A                       jmp     short loc_7FFF754DED9C
[stack]:00007FFF754DED92                             ;
-------------------------------------------------------------------------------
```

```
[stack]:00007FFF754DED92 48 31 D2                    xor     rdx, rdx
[stack]:00007FFF754DED95 48 31 C0                    xor     rax, rax
[stack]:00007FFF754DED98 B0 3C                       mov     al, 3Ch
[stack]:00007FFF754DED9A 0F 05                       syscall
[stack]:00007FFF754DED9C
[stack]:00007FFF754DED9C                             loc_7FFF754DED9C:                       ; CODE XREF:
[stack]:00007FFF754DED90#j
[stack]:00007FFF754DED9C C0 08 F2                    ror     byte ptr [rax], 0F2h
[stack]:00007FFF754DED9F 80 38 1B                    cmp     byte ptr [rax], 1Bh
[stack]:00007FFF754DEDA2 74 02                       jz      short loc_7FFF754DEDA6
[stack]:00007FFF754DEDA4 FF E3                       jmp     rbx
[stack]:00007FFF754DEDA6                             ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEDA6
[stack]:00007FFF754DEDA6                             loc_7FFF754DEDA6:                       ; CODE XREF:
[stack]:00007FFF754DEDA2#j
[stack]:00007FFF754DEDA6 48 83 C0 01                 add     rax, 1
[stack]:00007FFF754DEDAA 80 30 40                    xor     byte ptr [rax], 40h
[stack]:00007FFF754DEDAD 80 30 F2                    xor     byte ptr [rax], 0F2h
[stack]:00007FFF754DEDB0 80 30 B3                    xor     byte ptr [rax], 0B3h
[stack]:00007FFF754DEDB3 80 38 30                    cmp     byte ptr [rax], 30h
[stack]:00007FFF754DEDB6 74 02                       jz      short loc_7FFF754DEDBA
[stack]:00007FFF754DEDB8 FF E3                       jmp     rbx
[stack]:00007FFF754DEDBA                             ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEDBA
[stack]:00007FFF754DEDBA                             loc_7FFF754DEDBA:                       ; CODE XREF:
[stack]:00007FFF754DEDB6#j
[stack]:00007FFF754DEDBA 48 83 C0 01                 add     rax, 1
[stack]:00007FFF754DEDBE 80 30 71                    xor     byte ptr [rax], 71h
[stack]:00007FFF754DEDC1 80 38 1F                    cmp     byte ptr [rax], 1Fh
[stack]:00007FFF754DEDC4 74 02                       jz      short loc_7FFF754DEDC8
[stack]:00007FFF754DEDC6 FF E3                       jmp     rbx
[stack]:00007FFF754DEDC8                             ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEDC8
[stack]:00007FFF754DEDC8                             loc_7FFF754DEDC8:                       ; CODE XREF:
[stack]:00007FFF754DEDC4#j
[stack]:00007FFF754DEDC8 48 83 C0 01                 add     rax, 1
[stack]:00007FFF754DEDCC 80 00 A3                    add     byte ptr [rax], 0A3h
[stack]:00007FFF754DEDCF C0 08 BC                    ror     byte ptr [rax], 0BCh
[stack]:00007FFF754DEDD2 80 38 B0                    cmp     byte ptr [rax], 0B0h
[stack]:00007FFF754DEDD5 74 02                       jz      short loc_7FFF754DEDD9
[stack]:00007FFF754DEDD7 FF E3                       jmp     rbx
[stack]:00007FFF754DEDD9                             ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEDD9
[stack]:00007FFF754DEDD9                             loc_7FFF754DEDD9:                       ; CODE XREF:
[stack]:00007FFF754DEDD5#j
[stack]:00007FFF754DEDD9 48 83 C0 01                 add     rax, 1
[stack]:00007FFF754DEDDD 80 28 79                    sub     byte ptr [rax], 79h
[stack]:00007FFF754DEDE0 80 38 E8                    cmp     byte ptr [rax], 0E8h
[stack]:00007FFF754DEDE3 74 02                       jz      short loc_7FFF754DEDE7
[stack]:00007FFF754DEDE5 FF E3                       jmp     rbx
[stack]:00007FFF754DEDE7                             ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEDE7
[stack]:00007FFF754DEDE7                             loc_7FFF754DEDE7:                       ; CODE XREF:
[stack]:00007FFF754DEDE3#j
[stack]:00007FFF754DEDE7 48 83 C0 01                 add     rax, 1
[stack]:00007FFF754DEDEB C0 08 82                    ror     byte ptr [rax], 82h
