

- [Reverse Lab 1](#)
 - [1. Task 1](#)
 - [2. Task 2](#)
 - [3. Task 3](#)
 - [4. Bonus](#)

Reverse Lab 1

1. Task 1

gcc+llvm-mc可以正常生成目标文件：

```
→ Reverse llvm-mc-14 -filetype=obj hello_gcc.s -o hello_gcc_clang.o
```

但clang+as却不能正常生成目标文件：

```
→ Reverse as hello_clang.s -o hello_clang_gcc.o
hello_clang.s: Assembler messages:
hello_clang.s:36: Error: unknown pseudo-op: `.addrsig'
hello_clang.s:37: Error: unknown pseudo-op: `.addrsig_sym'
```

从报错信息中可以看出，这个问题是clang中使用了as无法理解的伪代码操作引起的，即.addrSIG和.addrSIG_sym。根据变量名判断是与地址签名相关的内容，用vim打开能在结尾看到以下两行代码：

```
.addrsig
.addrSIG_sym printf
```

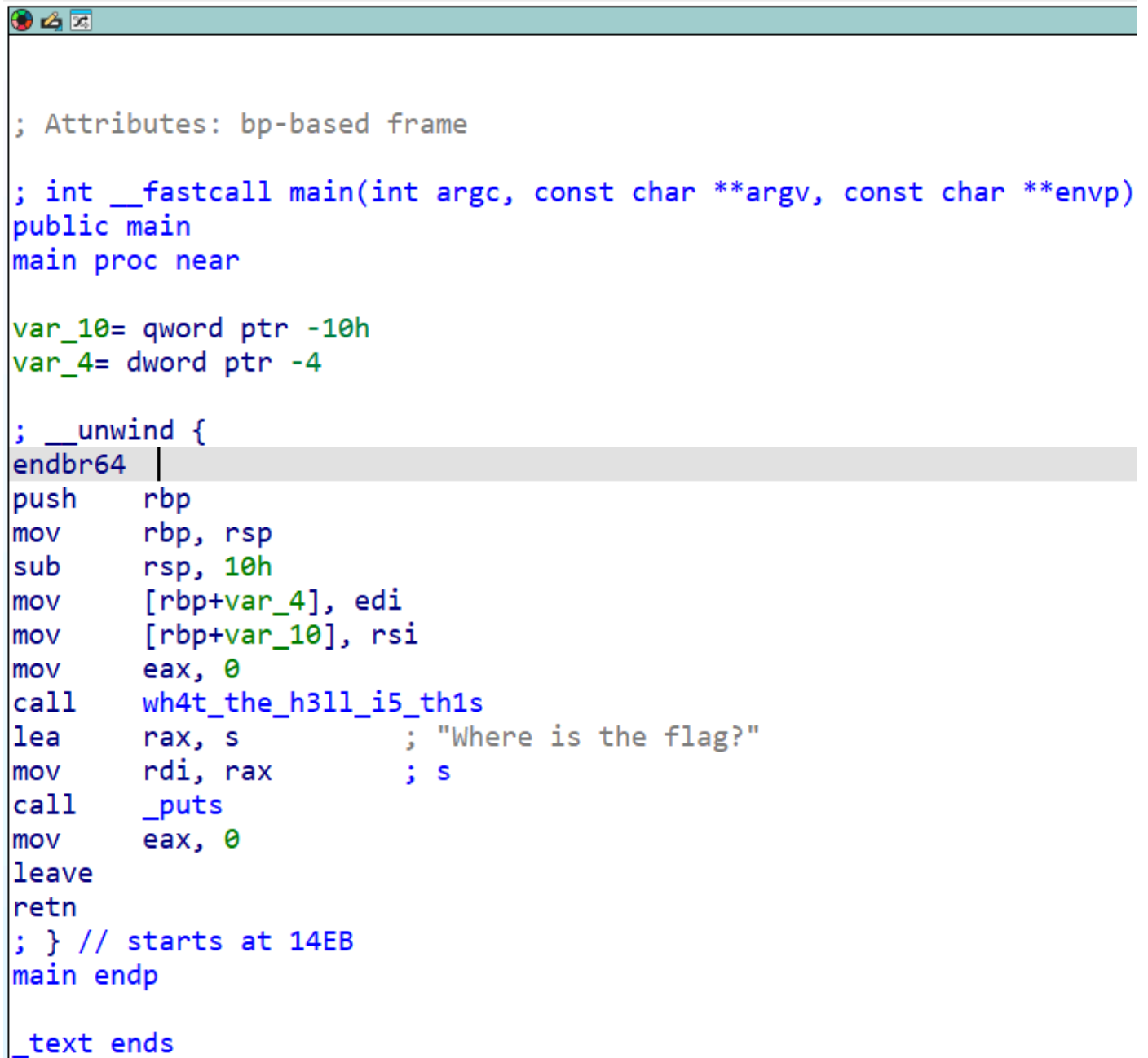
删除后重新用as生成就不会报错了，而且最后生成的可执行文件可以正常运行：

```
→ Reverse gcc hello_clang_gcc.o -o hell.elfo
→ Reverse ./hell.elfo
Hello, World!
```

