Capstone反汇编引擎数据类型及API分析及示例(二)

kabeor / 2019-07-27 09:34:00 / 浏览数 3709

Capstone反汇编引擎数据类型及API分析及示例(二)

Capstone反汇编引擎数据类型及API分析及示例(一)
上篇分析了Capstone开放的数据类型，下面就来正式看看API吧
官方开放的API只有二十个左右，但为了能写的更易懂，我将结合实例，分多篇写。
API中作者将capstone缩写为cs，下面我也用这种方式描述

API分析

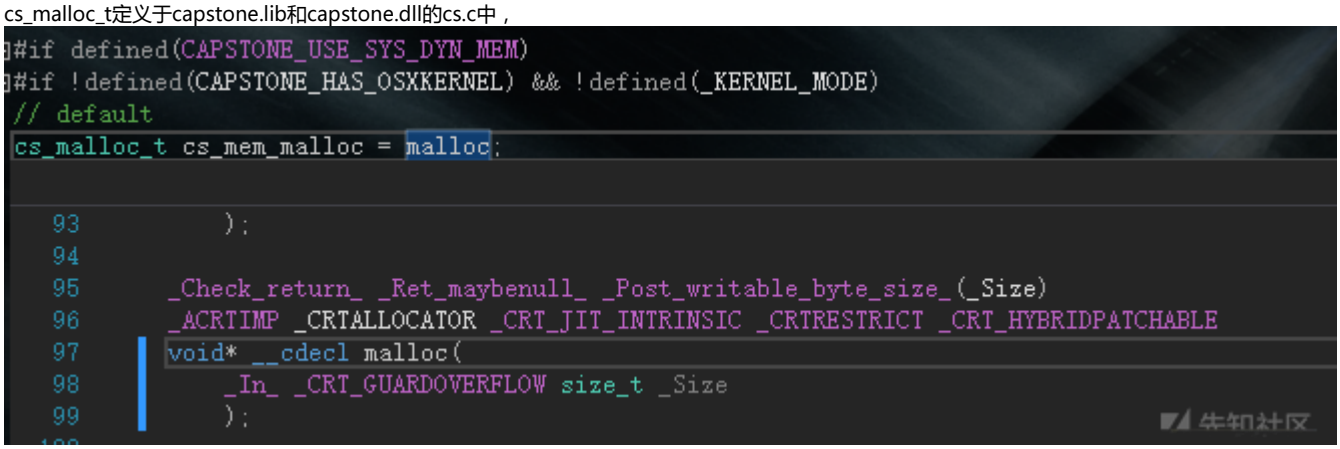
cs_malloc_t

```
void* (CAPSTONE_API *cs_malloc_t)(size_t size);
```

cs的动态内存分配，用于

```
struct cs_opt_mem {
   cs_malloc_t malloc;
   cs_calloc_t calloc;
   cs_realloc_t realloc;
   cs_free_t free;
   cs_vsnprintf_t vsnprintf;
} cs_opt_mem;
```

cs_malloc_t定义于capstone.lib和capstone.dll的cs.c中，



在用户模式下，cs_mem_malloc默认使用系统malloc

Windows driver模式下，`cs_malloc_t cs_mem_malloc = cs_winkernel_malloc;`
cs_winkernel_malloc定义于\capstone-4.0.1\windows\winkernel_mm.c,

实现代码

```
void * CAPSTONE_API cs_winkernel_malloc(size_t size)
{
   // ■■■■■■■0
   NT_ASSERT(size);

   // FP; NonPagedPool■■■■ Windows 7
#pragma prefast(suppress : 30030)       // ■■■■■■POOL_TYPE■■
   size_t number_of_bytes = 0;
   CS_WINKERNEL_MEMBLOCK *block = NULL;
   // ■■■■■■■■■■
   // ■■value■■■■■■■■■■■■■■■■■■NULL■
   if (!NT_SUCCESS(RtlSizeTAdd(size, sizeof(CS_WINKERNEL_MEMBLOCK), &number_of_bytes))) {
       return NULL;
   }
   block = (CS_WINKERNEL_MEMBLOCK *)ExAllocatePoolWithTag(
           NonPagedPool, number_of_bytes, CS_WINKERNEL_POOL_TAG);
   if (!block) {
```

```
        return NULL;
    }
    block->size = size;

    return block->data;
}
```

