# Dark CTF 2020-Rev/strings-writeup

原创

y4ung 于 2020-10-10 19:48:07 发布 194 收藏

分类专栏： ctf 文章标签： ctf

本文链接：https://blog.csdn.net/qq_35056292/article/details/109003201

版权

ctf 专栏收录该内容

35 篇文章 0 订阅

订阅专栏

## 1. 介绍

本题是dark ctf Reverse的第三题：`strings`，网址：https://ctf.darkarmy.xyz/challs

题目描述：Just manipulation of couple of strings…Note: Enclose the final output inside darkCTF{}

## 2. 分析

```
$ file strings
strings: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux
-x86-64.so.2, BuildID[sha1]=ab02c0f8411dc8e2b15e968f6eb7f4b98722bf41, for GNU/Linux 3.2.0, not stripped
$ chmod +x ./strings
$ ./strings
Use this as ur input:  !8u)05/>!#,.W/%H-G
!8u)05/>!#,.W/%H-G
```

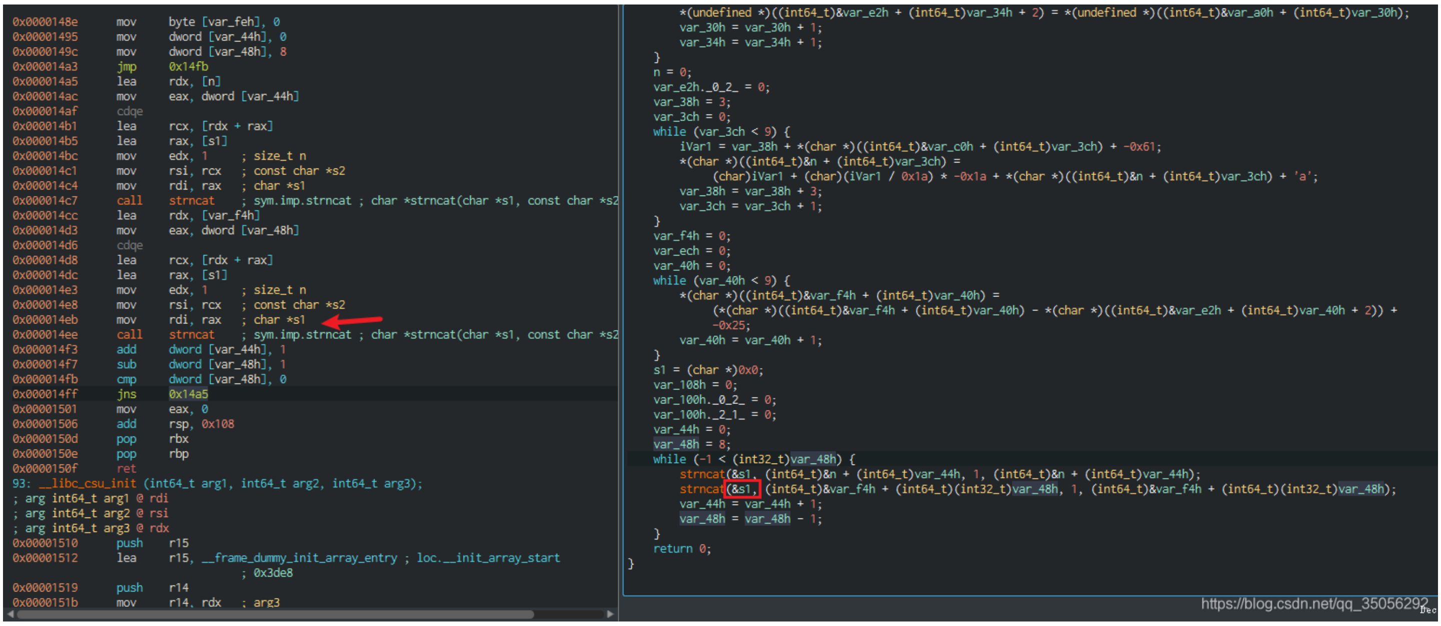
扔进IDA里，`Use this as ur input: %s` 在main函数中被使用。并且该提示语指定了我们的输入为：`!8u)05/>!#,.W/%H-G`

