基于数据流分析的静态漏洞挖掘

funnywei@xfocus.org

内容

- 回顾
- 体系结构
- 中间表示语言
- 数据流分析
- 结束

回顾

- 早期的自动化脚本，工具
- 优点
- 存在的问题
- 我们的不足
- 开发商在做什么

早期的自动化脚本，工具

- 基本思想
  - 简单的模式匹配；
  - 有限的回溯。
- 举例
  - 源码级别--FlowFinder，RATS等。
  - 二进制级别—IDC等脚本。

工具的优点

- 容易开发；
- 运行速度快；
- 占用资源少。

存在的问题

- CPU的依赖问题；
- 现实问题的复杂性，产生漏报；
- 大量的误报，花费很大的精力在没有价值的地方；
- 无法理解程序的语义，我们以往的分析停留在过程内部这样的阶段，无法做得更细致。

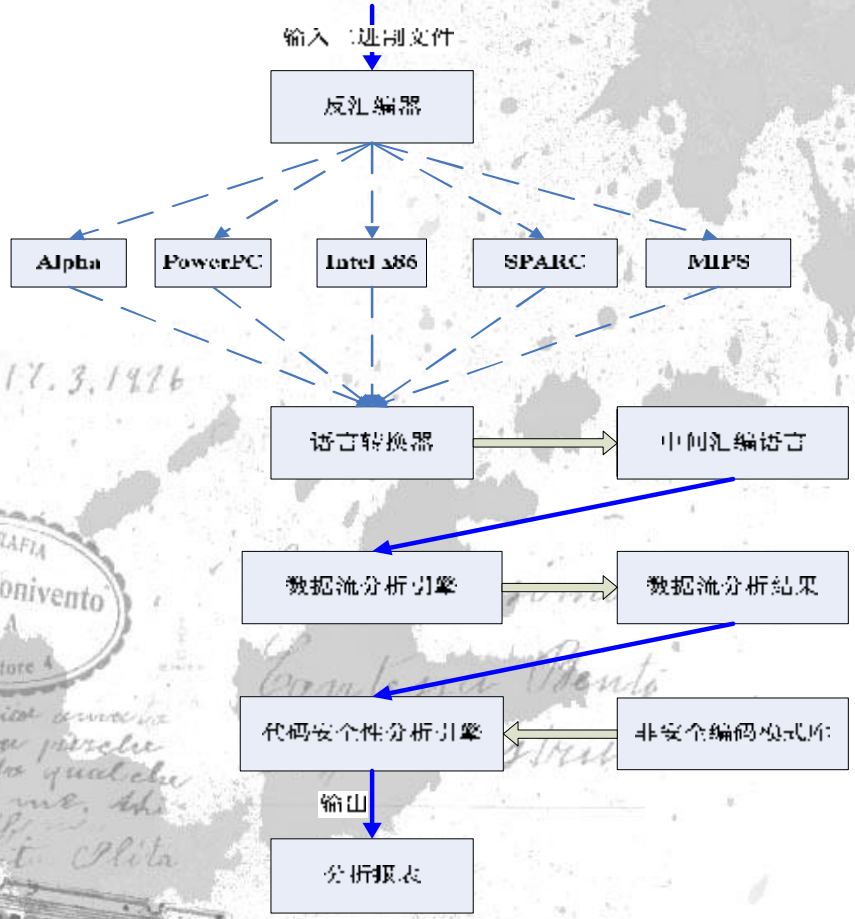
我们的不足

- 在挖掘漏洞时，纯粹的经验主义；
- 缺乏理论的支持；不知道怎样做得更好，陷入困境；
- 分析周期长，我们需要更加自动化的工具。
- 需要工具对于程序的理解。

开发商都在做什么

- 微软的SLAM
  - 1999年，花费6000万美金买下Intrinsa的Prefix用来控制Windows 2000的bug。
  - 开发SLAM，检测驱动程序的错误。
  - 方法：是否遵循了API使用规则。
  - 理论：Program Analysis, Model Checking, Automated Deduction
- IBM的BEAM
  - 是“Bugs Errors and Mistakes”的缩写；
  - 理论：Restrict data flow analysis to path that are executable

系统体系结构


输入二进制文件
反汇编器
Alpha
PowerPC
Intel x86
SPARC
MIPS
语言转换器
中间汇编语言
数据流分析引擎
数据流分析结果
代码安全性分析引擎
非安全编码模式库
输出
分析报表

