个人资料

kingzone_2008

访问： 296717次
积分： 5009
等级： BLOG 6
排名： 第2375名

原创： 109篇 转载： 15篇
译文： 19篇 评论： 86条

BitBlaze（三） - 静态分析组件Vine

分类： Security&Privacy 2013-04-23 10:54 1796人阅读 评论(11) 收藏 举报

计算机安全 二进制代码分析 静态分析 BitBlaze Vine

目录(?) [+]

3 Vine：静态分析组件

这一部分主要介绍Vine，BitBlaze平台的静态分析组件，描述它的中间语言（IL），它的前端组件、后端组件以及其实现。

3.1 Vine概述

图二Vine的高级结构图。Vine分为平台相关的前端和平台无关的后端。Vine的核心是一种平台无关的中间语言（IL）。IL是一种完全对应汇编语言的小规模的语言。汇编语言通过通过前端转换成IL，后端的分析都是建立在IL之上的。因此，可以实现与计算机架构无关的分析，而且不需要处理复杂的指令集（如x86）。这种设计还具有很强的扩展性—利用Vine提供的核心组件用户可以简单的实现自己的基于IL的分析。

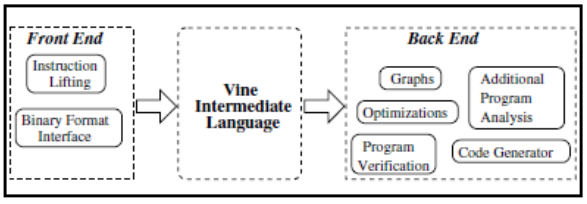


Fig. 2. Vine Overview

Vine前端现在支持把x86和ARMv4转换成IL。它利用一系列第三方库来分析不同的二进制文件并生成汇编代码。汇编代码再通过语法导向的方式转换成IL。

Vine后端支持多种核心的程序分析工具。这些工具可以创建不同的图，比如控制流图和程序依赖图。后端也包含一个优化框架。此框架通常用来简化特定的指令集。它也提供程序校验能力，这包括符号执行，计算最弱前提，以及决策过程。Vine也能通过后端的代码生成器生成c代码。

为了把静态分析和动态分析结合起来，我们也提供了一个接口以使Vine能够使用动态分析组件（如TEMU）生成的执行跟踪。执行跟踪文件可以转换成IL以支持更进一步的分析。

3.2 Vine中间语言

IL是转换的目标语言，也 斤的分析语言。IL的语义是与汇编语言对应的。表一展示了IL。

IL中的基本类型是1位 32位和64位寄存器（如n位向量）和内存。内存类型可以根据字节序分为little（如x86架构的小端模 如PowerPC架构的大端模式）和norm（稍后将介绍）三类。内存类型也可由其索引类型（必须是寄存器类型）进行区别。例如，mem_t(little,reg32_t)表示一个小端模式的地址为32位的内存类型。

Vine支持三种数值类型。第一种是τreg类型。第二，Vine有内存数值{*na*1 → *nv*1, *na*2→ *nv*2, ...},其中*nai*表示地址值，*nvi*表示该地址处的数值。另外，Vine还有一种特有的类型⊥。该数值对于用户是透明的。它被用来指示一次

Security&Privacy (5)
笔试题 (7)
Python (0)
ACM (30)
Linux (3)
Maven (3)
汇编 (1)

文章存档

2015年05月 (1)
2015年04月 (2)
2015年03月 (4)
2015年01月 (1)
2014年12月 (1)
展开

阅读排行

GitHub入门：如何上传与 (16046)
Java：String、StringBul (11815)
掌握VS2010调试 -- 入门 (10390)
jQuery动态添加删除sele (8884)
Java：Date、Calendar、(7928)
Java List与数组之间的转 (6680)
Java连接SQL Server：j (6627)
Spring MVC视图层：thy (6152)
数据挖掘（六）：预测 (5912)
Nexus创建本地Maven仓 (5527)

