




Dark CTF 2020-Rev/HelloWorld-writeup

原创

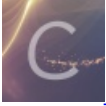
y4ung  于 2020-10-07 14:20:34 发布  569  收藏 2

分类专栏: [ctf](#) 文章标签: [ctf](#)

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

本文链接: https://blog.csdn.net/qq_35056292/article/details/108950370

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的第二题: [HelloWorld](#), 网址: <https://ctf.darkarmy.xyz/challs>

题目描述: taking small Bites of Bytes

2. 分析

```
$ file hw
hw: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=36a6e5cf577867ece4808b6c013e80e6ce8a470a, for GNU/Linux 3.2.0, not stripped
$ chmod +x ./hw
$ # 云服务器运行
$ ./hw
./hw: /lib/x86_64-linux-gnu/libm.so.6: version `GLIBC_2.29' not found (required by ./hw)
$ # 本地kali运行
$ ./hw 123 456
invalid argument: 123
```

2.1 静态分析

因为把文件放进IDA里分析的时候出现了和 [Rev/so_much](#) 这题的文件一样的警告。因此, 这里我仍然选用的ghidra进行逆向分析。

main函数中, 需要用户在命令行执行时输入两个参数: argv[1]和argv[2]。然后通过check函数分别对两个参数进行检查, 只有当check函数都返回0时, 才能通过。

```

undefined8 main(uint32_t argc, char **argv)
{
    int32_t iVar1;
    char **var_10h;
    uint32_t var_4h;

    if (argc == 3) {
        iVar1 = check(argv[1], 1);
        if (iVar1 == 0) {
            iVar1 = check(argv[2], 2);
            if (iVar1 == 0) {
                print();
            } else {
                printf("invalid argument: %s\n", argv[2]);
            }
        } else {
            printf("invalid argument: %s\n", argv[1]);
        }
    } else {
        puts("Enter the right arguments");
    }
    return 0;
}
https://blog.csdn.net/qq_35056292

```

跟进check函数看一下。参数arg2表明要检查的arg1是argv[1]还是argv[2]。

- 如果arg2为1，则检查argv[1];
- 如果arg2为2，则检查argv[2]

然后在while循环中对s1进行赋值，最后将s1与arg1进行比较。

对于arg2==1的情况：

var_48h 作为索引，初始值为0，范围是[0, 4]。在while循环中：

```

dVar1 = pow(&var_40h + var_48h * 4), *(undefined8 *)0x2040);
sprintf(s1 + var_48h, 0x2038, (int32_t)dVar1, s1 + var_48h);

```

sprintf函数主要功能是把格式化的数据写入某个字符串中：`int sprintf(char *string, char *format [,argument,...]);`

s1 + var_48h 就是对s1字符串的每个位置进行赋值，那么0x2038应该是 %c，赋的值就是dVar1。

ok，那dVar1怎么得到的？根据ghidra的分析，是调用了pow函数。第一个参数是 `var_40h + var_48h * 4`，第二个参数是 `*(undefined8 *)0x2040`。但是第二个参数我没找到具体的内容是啥。感觉还是得上动态调试看一下。

```

> x/16xb 0x00002040
- offset -  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0x00002040  00 00 00 00 00 00 e0 3f 01 1b 03 3b 54 00 00 00  .....?...;T...

```

```

undefined4 var_30h;
undefined4 var_2ch;
undefined4 var_28h;
undefined4 var_24h;
undefined4 var_20h;
undefined4 var_1ch;
undefined s1 [6];
int64_t canary;

canary = *(int64_t*)(in_FS_OFFSET + 0x28);
var_40h._0_4_ = 0x1440;
var_40h._4_4_ = 0xa29;
var_38h = 0x2d90;
var_34h = 0x2d90;
var_30h = 0x900;
var_2ch = 0x1d91;
var_28h = 0x900;
var_24h = 0x32c4;
var_20h = 0x2d90;
var_1ch = 10000;
if (arg2 == 1) {
    var_48h = 0;
    while (var_48h < 5) {
        dVar1 = (double)pow((double)*(int32_t*)((int64_t)&var_40h + (int64_t)var_48h * 4), *(undefined8 *)0x2040);
        sprintf(s1 + var_48h, 0x2038, (int32_t)dVar1, s1 + var_48h);
        var_48h = var_48h + 1;
    }
    strcmp(s1, arg1, arg1);
} else {
    if (arg2 == 2) {
        var_44h = 0;
        while (var_44h < 5) {
            dVar1 = (double)pow((double)*(int32_t*)((int64_t)&var_40h + (int64_t)(var_44h + 5) * 4),
                *(undefined8 *)0x2040);
            sprintf(s1 + var_44h, 0x2038, (int32_t)dVar1, s1 + var_44h);
            var_44h = var_44h + 1;
        }
        strcmp(s1, arg1, arg1);
    }
}
if (canary != *(int64_t*)(in_FS_OFFSET + 0x28)) {
    __stack_chk_fail();
}
return;
}

