

北邮网安杯线下决赛 Writeup

原创

疯疯芸 于 2019-07-08 13:20:03 发布 421 收藏 1

分类专栏: [CTF](#) 文章标签: [CTF Re](#)

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_45262739/article/details/95054976

版权



[CTF 专栏收录该内容](#)

5 篇文章 0 订阅

订阅专栏

北邮网安杯 Writeup

有幸参加了北京邮电大学的北邮网安杯线下决赛

虽然自己很菜, 得分很低, 但是这次比赛让我学到了很多, 也对网络空间安全和 CTF 有了更加深入的了解

最重要的是在这个过程中我认识了一些兴趣、爱好相同很有趣的人, 原来即使是在同龄人中, 还是有这么多的高手, 自己需要学习的东西还很多

因为知识浅薄, CTF 我只完成了最简单的一道, 下面是我对这道题不成熟的解法

如果有什么错误之处, 欢迎下方留言

Re1

下载之后发现不是 .exe 文件

用 Winhex 打开, 发现是 ELF 文件

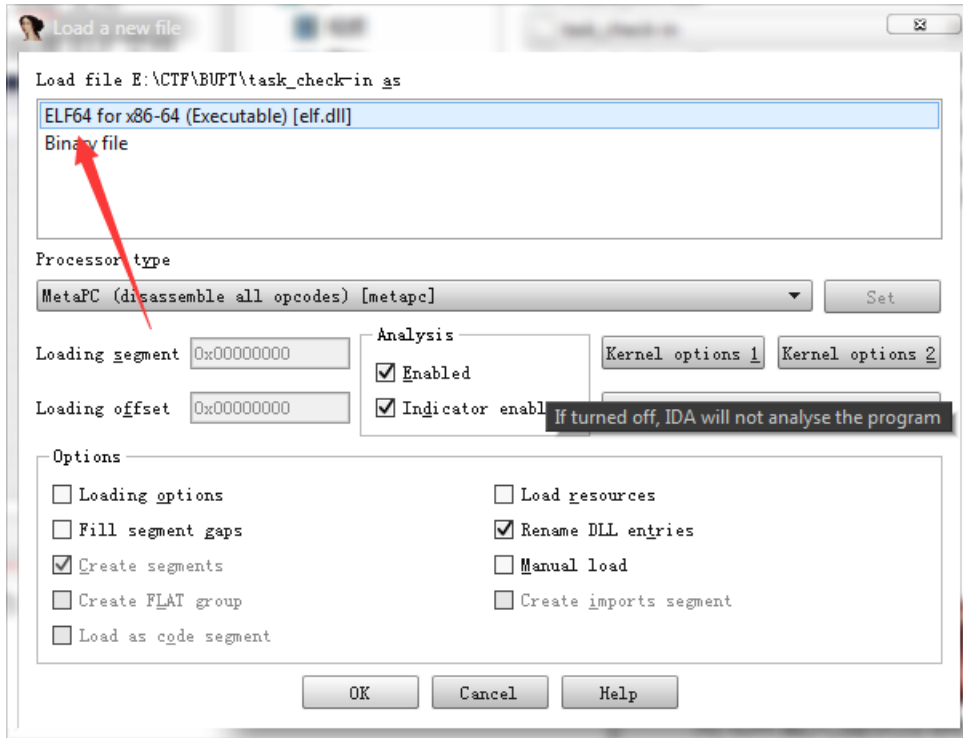
WinHex - [task_check-in]