中间表示语言

- 如何设计
- 中间表示语言的形式
- 我们的设计
- 存在的问题

如何设计

- Steven S. Muchnick说过Intermediate-language design is largely an art, not a science。
- 参考已有的设计。
  - Halvar Flake的Meta CPU设计得很好。

中间表示语言的形式

- 已有的中间表示
  - 逆波兰、四元式、三元式等。
- 四元式实际上是一种“三地址语句”的等价表示，一般形式为：
  - (op, arg1, arg2, result)
- 当op为一元、零元运算符（如无条件转移）时，arg2甚至arg1应缺省，
  - (op, arg1, ---, result)或者(op, ---, ---, result)
- 三元式是四元式的一种变形，为了减少临时变量。

我们的设计

- 参考Halvar Flake的设计
  - 类似Risc
  - 类似Sparc
  - 没有寄存器数目的限制
    - 256个全局寄存器，%g00-%gFF
    - 256个临时寄存器，%t00-%tFF
    - 256个输入/输出寄存器，%i00-%iFF/%o00-%oFF
    - 256个标志寄存器，%f00-%fFF
    - PC，SP，FP

我们的设计

- 清晰的访存指令
  - 内存地址为源地址
    - mov reg, mem
    - ldm mem, ---, reg
  - 内存地址为目的地址
    - mov mem, reg
    - stm reg, ---, mem
- 清晰的循环拷贝指令
  - rep movsd 指令
    - cmp gXX, 0
    - br_nz XXXXXXXX
    - dec gXX, 000004, gXX
    - ldm/stm
    - loop XXXXXXXX

存在的问题

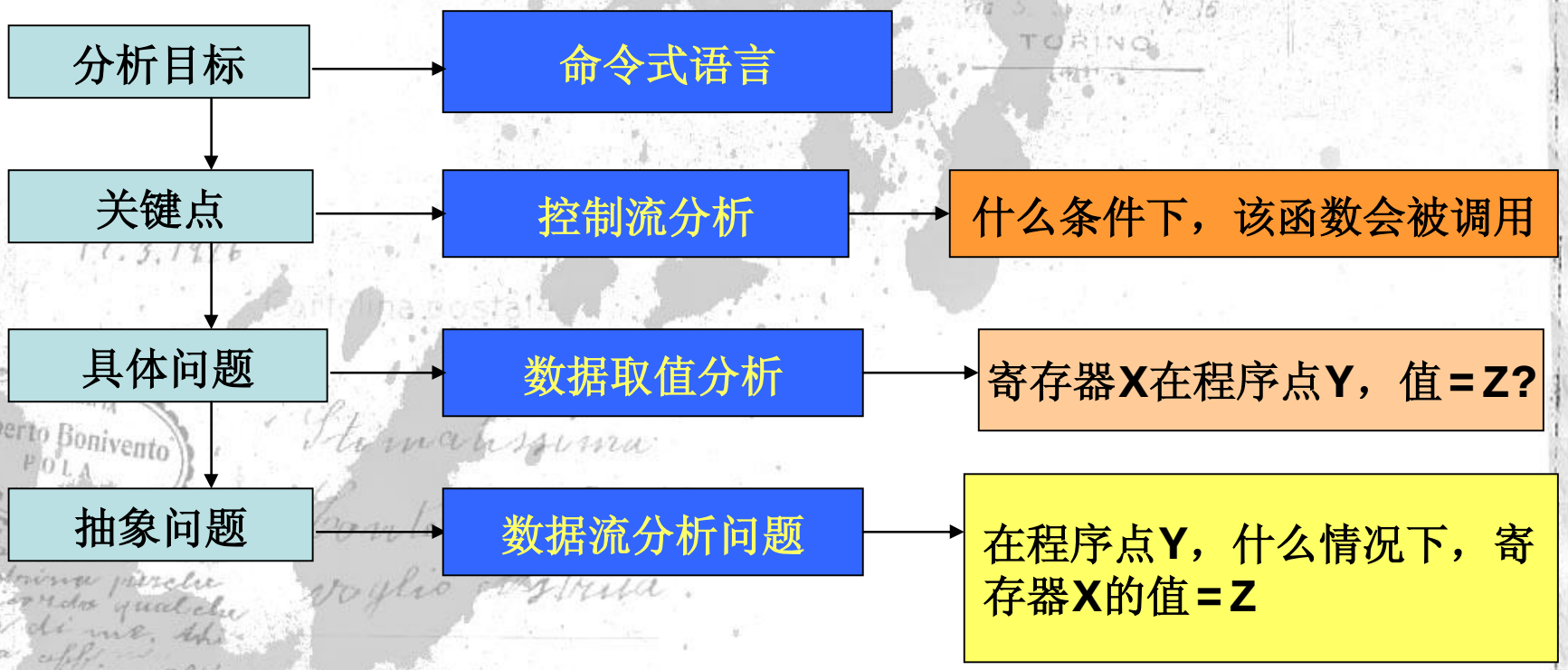
- 理论上：
  - 无法证明翻译的正确性；
  - 上下文无关文法识别--〉上下文相关文法识别。
- 实践中：
  - 比如编译优化问题，需要识别。
  - 不同编译器的优化带来很大的问题。
  - 举例：mov替代push。我们需要翻译mov [esp+offset]到正确的输出编译器。

