第四届全国网络与信息安全防护峰会

基于LLVM的移动应用软件安全保护方案

秦皓

猎豹移动

cheetahmobile

XDef2015 对话·交流·合作

个人介绍

- 姓名：秦皓
- ＩＤ ：QEver
- 工作：猎豹安全实验室 

- 职位：高级安全研究员
- 方向：移动端安全研究，主要包括
  应用加固，系统漏洞研究与挖掘等
- E-Mail: qinhao@cmcm.com / QEver.cn@gmail.com

提纲

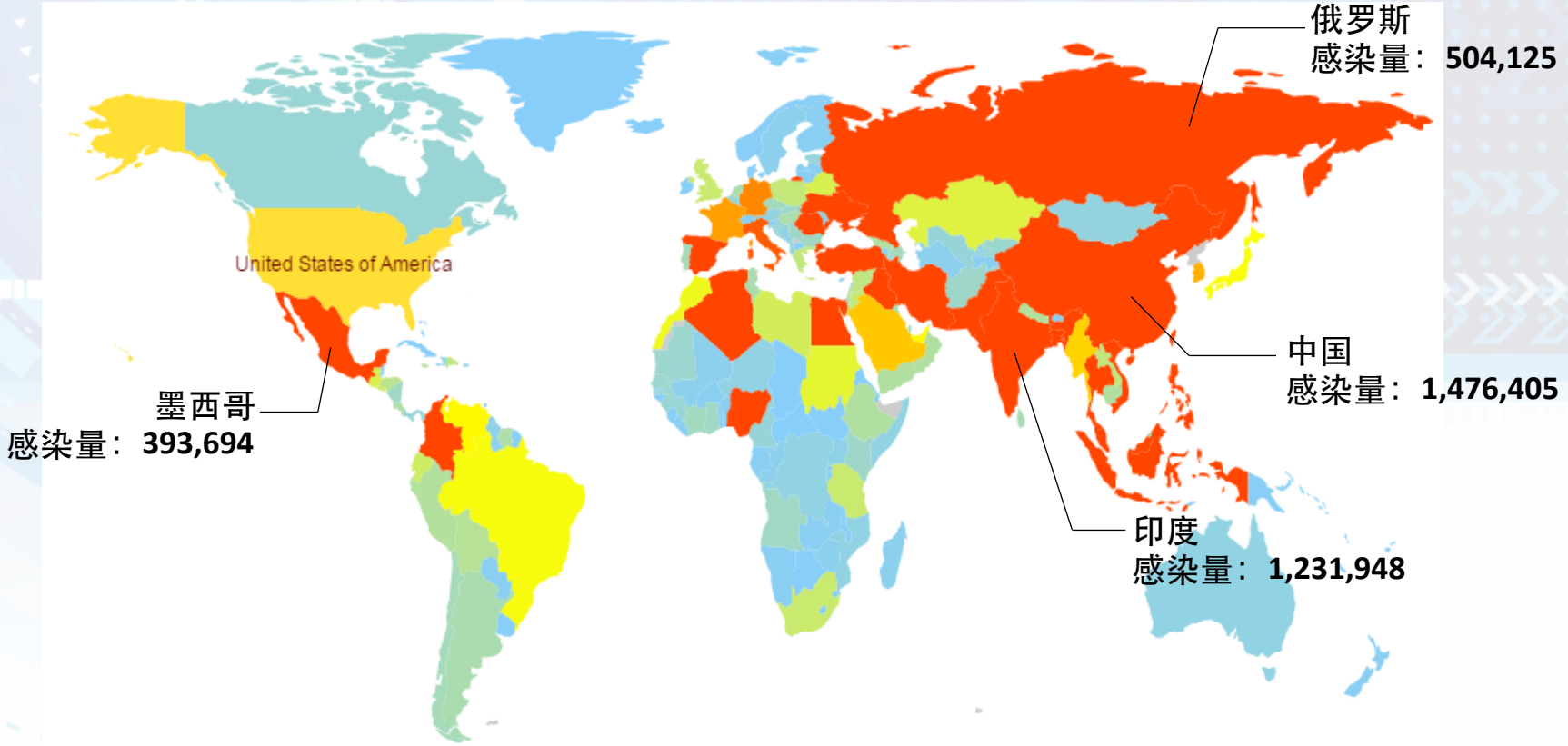
- Android软件防护加固现状
- ELK——基于LLVM的Android软件保护方案

Android软件防护加固现状

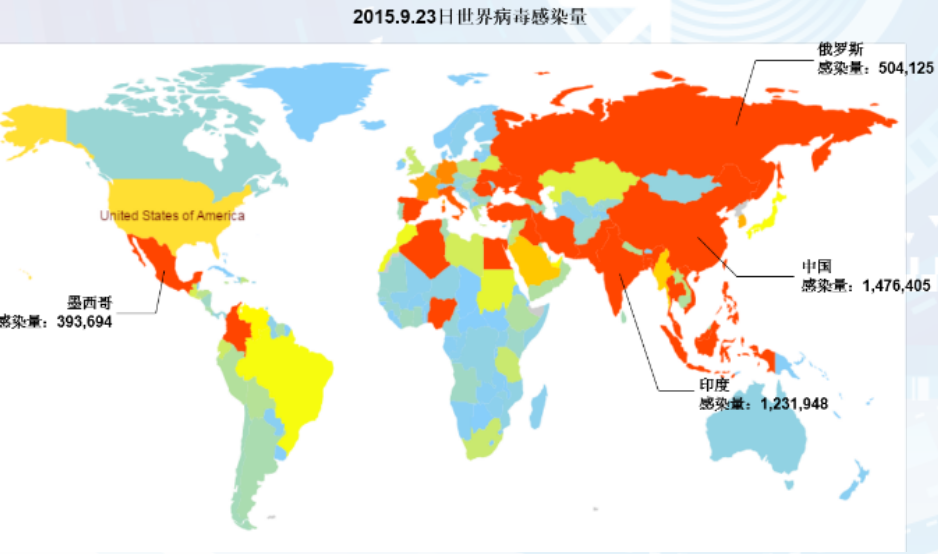
Android软件安全防护路在何方？

Android软件面临的安全威胁

2015.9.23日世界病毒感染量



Android软件面临的安全威胁



绝大多数是通过对正常软件**破解、二次打包**插入恶意代码造成的

Android软件面临的安全威胁

而**破解、二次打包**不仅仅可以插入恶意代码，还可以...

软件盗版，如植入广告、解除付费

数据篡改，如动态注入、发送假数据

软件山寨，危害开发者权益

Android软件防护加固现状

- Dex保护
  - 类加载

对完整的Dex文件进行加密，并放入资源文件中。

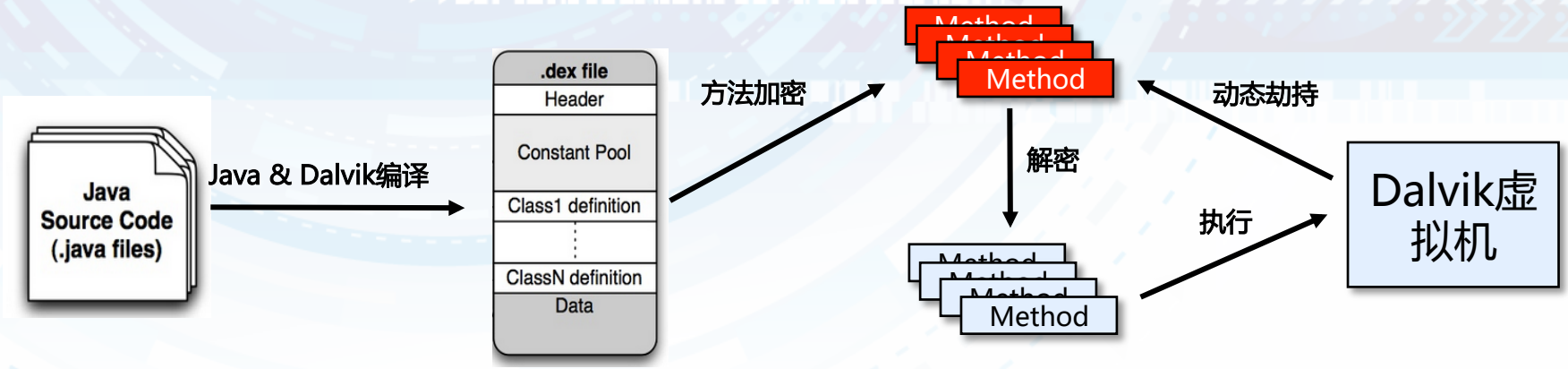
通过壳代码行进载入并解密运行。

Android软件防护加固现状

- Dex保护
  - 方法替换

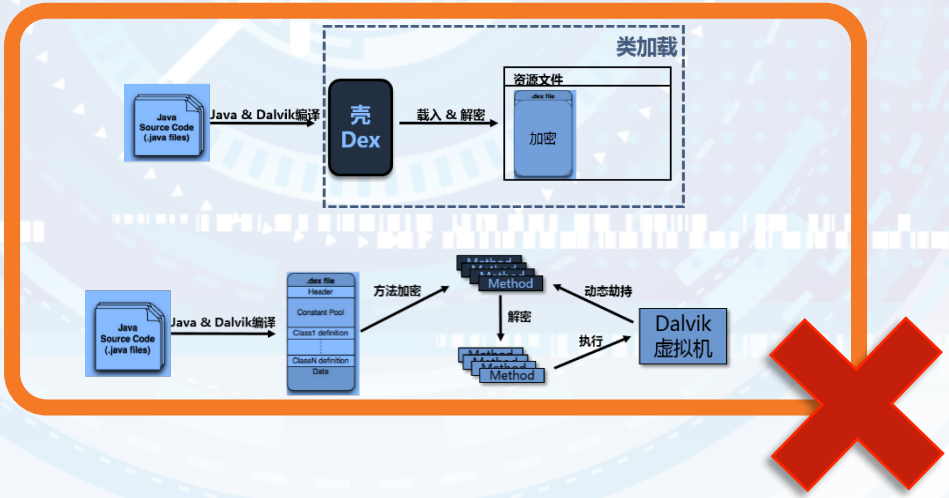
    仅提取dex文件中的所有方法进行加密。

    动态劫持虚拟机中的方法解析，将解密后的代码交给虚拟机执行。



Android软件防护加固现状

- Dex保护



1、通用脱壳机；
2、成熟的手工脱壳；

Android软件防护加固现状

- SO保护
  - 修改特殊文件或特殊标记值，阻止反编译。

    如修改ELF文件。

（1）使用程序段来描述ELF加载信息

程序段项	值
enum p_type32_e p_type	PT_NOTE (4)
Elf32_Off p_offset_FROM_FILE_BEGIN	33B45174h
Elf32_Addr p_vaddr_VIRTUAL_ADDRESS	0x7B4C4097
Elf32_Addr p_paddr_PHYSICAL_ADDRESS	0x0D7AB600
Elf32_Word p_filesz_SEGMENT_FILE_LENGTH	1788279023
Elf32_Word p_memsz_SEGMENT_RAM_LENGTH	72813938
enum p_flags32_e p_flags	PF_Read (4)
Elf32_Word p_align	1