task_check-in

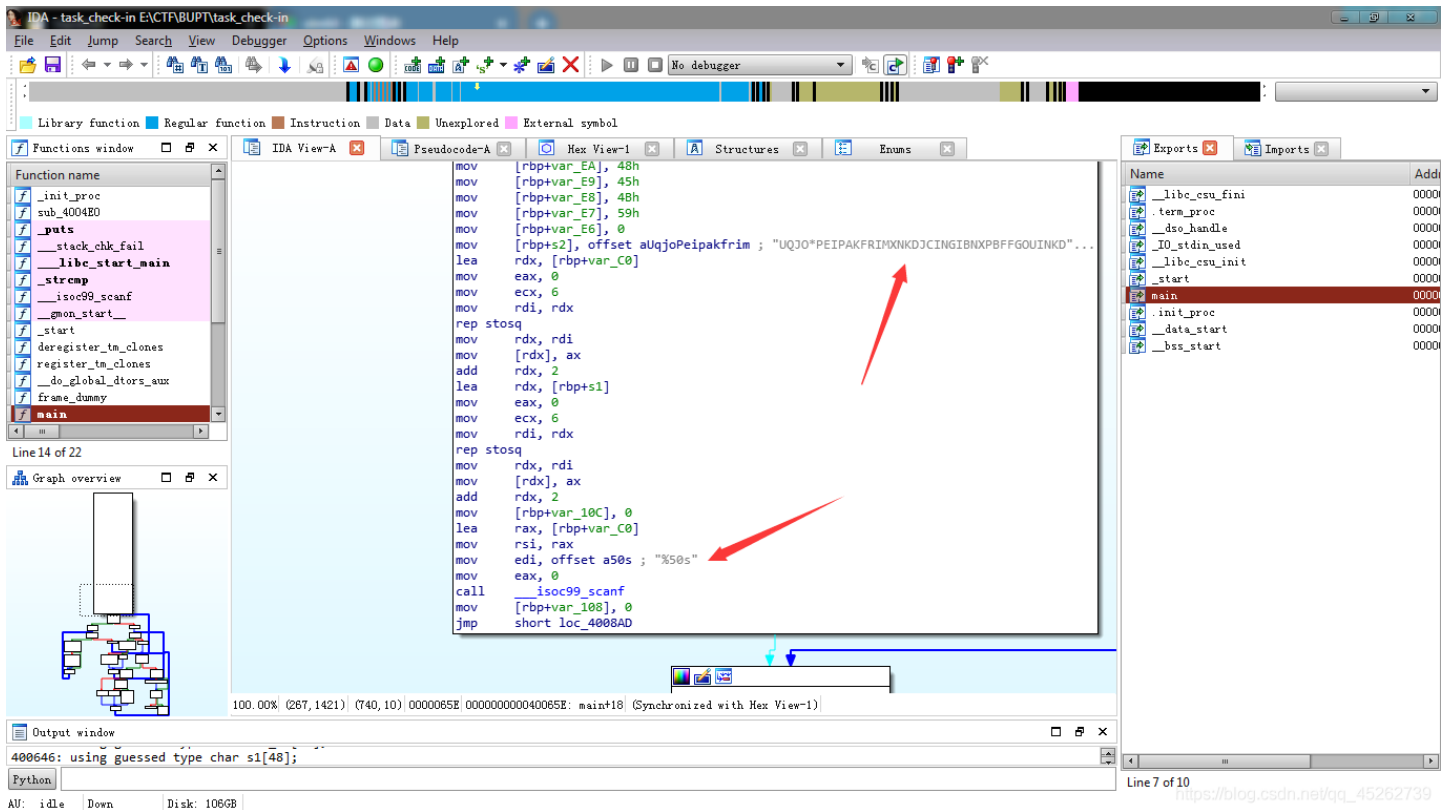
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI	ASCII
00000000	7F	45	4C	46	02	01	01	00	00	00	00	00	00	00	00	00	ELF	
00000010	02	00	3E	00	01	00	00	00	50	05	40	00	00	00	00	00		
00000020	40	00	00	00	00	00	00	00	88	1A	00	00	00	00	00	00		
00000030	00	00	00	00	40	00	38	00	09	00	40	00	1F	00	00	00		
00000040	06	00	00	00	05	00	00	00	40	00	00	00	00	00	00	00		
00000050	40	00	40	00	00	00	00	00	40	00	40	00	00	00	00	00		
00000060	F8	01	00	00	00	00	00	00	F8	01	00	00	00	00	00	00		
00000070	08	00	00	00	00	00	00	00	03	00	00	00	04	00	00	00		
00000080	38	02	00	00	00	00	00	00	38	02	40	00	00	00	00	00		
00000090	38	02	40	00	00	00	00	00	1C	00	00	00	00	00	00	00		
000000A0	1C	00	00	00	00	00	00	00	01	00	00	00	00	00	00	00		
000000B0	01	00	00	00	05	00	00	00	00	00	00	00	00	00	00	00		
000000C0	00	00	40	00	00	00	00	00	00	00	40	00	00	00	00	00		
000000D0	FC	0B	00	00	00	00	00	00	FC	0B	00	00	00	00	00	00		
000000E0	00	00	20	00	00	00	00	00	01	00	00	00	06	00	00	00		
000000F0	10	0E	00	00	00	00	00	00	10	0E	60	00	00	00	00	00		
00000100	10	0E	60	00	00	00	00	00	40	02	00	00	00	00	00	00		
00000110	48	02	00	00	00	00	00	00	00	00	20	00	00	00	00	00		
00000120	02	00	00	00	06	00	00	00	28	0E	00	00	00	00	00	00		
00000130	28	0E	60	00	00	00	00	00	28	0E	60	00	00	00	00	00		
00000140	D0	01	00	00	00	00	00	00	D0	01	00	00	00	00	00	00		
00000150	08	00	00	00	00	00	00	00	04	00	00	00	04	00	00	00		
00000160	54	02	00	00	00	00	00	00	54	02	40	00	00	00	00	00		
00000170	54	02	40	00	00	00	00	00	44	00	00	00	00	00	00	00		
00000180	44	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00		
00000190	50	E5	74	64	04	00	00	00	D4	0A	00	00	00	00	00	00		
000001A0	D4	0A	40	00	00	00	00	00	D4	0A	40	00	00	00	00	00		
000001B0	34	00	00	00	00	00	00	00	34	00	00	00	00	00	00	00		
000001C0	04	00	00	00	00	00	00	00	51	E5	74	64	06	00	00	00		
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00		
000001F0	00	00	00	00	00	00	00	00	10	00	00	00	00	00	00	00		
00000200	52	E5	74	64	04	00	00	00	10	0E	00	00	00	00	00	00		
00000210	10	0E	60	00	00	00	00	00	10	0E	60	00	00	00	00	00		
00000220	F0	01	00	00	00	00	00	00	F0	01	00	00	00	00	00	00		
00000230	01	00	00	00	00	00	00	00	2F	6C	69	62	36	34	2F	6C		
00000240	64	2D	6C	69	6E	75	78	2D	78	38	36	2D	36	34	2E	73		