在main函数中，以`ZUB*aVrOUsCPS;$R=Q`和`!8u)05/>!#,.W/%H-G`这两个字符串为基础进行了提取，异或等运算。反正最后赋值给`dest`变量，这个变量应该就是我们需要查看结果的变量了。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  size_t v3; // rbx
  size_t v4; // rbx
  char dest[8]; // [rsp+0h] [rbp-110h]
  __int64 v7; // [rsp+8h] [rbp-108h]
  __int16 v8; // [rsp+10h] [rbp-100h]
  char v9; // [rsp+12h] [rbp-FEh]
  __int64 v10; // [rsp+1Ch] [rbp-F4h]
  __int16 v11; // [rsp+24h] [rbp-ECh]
  __int64 v12; // [rsp+26h] [rbp-EAh]
  __int16 v13; // [rsp+2Eh] [rbp-E2h]
  char v14[32]; // [rsp+30h] [rbp-E0h]
  char v15[32]; // [rsp+50h] [rbp-C0h]
  char v16[8]; // [rsp+70h] [rbp-A0h]
  __int64 v17; // [rsp+78h] [rbp-98h]
  __int16 v18; // [rsp+80h] [rbp-90h]
  char v19; // [rsp+82h] [rbp-8Eh]
  char v20[32]; // [rsp+90h] [rbp-80h]
  int v21; // [rsp+B0h] [rbp-60h]
  int v22; // [rsp+B4h] [rbp-5Ch]
const char *v23; // [rsp+B8h] [rbp-58h]
char *s; // [rsp+C0h] [rbp-50h]
int n; // [rsp+C8h] [rbp-48h]
int v26; // [rsp+CCh] [rbp-44h]
int m; // [rsp+D0h] [rbp-40h]
int l; // [rsp+D4h] [rbp-3Ch]
int v29; // [rsp+D8h] [rbp-38h]
int v30; // [rsp+DCh] [rbp-34h]
int v31; // [rsp+E0h] [rbp-30h]
int k; // [rsp+E4h] [rbp-2Ch]
int j; // [rsp+E8h] [rbp-28h]
int v34; // [rsp+ECh] [rbp-24h]
int i; // [rsp+F0h] [rbp-20h]
int v36; // [rsp+F4h] [rbp-1Ch]
int v37; // [rsp+F8h] [rbp-18h]
int v38; // [rsp+FCh] [rbp-14h]