（2）利用文件偏移和内存偏移的不同，隐藏文件信息

程序段项	值
enum p_type32_e p_type	PT_LOAD (1)
Elf32_Off p_offset_FROM_FILE_BEGIN	0h
Elf32_Addr p_vaddr_VIRTUAL_ADDRESS	0x00000000
Elf32_Addr p_paddr_PHYSICAL_ADDRESS	0x00000000
Elf32_Word p_filesz_SEGMENT_FILE_LENGTH	368640
Elf32_Word p_memsz_SEGMENT_RAM_LENGTH	368640
enum p_flags32_e p_flags	PF_Read_Write_Exec (7)
Elf32_Word p_align	4096
char p_data[368640]	0h

程序段项	值
enum p_type32_e p_type	PT_DYNAMIC (2)
Elf32_Off p_offset_FROM_FILE_BEGIN	47000h
Elf32_Addr p_vaddr_VIRTUAL_ADDRESS	0x00048000
Elf32_Addr p_paddr_PHYSICAL_ADDRESS	0x00048000
Elf32_Word p_filesz_SEGMENT_FILE_LENGTH	248
Elf32_Word p_memsz_SEGMENT_RAM_LENGTH	248
enum p_flags32_e p_flags	PF_Read_Write (6)
Elf32_Word p_align	4
char p_data[248]	47000h

Android软件防护加固现状

- SO保护
  - 修改特殊文件或特殊标记值，阻止反编译。

    如修改ELF文件。

（1）使用程序段来描述ELF加载信息

程序段项	值
enum p_type32_e p_type	PT_NOTE (4)
Elf32_Off p_offset_FROM_FILE_BEGIN	33B45174h
Elf32_Addr p_vaddr_VIRTUAL_ADDRESS	0x7B4C4097
Elf32_Addr p_paddr_PHYSICAL_ADDRESS	0x0D7AB600
Elf32_Word p_filesz_SEGMENT_FILE_LENGTH	1788279023
Elf32_Word p_memsz_SEGMENT_RAM_LENGTH	72813938
enum p_flags32_e p_flags	PF_Read (4)
Elf32_Word p_align	1

p_filesz_SEGMENT_FILE_LENGTH = 0

（2）利用文件偏移和内存偏移的不同，隐藏文件信息

程序段项	值
enum p_type32_e p_type	PT_LOAD (1)
Elf32_Off p_offset_FROM_FILE_BEGIN	0h
Elf32_Addr p_vaddr_VIRTUAL_ADDRESS	0x00000000
Elf32_Addr p_paddr_PHYSICAL_ADDRESS	0x00000000
Elf32_Word p_filesz_SEGMENT_FILE_LENGTH	368640
Elf32_Word p_memsz_SEGMENT_RAM_LENGTH	368640
enum p_flags32_e p_flags	PF_Read_Write_Exec (7)
Elf32_Word p_align	4096
char p_data[368640]	0h

程序段项	值
enum p_type32_e p_type	PT_DYNAMIC (2)
Elf32_Off p_offset_FROM_FILE_BEGIN	47000h
Elf32_Addr p_vaddr_VIRTUAL_ADDRESS	0x00048000
Elf32_Addr p_paddr_PHYSICAL_ADDRESS	0x00048000
Elf32_Word p_filesz_SEGMENT_FILE_LENGTH	248
Elf32_Word p_memsz_SEGMENT_RAM_LENGTH	248
enum p_flags32_e p_flags	PF_Read_Write (6)
Elf32_Word p_align	4
char p_data[248]	47000h

p_offset_FROM_FILE_BEGIN = p_vaddr_VIRTUAL_ADDRESS

Android软件防护加固现状

- SO保护
  - 修改特殊文件或特殊标记值，阻止反编译。

    如修改ELF文件。修改进程状态位。

（1）通过进程名，判定是否存在调试进程

```
if ( !strcmp((const char *)&v7, "./gdb")
    || !strcmp((const char *)&v7, "./gdbserver")
    || !strcmp((const char *)&v7, "./android_server")
    || strstr((const char *)&v7, "xposed") )
      exit(777);
```

（2）利用进程状态位，判断是否存在调试进程

```
Name:   pool-14-thread-
State:  S (sleeping)
Tgid:   7017
Pid:    7201
PPid:   176
TracerPid:      0
Uid:    10056   10056   10056   10056
Gid:    10056   10056   10056   10056
FDSize: 256
Groups: 1015 1028 3003 50056
VmPeak:   908532 kB
VmSize:   907772 kB
```

当该线程被调试时，值为T或t

当该线程被调试时，TracerPid

Android软件防护加固现状

- SO保护
  - 修改特殊文件或特殊标记值，阻止反编译。

    如修改ELF文件。修改进程状态位。

（1）通过进程名，判定是否存在调试进程

```
if ( !strcmp((const char *)&v7, "./gdb")
    || !strcmp((const char *)&v7, "./gdbserver")
    || !strcmp((const char *)&v7, "./android_server")
    || strstr((const char *)&v7, "xposed") )
      exit(777);
```

修改调试工具名

（2）利用进程状态位，判断是否存在调试进程

```
Name:   pool-14-thread-
State:  S (sleeping)
Tgid:   7017
Pid:    7201
PPid:   176
TracerPid:      0
Uid:    10056   10056   10056   10056
Gid:    10056   10056   10056   10056
FDSize: 256
Groups: 1015 1028 3003 50056
VmPeak:   908532 kB
VmSize:   907772 kB
```

当该线程被调试时，值为T或t

当该线程被调试时，TracerPid

定制ROM

Android软件防护加固现状

- 目前的加固方法，通过加密、隐藏等方式，来保护Dex文件
- 只要能够获得Dex文件，就能逆向得到Java代码，破解软件

Android软件防护加固现状

- 目前的加固方法，通过加密、隐藏等方式，来保护Dex文件
- 只要能够获得Dex文件，就能逆向得到Java代码，破解软件

Android软件安全保护**路在何方**？

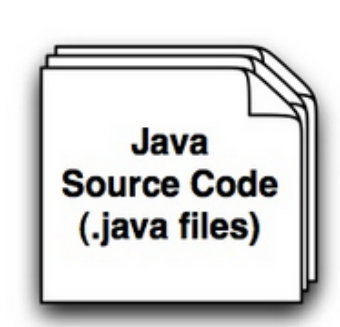
Android软件防护未来之路

- 目前的加固方法，通过加密、隐藏等方式，来保护Dex文件
- 只要能够获得Dex文件，就能逆向得到Java代码，破解软件

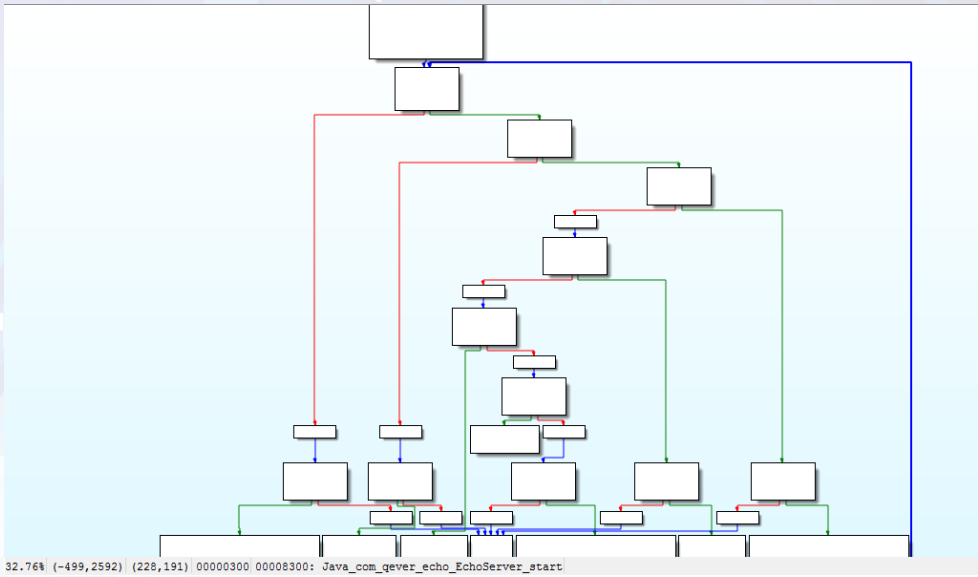
➡ Dex文件可逆，但Native方法**不可逆**

➡ 从 对Dex文件进行保护 转向 将Java方法编译成为Native方法

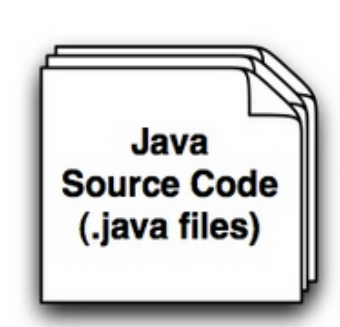
加固最终目的


Java
Source Code
(.java files)