Page 1 of 15 | Offset: 0 | = 127 | Block: | n/a | Size: | n/a

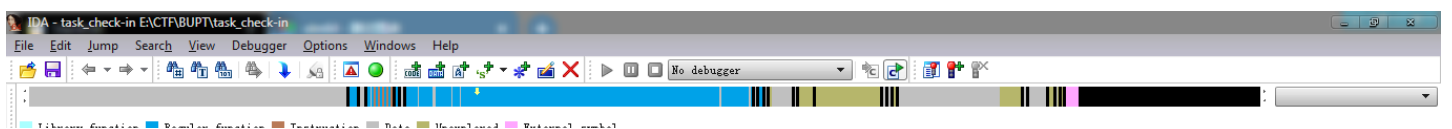
用 IDA 载入,提示为 64 位文件

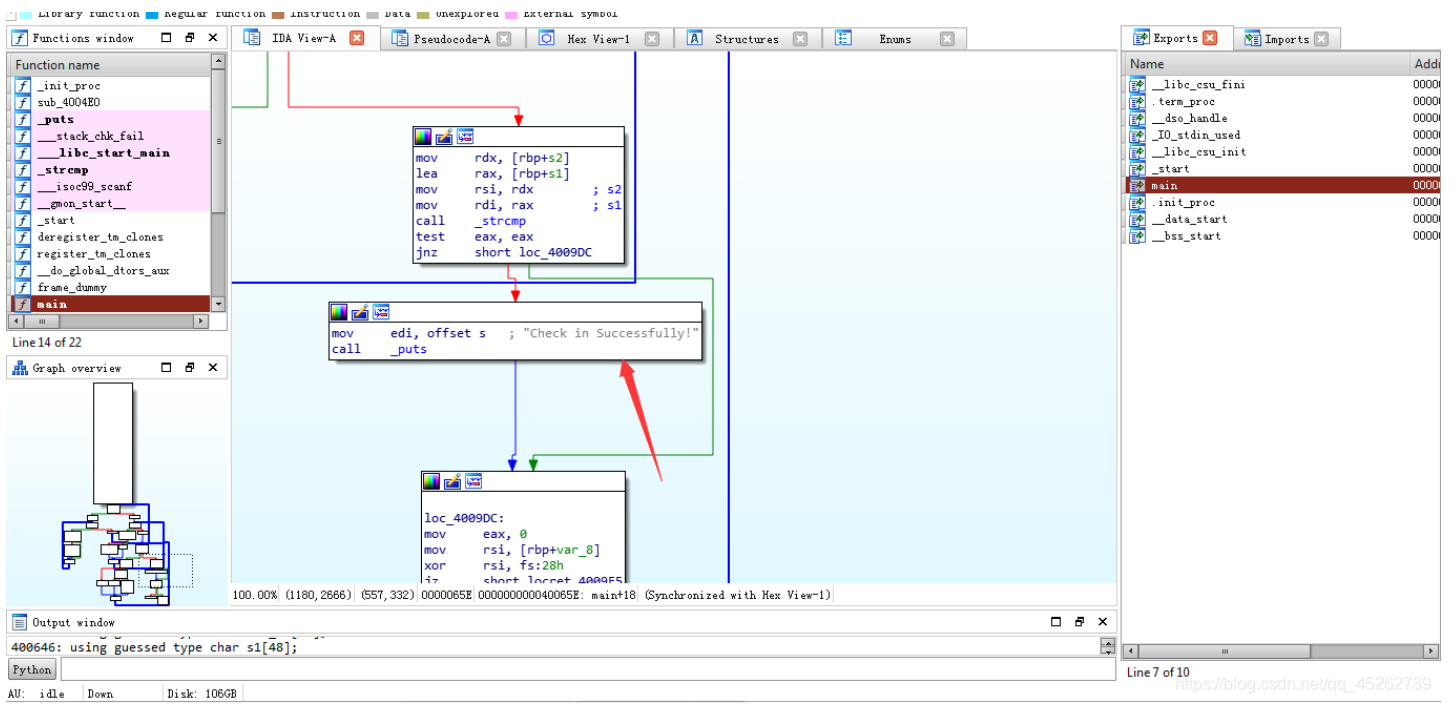


换成 IDA64 载入,找到主函数 main,进入查看
发现奇怪的相关字符串



继续查看,发现输入正确时的输出字符串





F5 反编译,将变量转化为字符

得到源码

```
int __cdecl main(int argc, const char **argv, const char **envp)
```

```
{
    int v3; // eax
    int v4; // eax
    int v6; // [rsp+4h] [rbp-10Ch]
    int i; // [rsp+8h] [rbp-108h]
    signed int j; // [rsp+Ch] [rbp-104h]
    signed int k; // [rsp+10h] [rbp-100h]
    int l; // [rsp+14h] [rbp-FCh]
    char v11; // [rsp+20h] [rbp-F0h]
    char v12; // [rsp+21h] [rbp-EFh]
    char v13; // [rsp+22h] [rbp-EEh]
    char v14; // [rsp+23h] [rbp-EDh]
    char v15; // [rsp+24h] [rbp-ECh]
    char v16; // [rsp+25h] [rbp-EBh]
    char v17; // [rsp+26h] [rbp-EAh]
    char v18; // [rsp+27h] [rbp-E9h]
    char v19; // [rsp+28h] [rbp-E8h]
    char v20; // [rsp+29h] [rbp-E7h]
    char v21; // [rsp+2Ah] [rbp-E6h]
    __int64 v22; // [rsp+30h] [rbp-E0h]
    __int64 v23; // [rsp+38h] [rbp-D8h]
    __int64 v24; // [rsp+40h] [rbp-D0h]
    __int16 v25; // [rsp+48h] [rbp-C8h]
    char v26; // [rsp+4Ah] [rbp-C6h]
    char v27[48]; // [rsp+50h] [rbp-C0h]
    __int16 v28; // [rsp+80h] [rbp-90h]