[stack]:00007FFF754DEDEE 80 28 28                    sub     byte ptr [rax], 28h
[stack]:00007FFF754DEDF1 80 38 F6                    cmp     byte ptr [rax], 0F6h
[stack]:00007FFF754DEDF4 74 02                       jz      short loc_7FFF754DEDF8
[stack]:00007FFF754DEDF6 FF E3                       jmp     rbx
[stack]:00007FFF754DEDF8                             ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEDF8
[stack]:00007FFF754DEDF8                             loc_7FFF754DEDF8:                       ; CODE XREF:
[stack]:00007FFF754DEDF4#j
[stack]:00007FFF754DEDF8 48 83 C0 01                 add     rax, 1
[stack]:00007FFF754DEDFC 80 28 B0                    sub     byte ptr [rax], 0B0h
[stack]:00007FFF754DEDFF C0 08 4D                    ror     byte ptr [rax], 4Dh
[stack]:00007FFF754DEE02 80 00 2C                    add     byte ptr [rax], 2Ch
[stack]:00007FFF754DEE05 80 38 1F                    cmp     byte ptr [rax], 1Fh
[stack]:00007FFF754DEE08 74 02                       jz      short loc_7FFF754DEE0C
[stack]:00007FFF754DEE0A FF E3                       jmp     rbx
[stack]:00007FFF754DEE0C                             ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEE0C
[stack]:00007FFF754DEE0C                             loc_7FFF754DEE0C:                       ; CODE XREF:
[stack]:00007FFF754DEE08#j
[stack]:00007FFF754DEE0C 48 83 C0 01                 add     rax, 1
[stack]:00007FFF754DEE10 80 00 54                    add     byte ptr [rax], 54h
[stack]:00007FFF754DEE13 C0 00 99                    rol     byte ptr [rax], 99h
[stack]:00007FFF754DEE16 80 30 B8                    xor     byte ptr [rax], 0B8h
[stack]:00007FFF754DEE19 C0 08 2A                    ror     byte ptr [rax], 2Ah
[stack]:00007FFF754DEE1C 80 00 3F                    add     byte ptr [rax], 3Fh
[stack]:00007FFF754DEE1F 80 38 AF                    cmp     byte ptr [rax], 0AFh
[stack]:00007FFF754DEE22 74 02                       jz      short loc_7FFF754DEE26
[stack]:00007FFF754DEE24 FF E3                       jmp     rbx
```

```
[stack]:00007FFF754DEE26                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEE26
[stack]:00007FFF754DEE26                                loc_7FFF754DEE26:                       ; CODE XREF:
[stack]:00007FFF754DEE22#j
[stack]:00007FFF754DEE26 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEE2A C0 08 BA                       ror     byte ptr [rax], 0BAh
[stack]:00007FFF754DEE2D 80 38 5D                       cmp     byte ptr [rax], 5Dh
[stack]:00007FFF754DEE30 74 02                          jz      short loc_7FFF754DEE34
[stack]:00007FFF754DEE32 FF E3                          jmp     rbx
[stack]:00007FFF754DEE34                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEE34
[stack]:00007FFF754DEE34                                loc_7FFF754DEE34:                       ; CODE XREF:
[stack]:00007FFF754DEE30#j
[stack]:00007FFF754DEE34 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEE38 80 30 ED                       xor     byte ptr [rax], 0EDh
[stack]:00007FFF754DEE3B C0 08 6C                       ror     byte ptr [rax], 6Ch
[stack]:00007FFF754DEE3E 80 00 30                       add     byte ptr [rax], 30h
[stack]:00007FFF754DEE41 80 38 29                       cmp     byte ptr [rax], 29h
[stack]:00007FFF754DEE44 74 02                          jz      short loc_7FFF754DEE48
[stack]:00007FFF754DEE46 FF E3                          jmp     rbx
[stack]:00007FFF754DEE48                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEE48
[stack]:00007FFF754DEE48                                loc_7FFF754DEE48:                       ; CODE XREF:
[stack]:00007FFF754DEE44#j
[stack]:00007FFF754DEE48 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEE4C 80 28 BF                       sub     byte ptr [rax], 0BFh
[stack]:00007FFF754DEE4F 80 38 B5                       cmp     byte ptr [rax], 0B5h
[stack]:00007FFF754DEE52 74 02                          jz      short loc_7FFF754DEE56
