Capstone反汇编引擎数据类型及API分析及示例(三)

kabeor / 2019-07-29 09:08:00 / 浏览数 3763 安全技术 二进制安全 顶(0) 踩(0)

Capstone反汇编引擎数据类型及API分析及示例(三)

Capstone反汇编引擎数据类型及API分析及示例(一)
Capstone反汇编引擎数据类型及API分析及示例(二)

API分析

cs_open

```
cs_err CAPSTONE_API cs_open(cs_arch arch, cs_mode mode, csh *handle);
```

初始化cs句柄

参数
arch: 架构类型 (CSARCH)
mode: 硬件模式. CSMODE在cs_mode数据类型中可查
handle: 指向句柄, 返回时更新
return: 创建成功返回CS_ERR_OK，否则返回cs_err枚举中对应的错误信息

实现代码

```
cs_err CAPSTONE_API cs_open(cs_arch arch, cs_mode mode, csh *handle)
{
   cs_err err;
   struct cs_struct *ud;
   if (!cs_mem_malloc || !cs_mem_calloc || !cs_mem_realloc || !cs_mem_free || !cs_vsnprintf)
       // Error: ■■cs_open()■, ■■■■cs_option(CS_OPT_MEM)■■■■■■■■■■■■
       return CS_ERR_MEMSETUP;

   if (arch < CS_ARCH_MAX && cs_arch_init[arch]) {
       // ■■■■■■■■■■■■■■■■■■■■■■■■■
       if (mode & cs_arch_disallowed_mode_mask[arch]) {
           *handle = 0;
           return CS_ERR_MODE;
       }

       ud = cs_mem_calloc(1, sizeof(*ud));
       if (!ud) {
           // ■■■■
           return CS_ERR_MEM;
       }

       ud->errnum = CS_ERR_OK;
       ud->arch = arch;
       ud->mode = mode;
       // ■■■■■■■■■detail■■
       ud->detail = CS_OPT_OFF;

       // ■■skipdata■■
       ud->skipdata_setup.mnemonic = SKIPDATA_MNEM;

       err = cs_arch_init[ud->arch](ud);
       if (err) {
           cs_mem_free(ud);
           *handle = 0;
           return err;
       }

       *handle = (uintptr_t)ud;

       return CS_ERR_OK;
   } else {
       *handle = 0;
```

```
        return CS_ERR_ARCH;
    }
}
```

其中，cs_struct结构体包含更多细节设定，如下

```
struct cs_struct {
    cs_arch arch;
    cs_mode mode;
    Printer_t printer;  // ■■asm
    void *printer_info; // ■■■■
    Disasm_t disasm;    // ■■■
    void *getinsn_info; // ■■■■■■
    GetName_t reg_name;
    GetName_t insn_name;
    GetName_t group_name;
    GetID_t insn_id;
    PostPrinter_t post_printer;
    cs_err errnum;
    ARM_ITStatus ITBlock;   // ARM■■■■
    cs_opt_value detail, imm_unsigned;
    int syntax; //ARM, Mips & PPC■■■■■■asm■■■■
    bool doing_mem; // ■InstPrinter■■■■■■■■■■
    unsigned short *insn_cache; //■mapping.c■■■■■■
    GetRegisterName_t get_regname;
    bool skipdata;  // ■■■■■■■■■■■■■■■■■■True
    uint8_t skipdata_size;  //■■■bytes■■■
    cs_opt_skipdata skipdata_setup; // ■■■skipdata■■
    const uint8_t *regsize_map; //■■register■■ (■■■■■x86)
    GetRegisterAccess_t reg_access;
    struct insn_mnem *mnem_list;    // ■■■■■■■■■■■list
};
```