OSX kernel模式下，`cs_malloc_t cs_mem_malloc = kern_os_malloc;`，这里暂且不探讨。

cs_calloc_t

```
void* (CAPSTONE_API *cs_calloc_t)(size_t nmemb, size_t size);
```

cs申请内存并初始化
用于`struct cs_opt_mem`，定义于cs.c
用户模式：`cs_calloc_t cs_mem_calloc = calloc;`,使用系统calloc
Windows driver模式：`cs_calloc_t cs_mem_calloc = cs_winkernel_calloc;`
实现代码

```
void * CAPSTONE_API cs_winkernel_calloc(size_t n, size_t size)
{
    size_t total = n * size;

    void *new_ptr = cs_winkernel_malloc(total);
    if (!new_ptr) {
        return NULL;
    }

    return RtlFillMemory(new_ptr, total, 0);
}
```

OSX kernel模式：`cs_calloc_t cs_mem_calloc = cs_kern_os_calloc;`

```
static void* cs_kern_os_calloc(size_t num, size_t size)
{
    return kern_os_malloc(num * size); // malloc bzeroes the buffer
}
```

直接调用kern_os_malloc了

cs_realloc_t

```
void* (CAPSTONE_API *cs_realloc_t)(void *ptr, size_t size);
```

cs重新分配内存
用于`struct cs_opt_mem`，定义于cs.c
用户模式：`cs_realloc_t cs_mem_realloc = realloc;`,调用系统realloc
Windows driver模式：`cs_realloc_t cs_mem_realloc = cs_winkernel_realloc;`

实现代码，可以看出是利用cs_winkernel_malloc重新申请

```
void * CAPSTONE_API cs_winkernel_realloc(void *ptr, size_t size)
{
    void *new_ptr = NULL;
    size_t current_size = 0;
    size_t smaller_size = 0;

    if (!ptr) {
        return cs_winkernel_malloc(size);
    }

    new_ptr = cs_winkernel_malloc(size);
    if (!new_ptr) {
        return NULL;
    }

    current_size = CONTAINING_RECORD(ptr, CS_WINKERNEL_MEMBLOCK, data)->size;
    smaller_size = (current_size < size) ? current_size : size;
    RtlCopyMemory(new_ptr, ptr, smaller_size);
    cs_winkernel_free(ptr);
```

```
    return new_ptr;
}
```

OSX kernel模式： `cs_realloc_t cs_mem_realloc = kern_os_realloc;`

cs_free_t

```
typedef void (CAPSTONE_API *cs_free_t)(void *ptr);
```

cs释放内存
用于`struct cs_opt_mem`，定义于cs.c
用户模式： `cs_free_t cs_mem_free = free;`,调用系统free
Windows driver模式： `cs_free_t cs_mem_free = cs_winkernel_free;`

实现代码

```
void CAPSTONE_API cs_winkernel_free(void *ptr)
{
    if (ptr) {
        ExFreePoolWithTag(CONTAINING_RECORD(ptr, CS_WINKERNEL_MEMBLOCK, data), CS_WINKERNEL_POOL_TAG);
    }
}
```

OSX kernel模式： `cs_free_t cs_mem_free = kern_os_free;`

cs_vsnprintf_t

```
int (CAPSTONE_API *cs_vsnprintf_t)(char *str, size_t size, const char *format, va_list ap);
```

按size大小输出到字符串str中

用户模式：

```
#if defined(_WIN32_WCE)
cs_vsnprintf_t cs_vsnprintf = _vsnprintf;
#else
cs_vsnprintf_t cs_vsnprintf = vsnprintf;
#endif  // defined(_WIN32_WCE)
```