[stack]:00007FFF754DEE54 FF E3                          jmp     rbx
[stack]:00007FFF754DEE56                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEE56
[stack]:00007FFF754DEE56                                loc_7FFF754DEE56:                       ; CODE XREF:
[stack]:00007FFF754DEE52#j
[stack]:00007FFF754DEE56 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEE5A C0 00 BC                       rol     byte ptr [rax], 0BCh
[stack]:00007FFF754DEE5D 80 00 8C                       add     byte ptr [rax], 8Ch
[stack]:00007FFF754DEE60 C0 00 7B                       rol     byte ptr [rax], 7Bh
[stack]:00007FFF754DEE63 80 28 31                       sub     byte ptr [rax], 31h
[stack]:00007FFF754DEE66 80 00 63                       add     byte ptr [rax], 63h
[stack]:00007FFF754DEE69 80 38 A5                       cmp     byte ptr [rax], 0A5h
[stack]:00007FFF754DEE6C 74 02                          jz      short loc_7FFF754DEE70
[stack]:00007FFF754DEE6E FF E3                          jmp     rbx
[stack]:00007FFF754DEE70                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEE70
[stack]:00007FFF754DEE70                                loc_7FFF754DEE70:                       ; CODE XREF:
[stack]:00007FFF754DEE6C#j
[stack]:00007FFF754DEE70 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEE74 C0 00 20                       rol     byte ptr [rax], 20h
[stack]:00007FFF754DEE77 C0 00 16                       rol     byte ptr [rax], 16h
[stack]:00007FFF754DEE7A 80 30 AE                       xor     byte ptr [rax], 0AEh
[stack]:00007FFF754DEE7D C0 00 98                       rol     byte ptr [rax], 98h
[stack]:00007FFF754DEE80 80 38 F3                       cmp     byte ptr [rax], 0F3h
[stack]:00007FFF754DEE83 74 02                          jz      short loc_7FFF754DEE87
[stack]:00007FFF754DEE85 FF E3                          jmp     rbx
[stack]:00007FFF754DEE87                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEE87
[stack]:00007FFF754DEE87                                loc_7FFF754DEE87:                       ; CODE XREF:
[stack]:00007FFF754DEE83#j
[stack]:00007FFF754DEE87 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEE8B C0 08 6E                       ror     byte ptr [rax], 6Eh
[stack]:00007FFF754DEE8E 80 00 D2                       add     byte ptr [rax], 0D2h
[stack]:00007FFF754DEE91 80 38 A6                       cmp     byte ptr [rax], 0A6h
[stack]:00007FFF754DEE94 74 02                          jz      short loc_7FFF754DEE98
[stack]:00007FFF754DEE96 FF E3                          jmp     rbx
[stack]:00007FFF754DEE98                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEE98
[stack]:00007FFF754DEE98                                loc_7FFF754DEE98:                       ; CODE XREF:
[stack]:00007FFF754DEE94#j
[stack]:00007FFF754DEE98 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEE9C 80 00 34                       add     byte ptr [rax], 34h
[stack]:00007FFF754DEE9F 80 38 62                       cmp     byte ptr [rax], 62h
[stack]:00007FFF754DEEA2 74 02                          jz      short loc_7FFF754DEEA6
[stack]:00007FFF754DEEA4 FF E3                          jmp     rbx
[stack]:00007FFF754DEEA6                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEEA6
[stack]:00007FFF754DEEA6                                loc_7FFF754DEEA6:                       ; CODE XREF:
[stack]:00007FFF754DEEA2#j
[stack]:00007FFF754DEEA6 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEEAA 80 00 CD                       add     byte ptr [rax], 0CDh
[stack]:00007FFF754DEEAD 80 28 10                       sub     byte ptr [rax], 10h
[stack]:00007FFF754DEEB0 80 00 62                       add     byte ptr [rax], 62h
[stack]:00007FFF754DEEB3 80 30 B2                       xor     byte ptr [rax], 0B2h
[stack]:00007FFF754DEEB6 80 38 32                       cmp     byte ptr [rax], 32h
```

```
[stack]:00007FFF754DEEB9 74 02                          jz      short loc_7FFF754DEEBD
