

CTFtime平台上发现的一场比赛，记录一下其中的2道逆向题

Feed_me

题目打开，发现一个scanf("%s",s)，明显有栈溢出倾向，而IDA将变量识别如下

```
char s[10]; // [rsp+32h] [rbp-6Eh]
char nptr; // [rsp+3Ch] [rbp-64h]
char v15; // [rsp+46h] [rbp-5Ah]
char v16; // [rsp+50h] [rbp-50h]
unsigned __int64 v17; // [rsp+88h] [rbp-18h]

v17 = __readfsqword(0x28u);
v3 = time(0LL);
srand(v3);
v6 = -2 * (rand() % 10000);
v7 = -2 * (rand() % 10000);
v8 = -2 * (rand() % 10000);
puts("Can you cook my favourite food using these ingredients :)");
printf("%d ; %d ; %d ;\n", v6, v7, v8);
__isoc99_scanf("%s", s);
for ( i = 0; i < strlen(s); ++i )
{
    if ( (s[i] <= '/' || s[i] > '9') && s[i] != 45 )
    {
        puts("Invalid input :( ");
        return 0;
    }
}
v9 = atoi(s);
v10 = atoi(&nptr);
v11 = atoi(&v15);
if ( v6 == v9 + v10 )
{
    if ( v7 == v10 + v11 )
    {
        if ( v8 == v11 + v9 )
```

注意后面的三个atoi，后两个的参数比较迷，相对s的偏移分别为10和20，这里我把s的类型重新定义为char s[30]

因为这题保护全开，不是考察pwn，应该是考察栈上变量偏移的识别

那么我们输入的字符串每10个字符被解析成3个int型数据

主要的判断可以看做三元一次方程

```
input1 + input2 = num1
input2 + input3 = num2
input3 + input1 = num3
```



```
=>
input1 = num1 + num3 - num2 / 2
input2 = num1 + num2 - num3 / 2
input3 = num2 + num3 - num1 / 2
```

多说一句，解方程时把三个式子加起来除以二后，分别减去每一个式子，即可得到方程的解

根据srand(time(0))和rand()的特性：随机种子确定，生成的序列也确定，写出python脚本扔程序跑就可以了

但是我不太熟悉python对rand的处理，直接C程序跑出来，手动喂给程序了

补充

解题过程中有一些想法

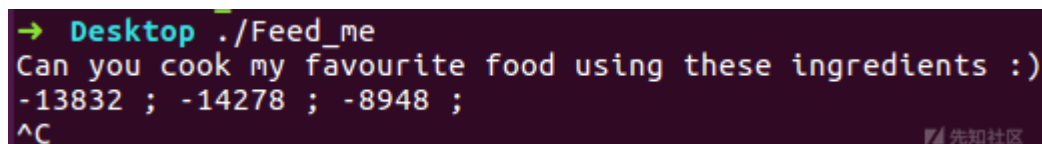
```
unsigned int num1; // [rsp+10h] [rbp-90h]
unsigned int num2; // [rsp+14h] [rbp-8Ch]
unsigned int num3; // [rsp+18h] [rbp-88h]
int input1; // [rsp+1Ch] [rbp-84h]
int input2; // [rsp+20h] [rbp-80h]
int input3; // [rsp+24h] [rbp-7Ch]
FILE *stream; // [rsp+28h] [rbp-78h]
char s[30]; // [rsp+32h] [rbp-6Eh]
char v14; // [rsp+50h] [rbp-50h]
unsigned __int64 v15; // [rsp+88h] [rbp-18h]
```

```
v15 = __readfsqword(0x28u);
v3 = time(0LL);
srand(v3);
num1 = -2 * (rand() % 10000);
num2 = -2 * (rand() % 10000);
num3 = -2 * (rand() % 10000);
puts("Can you cook my favourite food using these ingredients :)");
printf("%d ; %d ; %d ;\n", num1, num2, num3);
```

这里num1, num2, num3都是unsigned int，而它们都被以%d的形式输出

因为rand()返回值为int型，但是必定返回一个正数，模10000后再乘以-2，相当于把一个绝对值很小的负数赋值给了unsigned int

以%d有符号数输出结果，是三个负数，是应该的



```
→ Desktop ./Feed_me
Can you cook my favourite food using these ingredients :)
-13832 ; -14278 ; -8948 ;
^C
```

可是num系列数据的实际类型是无符号的，是一个很大的正数

而我们输入的字符串被解析成了3个int数据，而且都是负数，在判断时，如何比较一个■unsigned int和■int？

看一下汇编代码

```
.text:00000000000000BF1          mov     edx, [rbp+input1]
.text:00000000000000BF7          mov     eax, [rbp+input2]
.text:00000000000000BFA          add     eax, edx
.text:00000000000000BFC          cmp     [rbp+num1], eax
```

