KCon洞见 2017未来

KCon

第五代加固技术 ARM代码虚拟化保护技术

陈愉鑫

目录

CONTENTS

PART 01 Android 平台加固技术概述

PART 02 什么是虚拟机保护技术

PART 03 VYD 指令集设计

PART 04 VYD 虚拟机设计

PART 05 VYD 编译器设计

PART 06 ARM VM的问题

KCon洞见 2017未来

知乎： 无名侠

专栏：大话二进制安全

知道创宇 IA 实验室，Android 病毒分析

i春秋，特种行业逆向分析线下讲师

KCon洞见
2017未来

我们为什么要加固？

1.保护核心代码，防止被逆向,泄密

2.防止营销作弊

3.防止代码被修改

4.........

加固技术的发展历史

第一代 自定义ClassLoader

第二代 核心封装到So库 / 方法抽取 / 反调试

第三代 ELF变形 / Ollvm 混淆　/ 多进程保护

第四代 DEX虚拟化保护

第五代 ARM 虚拟化保护

Thumb 指令

ARM 指令

x86 指令

MIPS 指令

编译器

字节码

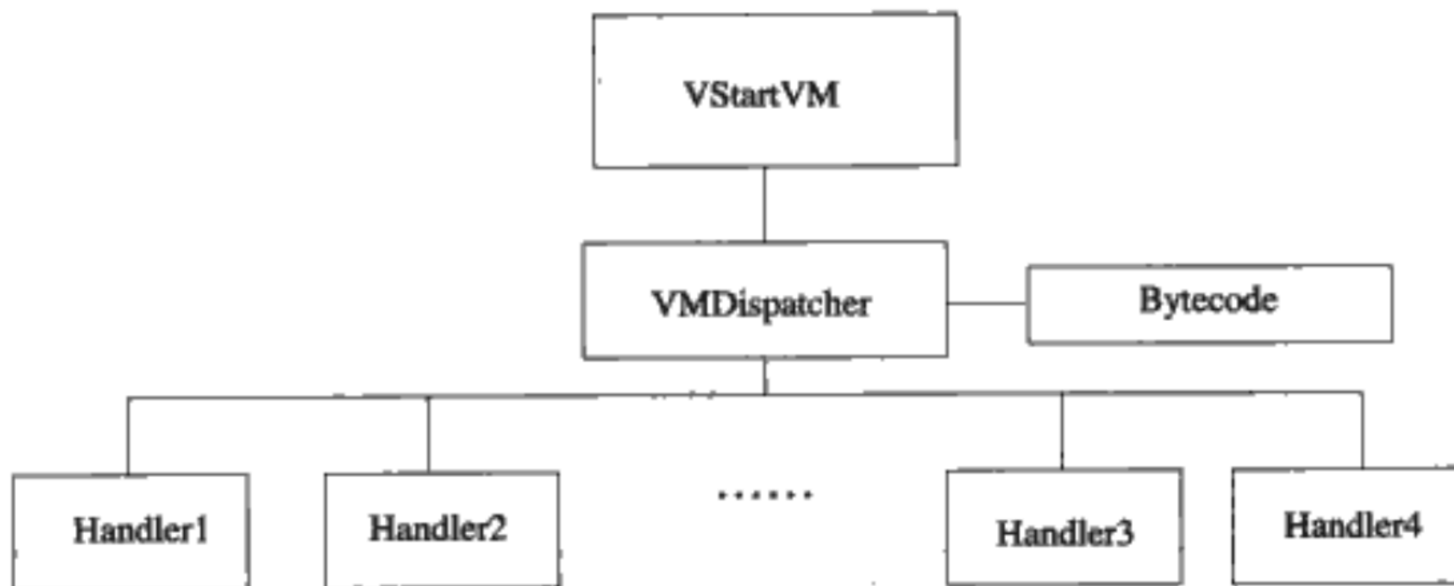
VM

```
Mov R1,#2
Mov R2,#3
Add R0,R1,R2
```

