DATACON 2020 僵尸网络方向答辩

答辩队伍

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contents

难点与挑战

1. 现有工作的不足：

- 传统方法对于同源跨架构二进制代码分析精度低：
  - 例如 CG，CFG，ACFG 等控制流相关特性
- 依赖深度网络的方法对样本要求高：
  - 需要大量有标签的样本
  - 需要样本具有代表性与普遍性

少量样本下的较高精度的特征工程？

难点与挑战

2. 数据集的特征：

- Rodata 段被抹去，符号表被 strip
  - 特征字符串和符号信息等直接信息缺失
- 编译器之间的差异
  - Function prologue和function epilogue存在差异
  - 内联函数的处理不同
- 一些源码中仅存在字符串长度的差异
  - 编译得到的binary控制流相似度高

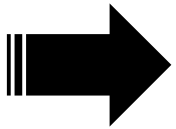
有限信息和噪声下的有效特征选择？

Contents

关键技术——用户定义函数识别

静态链接中

大量libc函数



噪声遮盖用户源代码特征

借助编译后用户定义**函数地址连续性**，筛出用户定义函数

其他	start	用户定义函数	fcntl	其他

借助**函数签名**进行libc函数识别，剔除libc函数

关键技术

青

大

借助编

其他

借助函

```
; Attributes: info_from_lumina

; int __cdecl fcntl(int fd, int cmd, struct flock *lock, char)
fcntl proc near

var_C= dword ptr -0Ch
fd= dword ptr  4
cmd= dword ptr  8
lock= dword ptr  0Ch
arg_C= byte ptr  10h

push    edi             ; Alternative name is '__GI___libc_fcntl'
push    ebx
sub     esp, 14h
mov     ecx, [esp+1Ch+cmd] ; cmd
lea     eax, [esp+1Ch+arg_C]
mov     [esp+1Ch+var_C], eax
mov     edi, [esp+1Ch+fd]
mov     edx, [esp+1Ch+lock] ; lock
lea     eax, [ecx-0Ch]
cmp     eax, 2
ja      short loc_804F603
```

```
loc_804F603:
push    ebx
mov     ebx, edi        ; fd
mov     eax, 37h ; '7'
int     80h             ; LINUX - sys_fcntl
pop     ebx
cmp     eax, 0FFFFF000h
mov     ebx, eax
jbe     short loc_804F623
```