2个int数据add后与unsigned int比较，实际上应该是比较的二进制数据

只要二进制的32个bit相等，那么就判断相等

做一个小实验

```
#include<stdio.h>
#include<stdlib.h>

int main(int argc,char**argv){
    int num1 = -1;
    unsigned int num2 = -1;
    if(num1==num2){
        printf("num1==num2");
    }
    return 0;
}
```

num1就是单纯的-1，而num2会是0xffffffff，是一个大正数

实际上if判断为真，会输出num1==num2

发现这个问题的原因主要是，我本地写C代码测试时，发现以%d输出预期的值input1、input2、input3时，都是长度为10的int数据，而连起来就是30长度的纯数字字符串

Super Secure Vault

IDA打开，主逻辑如下

```

v15 = 5;
v16 = 25;
v17 = 4;
v18 = 83;
v19 = 7;
v20 = 135;
v21 = 5;
printf("Enter the key: ", argv, envp);
__isoc99_scanf("%s", input);
if ( strlen(input) > 30 )
    fail(0LL);
v3 = getNum((__int64 *)"27644437104591489104652716127", 0, a3);
if ( (unsigned int)mod(input, v3) != v12 )
    fail(0LL);
a2 = a3;
v4 = getNum((__int64 *)"27644437104591489104652716127", a3, v15);
if ( (unsigned int)mod(input, v4) != v14 )
    fail(0LL);
a2a = v15 + a2;
v5 = getNum((__int64 *)"27644437104591489104652716127", a2a, v17);
if ( (unsigned int)mod(input, v5) != v16 )
    fail(0LL);
a2b = v17 + a2a;
v6 = getNum((__int64 *)"27644437104591489104652716127", a2b, v19);
if ( (unsigned int)mod(input, v6) != v18 )
    fail(0LL);
v8 = getNum((__int64 *)"27644437104591489104652716127", v19 + a2b, v21);
if ( (unsigned int)mod(input, v8) != v20 )
    fail(0LL);
printf("Enter password: ", v8);
__isoc99_scanf("%s", &password);
func2((__int64)&password, input, "27644437104591489104652716127");
return 0;
}

```

发现函数名都在，而且getNum和mod函数都接收了一个字符串作为参数，实际上是指向字符串的指针

先看一下getNum函数

```

__int64 __fastcall getNum(__int64 *str, int a2, int a3)
{
    unsigned int v4; // [rsp+18h] [rbp-8h]
    int i; // [rsp+1Ch] [rbp-4h]

    v4 = 0;
    for ( i = a2; i < a2 + a3; ++i )
        v4 = 10 * v4 + *((char *)str + i) - 48;
    return v4;
}

```

它的作用实际上是从字符串中取出一段数据并返回int64型

mod函数的行为和名字一样

```
__int64 __fastcall mod(const char *input, int a2)
{
    unsigned int v3; // [rsp+18h] [rbp-18h]
    int i; // [rsp+1Ch] [rbp-14h]

    v3 = 0;
    for ( i = 0; i < strlen(input); ++i )
        v3 = (signed int)(10 * v3 + input[i] - 48) % a2;
    return v3;
}
```

先知社区

注意到程序虽然用了scanf，但是保护全开，也就没有REpwn这种要改数据的可能性了，输入的长度姑且认为是30

因为getNum返回的值不受我们输入字符串的影响，只要动态调出来就可以了

Code

```
0x555555554d5a <main+267>:  mov     esi,ecx
0x555555554d5c <main+269>:  mov     rdi,rax
0x555555554d5f <main+272>:  call    0x5555555548e1 <getNum>
=> 0x555555554d64 <main+277>:  mov     DWORD PTR [rbp-0xc4],eax
0x555555554d6a <main+283>:  mov     edx,DWORD PTR [rbp-0xc4]
0x555555554d70 <main+289>:  lea     rax,[rbp-0x80]
0x555555554d74 <main+293>:  mov     esi,edx
0x555555554d76 <main+295>:  mov     rdi,rax
[rbp-0xc4] : 0x7fffffffddcf --> 0x0
```

Stack

0000	0x7fffffffddcf0	-->	0x7ffff7ffa280	(add	BYTE PTR ss:[rax],al)
0008	0x7fffffffddcf8	-->	0xf7ffe700		
0016	0x7fffffffdd00	-->	0x0		
0024	0x7fffffffdd08	-->	0x555555550e9	("27644437104591489104652716127")	
0032	0x7fffffffdd10	-->	0x8000000d5		
0040	0x7fffffffdd18	-->	0x5000000e5		
0048	0x7fffffffdd20	-->	0x400000019		
0056	0x7fffffffdd28	-->	0x700000053		

Legend: code, data, rodata, heap, value

Breakpoint 1, 0x0000555555554d64 in main ()

先知社区

比如第一个值是27644437，要求input % 27644437 == 213

同样的，可以得出以下5个线性同余方程组

```
input % 27644437 == 213
input % 10459    == 229
input % 1489     == 25
input % 1046527  == 83
input % 16127    == 135
```

