

基于 ARM 架构的信息流追踪系统 的设计与实现

施祖清, 张 涛, 王金双, 姚金魁, 袁志坚

(解放军理工大学 指挥自动化学院, 江苏 南京 210007)

摘 要:当前,智能手机平台面临着众多的安全威胁。动态信息流追踪是一种能够检测缓冲区溢出等安全威胁的有效技术。文中分析了动态信息流追踪技术的基本原理,设计并实现了基于 ARM 架构的信息流追踪系统。该系统通过在页表项上扩展添加污点标记位来标识来自不可信数据源的数据,扩充 ARM 架构指令集,在指令层追踪数据的传播过程并相应地完成污点传播。当系统跳转到来自不可信数据源的内存段执行时,CPU 将产生异常通知用户,根据系统安全策略决定是否允许该操作继续执行。研究表明该系统能够有效实现 ARM 架构智能平台的安全防护。

关键词:信息流追踪;ARM 架构;页表项;指令层;污点

中图分类号:TP309

文献标识码:A

文章编号:1673-629X(2012)06-0147-04

Design and Implementation of an Information Flow Tracking System Based on ARM Architecture

SHI Zu-qing, ZHANG Tao, WANG Jin-shuang, YAO Jin-kui, YUAN Zhi-jian

(Institute of Command Automation, PLA University of Science and Technology, Nanjing 210007, China)

Abstract: At present, smartphone platform is facing numerous security threats. Dynamic information flow tracking (DIFT) is an effective technique for detecting buffer overflow. It analyzed the basic principle of DIFT, designed and implemented an information tracking system for the ARM architecture. A taint marking bit was added to the page table entry, which was used to indicate data coming from untrusted source. The ARM instruction set was expanded to track the taint propagation accompanied with the propagation of the data. CPU will throw exceptions when the system jumps to memory segments containing data from untrusted source, and enforce the access decision according to system security policy. Thus the system can be protected from attacks originated from buffer overflows etc.

Key words: information flow tracking; ARM architecture; page table entry; instruction layer; taint

0 引 言

动态信息流追踪技术是一种通过为数据增加污点标识,并追踪污点在系统内的传播,从而服务于攻击检测和隐私泄露等用途的安全防护技术,能够检测缓冲区溢出等多种安全威胁的有效技术。目前,该技术的研究主要分为两类:利用动态标记技术跟踪恶意流量在内存中留下的痕迹,并触发报警机制产生日志,如基于 x86 架构的 Argos^[1]系统;通过跟踪和精确分析用户隐私数据的流向,确保隐私数据的安全,如基于 Android 的 TaintDroid^[2]。

文中以 Android 模拟器为实验平台,设计并实现

了基于 ARM 架构的信息流追踪系统。该系统追踪操作系统从不信任源上接收的数据,并监视这些数据在整个系统中的执行,当这些数据被用于安全策略不允许的操作时,系统将阻止该操作并抛出警报。

1 背景介绍

1.1 Android 模拟器

作为市场上主流的智能机操作系统,Android 越来越受到人们的青睐,然而,Android 也面临着越来越多的安全威胁^[3]。Google 提供了一款 Android 模拟器,该模拟器是基于 Qemu^[4]开发的,通过软件来模拟整个装载了 Android 系统的智能手机,Google 将其硬件部分命名为 Goldfish,其对应的源代码主要在 external/qemu 目录下,用来模拟包括 ARM926ej-S CPU、MMC 等功能的 ARM SoC。该模拟器功能齐全,包括电话本、通话、连接互联网等功能都可以正常使用,用户还可以使用键盘输入、鼠标点击、拖动屏幕等方式进行操作。An-

收稿日期:2011-11-01;修回日期:2012-02-06

基金项目:国家高技术研究发展计划“863”项目(2009AA01Z40)

作者简介:施祖清(1986-),男,硕士研究生,CCF 会员,研究方向为嵌入式系统应用技术;张 涛,教授,硕士研究生导师,研究方向为信息安全、计算机系统结构。

droid 模拟器为系统测试和应用程序开发提供了便利。

1.2 Qemu

Qemu 是一套由 Fabrice Bellard 所编写的模拟器处理器的自由软件,是一种能够支持多种 CPU 的机器模拟器^[4]。Qemu 采用动态翻译技术来实现不同处理器的模拟,从版本 0.10.0 开始,采用微型代码生成器(TCG)作为其翻译引擎。TCG 的全称为“Tiny Code Generator”,主要负责分析、优化目标机器代码以及生成主机代码,TCG 使得 Qemu 不再依赖特定的 GCC 版本,实现真正意义的动态翻译。