编译

```
vMov R1,2
vMov R2,3
vMov R20,R1
vAdd  R20,R2
vAdd  R0,R20
```

KCon 洞见未来 2017



出自《加密与解密》

KCon 2017 洞见未来

- 如何设计一个虚拟机？
- 用什么语言来开发虚拟机？
- 如何对编译后的ELF文件中指定函数进行虚拟化？
- 如何设计一个编译器？

KCon洞见 2017未来

PART 02 什么是虚拟化保护技术

几大模块：

VM 虚拟机核心
VM 编译器
VM 链接器
VM 各种stub

KCon洞见
2017未来

ARM 指令集的一些特性

1. 长度不一，Thumb / Thumb-2 / ARM
2. 条件执行
3. 多级流水线，Pc指向问题
4. 多寄存器寻址
5. 移位寻址

KCon 洞见 2017 未来

寄存器结构

VYD寄存器	ARM 对应寄存器	用途
vR0-vR12	R0-R12	通用寄存器
vFp	R11	栈帧寄存器
vSP	R13	栈指针寄存器
vLR	R14	链接寄存器
vPC1	R15	ARM指令同步PC
vPC	/	VYD 指令同步寄存器
v16-v32	/	临时寄存器
Flags	/	标志寄存器

VYD 虚拟机的指令编码格式

7	6 - 5	4 - 0
变长标记	位宽	opcode 5bit

指令中的寄存器:

7	6	5	0 - 5
变长标记	寄存器标记(1)	标志寄存器	reg id 5 bit

指令中的立即数

7	6	5	0 - 4
变长标记	立即数标记(0)	扩展立即数	The byte size of Imm

KCon洞见 2017未来

VYD 虚拟机的指令编码格式

```
1475    // MOV R1,1  1 00 01100   1 1 000001    00 000001    00000001   0x8C 0xC1 0x1 0x1
1476    // MOV R2,2  1 00 01100   1 1 000010    00 000001    00000010   0x8C 0xC2 0x1 0x2
1477    // MOV R3,R2 1 00 01100   1 1 000011    01 000010    ........   0x8C 0xC3 0x42
1478    // ADD R1,R2 1 00 00010   1 1 000001    01 000010    ........   0x82 0xC1 0x42
1479    // stop                                              ........   0XFF
1480    unsigned char opcodes[] = { 0x8c,0xc1,0x1,0x1,
1481                                0x8c,0xc2,0x1,0x2,
1482                                0x8C,0xC3,0x42,
1483                                0x82,0xc1,0x42,
1484                                0xff
1485    };
```