[stack]:00007FFF754DEEBB FF E3                          jmp     rbx
[stack]:00007FFF754DEEBD                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEEBD
[stack]:00007FFF754DEEBD                                loc_7FFF754DEEBD:                       ; CODE XREF:
[stack]:00007FFF754DEEB9#j
[stack]:00007FFF754DEEBD 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEEC1 80 30 B7                       xor     byte ptr [rax], 0B7h
[stack]:00007FFF754DEEC4 80 30 73                       xor     byte ptr [rax], 73h
[stack]:00007FFF754DEEC7 C0 08 07                       ror     byte ptr [rax], 7
[stack]:00007FFF754DEECA 80 38 EB                       cmp     byte ptr [rax], 0EBh
[stack]:00007FFF754DEECD 74 02                          jz      short loc_7FFF754DEED1
[stack]:00007FFF754DEECF FF E3                          jmp     rbx
[stack]:00007FFF754DEED1                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEED1
[stack]:00007FFF754DEED1                                loc_7FFF754DEED1:                       ; CODE XREF:
[stack]:00007FFF754DEECD#j
[stack]:00007FFF754DEED1 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEED5 80 00 34                       add     byte ptr [rax], 34h
[stack]:00007FFF754DEED8 80 28 61                       sub     byte ptr [rax], 61h
[stack]:00007FFF754DEEDB C0 08 36                       ror     byte ptr [rax], 36h
[stack]:00007FFF754DEEDE 80 00 5B                       add     byte ptr [rax], 5Bh
[stack]:00007FFF754DEEE1 80 28 4C                       sub     byte ptr [rax], 4Ch
[stack]:00007FFF754DEEE4 80 38 0B                       cmp     byte ptr [rax], 0Bh
[stack]:00007FFF754DEEE7 74 02                          jz      short loc_7FFF754DEEEB
[stack]:00007FFF754DEEE9 FF E3                          jmp     rbx
[stack]:00007FFF754DEEEB                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEEEB
[stack]:00007FFF754DEEEB                                loc_7FFF754DEEEB:                       ; CODE XREF:
[stack]:00007FFF754DEEE7#j
[stack]:00007FFF754DEEEB 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEEEF 80 00 5A                       add     byte ptr [rax], 5Ah
[stack]:00007FFF754DEEF2 80 38 9A                       cmp     byte ptr [rax], 9Ah
[stack]:00007FFF754DEEF5 74 02                          jz      short loc_7FFF754DEEF9
[stack]:00007FFF754DEEF7 FF E3                          jmp     rbx
[stack]:00007FFF754DEEF9                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEEF9
[stack]:00007FFF754DEEF9                                loc_7FFF754DEEF9:                       ; CODE XREF:
[stack]:00007FFF754DEEF5#j
[stack]:00007FFF754DEEF9 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEEFD C0 08 A2                       ror     byte ptr [rax], 0A2h
[stack]:00007FFF754DEF00 80 38 99                       cmp     byte ptr [rax], 99h
[stack]:00007FFF754DEF03 74 02                          jz      short loc_7FFF754DEF07
[stack]:00007FFF754DEF05 FF E3                          jmp     rbx
[stack]:00007FFF754DEF07                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEF07
[stack]:00007FFF754DEF07                                loc_7FFF754DEF07:                       ; CODE XREF:
[stack]:00007FFF754DEF03#j
[stack]:00007FFF754DEF07 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEF0B 80 30 7E                       xor     byte ptr [rax], 7Eh
[stack]:00007FFF754DEF0E 80 28 E7                       sub     byte ptr [rax], 0E7h
[stack]:00007FFF754DEF11 80 38 2B                       cmp     byte ptr [rax], 2Bh
[stack]:00007FFF754DEF14 74 02                          jz      short loc_7FFF754DEF18
[stack]:00007FFF754DEF16 FF E3                          jmp     rbx
[stack]:00007FFF754DEF18                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEF18
[stack]:00007FFF754DEF18                                loc_7FFF754DEF18:                       ; CODE XREF:
[stack]:00007FFF754DEF14#j
[stack]:00007FFF754DEF18 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEF1C 80 28 B8                       sub     byte ptr [rax], 0B8h