示例(创建一个x86_64类型的cs句柄)：

```
cs_open(CS_ARCH_X86, CS_MODE_64, &handle)
```

cs_close

```
cs_err CAPSTONE_API cs_close(csh *handle);
```

释放句柄
参数
handle: 指向一个cs_open()打开的句柄
return: 释放成功返回CS_ERR_OK,否则返回cs_err枚举的错误信息

实现代码，可以看出释放句柄实质为将句柄值设置为0

```
cs_err CAPSTONE_API cs_close(csh *handle)
{
    struct cs_struct *ud;
    struct insn_mnem *next, *tmp;

    if (*handle == 0)
        // ■■■■■
        return CS_ERR_CSH;

    ud = (struct cs_struct *)(*handle);

    if (ud->printer_info)
        cs_mem_free(ud->printer_info);

    // ■■■■■■■■■■■list
    tmp = ud->mnem_list;
    while(tmp) {
        next = tmp->next;
        cs_mem_free(tmp);
        tmp = next;
    }

    cs_mem_free(ud->insn_cache);
```

```
   memset(ud, 0, sizeof(*ud));
   cs_mem_free(ud);

   // handle■■■■0■■■■■■■■■cs_close()■■■■■■■■
   *handle = 0;

   return CS_ERR_OK;
}
```

示例：
```
cs_close(&handle);
```

cs_option

```
cs_err CAPSTONE_API cs_option(csh handle, cs_opt_type type, size_t value);
```

反编译引擎的运行时选项

handle: cs_open()打开的句柄
type: 设置选项的类型
value: 与type对应的选项值
return: 设置成功返回CS_ERR_OK,否则返回cs_err枚举的错误信息

注意: 在CS_OPT_MEM的情况下，handle可以是任何值，因此cs_option(handle, CS_OPT_MEM, value)必须在cs_open()之前被调用