KCon 洞见未来 2017

寻址支持方式

1. 对寻址地址表达式进行编译

例如：

```
str fp, [sp, #-4]!
```

编译为：

```
vMov    r16,sp
vAdd    r16,-4
vStr    fp,[r16]
vMov    sp,r16
```

KCon洞见 2017未来

标志寄存器与逻辑判断

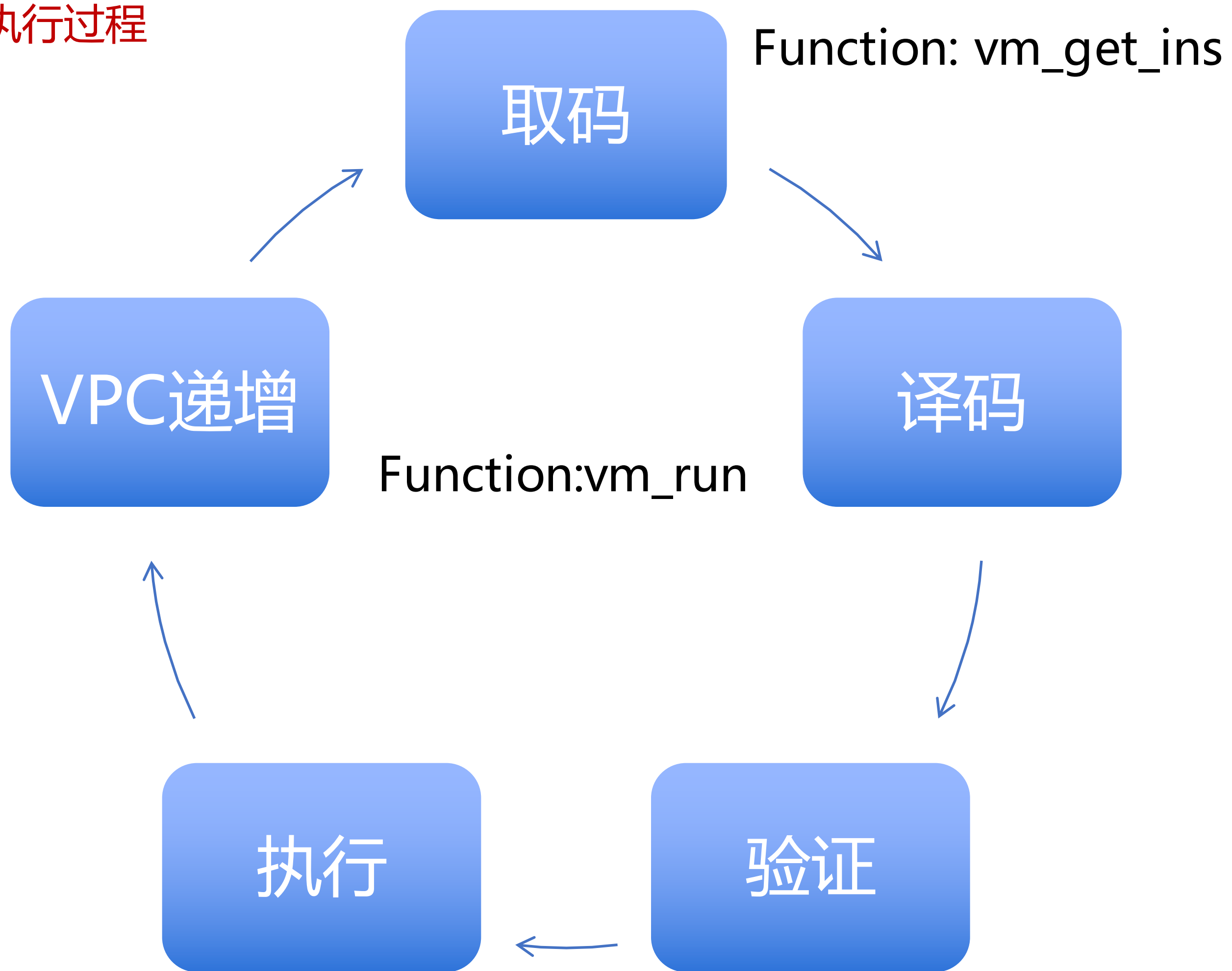
B类跳转指令：

vJmp cond，Offset/Rx

带条件的非B跳转指令 -> 编译 -> 多条VYD指令

```
moveq r0，r1   ->  vJne  s
                    vMov r0,r1
                  s: xxxxx
```

VYD 指令解码与执行过程



KCon洞见 2017未来

VYD 辅助解析指令

```
#define VM_EXTEND(x) (x & 0x80)  //字节是否有扩展
#define VM_GETW(x) ((x << 1)&0x60) //取OPCODE 位宽
#define VM_OPERAND_ISREG(x) (x & 0x40) //操作数是否为寄存器
#define VM_GETOPCODE(x)(x & 0x1f) //获取OPCODE
#define VM_GETREG(x) (x & 0x1f) // 获取REGId
#define VM_GETIMMSIZE(x) (x & 0x1f) //获取imm长度
#define VM_IMM_EXT(x) ( x & 0x20) //IMM后面是否有operand
#define VM_SET_FLAG(x) (x & 10) //是否影响标志寄存器
```

KCon洞见 2017未来

PART 04 VYD 虚拟机设计

VYD 虚拟机对于寄存器的定义

```
struct vm_context
{
    uint regs[REG_MAX];
    uint vPc;
    uint error_pc;
    vm_error_code error_code;
    char * bytecode;

    bool flag_z;
    bool flag_n;
    bool flag_c;
    bool flag_v;
};
```

```
enum vm_error_code
{
    VM_ERROR_UNDEF,
    VM_ERROR_DIV,
    VM_ERROR_REG,
    VM_ERROR_MEM
};
```

KCon 洞见 2017 未来

VYD 虚拟机的指令OPCODE

Opcode 为enum自动生成

```
enum vm_instruction_set
{ // The vm instructions set
    vPush = 0x0,
    vPop, vAdd, vSub, vMul, vDiv, vUNDEF, vUNDEF1, vCmp, vCall, vJmp, vXor, vMo
};
```