我的猜测是：addrSIG和.addrSIG_sym是clang独有的，因此as无法理解，究其原因，尽管两者使用了同样的asm语法，但两者使用的汇编器并不是同一个。

2. Task 2

对文件进行逆向，观察main函数发现调用了一个用户自定义函数：



```
; Attributes: bp-based frame

; int __fastcall main(int argc, const char **argv, const char **envp)
public main
main proc near

var_10= qword ptr -10h
var_4= dword ptr -4

; __unwind {
endbr64 |
push     rbp
mov      rbp, rsp
sub      rsp, 10h
mov      [rbp+var_4], edi
mov      [rbp+var_10], rsi
mov      eax, 0
call     wh4t_the_h3ll_i5_th1s
lea      rax, s          ; "Where is the flag?"
mov      rdi, rax        ; s
call     _puts
mov      eax, 0
leave
retn
; } // starts at 14EB
main endp

_text ends
```

转到该函数后可以发现又调用了一个函数：00000000

```
; Attributes: bp-based frame

public wh4t_the_h3ll_i5_th1s
wh4t_the_h3ll_i5_th1s proc near
; __unwind {
endbr64
push     rbp
mov      rbp, rsp
lea      rax, fl4g
mov      rdi, rax
call     00000000
nop
pop      rbp
retn
; } // starts at 14D1
wh4t_the_h3ll_i5_th1s endp
```

转到下一个函数后发现一行带有字符的汇编代码：

```
mov     byte ptr [rax], 41h ; 'A'
```

f	oo0o0o0
f	oo0o0oo
f	oo0oo00
f	oo0oo0o
f	oo0ooo0
f	oo0oooo
f	ooo0000
f	ooo000o
f	ooo00o0
f	ooo00oo
f	ooo0o00
f	ooo0o0o
f	ooo0oo0
f	ooo0ooo
f	oooo000
f	oooo00o
f	oooo0o0
f	oooo0oo
f	ooooo00
f	ooooo0o
f	oooooo0
f	ooooooo
f	wh4t_the_h3ll_i5_th1s
f	main

不断重复以上步骤，在每次调用的函数内找到一个字符，按顺序组合得到：
AAA{hope_u_have_fun~}。

3. Task 3

下载的文件不能直接逆向，先通过clang输出.o文件，再进行逆向得到：

```

int __fastcall main(int argc, const char **argv, const char **envp)
{
    int v3; // eax
    int v5[32]; // [rsp+0h] [rbp-F0h] BYREF
    char s[72]; // [rsp+80h] [rbp-70h] BYREF
    const char **v7; // [rsp+C8h] [rbp-28h]
    int v8; // [rsp+D4h] [rbp-1Ch]
    int v9; // [rsp+D8h] [rbp-18h]
    int i; // [rsp+DCh] [rbp-14h]
    int j; // [rsp+E0h] [rbp-10h]
    int k; // [rsp+E4h] [rbp-Ch]

    v8 = 0;
    v9 = argc;
    v7 = argv;
    memset(s, 0, 0x40uLL);
    memset(v5, 0, sizeof(v5));
    _isoc99_scanf(&unk_5E0, s);
    for ( i = 0; i < strlen(s); ++i )
    {
        if ( s[i] < 48 || s[i] > 57 )
        {
LABEL_15:
            printf("try again\n");
            exit(0);
        }
    }
    for ( j = 0; j < strlen(s); j += 2 )
    {
        v3 = xcrc32(&s[j], 2, 0x414243u);
        v5[j / 2] = v3;
    }
    for ( k = 0; (unsigned __int64)k < 8; ++k )
    {
        if ( v5[k] != target[k] )
            goto LABEL_15;
    }
    printf("awesome\n");
    return 0;
}

```

通过代码逻辑判断，需要输入的字符串应当全是数字（ASCII码值在48和57之间）。输入的数字会经过一个xcrc32的处理：以两位为单位输入函数，经过转换后输出到一个数组，全部数字转换完成之后将得到的数字与target数组中的数字一一比对，完全一致的情况下才会输出awesome。

根据k的取值知道需要比对的次数是8，对应的输入数字长度应为16，现在需要找到xcrc32的转换逻辑。

打开函数得到：

```
__int64 __fastcall xcrc32(_BYTE *a1, int a2, unsigned int a3)
{
    while ( a2-- )
        a3 = crc32_table[(unsigned __int8)(*a1++ ^ HIBYTE(a3))] ^ (a3 << 8);
    return a3;
}
```