setbuf(stdout, 0LL);
setbuf(stdin, 0LL);
setbuf(stderr, 0LL);
v38 = 0;
v37 = 1;
s = "ZUB*aVrOUsCPS;$R=Q";
v23 = "!8u)05/>!#,.W/%H-G";
printf("Use this as ur input:  %s\n", "!8u)05/>!#,.W/%H-G");
__isoc99_scanf("%[^\n]%*c", v20);
*(_QWORD *)v16 = 0LL;
v17 = 0LL;
v18 = 0;
v19 = 0;
v36 = 0;
for ( i = 1; ; i += 2 )
{
  v3 = i;
  if ( v3 >= strlen(s) )
    break;
  v16[v38] = s[i] ^ v20[v36];
  v38 += 2;
  v36 += 2;
}
v34 = 1;
for ( j = 0; ; j += 2 )
{
  v4 = j;
  if ( v4 >= strlen(s) )
    break;
  v16[v37] = s[j] ^ v20[v34];
  v37 += 2;
  v34 += 2;
}
v22 = strlen(v16);
v21 = v22 / 2;
for ( k = 0; k < v21; ++k )
  v15[k] = v16[k];
v15[k] = 0;
v31 = v21;
v30 = 0;
while ( v31 <= v22 )
  v14[v30++] = v16[v31++];
  v12 = 0LL;
  v13 = 0;
  v29 = 3;
  for ( l = 0; l <= 8; ++l )
  {
    *((_BYTE *)&v12 + l) += (v15[l] + v29 - 97) % 26 + 97;
    v29 += 3;
  }
  v10 = 0LL;
  v11 = 0;
  for ( m = 0; m <= 8; ++m )
    *((_BYTE *)&v10 + m) = *((_BYTE *)&v10 + m) - v14[m] - 37;
  *(_QWORD *)dest = 0LL;
  v7 = 0LL;
  v8 = 0;
  v9 = 0;
  v26 = 0;
  for ( n = 8; n >= 0; --n )
  {
    strncat(dest, (const char *)&v12 + v26, 1uLL);
    strncat(dest, (const char *)&v10 + n, 1uLL);
    ++v26;
  }
  return 0;
}
```

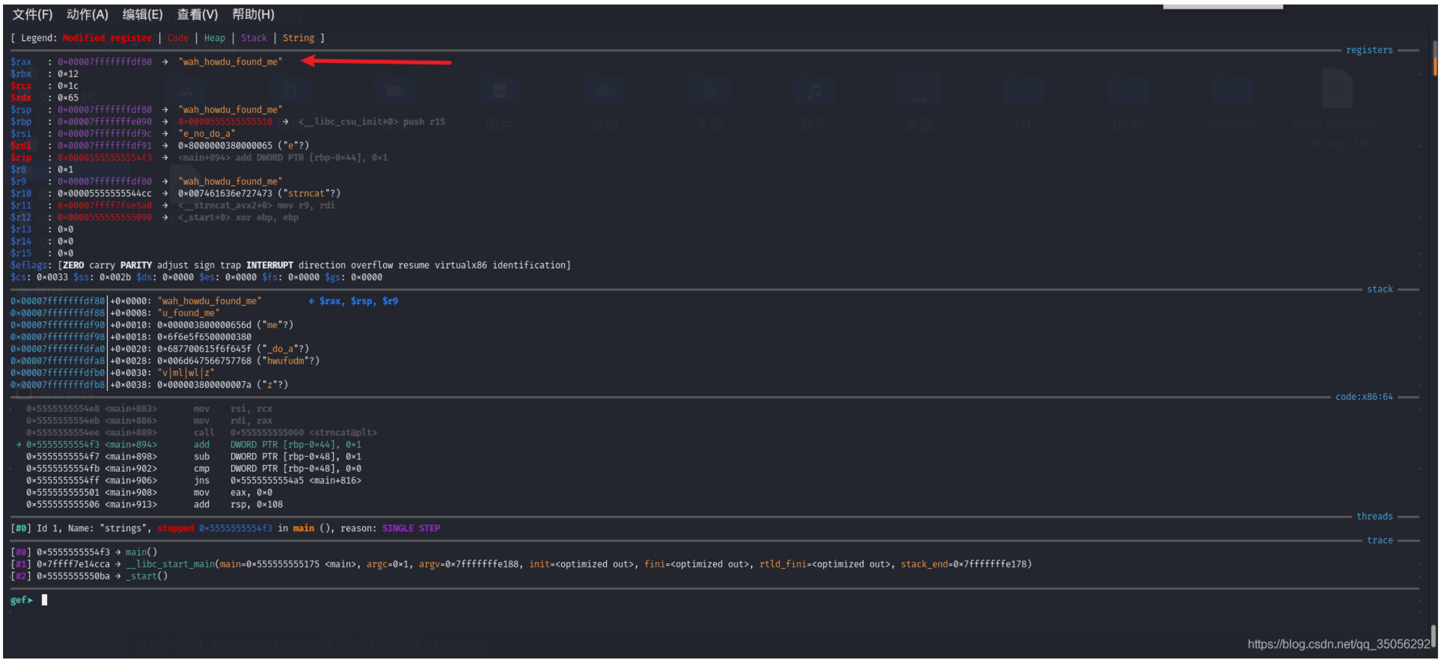
输入已经被指定了，也没有什么输出，那么直接动态调试看`dest`最后的结果即可。