执行模型

1.Handler表
2.Switch

```
switch (ins->opcode)
{
case vMov:
case vAdd:
case vSub:
case vMul:
case vDiv:
case vXor:
case vOr:
case vAnd:
case vShr:
case vShl:
case vCmp:
    if (!vm_math_operand(vm, ins))
        return false;
    break;
case vLdr:
case vStr:
    if (!vm_mem_str_ldr(vm, ins))
        return false;
    break;
case vPush:
case vPop:
    if (!vm_stack_push_pop(vm, ins))
        return false;
    break;
case vJmp:
    vm_jmp(vm, ins);
    continue;
    break;
default:
    vm_set_except(vm, VM_ERROR_UNDEF);
    return false;
}
vm->vPc += ins->ins_size;
```

标志位的设置

```
void vm_add_with_carry(uint a,uint b,char carry1,uint  * result,char *carry_out,char *overflow_out)
{
    char carry = 0;
    char overflow = 0;
    long long result64,a64,b64;
    a64 = a;
    b64 = b;
    result64 = a64;
    result64 += b64;
    result64 += carry1;
    if(result64 & 0x100000000)
        carry = 1;
    uint unsigned_result = a+b+carry1;
    int signed_result = (int)a+(int)b+carry1;
    overflow = unsigned_result != signed_result;
    *result = unsigned_result;
    *carry_out = carry;
    *overflow_out = overflow;
}
void vm_set_result_flag(vm_context * vm, uint value)
{
    int signed_value = (int)value;

    assert(vm && vm->bytecode);
    vm->flag_z = value == 0; // Zero 标志位
    vm->flag_n = signed_value < 0; // 负数标志位
}
```

KCon洞见
2017未来

编译器工作流程：

1.反汇编ARM

2.生成中间代码

3.处理定位

4.生成opcode

如何反汇编arm？

Capstone 跨平台开源反汇编引擎

Capstone 支持：
Arm, *Arm64* (*Armv8*), *M68K*, *Mips*, *PowerPC*, *Sparc*, *SystemZ*, *TMS320C64X*, *XCore*& *X86* (include *X86_64*)

提供了多种语言的编程接口：
Clojure, *F#*, *Common Lisp*, *Visual Basic*, *PHP*, *PowerShell*, *Haskell*, *Perl*, *Python*, *Ruby*, *C#*, *NodeJS*, *Java*, *GO*, *C++*, *OCaml*, *Lua*, *Rust*, *Delphi*, *Free Pascal*

https://github.com/aquynh/capstone

brew install capstone
sudo apt-get install libcapstone3

知乎文章：**用Python玩玩反汇编**

KCon 2017 洞见未来

Capstone的强大之处(反汇编 + 分析)

```
0x80001000:        bl        #0x80000fbc
        op_count: 1
                operands[0].type:  IMM = 0x80000fbc
////////////////////////////////////////////////////////////////////////////////
0x80001004:        str       lr, [sp, #-4]!
        op_count: 2
                operands[0].type:  REG = lr
                operands[1].type:  MEM
                        operands[1].mem.base:  REG = sp
                        operands[1].mem.disp:  0xfffffffc
        Write-back: True
////////////////////////////////////////////////////////////////////////////////
0x80001008:        andeq     r0, r0, r0
        op_count: 3
                operands[0].type:  REG = r0
                operands[1].type:  REG = r0
                operands[2].type:  REG = r0
        Code condition: 1
```

编译成中间文本形式代码，便于调试

```
str fp, [sp, #-4]!
```

```
{"old asm": "0x0: str fp, [sp, #-4]!", "opcode": "!", "address": 0}
{"opcode": "vMov", "operands": [{"isReg": 1, "data": 16}, {"isReg": 1, "data": 13}]}
{"opcode": "vAdd", "operands": [{"isReg": 1, "data": 16}, {"isReg": 0, "value": -4}]}
{"opcode": "vMov", "operands": [{"isReg": 1, "data": 13}, {"isReg": 1, "data": 16}]}
{"opcode": "vStr", "operands": [{"isReg": 1, "data": 11}, {"isReg": 1, "data": 16}]}
```

最后一步，处理偏移量，并编译为opcode

解析json数据，按照指令格式进行生成指令

KCon 2017 洞见未来

进入虚拟机入口处理

1.保存上下文环境，同步至对应vm寄存器

2.重新分配运行堆栈

3.设置vm_run 参数

何时退出虚拟机？
Pc寄存器发生切换（切换范围不在vm内）

1.被虚拟化函数返回　（完全退出）
2.调用其它未虚拟化函数（临时退出）

完全退出虚拟机：
1.恢复上下文，
2.切换原始堆栈
2.跳转回ARM or thumb 代码

临时退出：
1.恢复上下文
2.切换原始堆栈
3.设置Lr寄存器为vm stub
4.设置vm stub 返回vm 信息

KCon洞见 2017未来

VM 的“链接器”