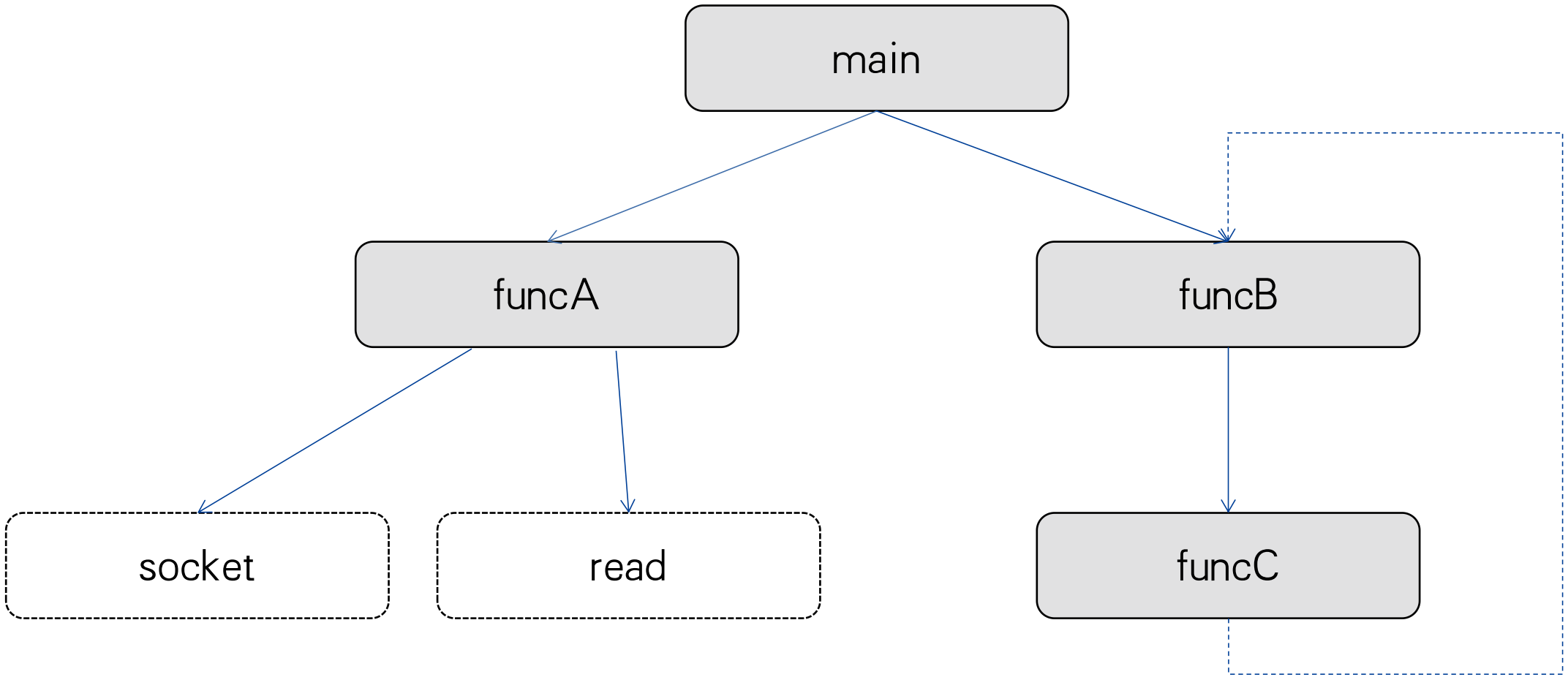
```
fcntl:

var_10= -0x10
var_8= -8
var_s0=  0
var_s4=  4
arg_8=  0x10
arg_C=  0x14

li      $gp, (off_4442F0+0x7FF0 - .)
addu    $gp, $t9
addiu   $sp, -0x28
sw      $ra, 0x20+var_s4($sp)
sw      $s0, 0x20+var_s0($sp)
sw      $gp, 0x20+var_10($sp)
addiu   $v1, $a1, -0x21
addiu   $v0, $sp, 0x20+arg_C
sltiu   $v1, 3
la      $t9, sub_402430
sw      $v0, 0x20+var_8($sp)
sw      $a3, 0x20+arg_C($sp)
beqz    $v1, loc_4023F0
sw      $a2, 0x20+arg_8($sp)
```

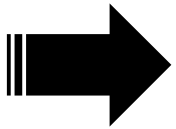
```
loc_4023F0:
li      $v0, 0xFD7
syscall
la      $t9, sub_4006B0
beqz    $a3, loc_402418
move    $s0, $v0
```

关键技术——用户定义函数Call Graph的建立



关键技术——rodata段的恢复

部分binary中仅字符串长度存在差异



代码段无法体现完整特征

遍历sections，找到被抹掉的rodata段的范围

LOAD	00400000	00400094
.init	00400094	00400120
.text	00400120	00403EB0
.fini	00403EB0	00403F0C
.ctors	00444000	00444008
.dtors	00444008	00444010
LOAD	00444010	00444020
.data	00444020	004442F0
.got	004442F0	004444E0
.sbss	004444E0	004444E8
LOAD	004444E8	004444F0
.bss	004444F0	00446638

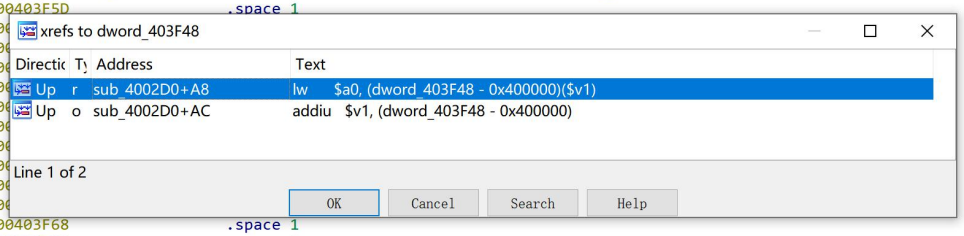
LOAD	00400000	00400094
.init	00400094	00400120
.text	00400120	00403EB0
.fini	00403EB0	00403F0C
roooodata	00403F0C	00444000
.ctors	00444000	00444008
.dtors	00444008	00444010
LOAD	00444010	00444020
.data	00444020	004442F0
.got	004442F0	004444E0
.sbss	004444E0	004444E8
LOAD	004444E8	004444F0
.bss	004444F0	00446638