Qemu 能够进行多系统模拟,而且支持多平台,因此常被人用来进行系统开发与测试,Android 模拟器就是使用 Qemu 进行底层硬件模拟。

1.3 动态信息流追踪

操作系统及应用软件存在许多漏洞^[5],黑客常利用这些漏洞对系统进行攻击。如黑客可通过往程序缓冲区中写入超出其长度的内容,造成缓冲区的溢出,促使程序崩溃或使程序转而执行其它指令,以达到攻击的目的^[6]。

针对缓冲区溢出漏洞,人们提出了多种解决思路^[7],动态信息流追踪技术就是其中之一。动态信息流追踪技术根据安全策略规定将系统中的部分数据打上标记,这些数据可以是从不信任源来的污点数据,也可以是用户要保护的隐私数据,由安全策略规定。这些标记在程序运行中将根据数据流和控制流进行传播。当带有标记的数据被用于安全策略不允许的操作时,系统将阻止这个操作并抛出警告^[8]。

2 系统的设计

2.1 总体方案设计

该系统是基于动态信息流追踪技术实现的,主要追踪从不信任源来的污点数据,系统包括三个模块:污点标记、污点传播和异常检测,如图 1 所示。

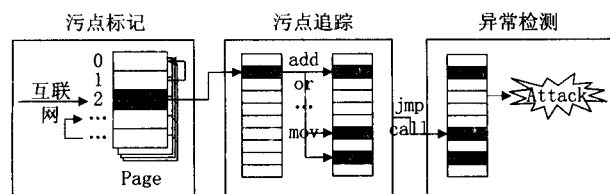


图 1 系统框架图

污点标记模块监控所有来自不信任数据源的数据,如从网络上下下载的第三方应用程序、接收的多媒体信息的附件、通过蓝牙接收的数据等等,并将这些数据统一打上污点标记。如图 1,污点标记模块的页表 2 中包含有非可信数据,用灰色块表示。

带有污点标记的数据在指令执行过程中被传播,污点传播模块追踪每个涉及数据转移的指令的执行,污点标记跟随指令的执行而转移。如图 1,污点标记在内存空间中传播(灰色块在扩散)。

当某些跳转、调用指令执行带有污点标记的数据时,异常检测模块将给出攻击警报。如图 1,当 jmp/call 指令地址为灰色块区域,系统将给出 Attack 警报。

2.2 污点标记模块

以网络下载数据为例,网卡驱动声明了一个结构 sk_buff 当做接收网络分组的缓存,所有网络上下载的数据都得通过该缓存进入系统中^[9]。当网卡接收到网络上来的数据,CPU 给操作系统一个中断,网卡驱动程序处理中断并接收数据。系统在此设置监测点,将所有接收到的数据都打上污点标记。

操作系统使用页表将虚拟地址空间映射到物理地址空间。文中扩展页表项功能,在页表项中增加污点标记位,以页表为单位区分污点数据和安全数据。在 arm linux 中,页表项为一个 32 位 unsigned long 类型的变量,前 20 位为物理页地址,后 12 位存储了一些与页相关的附加信息,如图 2 所示。

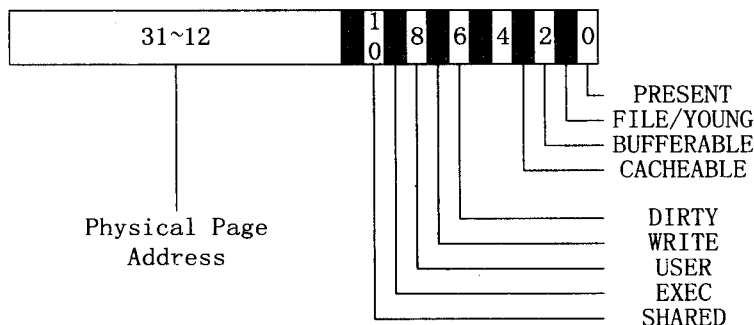


图 2 ARM Linux 页表项

其中,PRESENT 表示该页是否在内存中;DIRTY 表示该页是否为脏,即该页内容是否被修改过;WRITE 表示该页是否可写。

使用未定义的 4、5 位作为污点标记位。该页数据为安全数据时,4、5 位全置为 0;当该页数据为来自网络的污点数据时,置第 4 位为 1;来自信息服务时,设置第 5 位为 1;来自蓝牙红外时,第 4、5 位同时置为 1。这样,整个系统能够同时检测 3 种不同源的安全威胁。

2.3 污点传播模块

污点的标记在操作系统层实现,而污点传播主要在 ARM CPU 指令层上实现。ARM 指令集是 load/store 型的^[10],只能通过 load/store 指令实现对系统存储器的访问,而其它的指令都是基于处理器内部的寄存器操作完成的。因此,在操作系统层实现污点数据的标记后,通过 load 指令传播到 CPU,然后通过其它指令在 CPU 内部寄存器间进行传播,最后由 store 指令