优化前：	**push edx** **Push ecx** **call function** **add esp, 8**
优化后：	**sub esp, 8** **mov [esp+0], ecx** **mov [esp+4], eax** **call function**

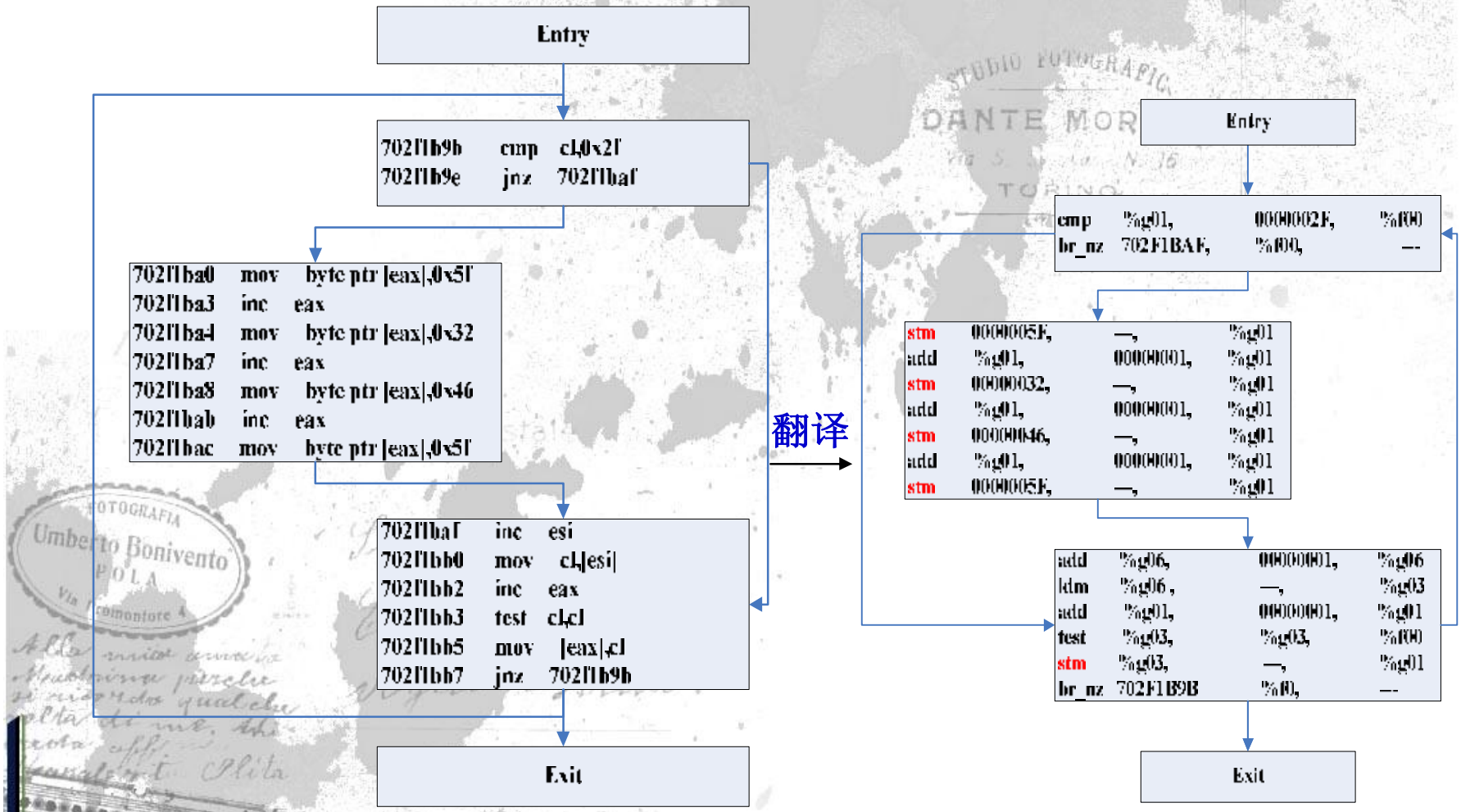
数据流分析

- 为什么要数据流分析
- 分析IE Object Type Property 溢出漏洞
- 基本块分析
- 循环检测
- 数据流分析的需求