关键技术——rodata段的恢复

部分binary中仅字符串长度存在差异 ➡ 代码段无法体现完整特征

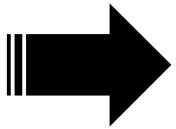
借助IDA的分析结果，获取rodata段中常量的数量、长度和使用位置

```
roooodata:00403F0C  # ============================================================================
roooodata:00403F0C
roooodata:00403F0C  # Segment type: Pure data
roooodata:00403F0C                  .data # roooodata
roooodata:00403F0C                  .align 4
roooodata:00403F10 byte_403F10:     .space 8                # DATA XREF: sub_4002D0+60↑o
roooodata:00403F18 byte_403F18:     .space 0x30             # DATA XREF: sub_4002D0+178↑o
roooodata:00403F48 dword_403F48:    .space 4                # DATA XREF: sub_4002D0+A8↑r
roooodata:00403F48                                          # sub_4002D0+AC↑o
roooodata:00403F4C dword_403F4C:    .space 4                # DATA XREF: sub_4002D0+B4↑r
roooodata:00403F50 byte_403F50:     .space 1                # DATA XREF: sub_4002D0+BC↑r
roooodata:00403F51                  .align 2
roooodata:00403F54 dword_403F54:    .space 4                # DATA XREF: sub_4002D0+FC↑r
roooodata:00403F54                                          # sub_4002D0+100↑o
roooodata:00403F58 dword_403F58:    .space 4                # DATA XREF: sub_4002D0+108↑r
roooodata:00403F5C unk_403F5C:      .space 1                # DATA XREF: sub_4002D0+120↑o
roooodata:00403F5D                  .space 1
roooodata:00403F68                  .space 1
```