实现代码

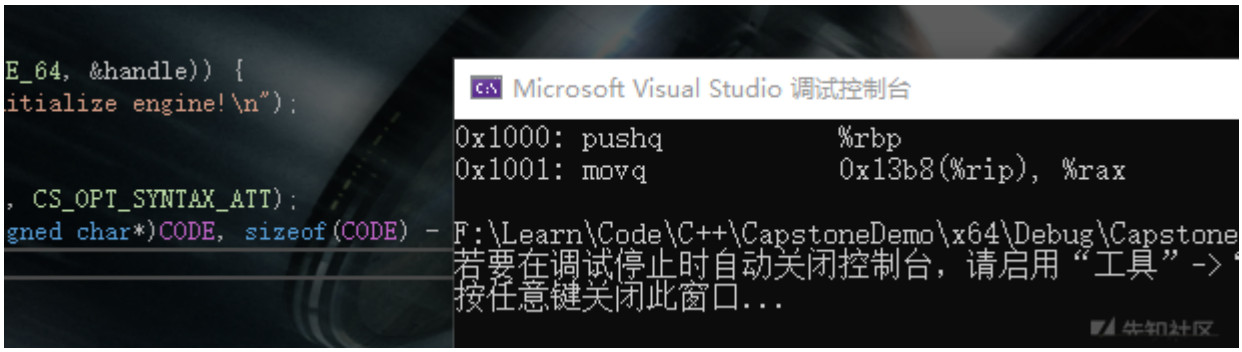
```
cs_err CAPSTONE_API cs_option(csh ud, cs_opt_type type, size_t value)
{
   struct cs_struct *handle;
   cs_opt_mnem *opt;

   // ■■■■■API■■■ (even cs_open())
   if (type == CS_OPT_MEM) {
       cs_opt_mem *mem = (cs_opt_mem *)value;

       cs_mem_malloc = mem->malloc;
       cs_mem_calloc = mem->calloc;
       cs_mem_realloc = mem->realloc;
       cs_mem_free = mem->free;
       cs_vsnprintf = mem->vsnprintf;

       return CS_ERR_OK;
   }

   handle = (struct cs_struct *)(uintptr_t)ud;
   if (!handle)
       return CS_ERR_CSH;

   switch(type) {
       default:
           break;

       case CS_OPT_UNSIGNED:
           handle->imm_unsigned = (cs_opt_value)value;
           return CS_ERR_OK;

       case CS_OPT_DETAIL:
           handle->detail = (cs_opt_value)value;
           return CS_ERR_OK;

       case CS_OPT_SKIPDATA:
           handle->skipdata = (value == CS_OPT_ON);
           if (handle->skipdata) {
               if (handle->skipdata_size == 0) {
                   handle->skipdata_size = skipdata_size(handle);
               }
           }
           return CS_ERR_OK;
```

```
        case CS_OPT_SKIPDATA_SETUP:
            if (value)
                handle->skipdata_setup = *((cs_opt_skipdata *)value);
            return CS_ERR_OK;

        case CS_OPT_MNEMONIC:
            opt = (cs_opt_mnem *)value;
            if (opt->id) {
                if (opt->mnemonic) {
                    struct insn_mnem *tmp;

                    // ■■■■■■■■■■■■■
                    // ■■■■insn■■■list■
                    tmp = handle->mnem_list;
                    while(tmp) {
                        if (tmp->insn.id == opt->id) {
                            // f■■■■■■■■■■■
                            (void)strncpy(tmp->insn.mnemonic, opt->mnemonic, sizeof(tmp->insn.mnemonic) - 1);
                            tmp->insn.mnemonic[sizeof(tmp->insn.mnemonic) - 1] = '\0';
                            break;
                        }
                        tmp = tmp->next;
                    }

                    // 2. ■■■■■■■■■■■■
                    if (!tmp) {
                        tmp = cs_mem_malloc(sizeof(*tmp));
                        tmp->insn.id = opt->id;
                        (void)strncpy(tmp->insn.mnemonic, opt->mnemonic, sizeof(tmp->insn.mnemonic) - 1);
                        tmp->insn.mnemonic[sizeof(tmp->insn.mnemonic) - 1] = '\0';
                        // ■■■■■list■■■
                        tmp->next = handle->mnem_list;
                        handle->mnem_list = tmp;
                    }
                    return CS_ERR_OK;
                } else {
                    struct insn_mnem *prev, *tmp;

                    tmp = handle->mnem_list;
                    prev = tmp;
                    while(tmp) {
                        if (tmp->insn.id == opt->id) {
                            // ■■■■
                            if (tmp == prev) {
                                handle->mnem_list = tmp->next;
                            } else {
                                prev->next = tmp->next;
                            }
                            cs_mem_free(tmp);
                            break;
                        }
                        prev = tmp;
                        tmp = tmp->next;
                    }
                }
            }
            return CS_ERR_OK;

        case CS_OPT_MODE:
            // ■■■■■■■■■■■■■
            if (value & cs_arch_disallowed_mode_mask[handle->arch]) {
                return CS_ERR_OPTION;
            }
            break;
    }

    return cs_arch_option[handle->arch](handle, type, value);
}
```

示例，更改反汇编后显示的语法：

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

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

using namespace std;

#define CODE "\x55\x48\x8b\x05\xb8\x13\x00\x00"

int main(void)
{
   csh handle;
   cs_insn* insn;
   size_t count;

   if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle)) {
       printf("ERROR: Failed to initialize engine!\n");
       return -1;
   }
   cs_option(handle, CS_OPT_SYNTAX, CS_OPT_SYNTAX_ATT);  // ■AT&T■■■■
   count = cs_disasm(handle, (unsigned char*)CODE, sizeof(CODE) - 1, 0x1000, 0, &insn);
   if (count) {
       size_t j;