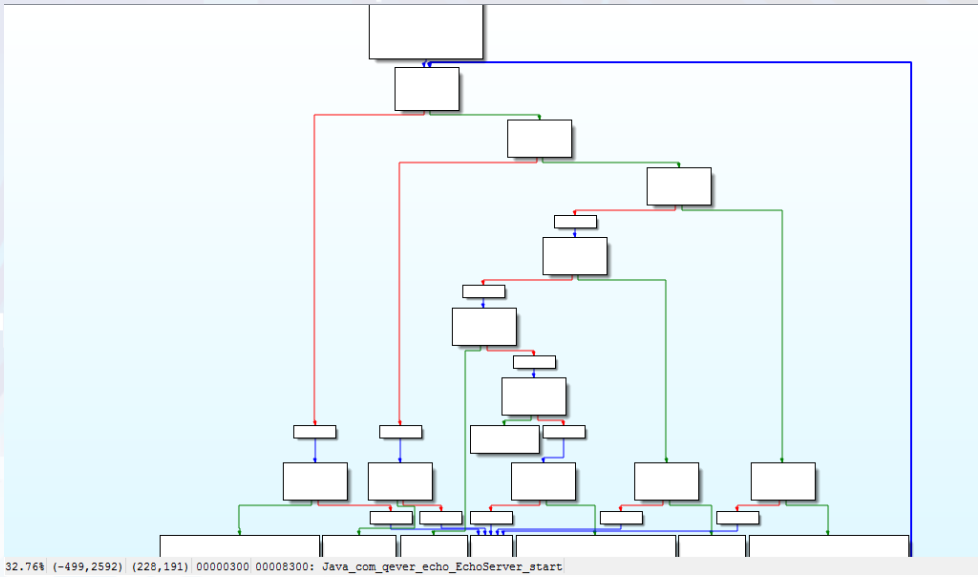
混淆加固


32.76% (-499,2592) (228,191) 00000300 00008300: Java_com_qever_echo_EchoServer_start

加固最终目的



混淆加固



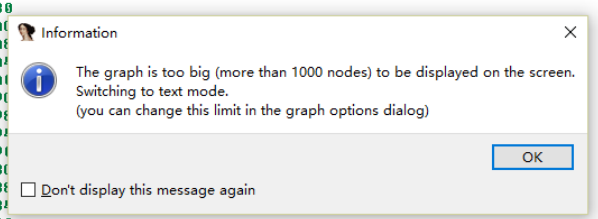
甚至，——

加固最终目的

甚至，可以这样 ——

混淆加固

```
.text:00008300
.text:00008300                 EXPORT Java_com_qever_echo_EchoServer_start
.text:00008300 Java_com_qever_echo_EchoServer_start
.text:00008300
.text:00008300 var_D0C         = -0xD0C
.text:00008300 var_D08         = -0xD08
.text:00008300 var_D04         = -0xD04
.text:00008300 var_D00         = -0xD00
.text:00008300 var_CFC         = -0xCFC
.text:00008300 var_BC          = -0xBC
.text:00008300 var_B8          = -0xB8
.text:00008300 var_B4          = -0xB4
.text:00008300 var_B0          = -0xB0
.text:00008300 var_AC          = -0xA
.text:00008300 var_A8          = -0xA
.text:00008300 var_A4          = -0xA
.text:00008300 var_A0          = -0xA
.text:00008300 var_9C          = -0x9
.text:00008300 var_98          = -0x9
.text:00008300 var_94          = -0x9
.text:00008300 var_90          = -0x9
.text:00008300 var_8C          = -0x8
.text:00008300 var_88          = -0x8
.text:00008300 var_84          = -0x8
.text:00008300 var_80          = -0x80
.text:00008300 var_7C          = -0x7C
.text:00008300 var_78          = -0x78
.text:00008300 var_74          = -0x74
.text:00008300 var_70          = -0x70
.text:00008300 var_6C          = -0x6C
.text:00008300 var_68          = -0x68
.text:00008300 var_64          = -0x64
.text:00008300 var_60          = -0x60
.text:00008300 var_5C          = -0x5C
```