Favorites

Java 6 Doc
Java SE 6 API
Java 6 中文Doc
C++ Reference
POJ
ZOJ
HDOJ
九度OJ
LeetCode OJ

编程语言

性能排名
TIOBE 流行度
LangPop 流行度
透明排行榜
Github top lang
Computer Programming Language Statistics
DB Engines

数据库与数据挖掘

Oracle Online Documentation
ACOUG
Ask Tom
ERI - Data Mining & Predictive Analytics

执行异常。

Vine中的表达式是没有副作用的。IL支持二元操作◇*b*（&和|是位操作符），一元操作◇*u*，常量，let绑定以及类型转换。类型转换用来改变数值的宽度。例如，x86中EAX的低8位是al。我们可以取出eax的低8位。

Table 1. The Vine Intermediate Language (Since '|' is an operator, for clarity we use commas to separate all operator elements in the productions for $\Diamond_b$ and $\Diamond_u$.)

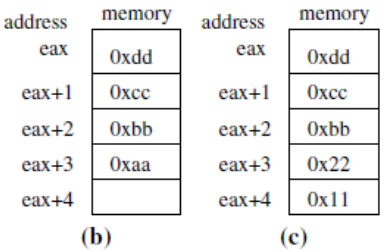
program	::= *decl* instr**
instr	::= *var* = *exp* | `jmp` *exp* | `cjmp` *exp*,*exp*,*exp* | `halt` *exp* | `assert` *exp*
	| `label` *integer* | `special` id_s
exp	::= `load`(*exp*, *exp*, τ_{reg}) | `store`(*exp*, *exp*, *exp*,τ_{reg}) | *exp* $\Diamond_b$ *exp* | $\Diamond_u$ *exp*
	| *const* | *var* | `let` *var* = *exp* `in` *exp* | `cast`(*cast_kind*,τ_{reg},*exp*)
cast_kind	::= `unsigned` | `signed` | `high` | `low`
decl	::= `var` *var*
var	::= (string, id_v, τ)
$\Diamond_b$	::= $+, -, *, /, /_s, \mathrm{mod}, \mathrm{mod}_s, \ll, \gg, \gg_a, \&, \mid, \oplus, ==, \neq, <, \leq, <_s, \leq_s$
$\Diamond_u$	::= $-$ (unary minus), ! (bit-wise not)
value	::= *const* | $\{ n_{a1} \rightarrow n_{v1}, n_{a2} \rightarrow n_{v2}, \ldots \}$: τ_{mem} | $\perp$
const	::= $n : \tau_{\mathrm{reg}}$
τ	::= τ_{reg} | τ_{mem} | Bot | Unit
τ_{reg}	::= `reg1_t` | `reg8_t` | `reg16_t` | `reg32_t` | `reg64_t`
τ_{mem}	::= `mem_t` (τ_{endian}, τ_{reg})
τ_{endian}	::= `little` | `big` | `norm`

在Vine中，load和store操作都是纯粹的。Load的语义通常是纯粹的，但store并非这样。每条store表达式都必须指定要用到的内存地址。例如，下列Vine的store操作mem1 =store（mem0， a， y），其中mem1和mem0基本相同，只是mem1中的地址a指向的数值是y。Vine采用纯粹的存储操作的优点在于可以从句法上区分那块内存被修改过或读取过。这可以用于计算SSA（单一静态指派，标量和内存都有唯一静态地地址）。

Vine程序一般格式是一系列变量声明，随后是指令。共有7中不同的指令，包括赋值、跳转、条件转移和标签。所有跳转和条件转移的目标必须是一个合法的标签，否则程序将终止。注意跳向一个未定义的位置也会导致程序终止。在任何时候都可以使用halt语句终止程序。Vine也支持assert，类似c语言的assert：断言的语句必须为真，否则将终止。