为什么需要数据流分析？


分析目标
命令式语言
关键点
控制流分析
什么条件下，该函数会被调用
具体问题
数据取值分析
寄存器X在程序点Y，值＝Z?
抽象问题
数据流分析问题
在程序点Y，什么情况下，寄存器X的值＝Z

IE Object Type Property 溢出漏洞



判断的标准

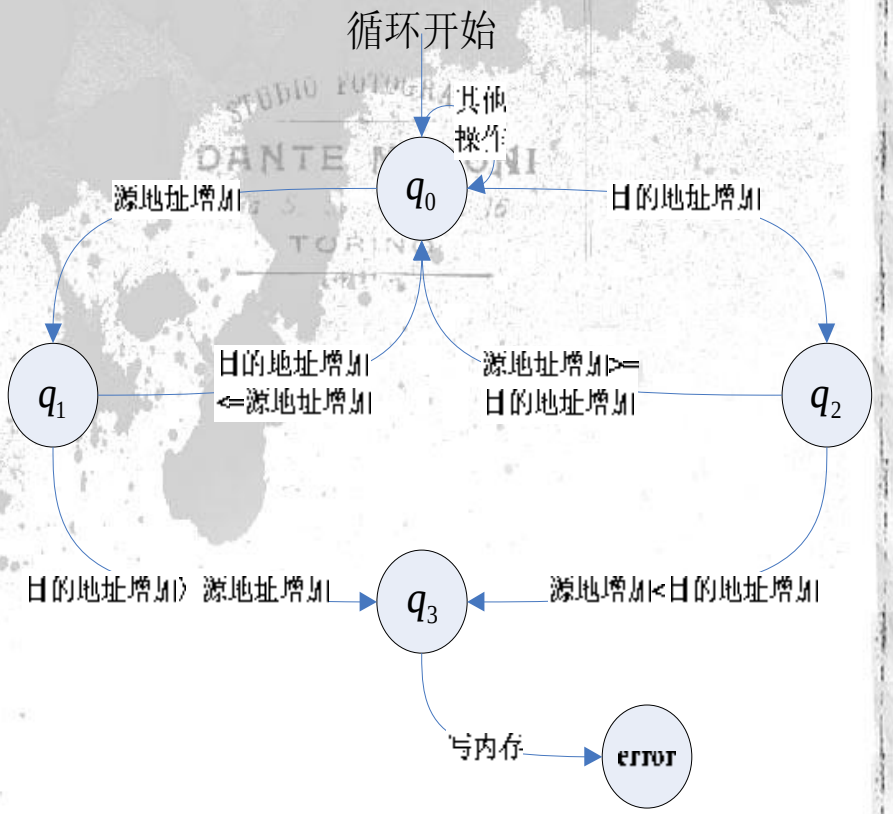
- 如果循环中没有stm语句，不感兴趣！
- 如果循环中，每次循环都写到同样的位置，不感兴趣！
- 循环写一个固定字节数，不感兴趣！