ELK——基于LLVM的Android软件保护方案

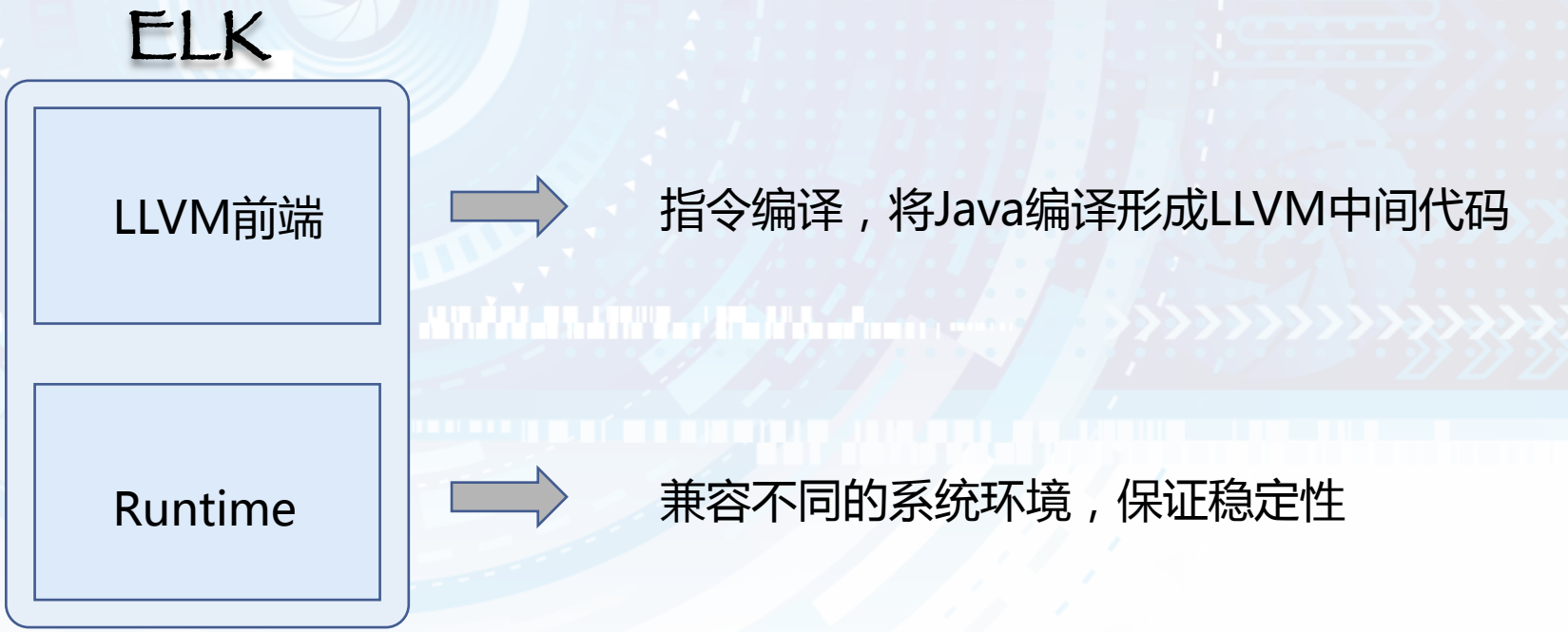
ELK —— 基于LLVM的Android软件保护方案

- 将Java方法直接编译成为Native方法，抹除Java方法的实现代码
- 利用LLVM成熟的代码混淆机制，对Native方法进行保护

cheetahmobile

XDef2015 对话·交流·合作

基于LLVM的Android软件保护方案



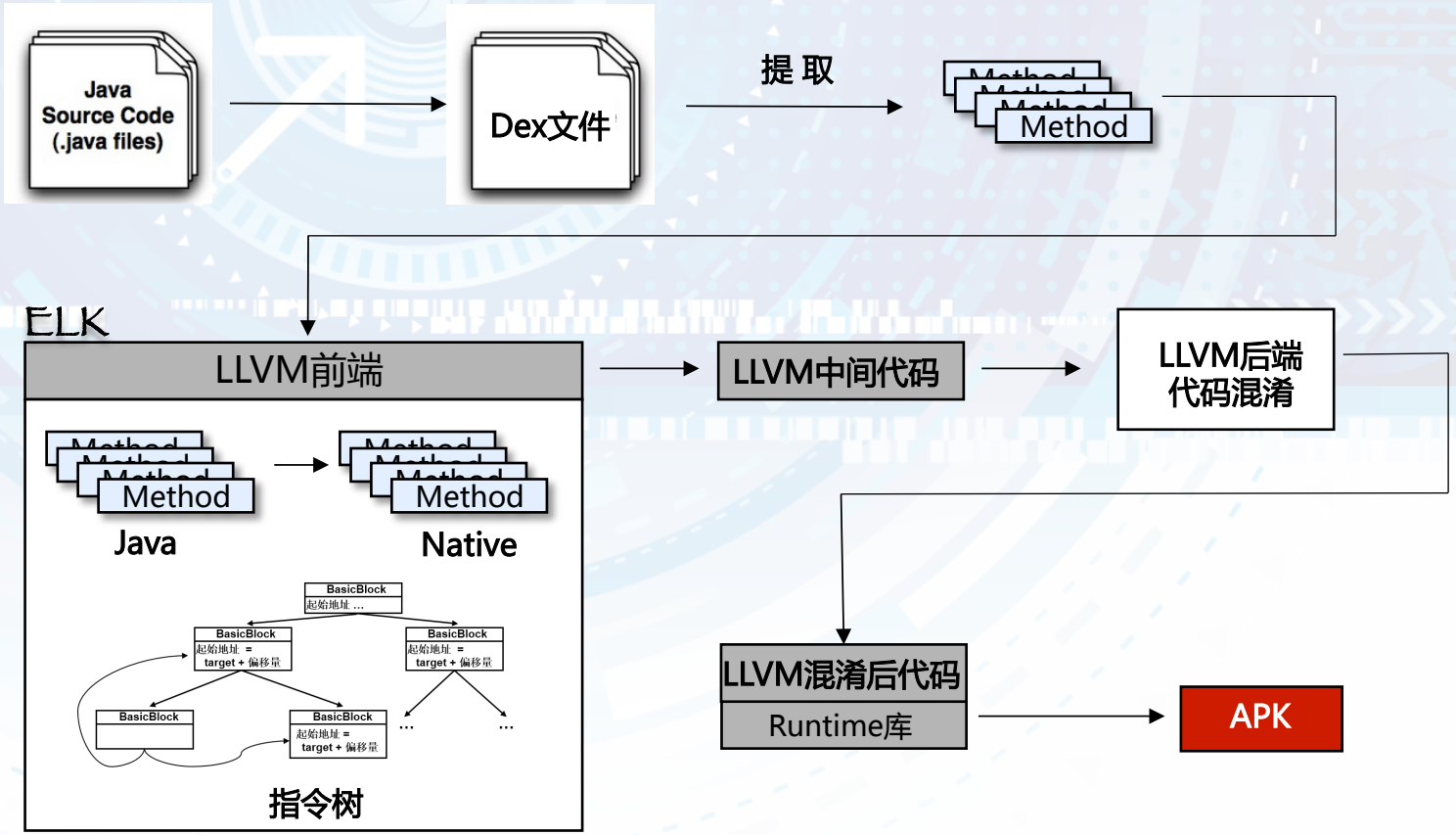
基于LLVM的Android软件保护方案

指令编译，将Java编译形成LLVM中间代码

兼容不同的系统环境，保证稳定性

基于LLVM的Android软件保护方案

- ELK前端 ----- 四步产生LLVM中间代码

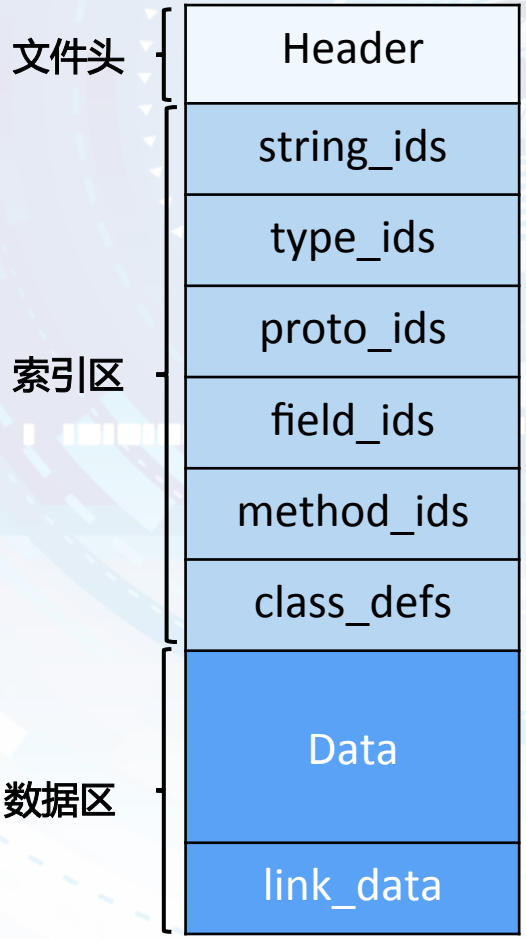


基于LLVM的Android软件保护方案

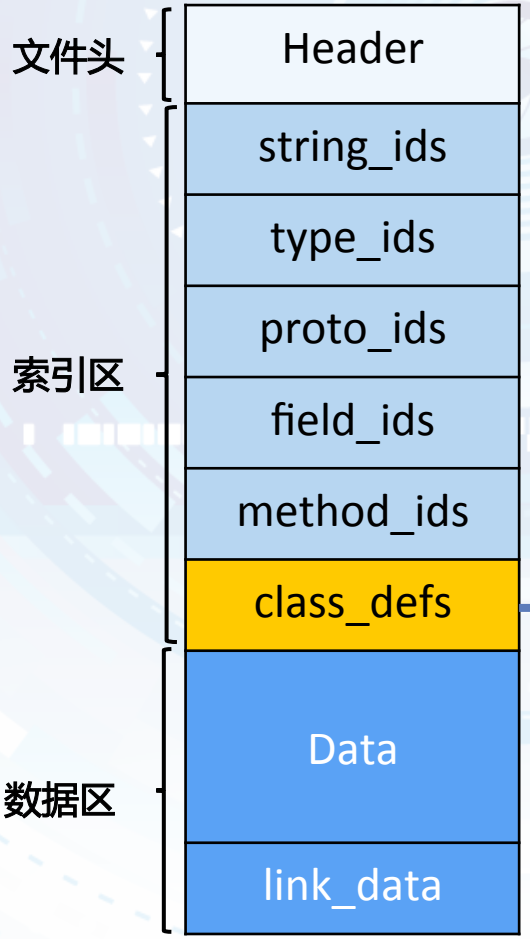
- 通过四步产生LLVM中间代码 ：

（1）Dex文件指令解析，提取方法；

Dex文件解析



Dex文件解析



提取DexCode

```
struct DexCode {
    u2  registersSize;
    u2  insSize;
    u2  outsSize;
    u2  triesSize;
    u4  debugInfoOff;       /* file offset to debug info stream */
    u4  insnsSize;          /* size of the insns array, in u2 units */
    u2  insns[1];
    /* followed by optional u2 padding */
    /* followed by try_item[triesSize] */
    /* followed by uleb128 handlersSize */
    /* followed by catch_handler_item[handlersSize] */
};
```