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

       cs_free(insn, count);
   }
   else
       printf("ERROR: Failed to disassemble given code!\n");

   cs_close(&handle);

   return 0;
}
```

输出



cs_errno

```
cs_err CAPSTONE_API cs_errno(csh handle);
```

API出错时返回错误消息
参数
handle: cs_open()打开的句柄
return: 无错误返回CS_ERR_OK,否则返回cs_err枚举的错误信息

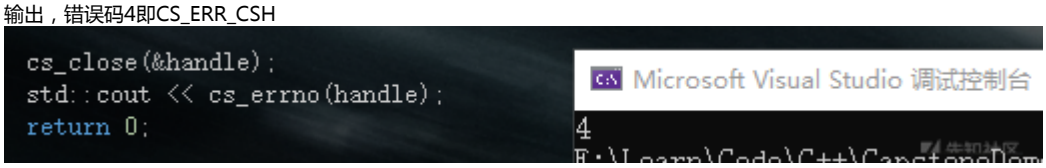
实现很简单，判断到句柄不存在直接返回CS_ERR_CSH

示例：

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

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

using namespace std;

#define CODE "\x55\x48\x8b\x05\xb8\x13\x00\x00"

int main(void)
{
   csh handle = 0;
   cs_insn* insn;
   size_t count;

   if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle)) {
       printf("ERROR: Failed to initialize engine!\n");
       return -1;
   }

   cs_close(&handle);
   std::cout << cs_errno(handle);    //■■■■■■■■■■■
   return 0;
}
```

输出，错误码4即CS_ERR_CSH



cs_strerror

```
const char * CAPSTONE_API cs_strerror(cs_err code);
```

将上个API输出的错误码转换为详细错误信息

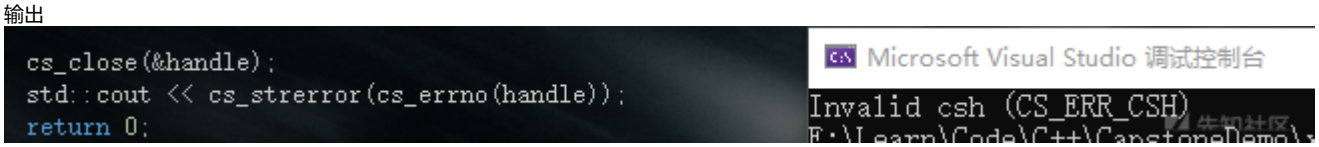
```
const char * CAPSTONE_API cs_strerror(cs_err code)
{
   switch(code) {
       default:
           return "Unknown error code";
       case CS_ERR_OK:
           return "OK (CS_ERR_OK)";
       case CS_ERR_MEM:
           return "Out of memory (CS_ERR_MEM)";
       case CS_ERR_ARCH:
           return "Invalid/unsupported architecture(CS_ERR_ARCH)";
       case CS_ERR_HANDLE:
           return "Invalid handle (CS_ERR_HANDLE)";
       case CS_ERR_CSH:
           return "Invalid csh (CS_ERR_CSH)";
       case CS_ERR_MODE:
           return "Invalid mode (CS_ERR_MODE)";
       case CS_ERR_OPTION:
           return "Invalid option (CS_ERR_OPTION)";
       case CS_ERR_DETAIL:
           return "Details are unavailable (CS_ERR_DETAIL)";
       case CS_ERR_MEMSETUP:
           return "Dynamic memory management uninitialized (CS_ERR_MEMSETUP)";
       case CS_ERR_VERSION:
           return "Different API version between core & binding (CS_ERR_VERSION)";
       case CS_ERR_DIET:
           return "Information irrelevant in diet engine (CS_ERR_DIET)";
       case CS_ERR_SKIPDATA:
           return "Information irrelevant for 'data' instruction in SKIPDATA mode (CS_ERR_SKIPDATA)";
       case CS_ERR_X86_ATT:
           return "AT&T syntax is unavailable (CS_ERR_X86_ATT)";
       case CS_ERR_X86_INTEL:
           return "INTEL syntax is unavailable (CS_ERR_X86_INTEL)";
```

```
        case CS_ERR_X86_MASM:
            return "MASM syntax is unavailable (CS_ERR_X86_MASM)";
    }
}
```

示例，结合cs_errno使用：

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

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

using namespace std;

#define CODE "\x55\x48\x8b\x05\xb8\x13\x00\x00"

int main(void)
{
   csh handle = 0;
   cs_insn* insn;
   size_t count;

   if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle)) {
       printf("ERROR: Failed to initialize engine!\n");
       return -1;
   }

   cs_close(&handle);
   std::cout << cs_strerror(cs_errno(handle));  //■■■■■■■■■
   return 0;
}
```

输出



cs_disasm