值得注意的是，如果系统为wince，将使用_vsnprintf函数
vsnprintf ()和_vsnprintf()对于驱动程序都是可用的，但是它们有一些不同。
在需要返回值和设置空终止符时应使用vsnprintf()

vsnprintf定义在stdio.h

```
#if defined vsnprintf
    // This definition of vsnprintf will generate "warning C4005: 'vsnprintf': macro
    // redefinition" with a subsequent line indicating where the previous definition
    // of vsnprintf was.  This makes it easier to find where vsnprintf was defined.
    #pragma warning(push, 1)
    #pragma warning(1: 4005)
    #define vsnprintf Do not define vsnprintf as a macro
    #pragma warning(pop)
    #error Macro definition of vsnprintf conflicts with Standard Library function declaration
#endif

_Success_(return >= 0)
_Check_return_opt_
_CRT_STDIO_INLINE int __CRTDECL vsnprintf(
    _Out_writes_opt_(_BufferCount) _Always_(_Post_z_) char*       const _Buffer,
    _In_                                              size_t      const _BufferCount,
    _In_z_ _Printf_format_string_                     char const* const _Format,
                                                      va_list           _ArgList
    )
#if defined _NO_CRT_STDIO_INLINE
```

Windows driver模式： `cs_vsnprintf_t cs_vsnprintf = cs_winkernel_vsnprintf;`

代码实现

```
int CAPSTONE_API cs_winkernel_vsnprintf(char *buffer, size_t count, const char *format, va_list argptr)
{
    int result = _vsnprintf(buffer, count, format, argptr);

        // _vsnprintf()[redacted]-1[redacted]“buffer”[redacted]“\0”[redacted]“count”[redacted]
    if (result == -1 || (size_t)result == count) {
        buffer[count - 1] = '\0';
    }

    if (result == -1) {
        // [redacted]-1[redacted]temp buffer[redacted]
        char* tmp = cs_winkernel_malloc(0x1000);
        if (!tmp) {
            return result;
        }

        result = _vsnprintf(tmp, 0x1000, format, argptr);
        NT_ASSERT(result != -1);
        cs_winkernel_free(tmp);
    }

    return result;
}
```

OSX kernel模式： `cs_vsnprintf_t cs_vsnprintf = vsnprintf;`，使用默认vsnprintf

cs_skipdata_cb_t

size_t (CAPSTONE_API cs_skipdata_cb_t)(const uint8_t code, size_t code_size, size_t offset, void *user_data);

SKIPDATA选项的用户自定义回调函数。

code:包含要分解的代码的输入缓冲区。和传递给cs_disasm()的缓冲区相同。
code_size:上面的code缓冲区的大小(以字节为单位)。
offset:上面提到的输入缓冲区code中当前检查字节的位置。
user_data:用户数据通过cs_opt_skipdata结构中的@user_data字段传递给cs_option()。
return:返回要跳过的字节数，或者0表示立即停止反汇编。

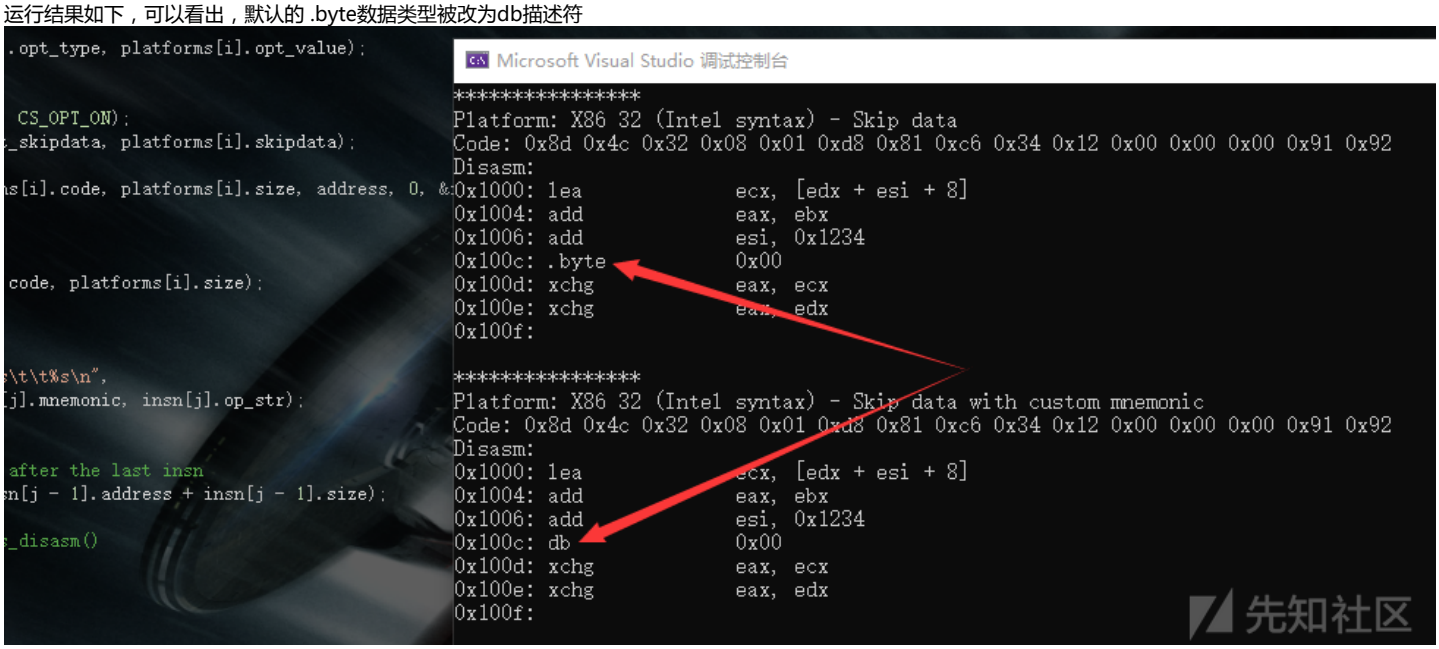
cs_skipdata_cb_t在`struct cs_opt_skipdata`中调用，下面来看一个例子
分析写在注释中