```

https://blog.csdn.net/qq_35056292

2.2 动态分析

两种动态调试的思路：

1. 从main函数来看，其实，输入的参数只是为了保证能通过check函数，进入到print()函数，这个print()函数应该是打印出flag的。因此，用户输入是啥其实没必要去分析，只需要用动态调试，修改strcmp的返回值，然后直接跳到 print()函数运行即可。
2. 既然最后是在check函数里比较生成的字符串与用户输入是否相等，那只要动态调试，运行到strcmp传参的时候，就能知道用户的正确输入应该是啥了。我这里采用的是这种方法。

```

gef> set args 123 456
gef> b check
Breakpoint 1 at 0x14c5
gef> r

```

```

0x555555554e1 <check+28>    mov     QWORD PTR [rbp-0x8], rax
0x555555554e5 <check+32>    xor     eax, eax
0x555555554e7 <check+34>    mov     DWORD PTR [rbp-0x40], 0x1440
→ 0x555555554ee <check+41>    mov     DWORD PTR [rbp-0x3c], 0xa29
0x555555554f5 <check+48>    mov     DWORD PTR [rbp-0x38], 0x2d90
0x555555554fc <check+55>    mov     DWORD PTR [rbp-0x34], 0x2d90
0x55555555503 <check+62>    mov     DWORD PTR [rbp-0x30], 0x900
0x5555555550a <check+69>    mov     DWORD PTR [rbp-0x2c], 0x1d91
0x55555555511 <check+76>    mov     DWORD PTR [rbp-0x28], 0x900

[#0] Id 1, Name: "hw", stopped 0x555555554ee in check (), reason: SINGLE STEP

[#0] 0x555555554ee → check()
[#1] 0x55555555248 → main()
[#2] 0x7ffff7cd0cca → __libc_start_main(main=0x55555555209 <main>, argc=0x3, argv=0
[#3] 0x5555555514e → _start()

gef> p 0x7fffffe060
$1 = 0x7fffffe060
gef> x/16xb 0x7fffffe060
0x7fffffe060: 0x40 0x14 0x00 0x00 0x00 0x00 0x00 0x00
0x7fffffe068: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
gef>
https://blog.csdn.net/qq_35056292

```

check函数里，最终debug到strcmp语句时，s1的值为 **H3110**，因此用户输入的第一个参数为 **H3110**

```

$rbp : 0x00007fffffe0a0 → 0x00007fffffe0c0 → 0x000055555555630 → <__libc_csu_init+0> endbr64
$rsi : 0x00007fffffe4cf → 0x0036353400333231 ("123"? )
$rdi : 0x00007fffffe092 → 0xff0000306c6c3348 ("H3110"? )
$rip : 0x000055555555595 → <check+208> call 0x55555555100 <strcmp@plt>

```

继续调试第二个参数：

```

gef> set args H3110 456
gef> r

```

可以得到第二个参数应该为：**W0rld**

```

$rsp : 0x00007fffffe030 → 0x0000000200000000
$rbp : 0x00007fffffe090 → 0x00007fffffe0b0 → 0x000055555555630 →
$rsi : 0x00007fffffe4d3 → 0x4c45485300363534 ("456"? )
$rdi : 0x00007fffffe082 → 0x9f0000646c723057 ("W0rld"? )
$rip : 0x000055555555607 → <check+322> call 0x55555555100 <strcmp@plt>

```

运行程序，得到flag为：**darkCTF{4rgum3nts_are_v3ry_1mp0rt4nt!!!}**

```

$ ./hw H3110 W0rld
darkCTF{4rgum3nts_are_v3ry_1mp0rt4nt!!!}

```

3. 补充

3.1 Glibc 2.29安装

1. glibc2.29 安装：<http://www.linuxfromscratch.org/lfs/view/8.4/chapter05/glibc.html>
2. bison安装：<https://stackoverflow.com/questions/53735137/glibc-configure-error-yacc-bison-missing>
3. M4安装：<https://cloud.tencent.com/developer/article/1550499>

安装的时候可能会出现bison未安装啊，M4未安装等报错。如果报错了，就分别安装2和3中的命令进行安装即可。

需要注意的是，如果安装完后回过头去安装glibc时提示bison版本太低等问题，就根据INSTALL文件中提示的版本安装：
<https://ftp.gnu.org/gnu/bison/bison-2.7.tar.gz>
 也可以直接sudo apt-get install bison。但是安装完以后还是报错

不过我在我的kali 2020.2上是可以运行的。。所以就直接在本地的kali上调试了。

3.2 gdb peda, gef安装

1. peda:

```
git clone https://github.com/longld/peda.git ~/peda
echo "source ~/peda/peda.py" >> ~/.gdbinit
```

2. gdf

```
wget -q -O ~/.gdbinit-gef.py https://github.com/hugsy/gef/raw/master/gef.py
echo source ~/.gdbinit-gef.py >> ~/.gdbinit
```

3.3 手动修复plt表, 解决IDA的报错

这个比赛的文件每次用IDA打开都会报错, 网上说是需要修复plt表。先开个坑, 啥时候有时间研究一下