关键技术——binary跨架构特征构建

由于不同架构的编译器处理不同（函数内联等）



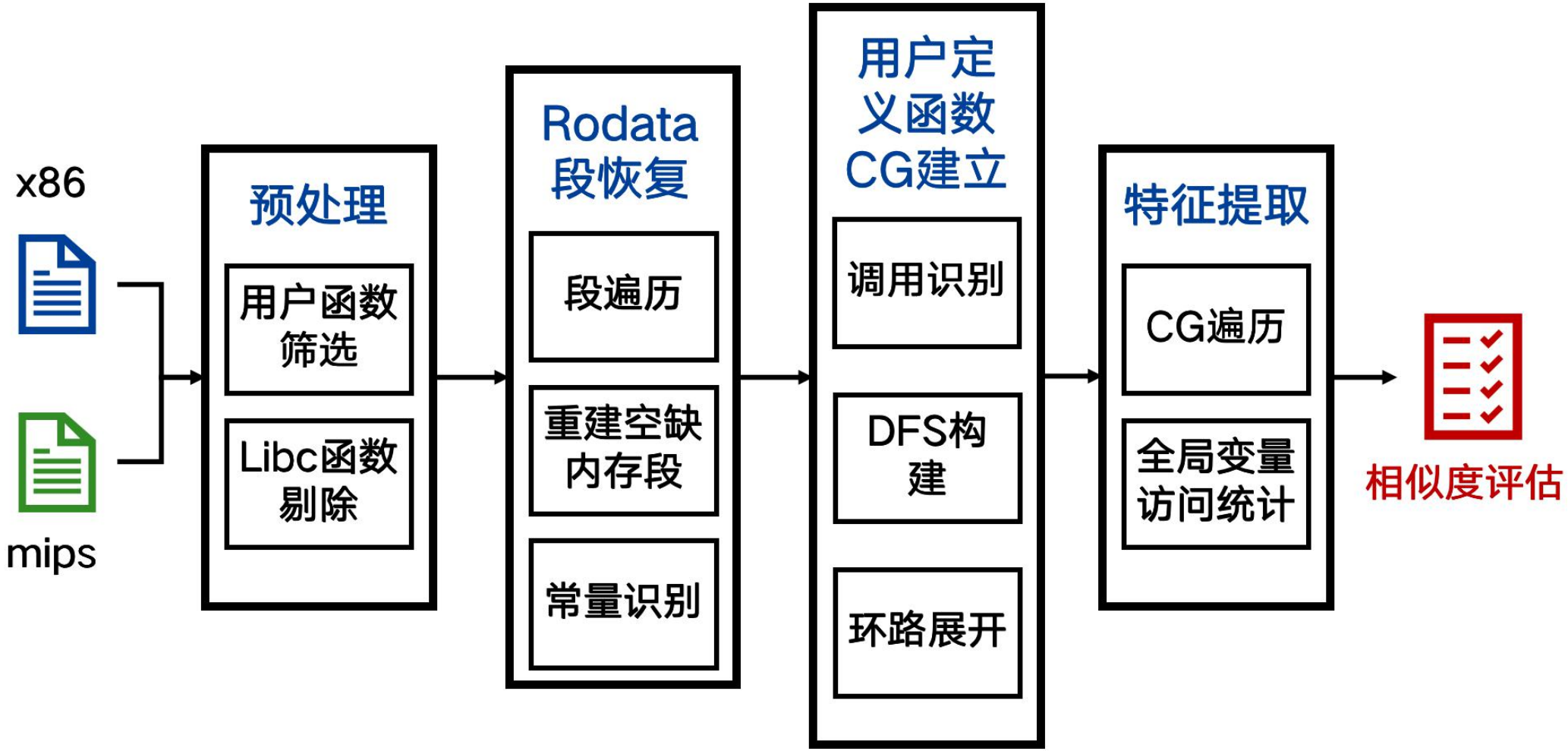
CFG存在差异

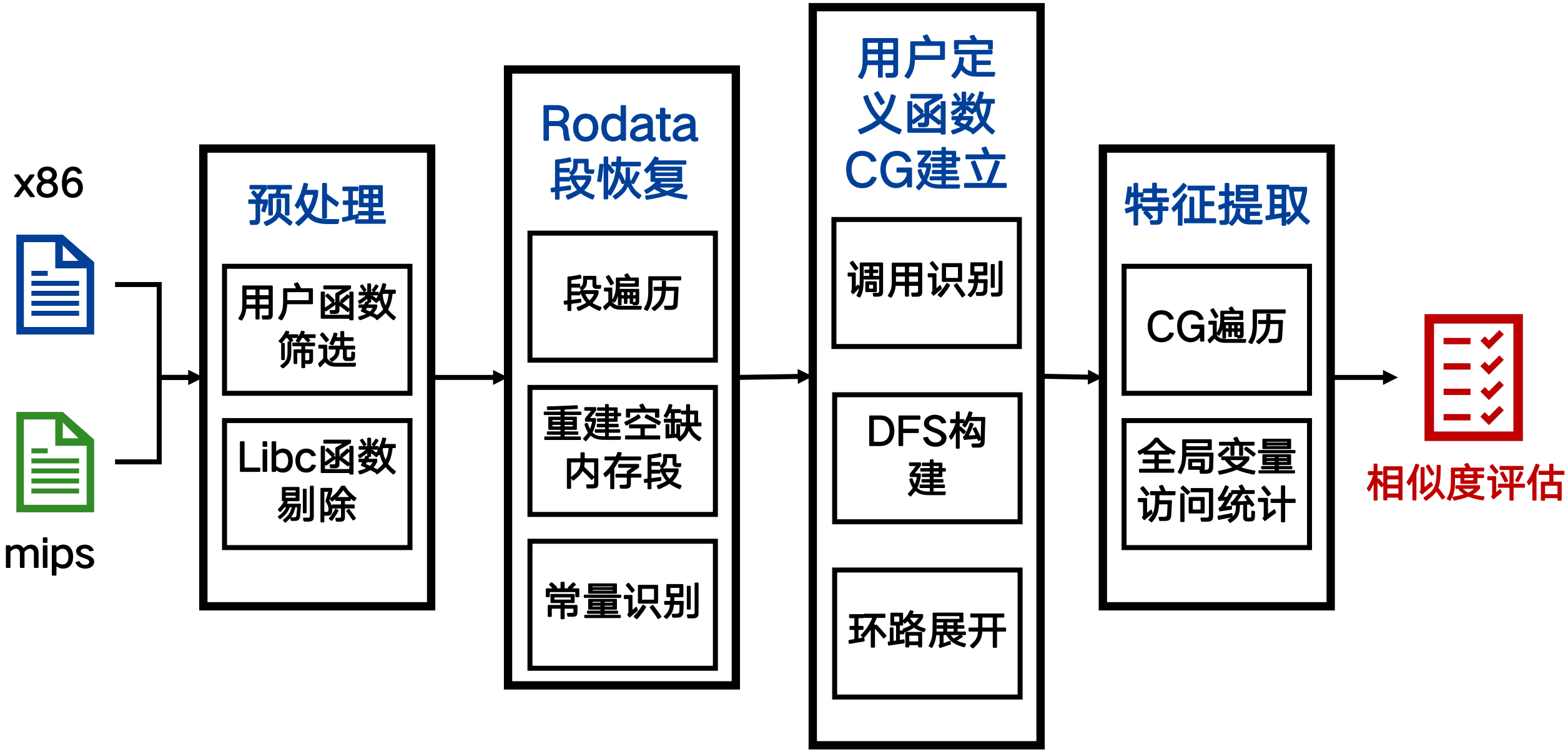
我们选取用户定义全局变量的访问作为核心特征

```
0x4002d0
['unk_403F10', 'unk_403F18', 'dword_403F48', 'dword_403F48', 'dword_403F4C', 'byte_403F50', 'dword_403F54', 'dword_403F54', 'dword_403F58', 'unk_403F5C']
0x400d70
['dword_403FC0', 'dword_403FC0']
0x401720
['unk_403FE0', 'unk_403FE8', 'unk_403FEC']
0x401e34
['unk_403FF0']
0x400120
['unk_403FFC', 'unk_440000', 'unk_440000']
0x4001dc
['unk_403FFC', 'unk_440000', 'unk_440000']
0x4007e0
['unk_440000']
0x401ebc
['unk_440000']
0x403630
['unk_440000', 'unk_440000', 'unk_440000', 'unk_440000']
0x403e40
['unk_440000']
Function 0x0 cannot be identified!
Counter()