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

#include "platform.h"
#include "capstone.h"

struct platform {
    cs_arch arch;
    cs_mode mode;
    unsigned char* code;
    size_t size;
    const char* comment;
    cs_opt_type opt_type;
    cs_opt_value opt_value;
    cs_opt_type opt_skipdata;
    size_t skipdata;
};

static void print_string_hex(unsigned char* str, size_t len)  //[redacted]
{
    unsigned char* c;

    printf("Code: ");
    for (c = str; c < str + len; c++) {
        printf("0x%02x ", *c & 0xff);
    }
    printf("\n");
}
```

```
static void test()
{

#define X86_CODE32 "\x8d\x4c\x32\x08\x01\xd8\x81\xc6\x34\x12\x00\x00\x00\x91\x92"  //■■■■■■■

#define RANDOM_CODE "\xed\x00\x00\x00\x00\x1a\x5a\x0f\x1f\xff\xc2\x09\x80\x00\x00\x00\x07\xf7\xeb\x2a\xff\xff\x7f\x57\xe3\x01\

   cs_opt_skipdata skipdata = {
       // ■■■ "data" ■■■■■ ".byte" ■■■■■ "db"
       "db",
   };

   struct platform platforms[2] = {         //■■■■■■■■■■■■■■■■■■■■■■■■■
       {
           CS_ARCH_X86,
           CS_MODE_32,
           (unsigned char*)X86_CODE32,
           sizeof(X86_CODE32) - 1,
           "X86 32 (Intel syntax) - Skip data",
       },
       {
           CS_ARCH_X86,
           CS_MODE_32,
           (unsigned char*)X86_CODE32,
           sizeof(X86_CODE32) - 1,
           "X86 32 (Intel syntax) - Skip data with custom mnemonic",
           CS_OPT_INVALID,
           CS_OPT_OFF,
           CS_OPT_SKIPDATA_SETUP,
           (size_t)& skipdata,
       },

   };

   csh handle;   //■■capstone■■
   uint64_t address = 0x1000;  //■■■■■■■
   cs_insn* insn;  //■■■■■■■■
   cs_err err;  //■■■■■
   int i;
   size_t count;  //■■■■■■■■

   for (i = 0; i < sizeof(platforms) / sizeof(platforms[0]); i++) {
       printf("****************\n");
       printf("Platform: %s\n", platforms[i].comment);
       err = cs_open(platforms[i].arch, platforms[i].mode, &handle);  //■■■■■
       if (err) {
           printf("Failed on cs_open() with error returned: %u\n", err);
           abort();
       }

       if (platforms[i].opt_type)
           cs_option(handle, platforms[i].opt_type, platforms[i].opt_value);