返回给操作系统并写入相应的页表项中。

load/store 指令中污点数据的传播可以通过页表转换来实现。其它指令执行时,追踪污点数据传播的规则是:对于二操作数指令,如果源操作数是污点数据,那么目的操作数也是污点数据;对于三操作数指令,任一操作数是污点数据,那么其结果也是污点数据。其中有一些特殊情况需要排除,如 XOR 指令,CPU 使用 XOR 指令来置零,该指令需要特殊对待。

图 3 中为数据处理指令机器码,Rd 为目的寄存器,Rn、Rm 为源寄存器,opcode 为操作码。在执行该指令时,先判断 Rn、Rm 是否含有污点标记,如果其中之一为污点数据,则给目的寄存器 Rd 也打上污点标记,其它指令的传播类似。

31	2827	2524	212019	1615	1211	7	6	5	4	3	0
Cond	0	0	0	opcode	S	Rn	Rd	amount	shift	0	Rm

图 3 ARM 数据处理指令编码格式

2.4 异常检测模块

黑客利用缓冲区溢出等系统漏洞进行攻击时,需要改变程序的正常控制流以及涉及对不信任数据的非法使用,该模块利用这个特点,通过识别非法使用污点数据来实现异常检测。

一般来说,检测规则主要是查看程序是否有改变跳转对象(如返回地址、函数指针等)进行攻击。当检测到 jmp/call 等指令目标地址为污点数据时,CPU 产生一个异常通知操作系统可能有攻击发生,在操作系统中增加异常处理函数响应该异常,阻止程序继续运行,要求用户确认信息的可靠性,自行判断是否允许其执行。

3 系统的具体实现

3.1 标记的生成

Linux 系统响应 CPU 中断开始接收网络数据,这些数据都要在协议栈的 netif_rx 函数中进行处理^[11],文中在该函数中实现污点标记,通过页表映射获得网络数据所在的页,将该页页表项指定位置 1,表示该页数据为来自网络上的不安全数据。

Linux 中,每个进程都有一个 task_struct 任务结构,和一片用于系统空间堆栈的存储空间,通过获取当前进程块及其存储空间,即可修改其页表项。具体实现如下:

- 获取当前进程块及其虚拟空间:

```
struct task_struct * cur_task = get_current();
struct mm_struct * mm = cur_task->mm
```

- 通过 mm_struct 和线性地址得到地址对应的页目录项:

```
pgd = pgd_offset(mm, vaddr);
```

```
pmd = pmd_offset(pgd, vaddr)
```

- 根据线性地址和 pmd,找到该线性地址对应的页表项,即网络数据的存储页,并获取页表项的值。

```
pte = pte_offset_kernel(pmd, vaddr);
```

```
pa = pte_val(*pte)
```

- 修改页表项值,打上“污点”标记,即置页表项第 4 位为 1。

```
pte = pte_modify(pte, newprot)
```

当进程撤销申请的内存空间或者该进程结束时,将相对应的污点标记撤销。

3.2 污点的传播

3.2.1 存储器传播

ARM CPU 执行 load 指令时,进行存储器操作,获取由操作系统生成的污点标记,将其转移到 CPU 中。

- load 指令,CPU 从指定地址 addr 中取出数据。

```
tcg_gen_qemu_ld32u(TCGv ret, TCGv addr, int index)
```

- 利用地址转换可以得到相对应的页面信息^[12]。

```
p = (void **)l1_phys_map;
```

```
index = addr >> TARGET_PAGE_BITS;
```

```
lp = p + ((index >> L2_BITS) & (L1_SIZE - 1));
```

```
pd = *lp;
```

```
return ((PhysPageDesc *)pd) + (index & (L2_SIZE - 1))
```

其中,l1_phys_map 为一级页表的首地址,TARGET_PAGE_BITS 为页大小,L1_BITS、L2_BITS、L1_SIZE、L2_SIZE 分别为一级页表和二级页表的 bits 与 size。获取页面信息后,从中提取出污点标记,并将其传播到 CPU 中的寄存器中,执行下节所述的寄存器传播。

执行 store 指令时,若发现存入某地址的数据为污点数据,CPU 将产生一个异常,通知操作系统将该污点标记写入该地址所在页表项中。

3.2.2 寄存器传播

ARM 微处理器有 37 个寄存器,其中 31 个为通用寄存器,为每个通用寄存器添加一个标签结构 rtag,里面包含三个变量,smsidx、conidx 和 netidx,分别表示来自三个不信任源的数据。

```
struct Info_Tag {
```

```
uint_t smsidx; // data from SMS or MMS
```

```
uint_t conidx; // data from Bluetooth
```

```
uint_t netidx; // data from Network
```

```
}
```

