HGAME Week4 WriteUp

RE

virtual_waifu

拿到题目 首先还是一样拖入IDA看一下

```
printf("Input your flag: \n");
inputString_1 = 0i64;
v168 = 0i64;
scanf_s("%s", &inputString_1, 32);
v169 = 0x3040500;
v170 = 0x50E050F;
v171 = 0x103020C;
v172 = 0x1050608;
v173 = 0xA050809;
v174 = 0x7080103;
v175 = 0x2080501;
v176 = 0xD0B0501;
inputString = &inputString_1;
v6 = 0;
v7 = 0x16;
v8 = 0x17;
v9 = 0xCC;
v10 = 1;
v11 = 0xFFFFFFE7;
v12 = 0x16;
v13 = 0x17;
v14 = 0xCC;
v15 = 1;
v16 = 0xFFFFFFE7;
v17 = 0x16;
v18 = 0x17;
v19 = 0xCC;
v20 = 1;
v21 = 0xFFFFFFE7;
v22 = 0x16;
v23 = 23;
v24 = 204;
v25 = 1;
v26 = -25;
v27 = 22;
v28 = 23;
```

```
  v143 = 23;
  v144 = 204;
  v145 = 1;
  v146 = -25;
  v147 = 22;
  v148 = 23;
  v149 = 204;
  v150 = 1;
  v151 = -25;
  v152 = 22;
  v153 = 23;
  v154 = 204;
  v155 = 1;
  v156 = -25;
  v157 = 22;
  v158 = 23;
  v159 = 204;
  v160 = 1;
  v161 = -25;
  v162 = 22;
  v163 = 23;
  v164 = 204;
  v165 = 1;
  v166 = -25;
  sub_4017B0(&v169, &inputString);
  v3 = 0;
  while ( *(&inputString_1 + v3) == byte_40318C[v3] )
  {
    if ( ++v3 >= 24 )
      goto LABEL_6;
  }
  printf("Never Give Up\n");
LABEL_6:
  system("pause");
  return 0;
}
```

00000B24 _main:360 (401724)

除去中间的重复数据 整个代码逻辑差不多就是这样
看起来也像一道经典的Crackme 最后只是一个比较
这里可以注意到传入(伪)加密函数的参数 &v169

```
  int v41; // [esp+20h] [ebp-8h]

  choice = goodString;
  _inputString = inputString;
  v3 = malloc(0xCu);
  if ( !v3 )
    sub_401000("Create Stack Malloc Fail CODE 1");
  v4 = malloc(0x100u);
  v3[2] = v4;
  if ( !v4 )
    sub_401000("Create Stack Malloc Fail CODE 2");
  v5 = 0;
  *v3 = 64;
  v6 = 0;
  v3[1] = 0;
  v41 = 0;
  v38 = 0;
  v40 = 0;
  while ( choice )
  {
    v7 = *(choice++ + 1);
    switch ( v7 )
    {
      case 1:
        *(v3[2] + 4 * ++v3[1]) = v5;            // A[++num] = v5
        break;
      case 2:                                   // v5 = v41 = A[num--]
        v8 = v3[1];
        v5 = *(v3[2] + 4 * v8);
        v41 = *(v3[2] + 4 * v8);
        v3[1] = v8 - 1;
        break;
      case 3:                                   // num++,A[num] = v38
        *(v3[2] + 4 * ++v3[1]) = v38;
        goto LABEL_22;
      case 4:                                   // v38 = A[num--]
```