这个例子，可疑！

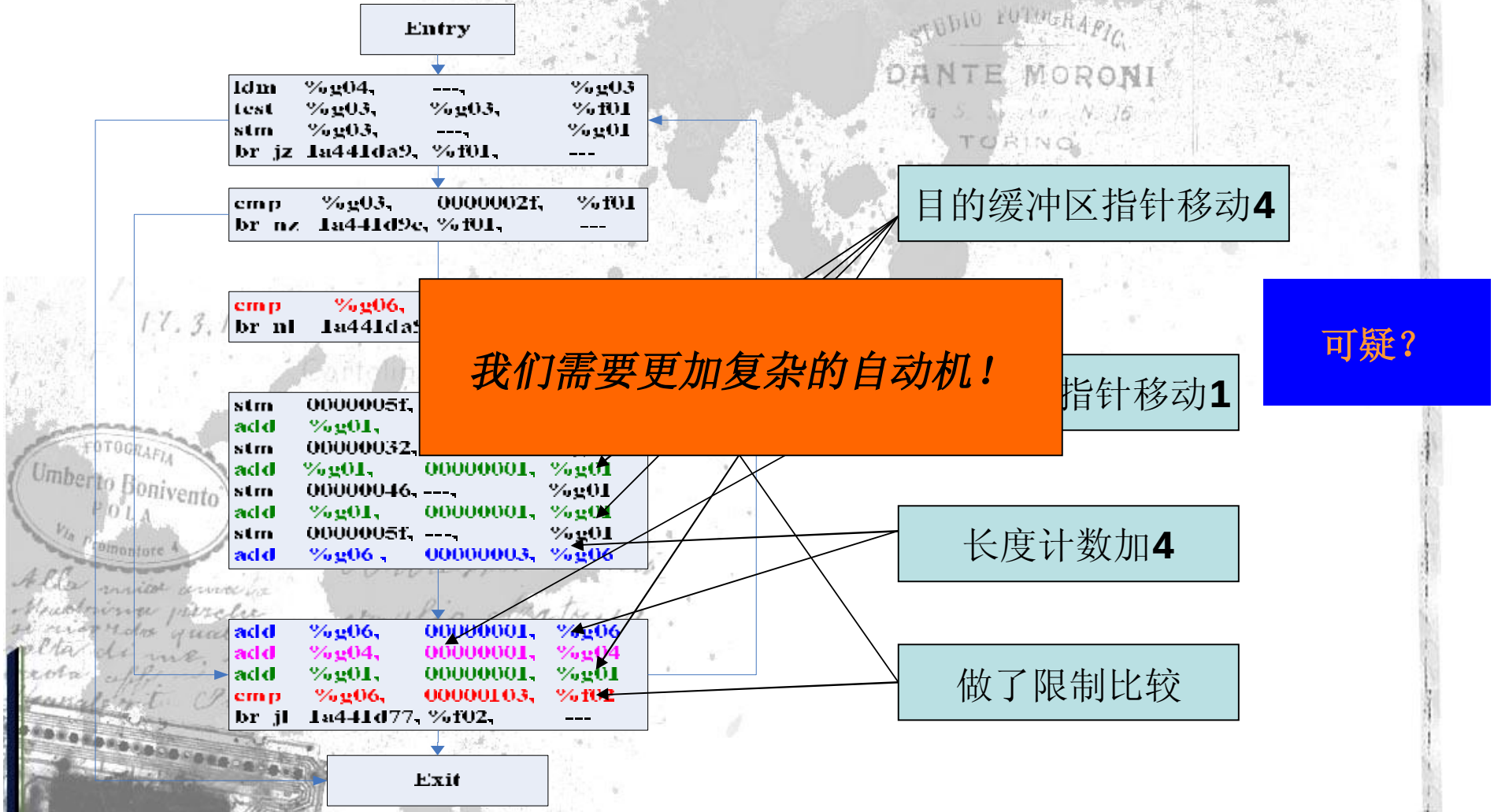
用简化的自动机模型描述这种思想。

下推自动机检测模型

- PDA是一个七元组
- 初始状态：q_0
- 终结状态：error，q_0
- 执行过程实质就是状态转换的函数。
- 设计更复杂的自动机
  - 对长度的检测的状态转移；
  - 写内存位置是否固定状态转移。



IE漏洞补丁情况分析


Entry
ldm %g04, ---, %g03
test %g03, %g03, %f01
stm %g03, ---, %g01
br jz 1a441da9, %f01, ---
cmp %g03, 0000002f, %f01
br nz 1a441d9e, %f01, ---
cmp %g06,
br nl 1a441da
stm 0000005f,
add %g01,
stm 00000032,
add %g01, 00000001, %g01
stm 00000046, ---, %g01
add %g01, 00000001, %g01
stm 0000005f, ---, %g01
add %g06 , 00000003, %g06
add %g06, 00000001, %g06
add %g04, 00000001, %g04
add %g01, 00000001, %g01
cmp %g06, 00000103, %f02
br jl 1a441d77, %f02, ---
Exit
目的缓冲区指针移动4
我们需要更加复杂的自动机！
指针移动1
可疑？
长度计数加4
做了限制比较