如果判定源操作数已经被“污染”,则直接将源操作数污点标记值传递给目的操作数。这里利用 tag_copy() 函数来实现,以 add 指令为例,当执行 add 微操作时,系统同时执行污点传输函数:

```
void tcg_gen_add_i32(TCGv_i32 ret, TCGv_i32 arg1, TCGv_i32 arg2)
```

```

}
tcg_gen_op3_i32(op_add_i32, ret, arg1, arg2);
tag_comb(ret, arg1, arg2);
}

```

污点判定过程:

```

#define tag_comb(dst, src1, src2)
do {
//判断源操作数是否为污点数据
if (tag_isdirty(src1))
tag_copy(dst, src1); //标记传播
else if (tag_isdirty(src2))
tag_copy(dst, src2);
} while (0)

```

如果源操作数为被污染的数据,则执行 tag_copy 函数,进行污点传播。污点传播函数:

```

#define tag_copy(dst, src)
do {
(dst)-> smsidx = (src)-> smsidx;
(dst)-> conidx = (src)-> conidx;
(dst)-> netidx = (src)-> netidx;
} while (0)

```

3.3 异常处理

CPU 在执行跳转指令时,获取转移地址,并将转移地址存入 CPU 通用寄存器 regs[15]。加入异常检测机制后,首先调用 InfoTrack_Check() 函数,判断是否有污点数据用于该操作。如果没有污点数据用于跳转操作,正常执行跳转操作,否则,CPU 将产生一个异常 raise_exception,通知操作系统进行异常处理。

```

void gen_bx_im(DisasContext *s, uint32_t addr)
{
TCGv tmp;
//判断污点数据是否用于不安全操作
if (InfoTrack_Check(addr, ALERT_JMP))
//产生异常,通知操作系统
raise_exception(env->exception_index);
else {
s->is_jmp = DISAS_UPDATE;
tmp = new_tmp();
tcg_gen_movi_i32(tmp, addr & ~1);
tcg_gen_st_i32(tmp, cpu_env, offsetof(CPUState, regs
[15]));
dead_tmp(tmp);
}
}

```

操作系统收到 CPU 的异常信号后,执行报警操作,通知用户有非法程序即将执行,授权用户进行处理。

4 结束语

文中主要以 Android 模拟器为试验平台,研究 ARM 架构智能手机平台的安全机制,主要利用动态信息流追踪思想,通过在页表项上扩展添加污点标记位来标识那些来自不可信数据源的数据;并在指令层监视这些不可信数据在整个系统中的执行,确保这些数据不被用于安全策略不允许的操作,达到安全防护的目标。

目前,该系统还处在初步的研究阶段,还有一些缺陷需要改进,例如:动态信息流追踪技术对系统性能的影响较大,需要在实际实验中进一步优化,寻求性能和安全方面的平衡点。

参考文献:

- [1] Portokalidis G, Slowinska A, Bos H. Argos: an emulator for fingerprinting zero-day attacks[C]//EuroSys 2006. [s. l.]: [s. n.], 2006.
- [2] Enck W, Gilbert P, Chun B G, et al. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones[C]//Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI). Vancouver: [s. n.], 2010.
- [3] 宋杰,党李成,郭振朝,等. Android OS 手机平台的安全机制分析和应用研究[J]. 计算机技术与发展, 2010, 20(6): 152-155.
- [4] Bellard F. QEMU, a fast and portable dynamic translator [C]//Proc. of the 2005 USENIX. Berkeley, CA: USENIX Association, 2005.
- [5] 王雨晨. 系统漏洞原理与常见攻击方法[J]. 计算机工程与应用, 2001, 37(3): 62-64.
- [6] 蒋卫华,李伟华,杜君. 缓冲区溢出攻击:原理,防御及检测[J]. 计算机工程, 2003, 29(10): 5-7.
- [7] 黄玉文. 缓冲区溢出攻击原理和现有检测技术[J]. 科技信息, 2010(23): 89-89.
- [8] Suh G E, Lee J, Devadas S. Secure Program Execution via Dynamic Information Flow Tracking[C]//Proc. of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). Boston, MA: [s. n.], 2004.
- [9] 刘宝宝,徐国雄,卜应敏. 移动 IPv4 与 IPv6 工作机制的研究[J]. 计算机技术与发展, 2011, 21(4): 126-128.
- [10] ARM Ltd. ARM926EJ-S Technical Reference Manual[EB/OL]. 2008-06. <http://www.arm.com/>.
- [11] Maunder W. Professional Linux Kernel Architecture[M]. [s. l.]: Wiley Publishing Inc., 2008.
- [12] 王庆民,刘福岩. ARM MMU 中虚地址转换研究[J]. 机械工程与自动化, 2007(1): 44-46.