指向方法的字节码

基于LLVM的Android软件保护方案

- 通过四步产生LLVM中间代码 ：

（1）Dex文件指令解析，提取方法；

（2）生成JNI方法；

Dex方法 ➜JNI方法

Dex方法

- 常用Method表示方法：

Lpackage/name/ObjectName; → MethodName(III)Z

例如， Ljava/io/PrintStream; → println(Ljava/lang/String;)V

JNI方法

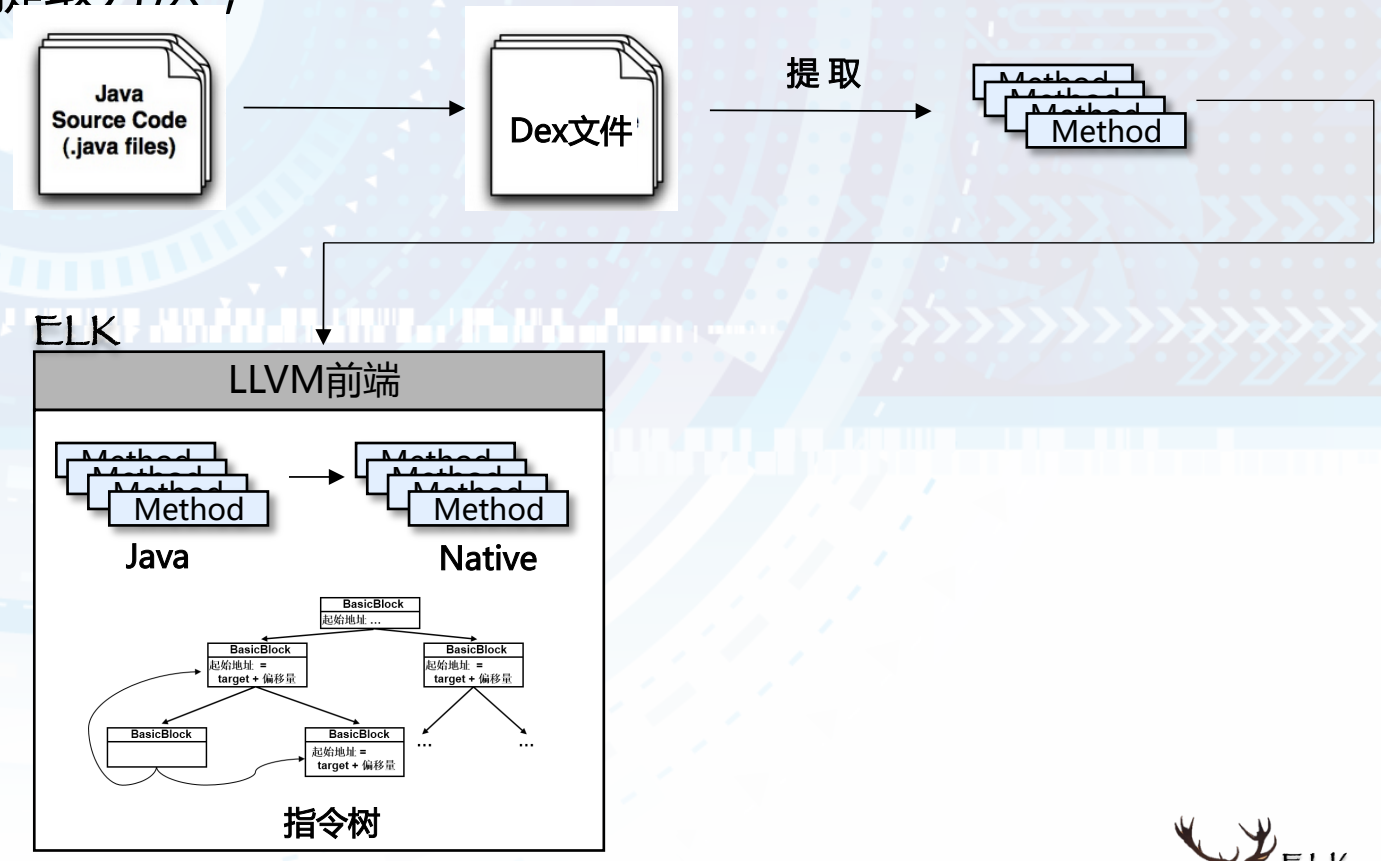
- 函数名：Java_包名_类名_方法名

例如， com.cm.ELKTest.method1 → Java_com_cm_ELKTest_method1

- 参数列表中增加JNIEnv* 和 jobject

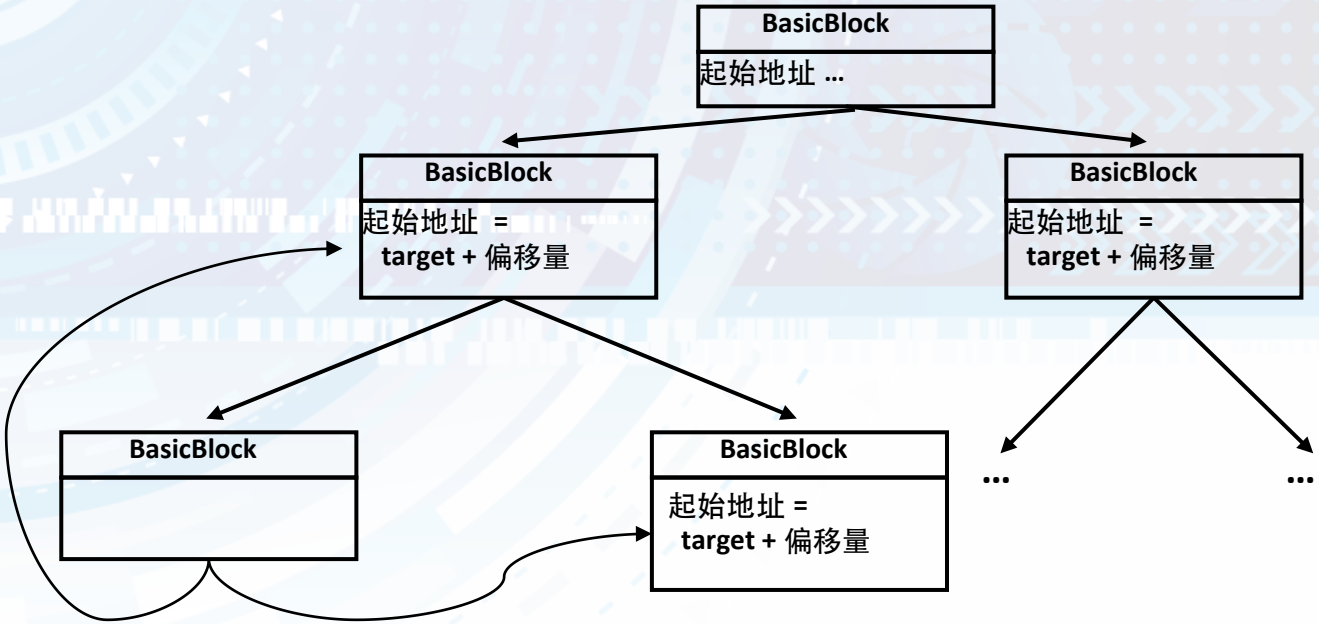
基于LLVM的Android软件保护方案

- 通过四步产生LLVM中间代码 ：
  （1）Dex文件指令解析，提取方法；
  （2）生成JNI方法；
  （3）构造指令树；



构造指令树

- 指令树的**节点**是BasicBlock；
- LLVM中将**顺序执行**的指令放在**一个BasicBlock**中；
- **跳转指令**决定指令树的分支。



跳转指令

Dex操作码跳转指令：

（1）直接跳转，如OP_GOTO；

操作数：1个。vA存放地址偏移。

```
case OP_GOTO:
case OP_GOTO_16:
case OP_GOTO_32:
    target += (int) dalvikInsn->vA;
    break;
```

跳转指令

Dex操作码跳转指令：

（1）直接跳转，如OP_GOTO；

操作数：1个。vA存放地址偏移。

（2）比较跳转，如OP_IF_EQ ；

操作数：3个。vA、vB存放比较值，vC存放地址偏移。

```
case OP_IF_EQ:
case OP_IF_NE:
case OP_IF_LT:
case OP_IF_GE:
case OP_IF_GT:
case OP_IF_LE:
    target += (int) dalvikInsn->vC;
    break;
```