如果感兴趣，更深入的话题

- 为什么不采用FSA
  - FSA对应3型文法，正规文法，无法描述该问题。
  - 因为需要存储符号，需要栈！
- 为什么不采用图灵机
  - 图灵机等价于0型文法，太强了，没必要。
  - PDA等价于2型文法，上下文无关文法，够用。
- PDA分类
  - 按终结状态方式结束（本模型使用）；
  - 按空栈方式结束；
  - 两者是等价的，没有优劣。

基本块

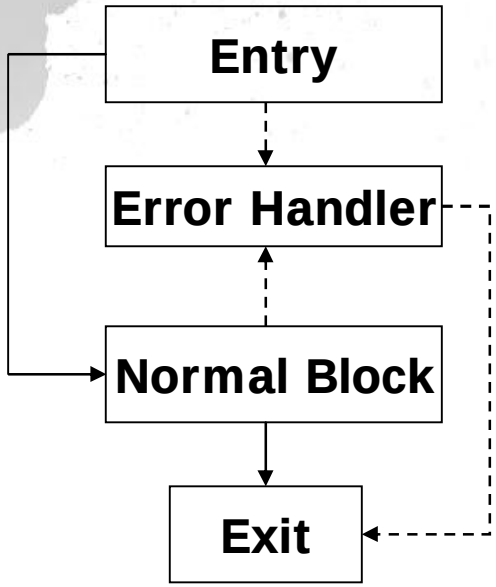
- 一个线性的指令序列，只能从唯一的入口点进入，且从唯一的出口点退出。
- 基本块的第一个指令可以是：
  - 例程的入口点；
  - 分支目标；
  - 紧跟在分支指令或者返回指令后面的指令。

基本块确定算法

- 根据上面的规则，确定各基本块的入口；
- 从没一个入口开始，分别确定各基本块应包含的指令，或者直至退出位置的所有指令都属于相应的基本块；
- 执行前两步之后，凡未包含在任何基本块中的指令，都是不可到达的，故可以被删除。

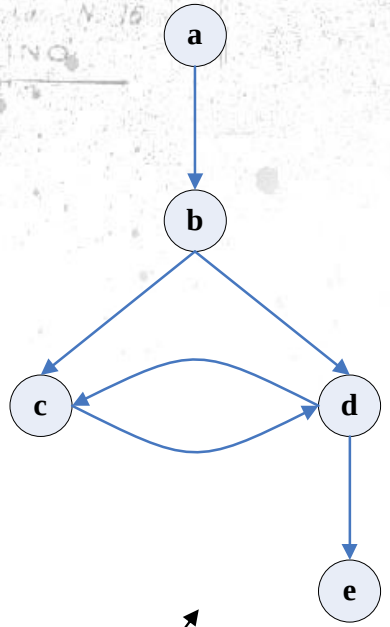
基本块确定可能遇到的问题

- C语言中的异常机制
  - 例如对setjmp的call，成为一个block的引导指令。
  - Windows的异常处理指令



循环的检测

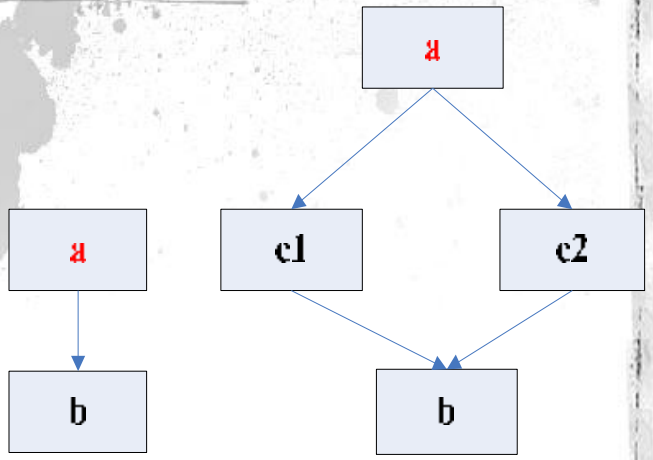
- 什么是循环？
  - 有唯一的入口节点，使得从循环外到循环内任何结点的所有通路，都必须经过此入口节点。
  - 这一组节点是强连通。强连通指从这组节点内任一节点出发，都能达到组中任一其余的节点（特别，当这组节点仅含一个节点时，必有从此节点到自身的有向边）。