{4210448: 8, 4210456: 48, 4210504: 4, 4210508: 4, 4210512: 4, 4210516: 4, 4210520: 4, 4210524: 100, 4210624: 32, 4210656: 8, 4210664: 4, 4210668: 4, 4210672: 12, 4210684: 245764, 4456448: 16384}
```

Contents

跨框架同源二进制识别流程




x86
mips
预处理
用户函数筛选
Libc函数剔除
Rodata段恢复
段遍历
重建空缺内存段
常量识别
用户定义函数CG建立
调用识别
DFS构建
环路展开
特征提取
CG遍历
全局变量访问统计
相似度评估

Contents

工作总结与展望

- 本项目中我们使用user define functions识别、建立Call Graph、全局变量统计还原的方法对同源跨架构的binary进行了相似性分析，以达到细粒度的同源配对分析。
- 由于时间原因，我们在研究过程中还有许多ideas没能实践...

Ideas

- 将不同架构的binary提升到IR，再在IR上进行程序分析，以提高对多架构binary的适应性

- 同架构下两个bianry代码相似性聚类的简单方法：在fileA中多次选取数条地址无关指令，在fileB中寻找是否出现过该地址无关指令，以此计算两个binary的代码相似度。

- 对单一函数进行动态检查（黑盒），如果能保证该函数及其子调用不会有外部输入（全局变量、input、syscall ），那么它的运行结果应该由其输入参数决定，我们可以写一个wrapper，构造好传入参数，以不同的输入多次调用该函数，使该函数成功执行，若两个binary中的该函数返回值相同，则可以认为这两个函数相同

谢谢！