00000BB0 sub_4017B0:55 (4017B0)

```
      break;
    case 5:                                 // 指向一段数据
      v10 = v3[2];
      v11 = *_inputString;
      ++v3[1];
      ++_inputString;
      *(v10 + 4 * v3[1]) = v11;
      goto LABEL_22;
    case 6:                                 // 取一位字符
      v29 = v3[1];
      v30 = v3[2];
      v31 = *(v30 + 4 * v29--);
      v3[1] = v29++;
      v32 = *v31;
      v3[1] = v29;
      *(v30 + 4 * v29) = v32;
      goto LABEL_22;
    case 7:                                 // 将改变后的值带回
      v33 = v3[1];
      v34 = v3[2];
      v35 = *(v34 + 4 * v33);
      v3[1] = v33 - 1;
      v36 = *(v34 + 4 * (v33 - 1));
      v3[1] = v33 - 2;
      *v35 = v36;
      goto LABEL_21;
    case 8:                                 // num--,A[num] += A[num + 1]
      v12 = v3[1];
      v13 = v3[2];
      v14 = *(v13 + 4 * v12--);
      v3[1] = v12;
      *(v13 + 4 * v12) += v14;
      goto LABEL_11;
    case 9:                                 // num--,A[num] -= A[num + 1]
      v15 = v3[1];
      v16 = v3[2];
```

```
    case 9:                                 // num--,A[num] -= A[num + 1]
      v15 = v3[1];
      v16 = v3[2];
      v17 = *(v16 + 4 * v15--);
      v3[1] = v15;
      *(v16 + 4 * v15) -= v17;
LABEL_11:
      v40 = *(v3[2] + 4 * v3[1]) == 0;      // v40 = 0/1
      goto LABEL_21;
    case 0xA:                               // num--,A[num] ^= A[num + 1]
      v18 = v3[1];
      v19 = v3[2];
      v20 = *(v19 + 4 * v18--);
      v3[1] = v18;
      *(v19 + 4 * v18) ^= v20;
      goto LABEL_22;
    case 0xB:                               // choice += A[num--]
      v21 = v3[1];
      v22 = *(v3[2] + 4 * v21);
      v3[1] = v21 - 1;
      choice += v22;
      goto LABEL_22;
    case 0xC:                               // v6 == 0?,choice += A[num]
      v23 = v3[1];
      v24 = *(v3[2] + 4 * v23);
      v3[1] = v23 - 1;
      if ( v6 )
        choice += v24;
      goto LABEL_22;
    case 0xD:                               // free
      free(v3[2]);
      free(v3);
      return 1;
    case 0xE:
      v28 = (v3[2] + 4 * v3[1]);
      v40 = *(v28 - 1) == *v28;             // v40 = 0/1
00000BB0 sub_4017B0:144 (4017B0)
```

```
        v3[1] = v21 - 1;
        choice += v22;
        goto LABEL_22;
      case 0xC:                                 // v6 == 0?,choice += A[num]
        v23 = v3[1];
        v24 = *(v3[2] + 4 * v23);
        v3[1] = v23 - 1;
        if ( v6 )
          choice += v24;
        goto LABEL_22;
      case 0xD:                                 // free
        free(v3[2]);
        free(v3);
        return 1;
      case 0xE:
        v28 = (v3[2] + 4 * v3[1]);
        v40 = *(v28 - 1) == *v28;               // v40 = 0/1
LABEL_21:
        v6 = v40;
        goto LABEL_22;
      case 0xF:                                 // A[num] = strlen(A + num)
        v25 = v3[1];
        v26 = v3[2];
        v27 = *(v26 + 4 * v25);
        v3[1] = v25 - 1;
        v3[1] = v25;
        *(v26 + 4 * v25) = strlen(v27);
LABEL_22:
        v5 = v41;
        break;
      default:
        break;
    }
  }
  return 0;
}
```

结合函数内容看 很容易可以得出 这个参数类似于指令 也就是说这是一道vm题

在我做这一题的时候 差不多只分析到这里 由于搞错了传入函数的第一个参数 误以为这里传入的是inputString 导致始终无法将逻辑整理清楚 后来利用动态调试分析出加密函数把这道题给偷鸡偷过了..

这就不上代码了..只是做了一个

((inputArray[i] + 0x17 - len(now) xor 0xcc)这样的一个操作 只要写其求反就行

得到flag: `3z_vm_u_cr4ck5d_g00d_J0b`

现在就尝试着执果推因了QAQ

既然是一道VM题目 那么我们可以根据IDA初步整理出来的逻辑 分析一下这里的指令

```
[0x5,0x4,0x3,0xf, #len
0x5,0xe, #判断len 是否为0x16 若是 则证明到达了字符串结尾 便将v40置1
0x5,0xc,0x2,0x3,0x1,0x8,0x6, #取出当前指针指向的字符 若到结尾 则直接跳转至0xd
0x5,0x1,0x9,0x8,#将目标数据 + 0x17 后 - len(now)
0x5,0xa,0x3,0x1,0x8,0x7,0x1,#将目标数据xor 0xcc 并将改变后的数据带回
0x5,0x8,0x2,0x1,#将当前数据的len -1 并将地址 + 1
0x5,0xb,#指令 -= 0x25(回退至0x5,0xe)
0xd]#结束
```

大体逻辑可以看成这样 这里我们可以注意到 每当那一串数据和v8…这些类似于寄存器的东西进行交互的时候 num指针会随之加减 这一点上看这很像是堆栈 后来v爷爷也证实了这一点(假装自己很接近答案了XD
最后说一下从这题学到的一些东西
首先 VM的题其实我蛮怕的~~其实怕的是bin~~..南邮三道题我只做出了第一道..做这种题的时候一脸懵逼但现在我认为可以分成这样的几个步骤:
1.分析各个指令背后的逻辑
2.判断出一个关键的指令
3.根据这个指令进行分段
4.分析出各个段落的逻辑

若是始终理不清逻辑 需要好好的看一下IDA!
可以通过动态调试观看堆栈和内存变化来更好的理解题目
更重要的 还要理解出这个vm的思想是什么(不偷鸡:(

写在最后 (~~因为只做出一(半)题来凑字数的~~)

转眼一个月的练习赛也过去了 虽然最后一周做的时间比较少(wp还是现在在动车上赶的)..做出的题目也是少得可怜(灯谜也算题目对吧！！！) (pwn很可惜..第二题就是出不来) 但是也能感受到自己的进步 这点是非常让我开心的 感谢各位学长出的题目!
这个月过后 自己的方向变得明确 对于二进制安全也有了自己的看法(虽然很稚嫩hh) 也发现了自己还是菜的可以QAQ 但是!至少可以很自豪的说 这个月过的不亏 是到目前过的最有意义的一个寒假hhh(手动比个heart)