不是循环

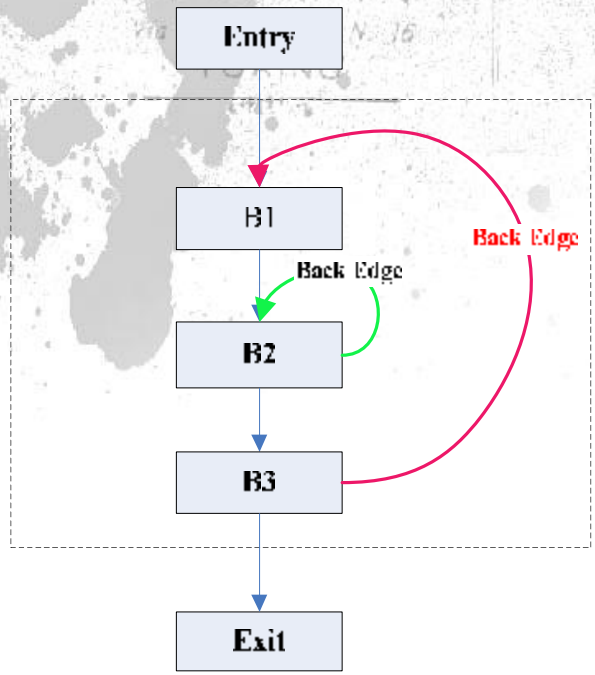
控制点

- 定义
  - 如果从入口点到b的所有可能的执行路径都包含了a，那么我们称节点a控制节点b。
- 计算控制点集合的两种方法
  - 节点a控制节点b，当且仅当 a=b，或者a是b的唯一的直接前驱。或者b有多个直接前驱并且b的所有直接前驱c，c不等于a并且a控制c。
  - Lengauer和Tarjan的方法，比第一种方法复杂，但速度快。



回边及循环检测的算法

- 设d是节点n的控制节点，若在流程图中，存在着从节点n到d的有向边，则称此有向边n→d为流程图中的一条回边。
- 设n→d为一条回边，不经过节点d而能到达节点n的所有节点，包括节点d和n本身，便构成了流程图中的一个循环。此循环以节点d为唯一入口。



最大强连通图

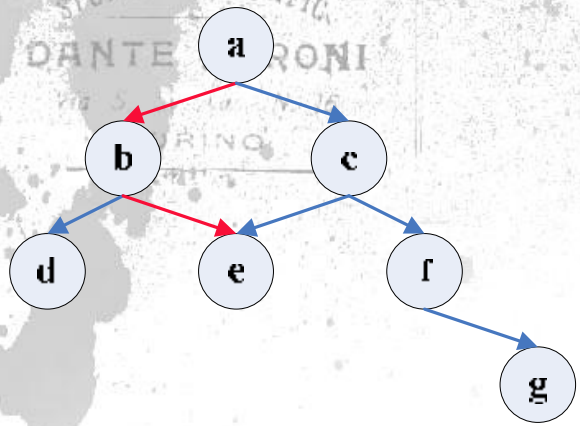
数据流分析的需求

- 能否作更细致的分析？
- Intra-procedure analysis
  - 漏洞最初被怀疑的位置
  - 包含：基本块分析，使用-定义链分析，结构化分析（后两者未作介绍）。
- inter-procedure analysis
  - 漏洞可能是多个函数造成
  - 缓冲区的组成的复杂原因。

➡ 必须两者结合进行分析！

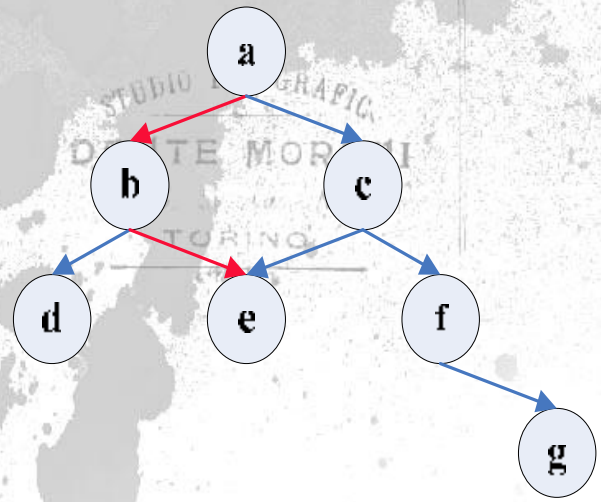
Inter-Procedure分析需求（1）

- 数据取值问题
- 如果函数e为有问题的函数
- 出错时的执行路径为
  - a->b->e
- 问题举例
  - 什么时候a会执行到b
  - a中寄存器X，在程序点Y什么时候会取值Z？