跳转指令

Dex操作码跳转指令：

（1）直接跳转，如OP_GOTO，直接GOTO至指定语句；

操作数：1个。vA存放地址偏移。

（2）比较跳转，如OP_IF_EQ，比较是否相等 ；

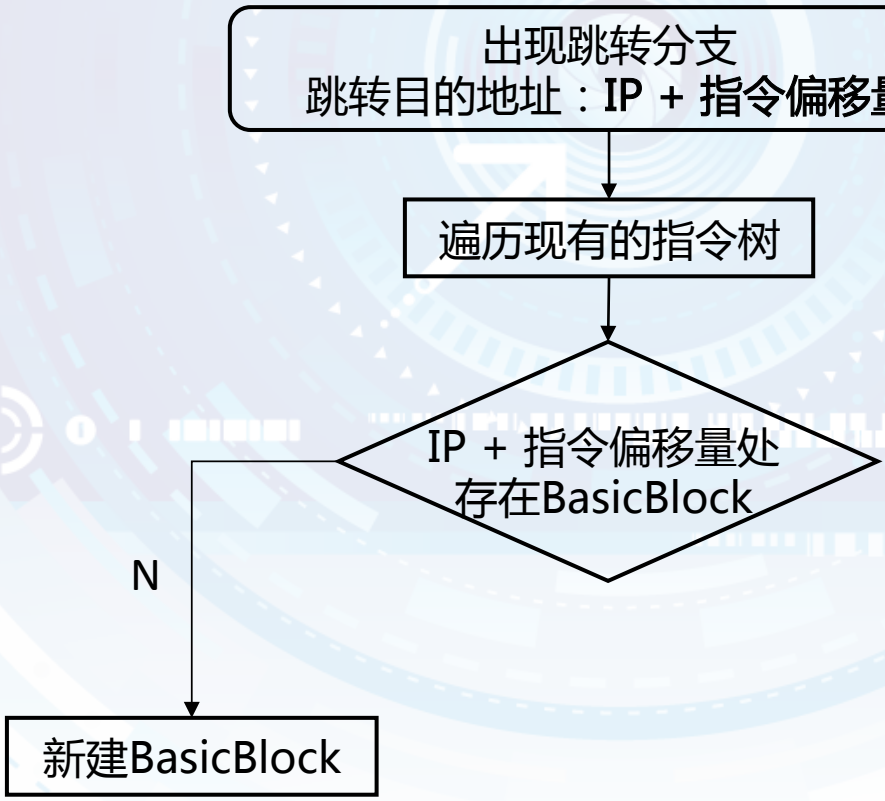
操作数：3个。vA、vB存放比较值，vC存放地址偏移。

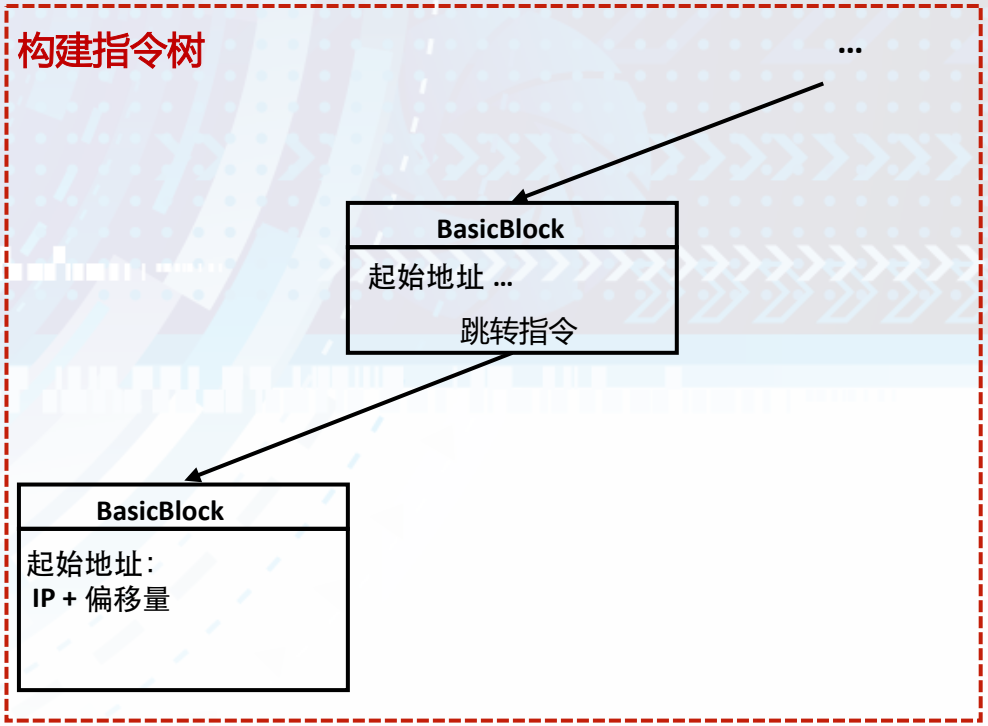
（3）0比跳转，如OP_IF_EQZ，比较是否等0 ；

操作数：2个。vA比较值，vB存放地址偏移。

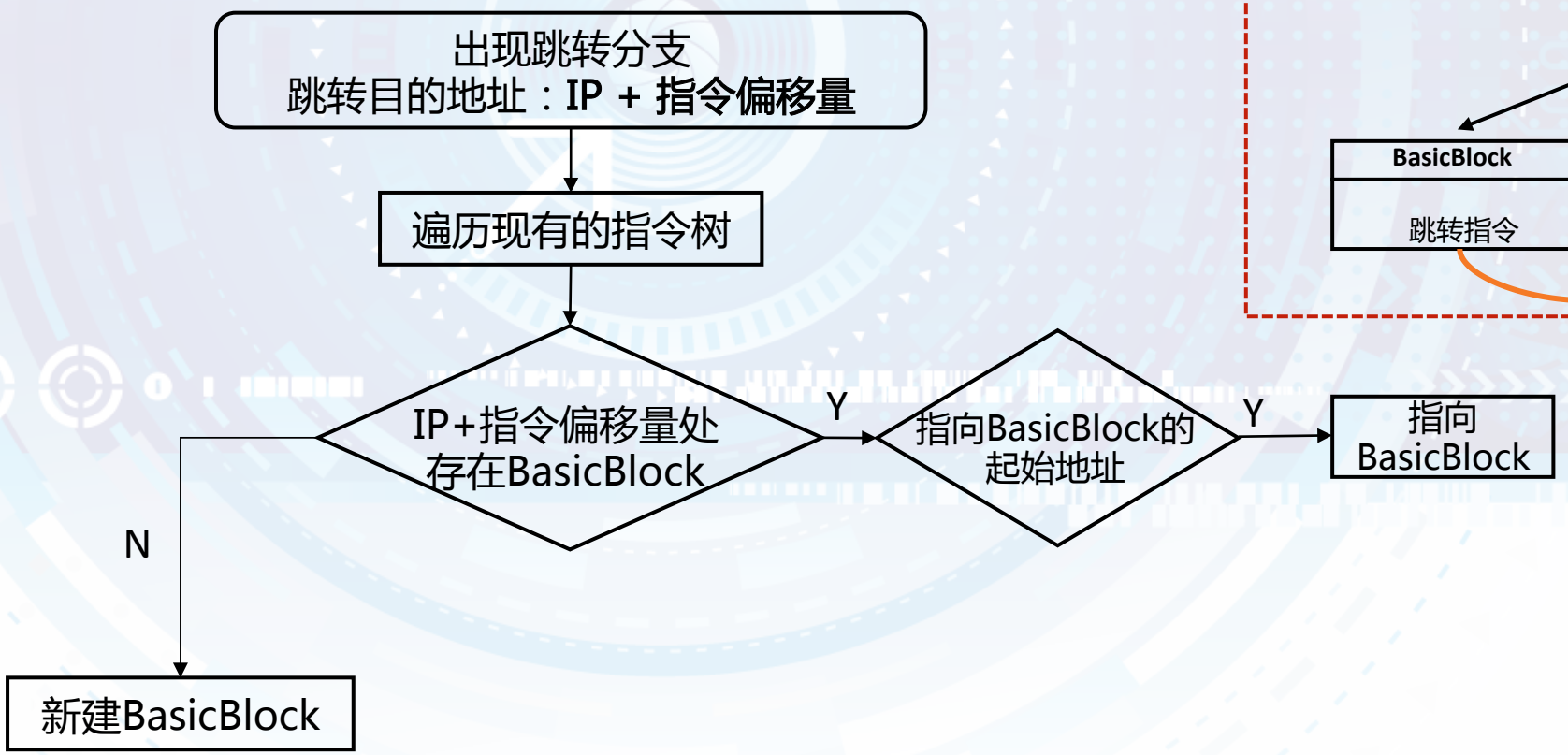
```
case OP_IF_EQZ:
case OP_IF_NEZ:
case OP_IF_LTZ:
case OP_IF_GEZ:
case OP_IF_GTZ:
case OP_IF_LEZ:
    target += (int) dalvikInsn->vB;
    break;
```

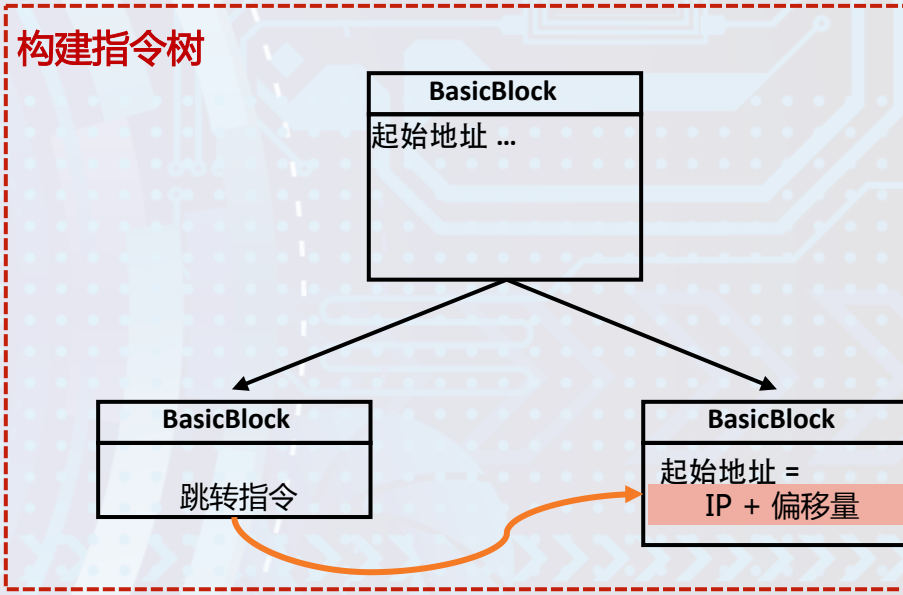
跳转处理


出现跳转分支
跳转目的地址：IP + 指令偏移量
遍历现有的指令树
IP + 指令偏移量处
存在BasicBlock
N
新建BasicBlock

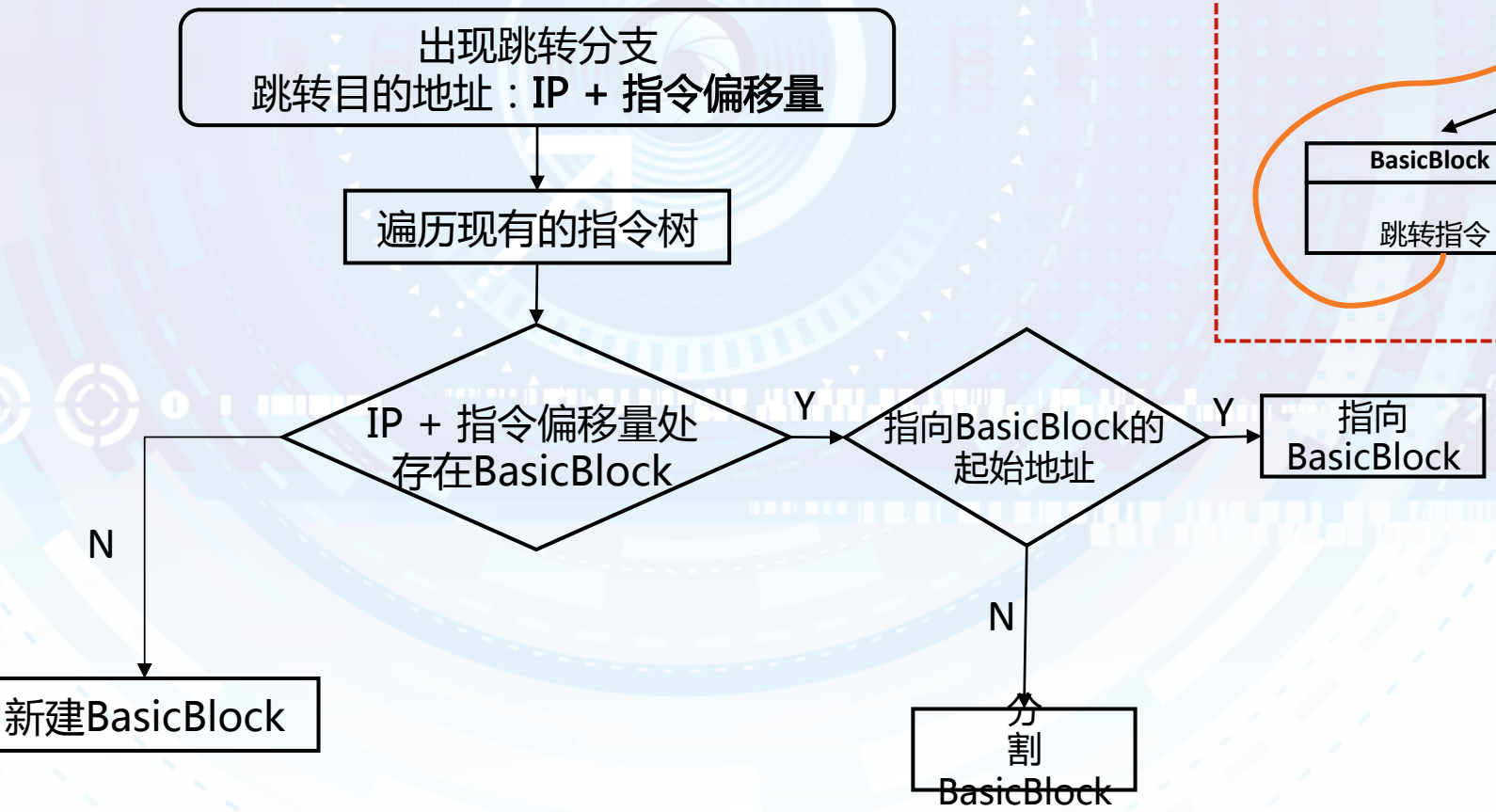

构建指令树
...
BasicBlock
起始地址 ...
跳转指令
BasicBlock
起始地址：
IP + 偏移量

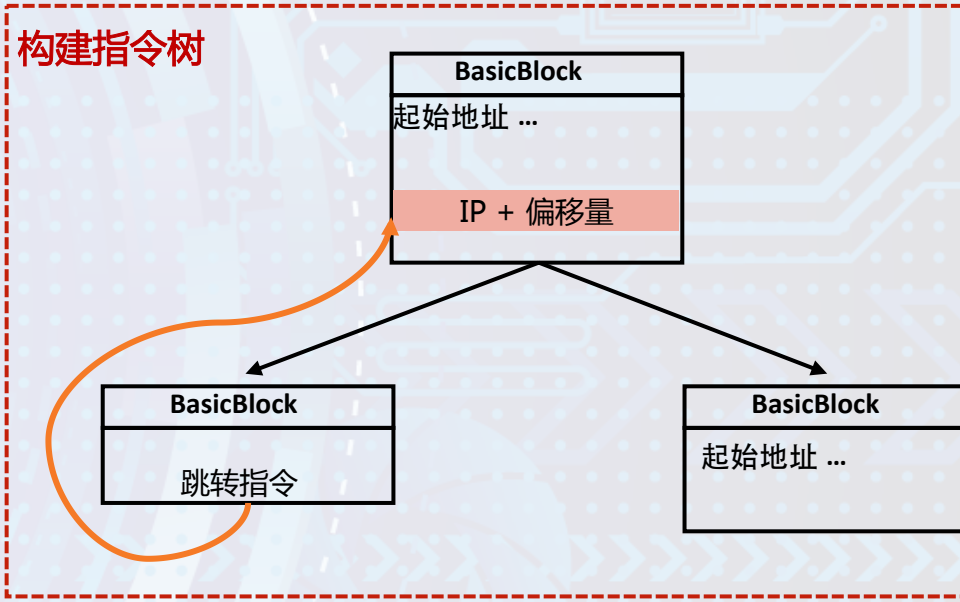
跳转处理


出现跳转分支
跳转目的地址：IP + 指令偏移量
遍历现有的指令树
IP+指令偏移量处
存在BasicBlock
Y
指向BasicBlock的
起始地址
Y
指向
BasicBlock
N
新建BasicBlock