Vine中的special相对于调用一个外部定义的过程或函数。Special的id代表相应的类型。Special的语义取决与分析过程；其操作语义是不确定的。之所以把special算作一种指令类型，就是要在发生那些会影响分析效果的调用是显式的辨别出来。处理special指令的典型方法是用IL编写一个适于分析的摘要函数。

```
// x86 instr dst,src
1. mov [eax], 0xaabbccdd
2. mov ebx, eax
3. add ebx, 0x3
4. mov eax, 0x1122
5. mov [ebx], ax
6. sub ebx, 1
7. mov ax, [ebx]
```



(a) (b) (c)

Fig. 3. An example of little-endian stores as found in x86 that partially overlap. (b) shows memory after executing line 1, and (c) shows memory after executing line 5. Line 7 will load the value 0x22bb.

正规化存储

机器的字节序通常由硬件的字节存储顺序决定。小端模式先存低序字节，大端模式先存高序字节。X86使用小端模式，而PowerPC使用大端模式。

在分析内存访问时必须考虑端模式问题。考虑图3a中的汇编代码。第二行的mov指令使用小端模式向内存中写入4字节（由于x86是小端模式）。执行完第二行后内存如b所示。第2、3行，ebx = eax+3。第4、5行把16位的值

0x1122写入ebx。分析这几行代码可知，第4行复写了第1行所写内容的最后一个字节，如c所示。

正规化内存是指对于b字节可寻址的内存load和store都是b字节对齐的。例如，x86是按字节寻址的，所以x86的正规化存储就是其所有的load和store都是字节级的。图3a中第1行的正规化形式如图4所示。注意第7行的当前存储是mem6。

```
1. mem4 = let mem1 = store(mem0,eax, 0xdd, reg8_t) in
          let mem2 = store(mem1, eax+1, 0xcc, reg8_t) in
          let mem3 = store(mem2, eax+2, 0xbb, reg8_t) in
              store(mem3, eax+3, 0xcc, reg8_t);
...
5. mem6 = let mem5 = store(mem4, ebx, 0x22, reg8_t) in
             store(mem5, ebx+1, 0x22, reg8_t)
...
7. value = let b1 = load(mem6, ebx, reg8_t) in
           let b2 = load(mem6, ebx+1, reg8_t) in
           let b1' = cast(unsigned, b1, reg16_t) in
           let b2' = cast(unsigned, b2, reg16_t) in
              (b2' ≪ 8) | b1';
```

Fig. 4. Vine normalized version of the store and load from Figure 3a

正规化内存简化了涉及内存的程序分析。正规化存储在句法上指出了那些本来隐含在端模式中的内存更新。Vine后端提供了正规化的工具。

3.3 Vine前端

Vine前端负责把二进制代码转换为IL。另外，前端也关联诸如GNU二进制文件描述符（libbfd）库之类的库以分析二进制文件的低级细节。

把二进制代码翻译成IL分为三步：

-第一步。首先对二进制文件按进行反汇编。Vine当前支持三种反汇编工具：IDA Pro，一款商业反汇编器，Kruegel er al.的可以反汇编x86混淆代码的汇编器，我们自己开发的基于GNU libopcodes的线性扫描反汇编器。

-第二步。反汇编完成后，由VEX（一个第三方库）把汇编代码转化为VEX中间语言（VEX　IL）。VEX IL是Valgrind动态测量工具的一部分。它有点像基于RISC的语言。因此，它只有少数几种指令类型，类似于Vine。然而，VEX IL本身不适合程序分析，因为它没有处理指令副作用的问题。这一步主要是为了简化Vine的生成：这一步生成一种基本的IL，第三步就只需处理指令副作用问题了。