Inter-Procedure分析需求(2)

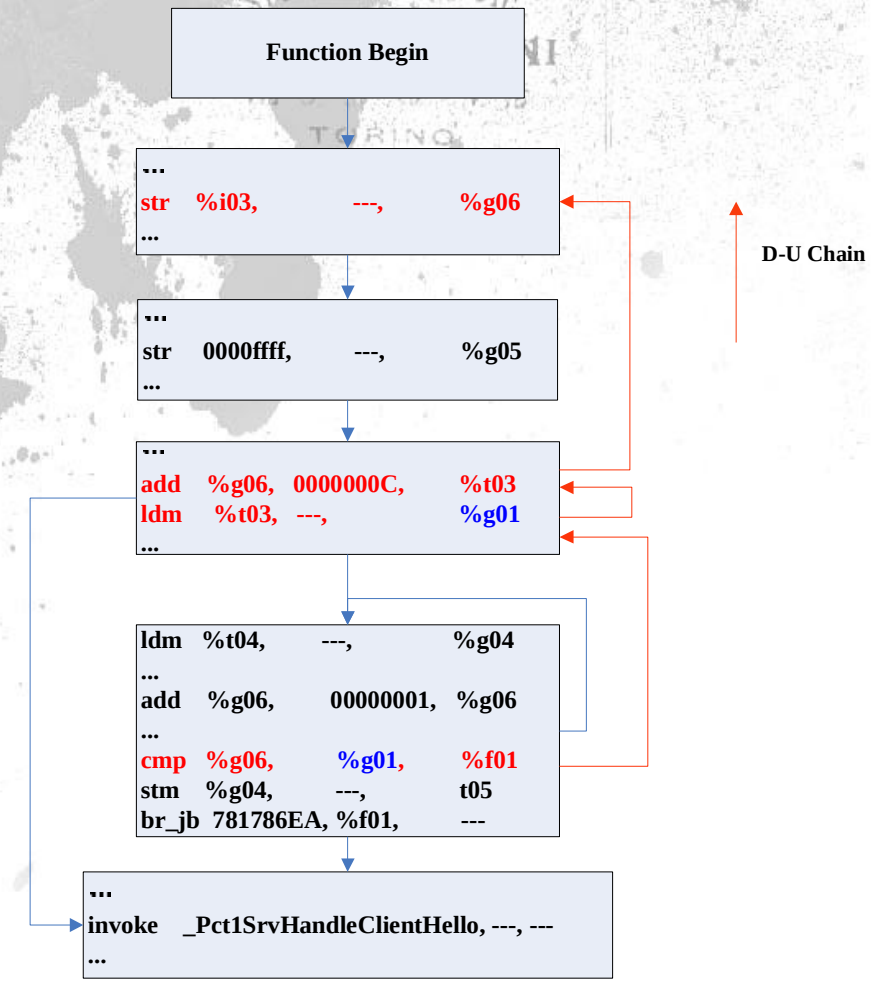
- 数据构成问题
- e中可能存在缓冲区溢出：
  - b中是否有过对缓冲区长度的定义，或者比较？
  - 或者比较是在a中？
  - buffer的构成复杂？a,b,c都参与了构造。



Intra和Inter-Procedure的结合分析

- U-D链和输入输出寄存器结合分析
- 问题会很复杂
- 完全的分析必定是NP问题

- 使用U-D链分析IIS PCT漏洞



为了更好的理解，建议学习

- 更高级的漏洞挖掘与分析涉及
  - 形式语言
  - 形式语义（主要关心操作语义）
  - 编译原理
  - 图论
  - 离散数学
  - 类型论
  - 程序分析

结束

- 谢谢！
- 我的E-mail
  - funnywei@xfocus.org
  - funnywei@163.com