    char s1[48]; // [rsp+90h] [rbp-80h]
    __int16 v30; // [rsp+C0h] [rbp-50h]
    _BYTE v31[6]; // [rsp+C2h] [rbp-4Eh]
    char v32[48]; // [rsp+D0h] [rbp-40h]
    int v33; // [rsp+100h] [rbp-10h]
    char v34; // [rsp+104h] [rbp-Ch]
    unsigned __int64 v35; // [rsp+108h] [rbp-8h]
}
```

```

v35 = __readsqword(0x28u);
memset(v32, 0, sizeof(v32));
v33 = 0;
v34 = 0;
v26 = 0;
v22 = 'HGFEDCBA';
v23 = 'PONMLKJI';
v24 = 'XWVUTSRQ';
v25 = 'ZY';
v11 = 'B';
v12 = 'U';
v13 = 'P';
v14 = 'T';
v15 = 'I';
v16 = 'S';
v17 = 'H';
v18 = 'E';
v19 = 'K';
v20 = 'Y';
v21 = 0;
memset(v27, 0, sizeof(v27));
v28 = 0;
memset(s1, 0, sizeof(s1));
v30 = 0;
v6 = 0;
__isoc99_scanf("%50s", v27, v31);
for ( i = 0; i < 25; ++i )
{
    for ( j = 0; j <= 25; ++j )
    {
        if ( *((_BYTE *)&v22 + j) == *((&v11 + i) )
        {
            *((_BYTE *)&v22 + j) = 0;
            v3 = v6++;
            v32[v3] = *((&v11 + i);
        }
    }
}
for ( k = 0; k <= 25; ++k )
{
    if ( *((_BYTE *)&v22 + k) )
    {
        v4 = v6++;
        v32[v4] = *((_BYTE *)&v22 + k);
    }
}
for ( l = 0; l < 25; ++l )
{
    if ( v27[l] == 123 )
    {
        s1[l] = 42;
    }
    else if ( v27[l] == 125 )
    {
        s1[l] = 35;
    }
    else
    {
        s1[l] = v32[v27[l] - 65];
    }
}