-第三步。把VEX IL翻译成Vine。

汇编指令中所有的副作用在翻译完成后都将作为Vine指令显式的表示出来。因而，一条汇编指令可能会翻译成一系列Vine指令。例如x86指令add eax，0x2

翻译成一下Vine指令：

```
tmp1 = EAX; EAX = EAX + 2;
//eflags calculation
CF:reg1_t = (EAX<tmp1);
tmp2 = cast(low, EAX, reg8_t);
PF =(!cast(low,
((((tmp2>>7)^(tmp2>>6))^((tmp2>>5)^(tmp2>>4)))^
(((tmp2>>3)^(tmp2>>2))^((tmp2>>1)^tmp2)))), reg1_t);
AF = (1==(16&(EAX^(tmp1^2))));
ZF = (EAX==0);
```

```
SF = (1==(1&(EAX>>31)));

OF = (1==(1&(((tmp1^(2^0xFFFFFFFF))&(tmp1^EAX))>>31)));
```

翻译完的指令列出了add指令的所有副作用，包括操作可能更新的六个eflags标志。

除了二进制文件，Vine也可以把指令跟踪（instruction trace）转化成IL。跟踪中的条件分支转化成assert语句。Vine和TEMU是联合设计的，因此TEMU生成的trace文件可以被Vine识别和使用。

3.4 Vine后端

在Vine后端，分析工具都是基于Vine IL开发的。Vine提供了基础的组件。下面我们将介绍Vine后端提供的分析工具和组件。

鉴定机。Vine提供了一个实现了Vine IL的操作语义的鉴定机。通过鉴定机，我们可以直接执行程序，而不需把IL重新编译成汇编。

图。Vine可以建立控制流图（CFG）、数据依赖图以及程序依赖图。

创建CFG面临的一个问题是如何判定非直接跳转（跳转到一个计算得到的地址）的后继指令。解决非直接跳转通常需要诸如VSA（值集分析）之类的程序分析工具。因此，可能出现循环依赖。注意，非直接跳转可能跳到任何地方，可能是堆，也可能是还未进行反汇编的代码。

一种解决方案是在CFG图中设置一个指向未确定非直接跳转后继的节点。这样基于CFG的分析工具可以得知我们不知道后继的状态。例如，数据流分析可以把所有事实都扩展到格底。更好的方法是先执行一个非直接跳转解析过程以生成一个更精确的CFG。Vine提供了一个分析工具，VSA。

单一静态赋值（**SSA**）。Vine支持与单一静态指派之间的相互转化。由于每个变量只能静态地定义一次，因而SSA格式使分析变得更加简单。我们将内存和标量都转化成SSA格式。之所以要把内存转化成SSA格式，是因为这样我们就能从句法上辨别在写操作执行前后的内存，而分析工具本身就不需要做类似的记录了。

截断。对于给定的源和汇，程序截断是一个包含了引起源的定义影响汇的使用的语句的图。例如，截断可以用作把后继分析限定在一部分代码中（这部分代码与给定的源和汇有关而非与整个程序相关）。

数据流和优化。Vine提供了一个通用的基于用户自定义结构的数据流引擎。Vine也实现了一些数据流分析。Vine目前实现了Simpson全局数值编号，常量传播及合并，不可达代码删除，动态变量分析，整数范围分析，以及值集分析（VSA）。VSA是一种在任何一个程序点逼近每个变量的值得数据流分析过程。值集分析对非直接跳转问题的解决大有裨益。也可用来做别名分析。两次内存访问值集的交集若不为空，则一个可能是另一个的别名。

优化用来简化或加速后继分析。例如，优化可以是决策过程STP返回一个查询结果的时间减半。

C代码生成器。Vine可以从IL生成合法的C代码。Vine可用作一个基本的反编译器，先把汇编语言转换为IL，再生成C代码。这也提供了一种编译Vine程序的方法：先把IL转换成C代码，在用C编译器进行编译。

C代码生成器把IL中的存储体转换成数组。Store操作是对数组的存储操作，load操作是从数组加载。因此，C代码模拟真实的内存。例如，一个易受缓冲区溢出攻击的程序提交给Vine并生成C代码，继而进行编译。原程序的越界将在相应的C数组中模拟，但并不会导致真正的缓冲区溢出。