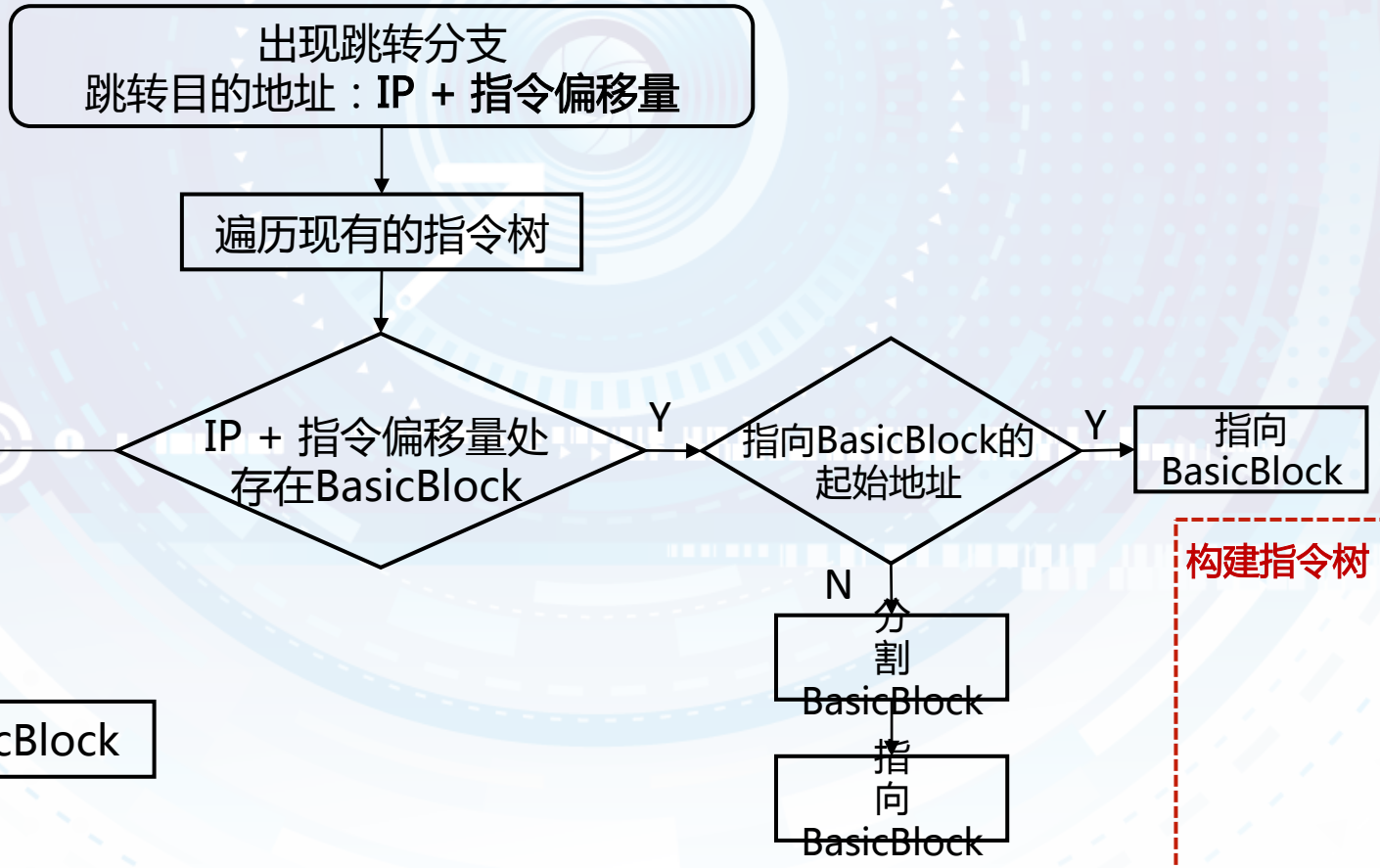

构建指令树
BasicBlock
起始地址 ...
BasicBlock
跳转指令
BasicBlock
起始地址 =
IP + 偏移量

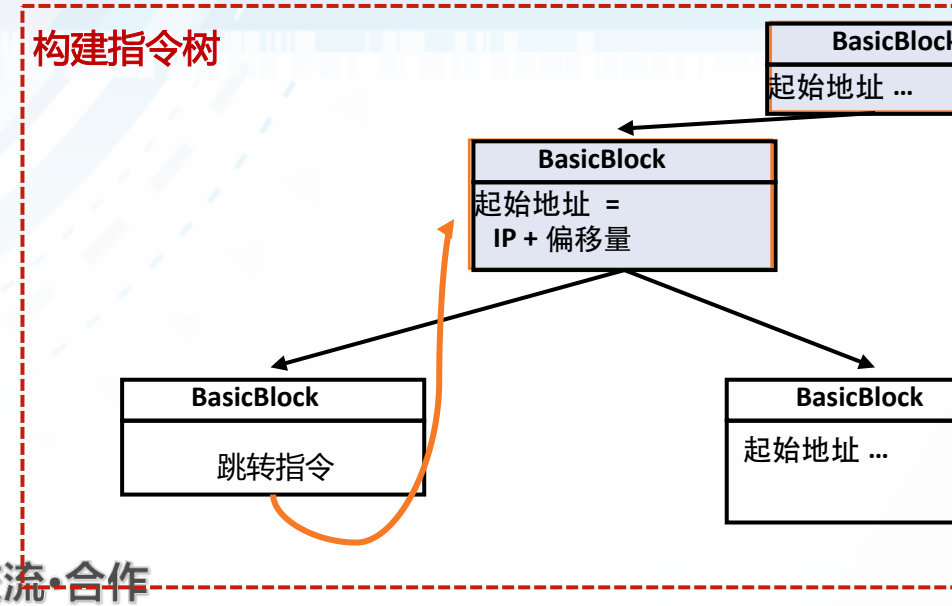
跳转处理


出现跳转分支
跳转目的地址：IP + 指令偏移量
遍历现有的指令树
IP + 指令偏移量处
存在BasicBlock
Y
指向BasicBlock的
起始地址
Y
指向
BasicBlock
N
新建BasicBlock
N
分
割
BasicBlock


构建指令树
BasicBlock
起始地址 ...
IP + 偏移量
BasicBlock
跳转指令
BasicBlock
起始地址 ...

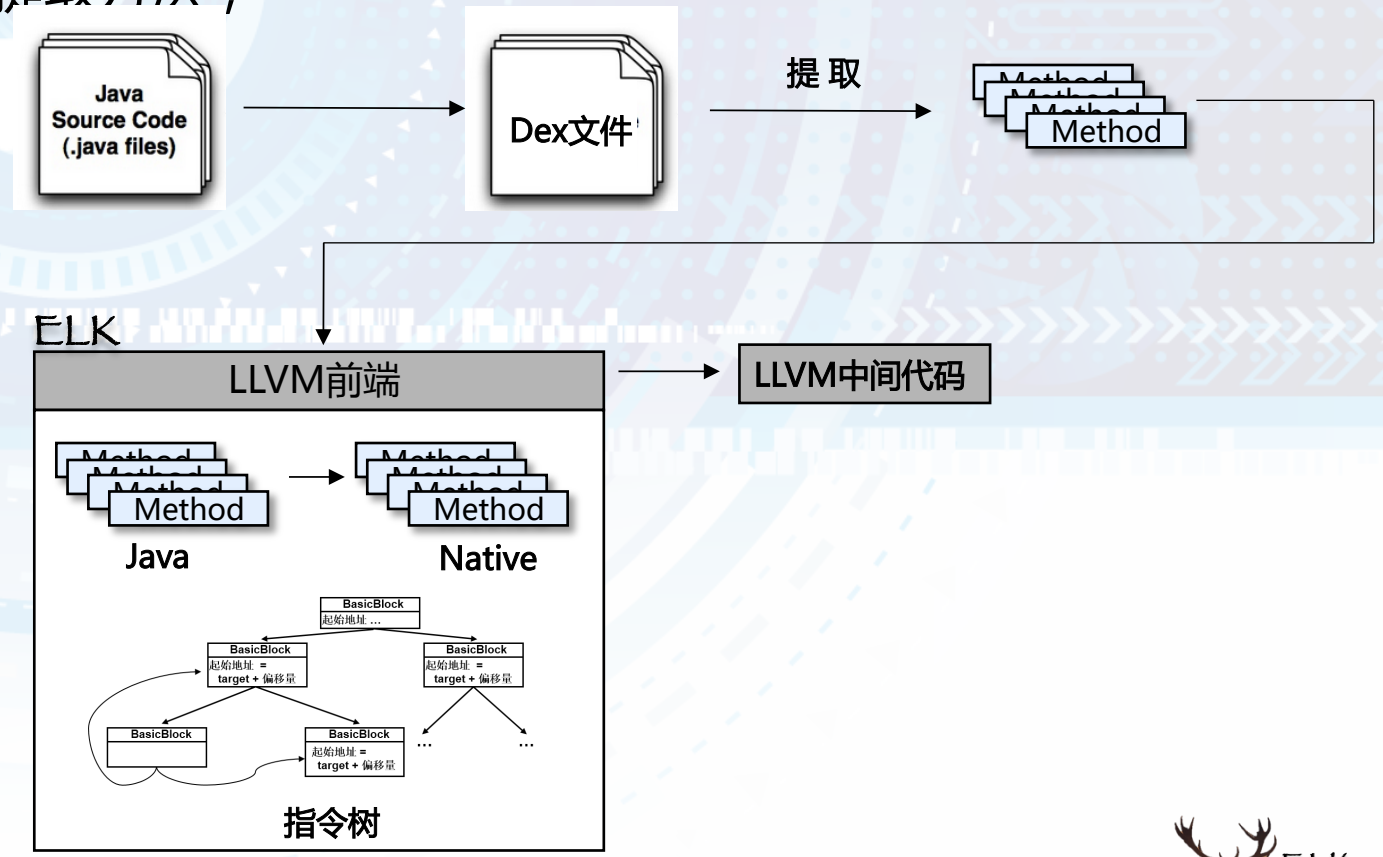
跳转处理


出现跳转分支
跳转目的地址：IP + 指令偏移量
遍历现有的指令树
IP + 指令偏移量处存在BasicBlock
Y
指向BasicBlock的起始地址
Y
指向BasicBlock
N
N
分割BasicBlock
指向BasicBlock
建BasicBlock


构建指令树
BasicBlock
起始地址 ...
BasicBlock
起始地址 =
IP + 偏移量
BasicBlock
跳转指令
BasicBlock
起始地址 ...