```
size_t CAPSTONE_API cs_disasm(csh handle,
       const uint8_t *code, size_t code_size,
       uint64_t address,
       size_t count,
       cs_insn **insn);
```

给定缓冲区、大小、地址和编号，反编译机器码
API动态地分配内存来包含分解的指令，生成的指令将放在*insn中

注意： 必须释放分配的内存，以避免内存泄漏。对于需要动态分配稀缺内存的系统(如OS内核或固件)，API cs_disasm_iter()可能是比cs_disasm()更好的选择。原因是，使用cs_disasm()时，基于有限的可用内存，必须预先计算要分解多少条指令。

handle: cs_open()返回的句柄
code: 包含要反汇编的机器码的缓冲区。
code_size:上面代码缓冲区的大小。
address:给定原始代码缓冲区中的第一条指令的地址。
insn: 由这个API填写的指令数组。注意: insn将由这个函数分配，应该用cs_free () API释放
count: 需要分解的指令数量，或输入0分解所有指令
return:成功反汇编指令的数量，如果该函数未能反汇编给定的代码，则为0，失败时，调用cs_errno()获取错误代码。

源码分析

```
size_t CAPSTONE_API cs_disasm(csh ud, const uint8_t *buffer, size_t size, uint64_t offset, size_t count, cs_insn **insn)
{
   struct cs_struct *handle;
   MCInst mci;
   uint16_t insn_size;
   size_t c = 0, i;
   unsigned int f = 0; // ■■■■■■■■■■■■
   cs_insn *insn_cache;    // ■■■■■■■■■■
```

```
   void *total = NULL;
   size_t total_size = 0;  //■■insn■■■■■■■■■■■
   bool r;
   void *tmp;
   size_t skipdata_bytes;
   uint64_t offset_org; // ■■■■■■■■■■■■■
   size_t size_org;
   const uint8_t *buffer_org;
   unsigned int cache_size = INSN_CACHE_SIZE;
   size_t next_offset;

   handle = (struct cs_struct *)(uintptr_t)ud;
   if (!handle) {
       // ■■■■:
       // handle->errnum = CS_ERR_HANDLE;
       return 0;
   }

   handle->errnum = CS_ERR_OK;

   // ■■ARM■■■IT block
   if (handle->arch == CS_ARCH_ARM)
       handle->ITBlock.size = 0;

#ifdef CAPSTONE_USE_SYS_DYN_MEM
   if (count > 0 && count <= INSN_CACHE_SIZE)
       cache_size = (unsigned int) count;
#endif

   // ■■SKIPDATA■■■■■
   buffer_org = buffer;
   offset_org = offset;
   size_org = size;

   total_size = sizeof(cs_insn) * cache_size;
   total = cs_mem_malloc(total_size);
   if (total == NULL) {
       // ■■■■
       handle->errnum = CS_ERR_MEM;
       return 0;
   }

   insn_cache = total;

   while (size > 0) {
       MCInst_Init(&mci);
       mci.csh = handle;

       mci.address = offset;

       if (handle->detail) {
           //■detail■■■■■■
           insn_cache->detail = cs_mem_malloc(sizeof(cs_detail));
       } else {
           insn_cache->detail = NULL;
       }

       // ■non-detailed■■■■■■■■
       mci.flat_insn = insn_cache;
       mci.flat_insn->address = offset;
#ifdef CAPSTONE_DIET
       //mnemonic & op_str0■■
       mci.flat_insn->mnemonic[0] = '\0';
       mci.flat_insn->op_str[0] = '\0';
#endif

       r = handle->disasm(ud, buffer, size, &mci, &insn_size, offset, handle->getinsn_info);
       if (r) {
           SStream ss;
```

```
            SStream_Init(&ss);

            mci.flat_insn->size = insn_size;