可以看出这是一个CRC32的操作，通过查表法进行转换，因此我们可以得到以下思路：通过每两位的穷举计算出对应的crc32值，并与target中的数字比对，需要的遍历次数只有800次，理论上是可行的。

在代码的数据部分得到crc32_table和target的值，并以此写一个程序来爆破对应的数字串：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef unsigned char _BYTE;
#define HIBYTE(x) (((x) >> 24) & 0xFF)
unsigned int crc32_table[]={...};
unsigned int target[]={...};

__int64 __fastcall xcrc32(_BYTE *a1, int a2, unsigned int a3)
{
    while ( a2-- )
        a3 = crc32_table[(unsigned __int8)(*a1++ ^ HIBYTE(a3))] ^ (a3 << 8);
    return a3;
}

int main(){

    __int8 i = 0;
    __int8 j = 0;
    __int8 k = 0;
    unsigned int v5;

    for (k = 0; k < 8; k++)
    {
        for(i = 0; i < 10; i++){
            for(j = 0; j < 10; j++){
                _BYTE *v4 = (_BYTE *)malloc(2);
                v4[0] = i + '0';
                v4[1] = j + '0';
                v5 = xcrc32(&v4[0], 2, 0x414243u);
                if(v5 == target[k]){
                    printf("%dFound: %x %x\n", k, i, j);
                }
            }
        }
    }
}
```

```

    }
}
}

return 0;
}

```

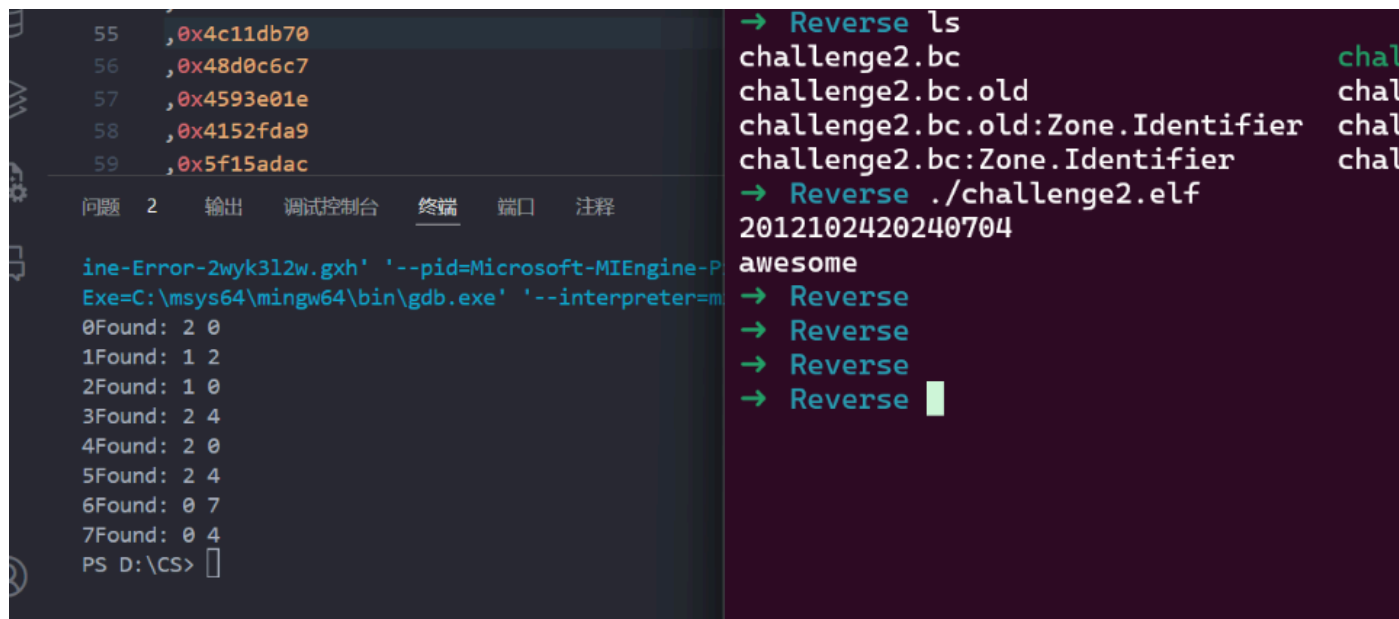
代码中的数据部分已略去，运行后就能得到正确的输入：2012102420240704

```

0Found: 2 0
1Found: 1 2
2Found: 1 0
3Found: 2 4
4Found: 2 0
5Found: 2 4
6Found: 0 7
7Found: 0 4

```

在程序中验证通过：



因此flag为AAA{2012102420240704}

4. Bonus