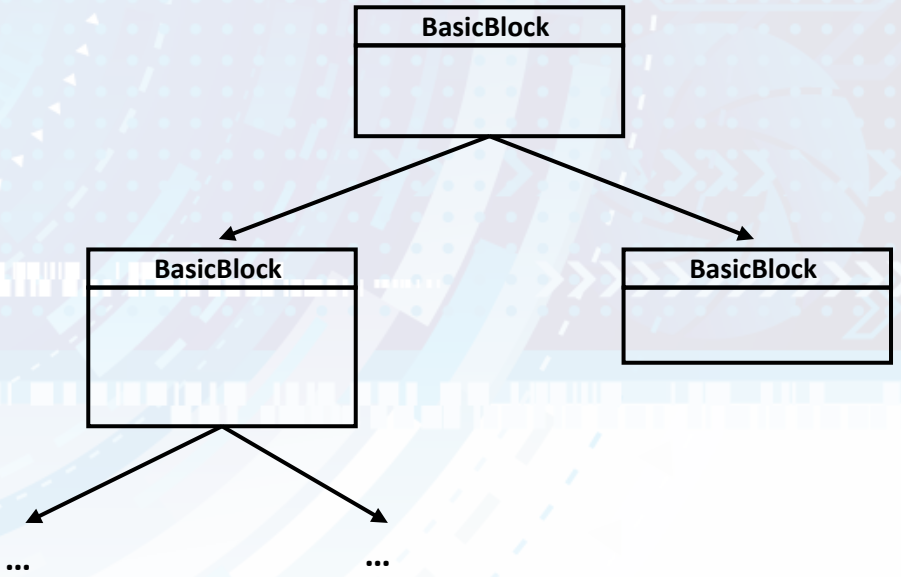
基于LLVM的Android软件保护方案

- 通过四步产生LLVM中间代码 ：
  （1）Dex文件指令解析，提取方法；
  （2）生成JNI方法；
  （3）构造指令树；
  （4）解析指令树。



解析BasicBlock

- 遍历指令树；
- 对每个BasicBlock逐一解析；
- 生成LLVM中间代码。



指令解析

- 数据操作指令
- 流程控制指令
- 系统相关指令

指令解析

- 数据操作指令
- 流程控制指令
- 系统相关指令

move: move vx, vy —— vy(4bits)的值赋给vx(4bits)

move/from16 vx, vy —— vy(8bits)的值赋给vx(16bits)

const: const/4 vA, #+B —— 将B(4bits)的值赋给VA(4bits)

const/high16 Vaa, #+BBBB0000 —— 将B(16-32bits)赋值给VAA（8bits）

add: add-int \ add-long \ add-float \ add-double

sub: sub-int \ sub-long \ sub-float \ sub-double

cmp*: cmpg-float vx, vy, vz —— 比较(float)vy, (float)vz，并设置v

指令解析

- 数据操作指令
- 流程控制指令
- 系统相关指令

goto +AA: 8bits 有符号分支偏移

goto/16 +AAAA: 16bits 有符号分支偏移

goto/32 +AAAAAAAA: 32bits 有符号分支偏移

if: if-eq / if-ne / if-lt / if-ge / if-gt / if-le / if-eqz / if-nez / if-ltz …

指令解析

- 数据操作指令
- 流程控制指令
- 系统相关指令
  - **instance:** instance-of vx, vy, type_id / new-instance vx, type
  - **array:** array-length vx, vy / new-array vx, vy, type_id …
  - **invoke:** invoke-virtual / invoke-super / invoke-direct …

指令格式

- 对每个指令，可通过dexDecodeInstruction()来获取其格式分类。

kFmt10x : op

kFmt12x : op vA, vB

kFmt11n : op vA, #+B

kFmt11x : op vAA

指令格式

- 对每个指令，可通过dexDecodeInstruction()来获取其格式分类。

kFmt10x : op

kFmt12x : op vA, vB

kFmt11n : op vA, #+B

kFmt11x : op vAA

- 根据指令格式进行分类处理。

基于LLVM的Android软件保护方案

指令编译，将Java编译形成LLVM中间代码

兼容不同的系统环境，保证稳定性

Runtime库

- 高稳定性 & 广适配性 面临的挑战：
  - Android系统碎片化；
  - 各种定制ROM；
  - 差异化的系统接口；
  - ...

Runtime库

- 高稳定性 & 广适配性 面临的挑战：
  - Android系统碎片化；
  - 各种定制ROM；
  - 差异化的系统接口；
  - ...

Runtime运行时库
适配不同的系统环境

Runtime库解析系统指令

- 系统相关指令

instance: instance-of vx, vy, type_id / new-instance vx, type

array: array-length vx, vy / new-array vx, vy, type_id …

invoke: invoke-virtual / invoke-super / invoke-direct …

Runtime库解析系统指令

- 系统相关指令

instance: instance-of vx, vy, type_id / new-instance vx, type

array: array-length vx, vy / new-array vx, vy, type_id …

invoke: invoke-virtual / invoke-super / invoke-direct …

Runtime库解析系统指令

- 系统相关指令

instance: instance-of vx, vy, type_id / new-instance vx, type

array: array-length vx, vy / new-array vx, vy, type_id …

invoke: invoke-virtual / invoke-super / invoke-direct …

Runtime库解析系统指令

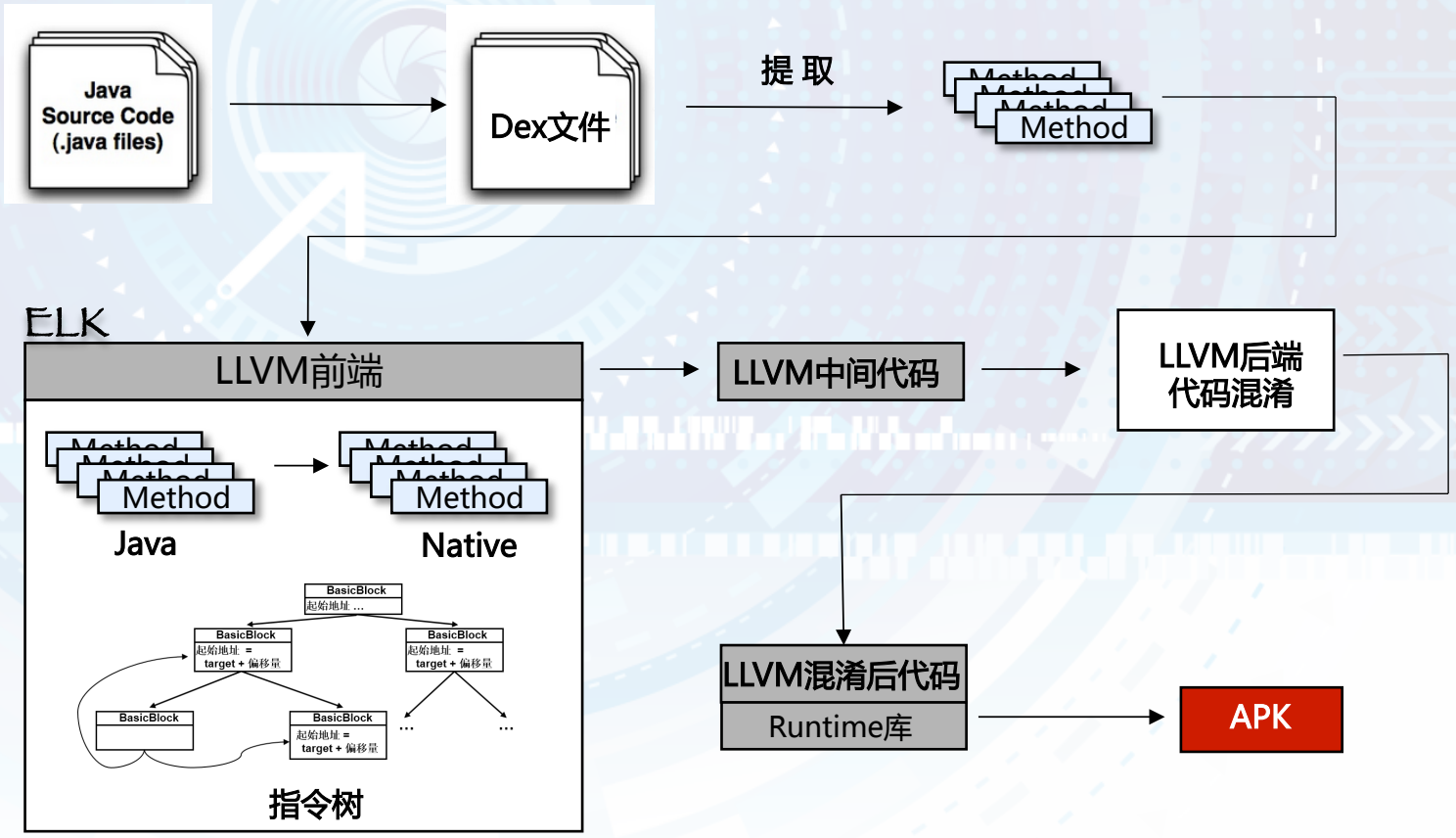
- 系统相关指令

instance: instance-of vx, vy, type_id / new-instance vx, type

array: array-length vx, vy / new-array vx, vy, type_id …

invoke: invoke-virtual / invoke-super / invoke-direct …

基于LLVM的Android软件保护方案



总结

- 当前Android软件加固现状
  - 加固需要深入底层
- ELK --- 基于LLVM，Java方法 → Native方法
  - 抵制目前的破解
  - 生成LLVM中间语言之后，可以利用成熟的LLVM代码混淆技术
  - 稳定性高，适用性广

感谢您的关注！

Thank you for your attention