```

```

}
if ( !strcmp(s1, "UQJO*PEIPAKFRIMXNKDJCINGIBNXPBFFGOUINKDJCIM#") )
    puts("Check in Successfully!");
return 0;
}

```

分析代码

第一个部分

```

for ( i = 0; *(&v11 + i); ++i )
{
    for ( j = 0; j <= 25; ++j )
    {
        if ( *((_BYTE *)&v22 + j) == *(&v11 + i) )
        {
            *((_BYTE *)&v22 + j) = 0;
            v3 = v6++;
            v32[v3] = *(&v11 + i);
        }
    }
}
}

```

在 *(v22+0) 到 *(v22+25) 之间查找将 *(v11+i),如果能找到,把它加在 v32 的后面,并把当前的 *(v22+j) 标记为 0,

第二个部分

```

for ( k = 0; k <= 25; ++k )
{
    if ( *((_BYTE *)&v22 + k) )
    {
        v4 = v6++;
        v32[v4] = *((_BYTE *)&v22 + k);
    }
}
}

```

将第一部分没有标记的字符依次加在 v32 的后面
得到 v32 : **BUPTISHEKYACDFGJLMNOQRVWXZ**

第三个部分

```

for ( l = 0; v27[l]; ++l )
{
    if ( v27[l] == 123 )//123='{'
    {
        s1[l] = 42;//42='*'
    }
    else if ( v27[l] == 125 )//125='}'
    {
        s1[l] = 35;//35='#'
    }
    else
    {
        s1[l] = v32[v27[l] - 65];
    }
}
if ( !strcmp(s1, "UQJO*PEIPAKFRIMXNKDJCINGIBNXPBFFGOUINKDJCIM#") )
    puts("Check in Successfully!");

```

最后的 if 语句判断当 s1 与 `UQJO*PEIPAKFRIMXNKDJCINGIBNXPBFFGOUINKDJCIM#` 相等时才输出正确

向上看 s1 的形成,涉及到 v27,v32 两个字符串

v27 为输入的字符串,也就是我们要提交的 flag

v32 为前两部分形成的字符串

当 v27=='{' 时 s1[i]='*' 对应 s1[4]

当 v32=='}' 时 s1[i]='#' 对应 s1[43]

其余情况 s1=v32[v27[i]-'A']