和中国剩余定理有关，网上找个脚本解一下

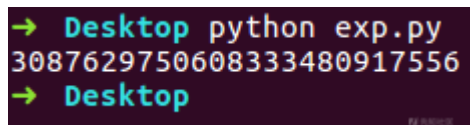
```
from functools import reduce
```

```
def egcd(a, b):
    if 0 == b:
        return 1, 0, a
    x, y, q = egcd(b, a % b)
    x, y = y, (x - a // b * y)
    return x, y, q
```

```
def Chinese_remainder(pairs):
    mod_list, remainder_list = [p[0] for p in pairs], [p[1] for p in pairs]
    mod_product = reduce(lambda x, y: x * y, mod_list)
    mi_list = [mod_product//x for x in mod_list]
    mi_inverse = [egcd(mi_list[i], mod_list[i])[0] for i in range(len(mi_list))]
    x = 0
    for i in range(len(remainder_list)):
        x += mi_list[i] * mi_inverse[i] * remainder_list[i]
        x %= mod_product
    return x

if __name__ == '__main__':
    print(Chinese_remainder([(27644437, 213), (10459, 229), (1489, 25), (1046527, 83), (16127, 135)]))
```

出结果：



```
→ Desktop python exp.py
3087629750608333480917556
→ Desktop
```

长度也符合要求，是一个符合条件的解

然后我们就被要求输入password

主要逻辑在func2中

```
// func2

v12 = strcat(input, a3);
v3 = (unsigned __int64)&v12[strlen(v12)];
*(_WORD *)v3 = 12344;
*(_BYTE *)(v3 + 2) = 0;
v7 = 0;
v8 = 0;
v10 = strlen(v12) >> 1;
while ( v8 < strlen(v12) >> 1 )
{
    if ( *((_BYTE *)password + v7) != matrix[100 * (10 * (v12[v8] - 48) + v12[v8 + 1] - 48)
        - 48
        + 10 * (v12[v10] - 48)
        + v12[v10 + 1]] )

        fail(1LL);
    ++v7;
    v8 += 2;
    v10 += 2;
}
v9 = 0;
v11 = strlen(v12) >> 1;
while ( v9 < strlen(v12) >> 1 )
{
    v4 = 10 * (v12[v9] - 48) + v12[v9 + 1] - 48;
    v5 = 10 * (v12[v11] - 48) + v12[v11 + 1] - 48;
    if ( *((_BYTE *)password + v7) != matrix[100 * (v4 * v4 % 97) + v5 * v5 % 97] )
        fail(1LL);
    ++v7;
    v9 += 2;
    v11 += 2;
}
puts("Your Skills are really great. Flag is:");
return printf("pctf{%s}\n", password);
}
```

我们第一轮输入的字符串会被追加"27644437104591489104652716127"

再被追加0x3038和\x00

```
str_appended = strcat(input, a3); // "27644437104591489104652716127"
v3 = (unsigned __int64)&str_appended[strlen(str_appended)];
*(_WORD *)v3 = 0x3038;
*(_BYTE *)v3 + 2 = 0;
```

先知社区

然后基本就是查表了，由于程序开了PIE，我不知道怎么用angr秒解它，于是只能patch程序，用gdb下断点看了

```
v10 = strlen(str_appended) >> 1;
while ( v8 < strlen(str_appended) >> 1 )
{
    if ( *((_BYTE *)password + v7) == matrix[100 * (10 * (str_appended[v8] - 48) + str_appended[v8 + 1] - 48)
        - 48
        + 10 * (str_appended[v10] - 48)
        + str_appended[v10 + 1]] )
    {
        fail(1LL);
        ++v7;
        v8 += 2;
        v10 += 2;
    }
    v9 = 0;
    v11 = strlen(str_appended) >> 1;
    while ( v9 < strlen(str_appended) >> 1 )
    {
        v4 = 10 * (str_appended[v9] - 48) + str_appended[v9 + 1] - 48;
        v5 = 10 * (str_appended[v11] - 48) + str_appended[v11 + 1] - 48;
        if ( *((_BYTE *)password + v7) == matrix[100 * (v4 * v4 % 97) + v5 * v5 % 97] )
        {
            fail(1LL);
            ++v7;
            v9 += 2;
            v11 += 2;
        }
    }
    puts("Your Skills are really great. Flag is:");
}
```

先知社区

这里把两行中的!=改成了==，并且下断点观察矩阵中给出的值

密码随便输入!!!!!!!!!!!!!!

然后RAX中的值，也就是cmp cl,al中会给出应该输入的字符

收集一下就是：pctf{R3v3rS1Ng_#s_h311_L0t_Of_Fun}

我猜测程序中可能是手动把符合同余方程组的解都算了一遍？然后把数据填入那个巨型矩阵？

最后其他位置乱放了一些字符？

总结一下

这两道题的基本是一个递进的关系，其中第二题的函数编写值得学习一下，C语言如何处理这种大数的mod，之前我没有仔细想过，这段代码实现也没有彻底的理解.....

Pragyan CTF 19 binary.rar (0.066 MB) [下载附件](#)

点击收藏 | 0 关注 | 1

[上一篇：一篇域攻击文章的复现](#) [下一篇：一篇域攻击文章的复现](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

[目录](#)

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)