       // ■■SKIPDATA ■■
       cs_option(handle, CS_OPT_SKIPDATA, CS_OPT_ON);
       cs_option(handle, platforms[i].opt_skipdata, platforms[i].skipdata);

       count = cs_disasm(handle, platforms[i].code, platforms[i].size, address, 0, &insn);
       if (count) {
           size_t j;

           print_string_hex(platforms[i].code, platforms[i].size);
           printf("Disasm:\n");

           for (j = 0; j < count; j++) {  //■■■■■
               printf("0x%" PRIx64 ":\t%s\t\t%s\n",
                   insn[j].address, insn[j].mnemonic, insn[j].op_str);
           }
```

```
            // ■■■■■■■■■■■■
            printf("0x%" PRIx64 ":\n", insn[j - 1].address + insn[j - 1].size);

            // ■■cs_disasm()■■■■■
            cs_free(insn, count);
        }
        else {
            printf("****************\n");
            printf("Platform: %s\n", platforms[i].comment);
            print_string_hex(platforms[i].code, platforms[i].size);
            printf("ERROR: Failed to disasm given code!\n");
            abort();
        }

        printf("\n");

        cs_close(&handle);
    }
}

int main()
{
    test();

    return 0;
}
```

运行结果如下，可以看出，默认的 .byte数据类型被改为db描述符



cs_version

```
unsigned int CAPSTONE_API cs_version(int *major, int *minor);
```

用来输出capstone版本号
参数
major: API主版本
minor: API次版本
return: 返回主次版本的16进制，如4.0版本返回 0x0400

通过分析源码发现

```
unsigned int CAPSTONE_API cs_version(int *major, int *minor)
{
    if (major != NULL && minor != NULL) {
        *major = CS_API_MAJOR;
        *minor = CS_API_MINOR;
    }

    return (CS_API_MAJOR << 8) + CS_API_MINOR;
}
```

```
// Capstone API version
#define CS_API_MAJOR 4
#define CS_API_MINOR 0

// Version for bleeding edge code of the Github's "next" branch.
// Use this if you want the absolutely latest development code.
// This version number will be bumped up whenever we have a new major change.
#define CS_NEXT_VERSION 5

// Capstone package version
#define CS_VERSION_MAJOR CS_API_MAJOR
#define CS_VERSION_MINOR CS_API_MINOR
#define CS_VERSION_EXTRA 1
```

该版本定义于cs.c中，编译后不可更改，不接受自定义版本

示例1：

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

#include "platform.h"
#include "capstone.h"

static int test()
{
   return cs_version(NULL, NULL);
}

int main()
{
   int version = test();
   printf("%X", version);
   return 0;
}
```

输出

```
int main()
{
    int version = test();
    printf("%X", version);
    return 0;
}
```

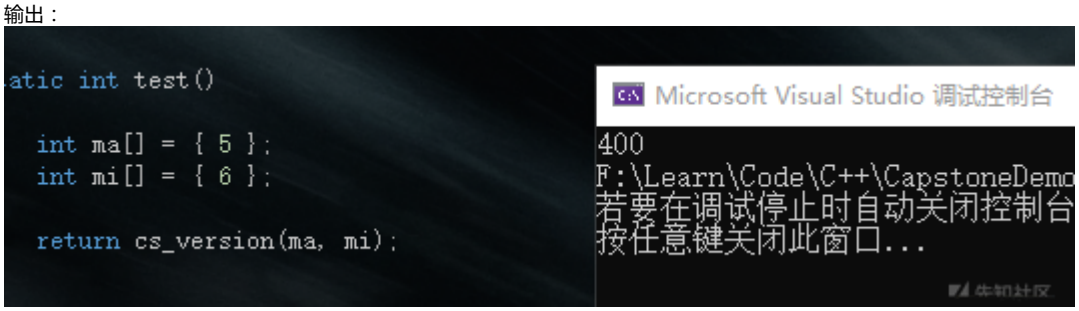
```
Microsoft Visual Studio 调试控制台
400
F:\Learn\Code\C++\CapstoneDemo
若要在调试停止时自动关闭控制台
按任意键关闭此窗口...
```

示例2，强行修改版本：

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

#include "platform.h"
#include "capstone.h"

static int test()
{
```

```
   int ma[] = { 5 };
   int mi[] = { 6 };

   return cs_version(ma, mi);
}

int main()
{
   int version = test();
   printf("%X", version);
   return 0;
}
```

输出：



可以看到并不能改变

cs_support

bool CAPSTONE_API cs_support(int query);

用来检查capstone库是否支持参数输入的架构或处于某编译选项
通过查看源码得知，共有四种查询参数