直至 v27[i] 为空,s1 形成

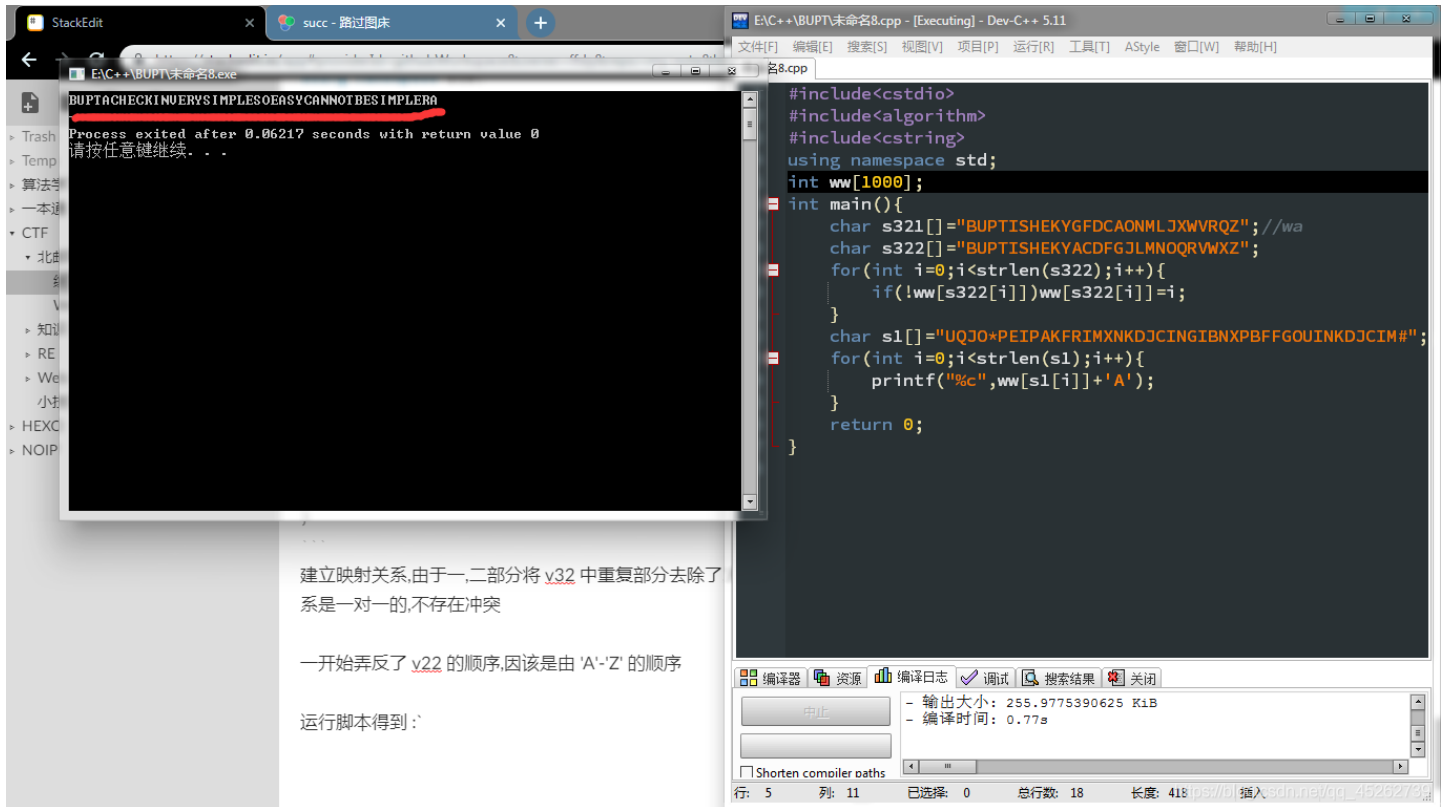
分析到这里,就可以写了脚本反解密了

```
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
int ww[1000];
int main(){
    char s321[]="BUPTISHEKYGFDCANMLJXWVRQZ";//这个是错误的顺序
    char s322[]="BUPTISHEKYACDFGJLMNOQRVWXZ";
    for(int i=0;i<strlen(s322);i++){
        if(!ww[s322[i]])ww[s322[i]]=i;
    }
    char s1[]="UQJO*PEIPAKFRIMXNKDJCINGIBNXPBFFGOUINKDJCIM#";
    for(int i=0;i<strlen(s1);i++){
        printf("%c",ww[s1[i]]+'A');
    }
    return 0;
}
```

建立映射关系,由于一,二部分将 v32 中重复部分去除了,所以映射关系是一对一的,不存在冲突

一开始忽略了大小端

导致 v22 的顺序出错,因该是由 'A'-'Z' 的顺序



The image shows a screenshot of a C++ IDE (Dev-C++ 5.11) and a terminal window. The IDE window displays the following C++ code:

```
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
int ww[1000];
int main(){
    char s321[]="BUPTISHEKYGFDCANMLJXWVRQZ";//wa
    char s322[]="BUPTISHEKYACDFGLMNOQRVWXZ";
    for(int i=0;i<strlen(s322);i++){
        if(!ww[s322[i]])ww[s322[i]]=i;
    }
    char s1[]="UQJ0+PEIPAKFRIMXNKDJCINGIBNXPBF GOUINKDJCIM#";
    for(int i=0;i<strlen(s1);i++){
        printf("%c",ww[s1[i]]+'A');
    }
    return 0;
}
```

The terminal window shows the output of the program:

```
BUPTACHECKINUERVSIMPLESOEASYNANOTBESIMPLERA
Process exited after 0.06217 seconds with return value 0
请按任意键继续. . .
```

Below the terminal window, there is a text box with the following content:

建立映射关系,由于一,二部分将 v32 中重复部分去除了
系是一一对一的,不存在冲突

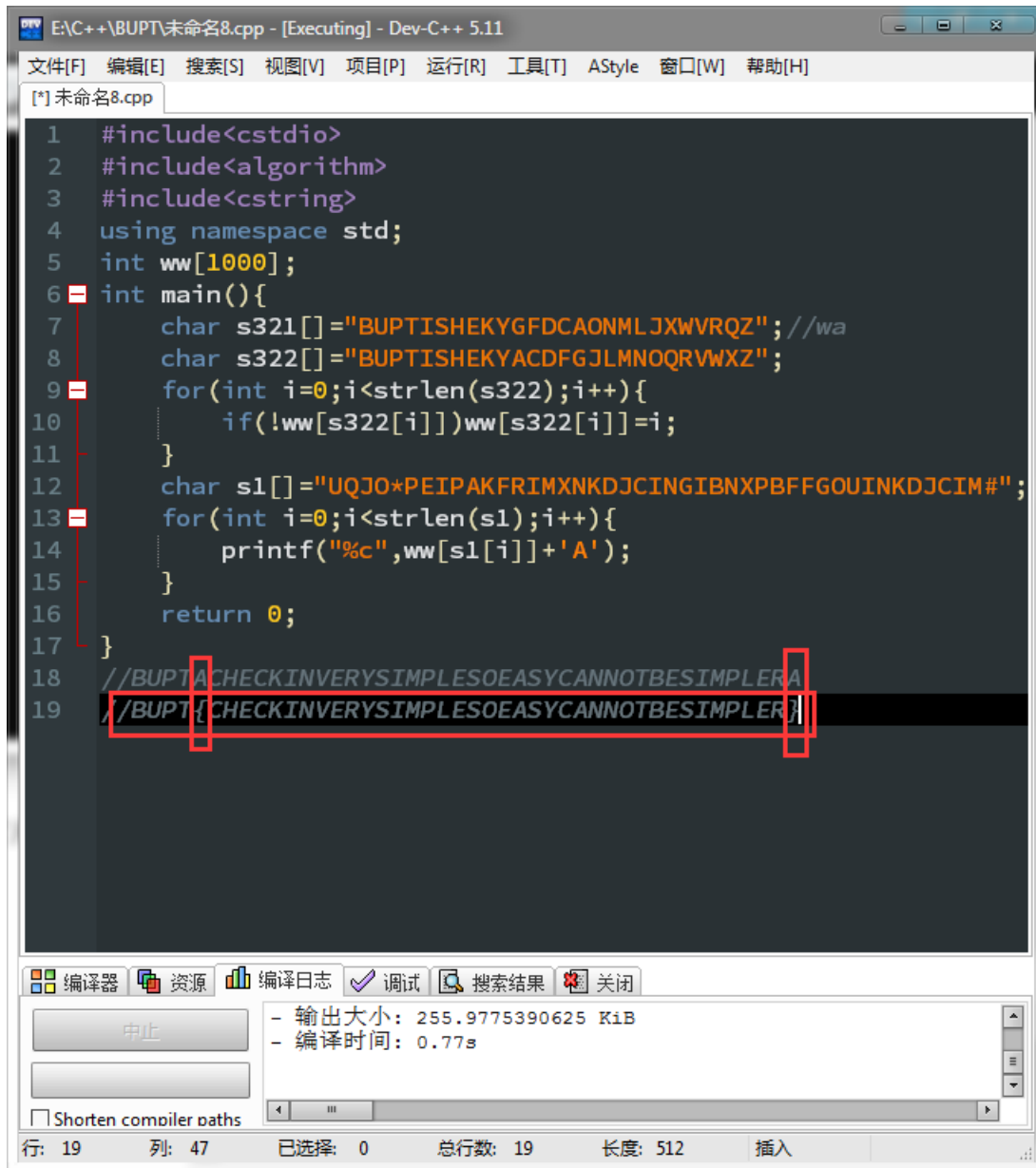
一开始弄反了 v22 的顺序,因该是由 'A'-'Z' 的顺序

运行脚本得到:

At the bottom of the IDE window, the compiler output shows:

```
- 输出大小: 255.9775390625 K1B
- 编译时间: 0.77s
```

运行脚本得到：BUPTACHECKINVERYSIMPLESOEASYCANNOTBESIMPLERA



```
1 #include<cstdio>
2 #include<algorithm>
3 #include<cstring>
4 using namespace std;
5 int ww[1000];
6 int main(){
7     char s321[]="BUPTISHEKYGFDCANMLJXWVRQZ"; //wa
8     char s322[]="BUPTISHEKYACDFGJLMNOQRWXZ";
9     for(int i=0;i<strlen(s322);i++){
10         if(!ww[s322[i]])ww[s322[i]]=i;
11     }
12     char s1[]="UQJO*PEIPAKFRIMXNKDJCINGIBNXPBFFGOUINKDJCIM#";
13     for(int i=0;i<strlen(s1);i++){
14         printf("%c",ww[s1[i]]+'A');
15     }
16     return 0;
17 }
18 //BUPTACHECKINVERYSIMPLESOEASYCANNOTBESIMPLERA
19 //BUPT{CHECKINVERYSIMPLESOEASYCANNOTBESIMPLER}
```