[stack]:00007FFF754DEF1F 80 30 86                       xor     byte ptr [rax], 86h
[stack]:00007FFF754DEF22 80 00 4E                       add     byte ptr [rax], 4Eh
[stack]:00007FFF754DEF25 C0 08 4A                       ror     byte ptr [rax], 4Ah
[stack]:00007FFF754DEF28 C0 00 57                       rol     byte ptr [rax], 57h
[stack]:00007FFF754DEF2B 80 38 AF                       cmp     byte ptr [rax], 0AFh
[stack]:00007FFF754DEF2E 74 02                          jz      short loc_7FFF754DEF32
[stack]:00007FFF754DEF30 FF E3                          jmp     rbx
[stack]:00007FFF754DEF32                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEF32
[stack]:00007FFF754DEF32                                loc_7FFF754DEF32:                       ; CODE XREF:
[stack]:00007FFF754DEF2E#j
[stack]:00007FFF754DEF32 48 83 C0 01                    add     rax, 1
[stack]:00007FFF754DEF36 C0 08 86                       ror     byte ptr [rax], 86h
[stack]:00007FFF754DEF39 80 30 E8                       xor     byte ptr [rax], 0E8h
[stack]:00007FFF754DEF3C C0 00 95                       rol     byte ptr [rax], 95h
[stack]:00007FFF754DEF3F 80 30 4A                       xor     byte ptr [rax], 4Ah
[stack]:00007FFF754DEF42 80 30 AD                       xor     byte ptr [rax], 0ADh
[stack]:00007FFF754DEF45 80 38 C3                       cmp     byte ptr [rax], 0C3h
[stack]:00007FFF754DEF48 74 02                          jz      short loc_7FFF754DEF4C
[stack]:00007FFF754DEF4A FF E3                          jmp     rbx
[stack]:00007FFF754DEF4C                                ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEF4C
```

```
[stack]:00007FFF754DEF4C                        loc_7FFF754DEF4C:                       ; CODE XREF:
[stack]:00007FFF754DEF48#j
[stack]:00007FFF754DEF4C 48 83 C0 01            add     rax, 1
[stack]:00007FFF754DEF50 C0 08 45               ror     byte ptr [rax], 45h
[stack]:00007FFF754DEF53 80 30 CC               xor     byte ptr [rax], 0CCh
[stack]:00007FFF754DEF56 80 00 1C               add     byte ptr [rax], 1Ch
[stack]:00007FFF754DEF59 80 38 03               cmp     byte ptr [rax], 3
[stack]:00007FFF754DEF5C 74 02                  jz      short loc_7FFF754DEF60
[stack]:00007FFF754DEF5E FF E3                  jmp     rbx
[stack]:00007FFF754DEF60                        ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEF60
[stack]:00007FFF754DEF60                        loc_7FFF754DEF60:                       ; CODE XREF:
[stack]:00007FFF754DEF5C#j
[stack]:00007FFF754DEF60 48 83 C0 01            add     rax, 1
[stack]:00007FFF754DEF64 80 28 4A               sub     byte ptr [rax], 4Ah
[stack]:00007FFF754DEF67 80 38 E3               cmp     byte ptr [rax], 0E3h
[stack]:00007FFF754DEF6A 74 02                  jz      short loc_7FFF754DEF6E
[stack]:00007FFF754DEF6C FF E3                  jmp     rbx
[stack]:00007FFF754DEF6E                        ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEF6E
[stack]:00007FFF754DEF6E                        loc_7FFF754DEF6E:                       ; CODE XREF:
[stack]:00007FFF754DEF6A#j
[stack]:00007FFF754DEF6E 48 83 C0 01            add     rax, 1
[stack]:00007FFF754DEF72 80 30 A5               xor     byte ptr [rax], 0A5h
[stack]:00007FFF754DEF75 C0 08 90               ror     byte ptr [rax], 90h
[stack]:00007FFF754DEF78 80 38 CA               cmp     byte ptr [rax], 0CAh
[stack]:00007FFF754DEF7B 74 02                  jz      short loc_7FFF754DEF7F
[stack]:00007FFF754DEF7D FF E3                  jmp     rbx
[stack]:00007FFF754DEF7F                        ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEF7F
[stack]:00007FFF754DEF7F                        loc_7FFF754DEF7F:                       ; CODE XREF:
[stack]:00007FFF754DEF7B#j
[stack]:00007FFF754DEF7F 48 83 C0 01            add     rax, 1
[stack]:00007FFF754DEF83 C0 08 DE               ror     byte ptr [rax], 0DEh
[stack]:00007FFF754DEF86 C0 00 36               rol     byte ptr [rax], 36h