0.识别需要VM的函数并提取代码数据

1.将代码数据送入编译器编译

2.设置Vm入口Stub

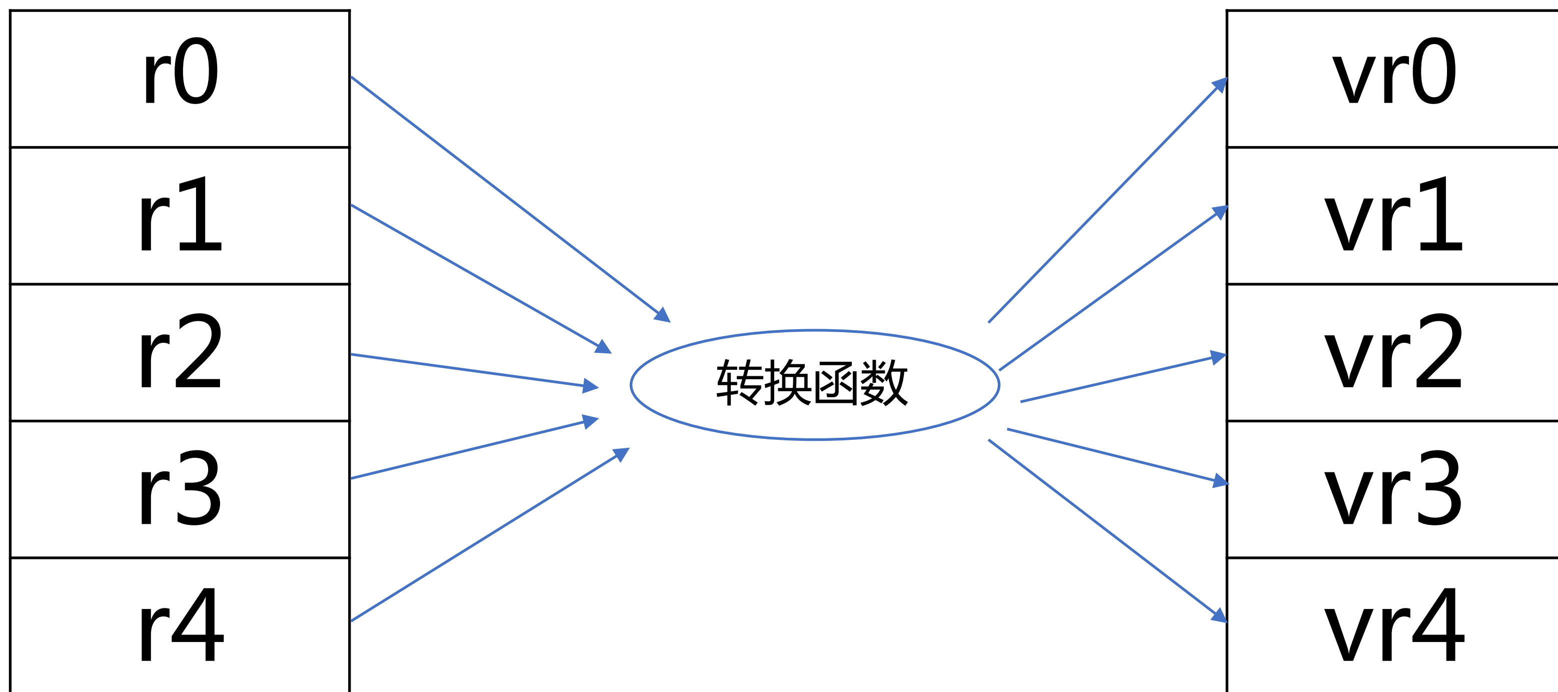
3.抽取vm 虚拟机elf的代码数据

4.嵌入目标elf中，修复重定位等各种细节

5.目标elf中添加opcode节表并映射

6..........

Vm加强方案 – 寄存器随机映射



KCon洞见未来 2017

VM 加强方案 – 字节码随机映射

动态生成一张map表

map<string,uint> opcodes;

vMov:xxxxx

Xxxx动态生成

KCon 洞见
2017 未来

THANK YOU

KCon 洞见
2017 未来