```
bool CAPSTONE_API cs_support(int query)
{
   if (query == CS_ARCH_ALL)
       return all_arch == ((1 << CS_ARCH_ARM) | (1 << CS_ARCH_ARM64) |
               (1 << CS_ARCH_MIPS) | (1 << CS_ARCH_X86) |
               (1 << CS_ARCH_PPC) | (1 << CS_ARCH_SPARC) |
               (1 << CS_ARCH_SYSZ) | (1 << CS_ARCH_XCORE) |
               (1 << CS_ARCH_M68K) | (1 << CS_ARCH_TMS320C64X) |
               (1 << CS_ARCH_M680X) | (1 << CS_ARCH_EVM));

   if ((unsigned int)query < CS_ARCH_MAX)
       return all_arch & (1 << query);

   if (query == CS_SUPPORT_DIET) {
#ifdef CAPSTONE_DIET
       return true;
#else
       return false;
#endif
   }

   if (query == CS_SUPPORT_X86_REDUCE) {
#if defined(CAPSTONE_HAS_X86) && defined(CAPSTONE_X86_REDUCE)
       return true;
#else
       return false;
#endif
   }

   // unsupported query
   return false;
}
```

示例1(CS_ARCH_ALL，检查是否支持所有架构)：

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

#include "platform.h"
#include "capstone.h"


static int test()
{
    return cs_support(CS_ARCH_ALL);
}

int main()
{
    printf("%d", test());
    return 0;
}
```

```
1
F:\Lea
若要在
按任意
```

示例2(CSARCH*，检查是否支持指定架构)

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

#include "platform.h"
#include "capstone.h"


static int test()
{
    return cs_support(CS_ARCH_X86);
}

int main()
{
    printf("%d", test());
    return 0;
}
```

```
1
F:\Lea
若要在
按任意
```

示例3(检查是否处于DIET编译模式)：

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

#include "platform.h"
#include "capstone.h"


static int test()
{
    return cs_support(CS_SUPPORT_DIET);
}

int main()
{
    printf("%d", test());
    return 0;
}
```

```
0
F:\
若要
按任
```

示例4(检查是否处于X86_REDUCE编译模式)：

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

#include "platform.h"
#include "capstone.h"


static int test()
{
    return cs_support(CS_SUPPORT_X86_REDUCE);
}

int main()
{
    printf("%d", test());
    return 0;
}
```

```
Micro
0
F:\Learn
若要在调
按任意键
```

)

下篇将介绍更多有用的API，望支持

上一篇：论菜鸡pwn手如何在无网环境（ps... 下一篇：Windows Kernel Ex...

1. 2 条回复

x14m1 2019-08-02 11:02:10

typedef enum cs_arch {
CS_ARCH_ARM = 0, ///< ARM architecture (including Thumb, Thumb-2)
CS_ARCH_ARM64, ///< ARM-64, also called AArch64
CS_ARCH_MIPS, ///< Mips architecture
CS_ARCH_X86, ///< X86 architecture (including x86 & x86-64)
CS_ARCH_PPC, ///< PowerPC architecture
CS_ARCH_SPARC, ///< Sparc architecture
CS_ARCH_SYSZ, ///< SystemZ architecture
CS_ARCH_XCORE, ///< XCore architecture
CS_ARCH_M68K, ///< 68K architecture
CS_ARCH_TMS320C64X, ///< TMS320C64x architecture
CS_ARCH_M680X, ///< 680X architecture
CS_ARCH_EVM, ///< Ethereum architecture
CS_ARCH_MOS65XX, ///< MOS65XX architecture (including MOS6502)
CS_ARCH_MAX,
CS_ARCH_ALL = 0xFFFF, // All architectures - for cs_support()
} cs_arch;

define CS_SUPPORT_DIET (CS_ARCH_ALL + 1)

define CS_SUPPORT_X86_REDUCE (CS_ARCH_ALL + 2)

0 回复Ta

kabeor 2019-08-02 15:01:34

@x14m1 我第一篇写了哈

0 回复Ta

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