[stack]:00007FFF754DEF89 80 30 78               xor     byte ptr [rax], 78h
[stack]:00007FFF754DEF8C 80 28 D8               sub     byte ptr [rax], 0D8h
[stack]:00007FFF754DEF8F 80 38 3E               cmp     byte ptr [rax], 3Eh
[stack]:00007FFF754DEF92 74 02                  jz      short loc_7FFF754DEF96
[stack]:00007FFF754DEF94 FF E3                  jmp     rbx
[stack]:00007FFF754DEF96                        ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEF96
[stack]:00007FFF754DEF96                        loc_7FFF754DEF96:                       ; CODE XREF:
[stack]:00007FFF754DEF92#j
[stack]:00007FFF754DEF96 48 83 C0 01            add     rax, 1
[stack]:00007FFF754DEF9A 80 00 B5               add     byte ptr [rax], 0B5h
[stack]:00007FFF754DEF9D 80 28 AD               sub     byte ptr [rax], 0ADh
[stack]:00007FFF754DEFA0 C0 08 89               ror     byte ptr [rax], 89h
[stack]:00007FFF754DEFA3 C0 00 A2               rol     byte ptr [rax], 0A2h
[stack]:00007FFF754DEFA6 C0 00 11               rol     byte ptr [rax], 11h
[stack]:00007FFF754DEFA9 80 38 D8               cmp     byte ptr [rax], 0D8h
[stack]:00007FFF754DEFAC 74 02                  jz      short loc_7FFF754DEFB0
[stack]:00007FFF754DEFAE FF E3                  jmp     rbx
[stack]:00007FFF754DEFB0                        ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEFB0
[stack]:00007FFF754DEFB0                        loc_7FFF754DEFB0:                       ; CODE XREF:
[stack]:00007FFF754DEFAC#j
[stack]:00007FFF754DEFB0 48 83 C0 01            add     rax, 1
[stack]:00007FFF754DEFB4 80 00 40               add     byte ptr [rax], 40h
[stack]:00007FFF754DEFB7 80 28 21               sub     byte ptr [rax], 21h
[stack]:00007FFF754DEFBA C0 08 C0               ror     byte ptr [rax], 0C0h
[stack]:00007FFF754DEFBD 80 38 82               cmp     byte ptr [rax], 82h
[stack]:00007FFF754DEFC0 74 02                  jz      short loc_7FFF754DEFC4
[stack]:00007FFF754DEFC2 FF E3                  jmp     rbx
[stack]:00007FFF754DEFC4                        ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEFC4
[stack]:00007FFF754DEFC4                        loc_7FFF754DEFC4:                       ; CODE XREF:
[stack]:00007FFF754DEFC0#j
[stack]:00007FFF754DEFC4 48 83 C0 01            add     rax, 1
[stack]:00007FFF754DEFC8 C0 00 E3               rol     byte ptr [rax], 0E3h
[stack]:00007FFF754DEFCB 80 38 7B               cmp     byte ptr [rax], 7Bh
[stack]:00007FFF754DEFCE 74 02                  jz      short loc_7FFF754DEFD2
[stack]:00007FFF754DEFD0 FF E3                  jmp     rbx
[stack]:00007FFF754DEFD2                        ;
-------------------------------------------------------------------------------
[stack]:00007FFF754DEFD2
[stack]:00007FFF754DEFD2                        loc_7FFF754DEFD2:                       ; CODE XREF:
[stack]:00007FFF754DEFCE#j
[stack]:00007FFF754DEFD2 48 83 C0 01            add     rax, 1
[stack]:00007FFF754DEFD6 80 28 78               sub     byte ptr [rax], 78h
[stack]:00007FFF754DEFD9 C0 08 F6               ror     byte ptr [rax], 0F6h
[stack]:00007FFF754DEFDC 80 38 D7               cmp     byte ptr [rax], 0D7h
[stack]:00007FFF754DEFDF 74 02                  jz      short loc_7FFF754DEFE3
[stack]:00007FFF754DEFE1 FF E3                  jmp     rbx
```

After discover this code is clear the job to do, only is needed do the reverse operations to know the values compared with the second parameter and this values get the next string : "l1nhax.hurt.u5.a1l@flare-on.com". This is a trivial job I do it using "ImmunityDebugger" riping the code using same method as previous paper.

. Final Conclusion and The Flag .

. Final Conclusion .

Great challenger and really painful, but IDA always have too much tools to help us to understand the code.

It curious the generation of a BASE64 string with the movs of all the functions of the program.

. The Flag .

```
l1nhax.hurt.u5.a1l@flare-on.com
```

The Flag is: "l1nhax.hurt.u5.a1l@flare-on.com"