编译器 资源 编译日志 调试 搜索结果 关闭

中止

- 输出大小: 255.9775390625 KiB
- 编译时间: 0.77s

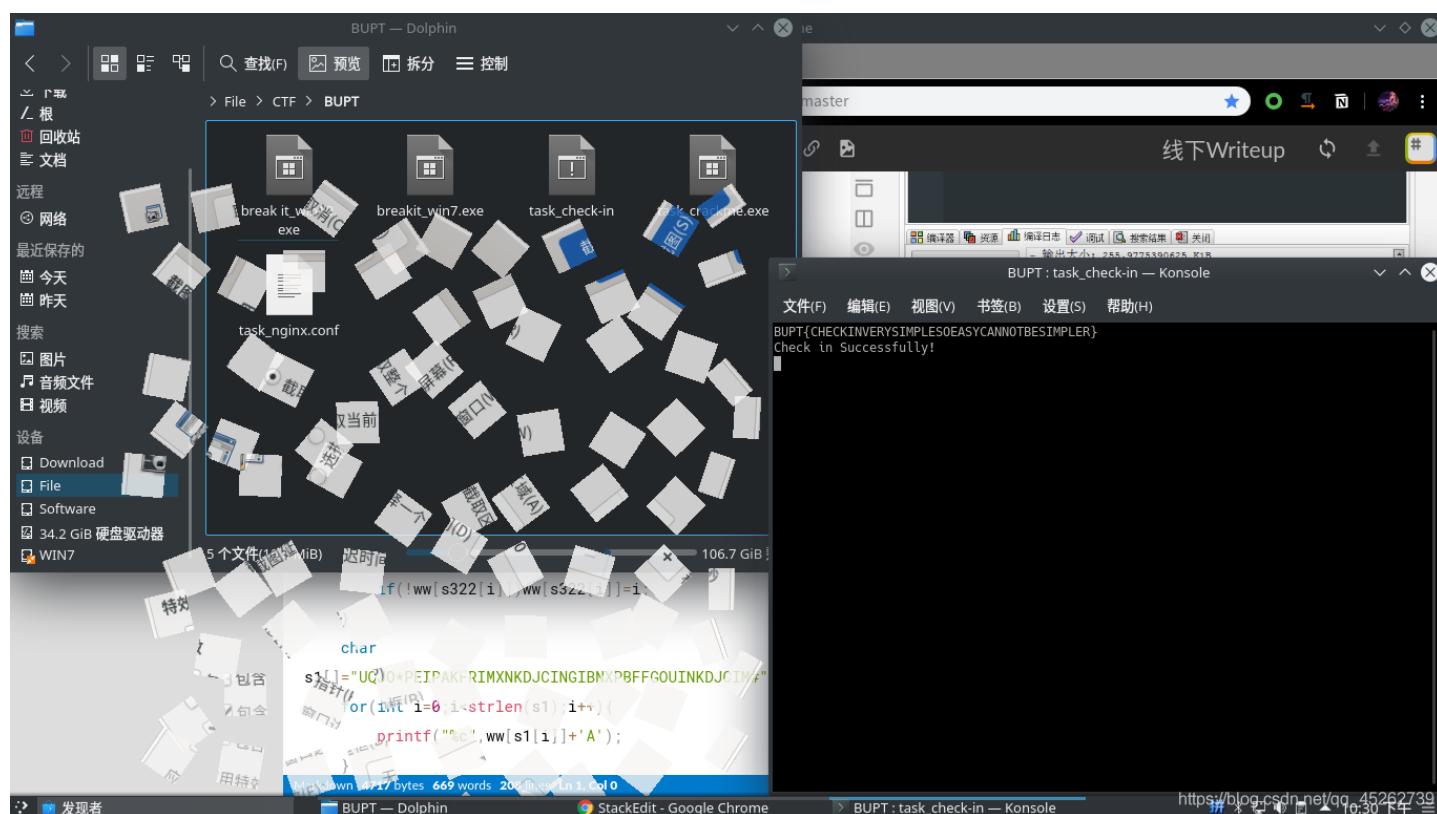
Shorten compiler paths

行: 19 列: 47 已选择: 0 总行数: 19 长度: 512 插入

将位置 4 和最后一位换成相应的 {}

得到 flag: BUPT{CHECKINVERYSIMPLESOEASYCANNOTBESIMPLER}

在 Kali 下测试,成功!



提交, 一百分到手。