```
0x0000148e      mov     byte [var_feh], 0
0x00001495      mov     dword [var_44h], 0
0x0000149c      mov     dword [var_48h], 8
0x000014a3      jmp     0x14fb
0x000014a5      lea     rdx, [n]
0x000014ac      mov     eax, dword [var_44h]
0x000014af      cdqe
0x000014b1      lea     rcx, [rdx + rax]
0x000014b5      lea     rax, [s1]
0x000014bc      mov     edx, 1      ; size_t n
0x000014c1      mov     rsi, rcx    ; const char *s2
0x000014c4      mov     rdi, rax    ; char *s1
0x000014c7      call    strncat     ; sym.imp.strncat ; char *strncat(char *s1, const char *s2
0x000014cc      lea     rdx, [var_f4h]
0x000014d3      mov     eax, dword [var_48h]
0x000014d6      cdqe
0x000014d8      lea     rcx, [rdx + rax]
0x000014dc      lea     rax, [s1]
0x000014e3      mov     edx, 1      ; size_t n
0x000014e8      mov     rsi, rcx    ; const char *s2
0x000014eb      mov     rdi, rax    ; char *s1
0x000014ee      call    strncat     ; sym.imp.strncat ; char *strncat(char *s1, const char *s2
0x000014f3      add     dword [var_44h], 1
0x000014f7      sub     dword [var_48h], 1
0x000014fb      cmp     dword [var_48h], 0
0x000014ff      jns     0x14a5
0x00001501      mov     eax, 0
0x00001506      add     rsp, 0x108
0x0000150d      pop     rbx
0x0000150e      pop     rbp
0x0000150f      ret
93: __libc_csu_init (int64_t arg1, int64_t arg2, int64_t arg3);
; arg int64_t arg1 @ rdi
; arg int64_t arg2 @ rsi
; arg int64_t arg3 @ rdx
0x00001510      push    r15
0x00001512      lea     r15, __frame_dummy_init_array_entry ; loc.__init_array_start
                            ; 0x3de8
0x00001519      push    r14
0x0000151b      mov     r14, rdx    ; arg3
```

```
        *(undefined *)((int64_t)&var_e2h + (int64_t)var_34h + 2) = *(undefined *)((int64_t)&var_a0h + (int64_t)var_30h);
        var_30h = var_30h + 1;
        var_34h = var_34h + 1;
    }
    n = 0;
    var_e2h._0_2_ = 0;
    var_38h = 3;
    var_3ch = 0;
    while (var_3ch < 9) {
        iVar1 = var_38h + *(char *)((int64_t)&var_c0h + (int64_t)var_3ch) + -0x61;
        *(char *)((int64_t)&n + (int64_t)var_3ch) =
             (char)iVar1 + (char)(iVar1 / 0x1a) * -0x1a + *(char *)((int64_t)&n + (int64_t)var_3ch) + 'a';
        var_38h = var_38h + 3;
        var_3ch = var_3ch + 1;
    }
    var_f4h = 0;
    var_ech = 0;
    var_40h = 0;
    while (var_40h < 9) {
        *(char *)((int64_t)&var_f4h + (int64_t)var_40h) =
             (*(char *)((int64_t)&var_f4h + (int64_t)var_40h) - *(char *)((int64_t)&var_e2h + (int64_t)var_40h + 2)) +
             -0x25;
        var_40h = var_40h + 1;
    }
    s1 = (char *)0x0;
    var_108h = 0;
    var_100h._0_2_ = 0;
    var_100h._2_1_ = 0;
    var_44h = 0;
    var_48h = 8;
    while (-1 < (int32_t)var_48h) {
        strncat(&s1, (int64_t)&n + (int64_t)var_44h, 1, (int64_t)&n + (int64_t)var_44h);
        strncat(&s1, (int64_t)&var_f4h + (int64_t)(int32_t)var_48h, 1, (int64_t)&var_f4h + (int64_t)(int32_t)var_48h);
        var_44h = var_44h + 1;
        var_48h = var_48h - 1;
    }
    return 0;
}
```

经过动态调试，最终得到flag为：`darkCTF{wah_howdu_found_me}`


文件(F) 动作(A) 编辑(E) 查看(V) 帮助(H)
https://blog.csdn.net/qq_35056292