程序校验分析。Vine目前支持两种正式的程序校验方式。第一，Vine能够把IL转化为Dijkstra守卫命令语言（GCL），并计算GCL程序的最弱前提。对于一个程序而言，关于谓词q的最弱前提是保证对任意满足该条件的输入都会得到一个满足q的状态的一个最普遍条件。目前只支持无循环的程序，如，不支持GCL的while。

Vine包含与决策过程的接口。Vine能够用CVC语法产生表达式（如，最弱前提）。Vine可以通过直接调用STP库与STP决策过程相关联。

3.5 Vine的实现

Vine采用C++和OCaml实现。前端主要由C++实现，包括大约17200行代码。后端由OCaml实现，包含约40000行代码。通过IDL生成的存根可以实现前端和后端的衔接。

前端采用Valgrind VEX协助转换指令，采用GNU BFD分析可执行对象，采用GNU libopcodes打印反汇编后的程序。

除了图1中的指令，Vine IL还包含几种构造器：

-Vine IL包含注释构造器。它可以打印每条反汇编的指令和用户自定义的注释。

-Vine IL支持跨块地变量辖域。

-Vine IL含有一个修饰用户自定义属性的语句和类型的构造器。

上一篇 BitBlaze（二） - BitBlaze架构
下一篇 C#实现打印功能

查看评论

1楼 lucase_yy 2013-11-22 10:39发表

借问下，楼主这些东西是自己总结的还是网上已有的资料？谢谢~

Re: kingzone_2008 2013-11-23 16:08发表

回复lucase_yy：这个是我翻译的，berkeley官网上有英文原版。
http://bitblaze.cs.berkeley.edu/papers/bitblaze_iciss08.pdf

Re: lucase_yy 2013-11-28 13:49发表

回复kingzone_2008：嗯，谢谢哈～最近导师说让我从源代码入手研究下这个工具，有些迷茫，看到您的博客，有点启发，不知道您对这个工具了解程度如何？从源代码入手，该怎么下手好点？

Re: kingzone_2008 2013-11-28 18:59发表

回复lucase_yy：哦，你是本科还是研究生？我这是本科毕设时搞的，现在已经基本忘干净了

Re: lucase_yy 2013-11-28 19:32发表

回复kingzone_2008：研究生..略头大，那个源码里面，我下载看了一下，不太清楚里面有的文件是干嘛的，有的文件夹的作用是什么也不太清楚..Readme里面也没有相关的介绍..当时你本科毕设做的是什么？有没有深入了解下..

Re: kingzone_2008 2013-11-29 22:55发表

回复lucase_yy：。。。我去，本科毕设。你懂的。我最近两周也在忙毕设。忙完了抽时间再看看，还来得及吗

Re: lucase_yy 2013-11-30 21:11发表

回复kingzone_2008：没事，你这么热心回复已经很感谢了~我现在处于开题报告阶段，下周二开题，所以想确定下要做什么，目前的想法是做个TEMU插件吧。。不知道如何？

Re: kingzone_2008 2013-11-30 22:31发表

回复lucase_yy：哦，是研二。我感觉有的研究，不过可能有些难度。你实验室里之前有人做过这个吗？

Re: lucase_yy 2013-12-01 00:05发表

回复kingzone_2008：sry,刚看到，之前没人做过，网上的有个bitblaze论坛，编写插件的话，不清楚具体要做什么功能呢？检测缓冲区溢出？没有做过，不知道难度如何？你觉得呢？

Re: kingzone_2008 2015-03-24 21:58发表

回复lucase_yy：做得怎么样？共享下成果吧

Re: kingzone_2008 2013-12-05 15:40发表

回复lucase_yy：我之前是做的病毒的加壳和脱壳研究。