            //■■■■■■■■■■■■■■insn ID
            handle->insn_id(handle, insn_cache, mci.Opcode);

            handle->printer(&mci, &ss, handle->printer_info);
            fill_insn(handle, insn_cache, ss.buffer, &mci, handle->post_printer, buffer);

            // ■■opcode (X86)
            if (handle->arch == CS_ARCH_X86)
                insn_cache->id += mci.popcode_adjust;

            next_offset = insn_size;
        } else  {
            // ■■■■■■

            // ■detail■■■■■■
            if (handle->detail) {
                cs_mem_free(insn_cache->detail);
            }

            if (!handle->skipdata || handle->skipdata_size > size)
                break;

            if (handle->skipdata_setup.callback) {
                skipdata_bytes = handle->skipdata_setup.callback(buffer_org, size_org,
                        (size_t)(offset - offset_org), handle->skipdata_setup.user_data);
                if (skipdata_bytes > size)
                    break;

                if (!skipdata_bytes)
                    break;
            } else
                skipdata_bytes = handle->skipdata_size;

            insn_cache->id = 0;
            insn_cache->address = offset;
            insn_cache->size = (uint16_t)skipdata_bytes;
            memcpy(insn_cache->bytes, buffer, skipdata_bytes);
#ifdef CAPSTONE_DIET
            insn_cache->mnemonic[0] = '\0';
            insn_cache->op_str[0] = '\0';
#else
            strncpy(insn_cache->mnemonic, handle->skipdata_setup.mnemonic,
                    sizeof(insn_cache->mnemonic) - 1);
            skipdata_opstr(insn_cache->op_str, buffer, skipdata_bytes);
#endif
            insn_cache->detail = NULL;

            next_offset = skipdata_bytes;
        }

        // ■■■■■■■■■
        f++;

        // ■■■■■■■■
        c++;
        if (count > 0 && c == count)
            break;

        if (f == cache_size) {
            cache_size = cache_size * 8 / 5;
            total_size += (sizeof(cs_insn) * cache_size);
            tmp = cs_mem_realloc(total, total_size);
            if (tmp == NULL) {  //■■■■
                if (handle->detail) {
                    insn_cache = (cs_insn *)total;
```

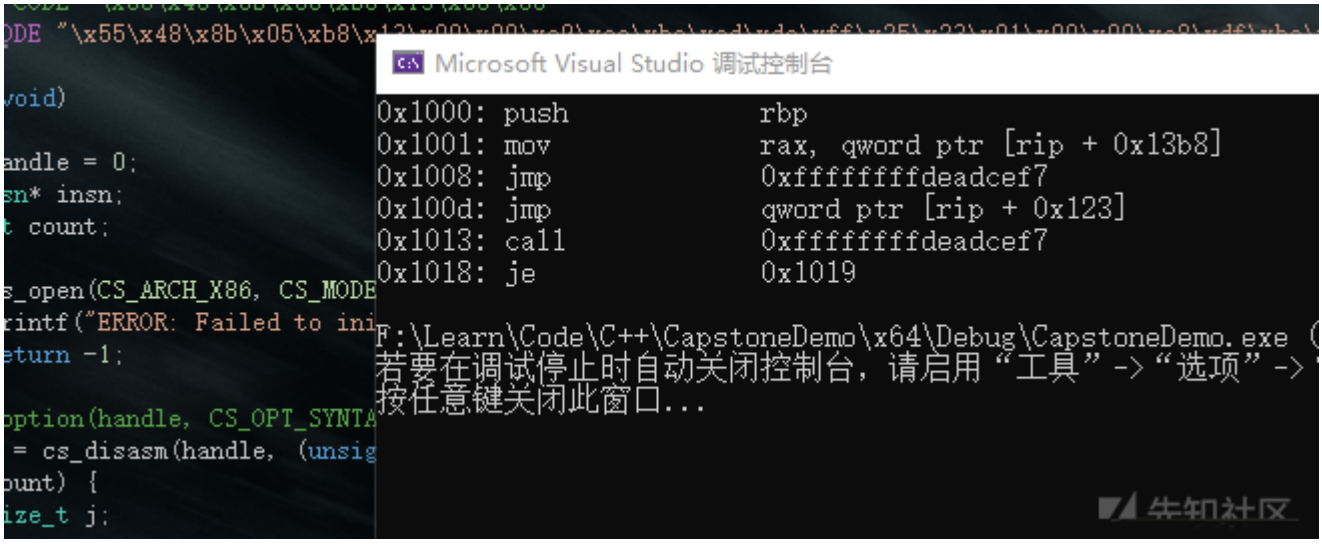
```
                for (i = 0; i < c; i++, insn_cache++)
                    cs_mem_free(insn_cache->detail);
            }