```
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────── registers ────
$rax   : 0×00007fffffffdf80  →  "wah_howdu_found_me"
$rbx   : 0×12
$rcx   : 0×1c
$rdx   : 0×65
$rsp   : 0×00007fffffffdf80  →  "wah_howdu_found_me"
$rbp   : 0×00007fffffffe090  →  0×0000555555555510  →  <__libc_csu_init+0> push r15
$rsi   : 0×00007fffffffdf9c  →  "e_no_do_a"
$rdi   : 0×00007fffffffdf91  →  0×8000000380000065 ("e"?)
$rip   : 0×00005555555554f3  →  <main+894> add DWORD PTR [rbp-0×44], 0×1
$r8    : 0×1
$r9    : 0×00007fffffffdf80  →  "wah_howdu_found_me"
$r10   : 0×00005555555544cc  →  0×007461636e727473 ("strncat"?)
$r11   : 0×00007ffff7f4e5a0  →  <__strncat_avx2+0> mov r9, rdi
$r12   : 0×0000555555555090  →  <_start+0> xor ebp, ebp
$r13   : 0×0
$r14   : 0×0
$r15   : 0×0
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0×0033 $ss: 0×002b $ds: 0×0000 $es: 0×0000 $fs: 0×0000 $gs: 0×0000
──────────────────────────────────────────────────────────────────── stack ────
0×00007fffffffdf80│+0×0000: "wah_howdu_found_me"        ← $rax, $rsp, $r9
0×00007fffffffdf88│+0×0008: "u_found_me"
0×00007fffffffdf90│+0×0010: 0×000003800000656d ("me"?)
0×00007fffffffdf98│+0×0018: 0×6f6e5f6500000380
0×00007fffffffdfa0│+0×0020: 0×687700615f6f645f ("_do_a"?)
0×00007fffffffdfa8│+0×0028: 0×006d647566757768 ("hwufudm"?)
0×00007fffffffdfb0│+0×0030: "v|ml|wl|z"
0×00007fffffffdfb8│+0×0038: 0×000003800000007a ("z"?)
────────────────────────────────────────────────────────────── code:x86:64 ────
   0×5555555554e8 <main+883>       mov    rsi, rcx
   0×5555555554eb <main+886>       mov    rdi, rax
   0×5555555554ee <main+889>       call   0×555555555060 <strncat@plt>
 → 0×5555555554f3 <main+894>       add    DWORD PTR [rbp-0×44], 0×1
   0×5555555554f7 <main+898>       sub    DWORD PTR [rbp-0×48], 0×1
   0×5555555554fb <main+902>       cmp    DWORD PTR [rbp-0×48], 0×0
   0×5555555554ff <main+906>       jns    0×5555555554a5 <main+816>
   0×555555555501 <main+908>       mov    eax, 0×0
   0×555555555506 <main+913>       add    rsp, 0×108
────────────────────────────────────────────────────────────────── threads ────
[#0] Id 1, Name: "strings", stopped 0×5555555554f3 in main (), reason: SINGLE STEP
──────────────────────────────────────────────────────────────────── trace ────
[#0] 0×5555555554f3 → main()
[#1] 0×7ffff7e14cca → __libc_start_main(main=0×555555555175 <main>, argc=0×1, argv=0×7fffffffe188, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0×7fffffffe178)
[#2] 0×5555555550ba → _start()
───────────────────────────────────────────────────────────────────────────────
gef➤ █
```

## 3. 总结

1. 对于一些让用户输入的字符串与程序中经过变换以后的字符串相比较的题目、或是用户输入进行变换的题，可以直接上动态调试，运行到检查的那一个地方，看一下要比较的字符串长啥样即可。
2. 根据gdb调试中main函数的地址和cutter中main函数的地址，算一个差值diff
3. 然后根据cutter中Decompiler窗口的内容，定位到 Dissassembly窗口中伪代码对应的地址。
4. 最后，只需要调用下面的函数，即可根据kali中gdb调试时汇编指令的地址计算出cutter中的地址，从而判断当前是运行到程序中的哪个地方了。

```python
def get_gdb(cut, diff):
 return hex(cut + diff)

def get_cut(gdb, diff):
 return hex(gdb - diff)
```

5. 群里问了下各位师傅，其中一位师傅提到，IDA中可以设置基地址：https://blog.csdn.net/hgy413/article/details/5856827