            cs_mem_free(total);
            *insn = NULL;
            handle->errnum = CS_ERR_MEM;
            return 0;
        }

        total = tmp;
        //■■■■■■■■■■■■■■■■■
        insn_cache = (cs_insn *)((char *)total + sizeof(cs_insn) * c);

        // ■f■■■0■■■■■■■■■■■
        f = 0;
    } else
        insn_cache++;

    buffer += next_offset;
    size -= next_offset;
    offset += next_offset;
}

if (!c) {
    //■■■■■■■■■
    cs_mem_free(total);
    total = NULL;
} else if (f != cache_size) {
    // ■■■■■■■■■■■■■■■■■■■
    tmp = cs_mem_realloc(total, total_size - (cache_size - f) * sizeof(*insn_cache));
    if (tmp == NULL) {  // ■■■■
        // ■■■■detail■■
        if (handle->detail) {
            insn_cache = (cs_insn *)total;
            for (i = 0; i < c; i++, insn_cache++)
                cs_mem_free(insn_cache->detail);
        }

        cs_mem_free(total);
        *insn = NULL;

        handle->errnum = CS_ERR_MEM;
        return 0;
    }

    total = tmp;
}

*insn = total;

return c;
}
```

示例，x86_64:

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

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

using namespace std;

#define CODE "\x55\x48\x8b\x05\xb8\x13\x00\x00\xe9\xea\xbe\xad\xde\xff\x25\x23\x01\x00\x00\xe8\xdf\xbe\xad\xde\x74\xff"

int main(void)
{
   csh handle = 0;
```

```
    cs_insn* insn;
    size_t count;

    if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle)) {
        printf("ERROR: Failed to initialize engine!\n");
        return -1;
    }

    count = cs_disasm(handle, (unsigned char*)CODE, sizeof(CODE) - 1, 0x1000, 0, &insn);   //[redacted]0x1000[redacted]insn
    if (count) {
        size_t j;

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

        cs_free(insn, count);
    }
    else
        printf("ERROR: Failed to disassemble given code!\n");

    cs_close(&handle);

    return 0;
}
```

输出



点击收藏 | 0 关注 | 1

上一篇：Linux Kernel Expl... 下一篇：QEMU虚拟化逃逸学习之：WCTF...

1. 2 条回复

x14m1 2019-08-02 11:34:44

```
771905C6        8975FC                          mov dword ptr [ebp
771905C9        EB0E                            jmp 0x771905d9
771905CB        33C0                            xor eax, eax
771905CD        40                              inc eax
771905CE        C3                              ret
771905CF        8B65E8                          mov esp, dword ptr
771905D2        C745FCFEFFFFFF                  mov dword ptr [ebp
771905D9        E81B22FBFF                      call 0x771427f9
771905DE        C3                              ret
                opcode                                  先知社区
```

```
//cs_insn中定义 uint16_t size;
for (uint16_t j = 0; j < 16; ++j)
{
if (j < ins[i].size)
// bytes 保存 OPCODE 编码
printf("%02X", ins[i].bytes[j]);
else
printf(" ");
}
```

0 回复Ta

kabeor 2019-08-02 15:06:45

@x14m1 输出opcode，感谢师傅

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