用户级动态的设计和实现ARM体系结构的二进制检测

Dongwoo Kim1 · Sangwho Kim1 · Jaecheol Ryou

摘要

我们在ARM体系结构上开发了用户级动态二进制检测（DBI）工具，以支持应用各种分析技术，如性能评估，配置和错误检测。大多数现有的DBI工具都基于x86架构。由于主要在x86中执行的类似方法的需求在嵌入式设备域中也越来越高，因此嵌入式设备尤其是ARM体系结构（其在嵌入式系统中占据大部分市场份额）非常需要DBI环境。ARM目前只有少数几种工具可用，但它们仅限于特定用途。因此，我们在ARM上设计并实现了一个通用DBI工具，以在目标进程的指令之间执行任意代码。我们的工具支持安全分析所需的本机执行环境。我们使用常见的UNIXutilities和Android应用程序进行了一些实验，通过跟踪和保存所需线程的每条指令的上下文信息来验证我们的工具。因此，我们确认它可以按预期工作，而不会影响目标流程的原始上下文。

关键词：动态二进制仪表，动态跟踪，程序分析，嵌入式设备，ARM

1简介

近年来，平台技术的发展使得许多服务无处不在。超高速移动网络和高性能嵌入式设备（如智能手机，平板电脑和可穿戴计算设备）的出现已经实现了无处不在的生活，但它仍然专注于快速推出新产品和服务以满足客户的需求，而无需对软件进行充分的测试。嵌入式系统在可靠性方面。随着嵌入式设备越来越多地涉及我们的现实生活，出现了有关隐私侵犯和财务损失的各种问题。现在需要支持嵌入式软件的分析技术以提供可靠性。然而，由于缺乏嵌入式系统的动态二进制检测（DBI）等专用工具，这种研究受到阻碍。

DBI是一种从分析过程中获取上下文信息的技术，无需源代码用于目标程序的重新编译或二进制修改，因此甚至适用于商业现成软件[1,2]。它通常表示动态地将任意代码插入到目标进程中，该目标进程不会影响目标进程的原始流程。 因此，它支持各种分析技术，如漏洞分析和动态执行信息[3-5]。

用户级DBI有一些有用的工具，但大多数只适用于像Pin和DynamoRio这样的x86架构[6,7]。因此，我们提出了一种用于ARM架构的DBI工具作为支持各种分析技术的基本技术。我们的最终目标是对真实设备上的嵌入式软件进行崩溃分析，以确定需要本地执行环境而非模拟环境的可利用性。

Li和Wang提出了CBASS-TREE [8]，这是与我们的最终目标相关的类似工作。CBASS-TREE被认为是一种基于中间语言的多平台二进制分析框架，称为REIL（逆向工程中间语言）[9]，它支持污点分析和符号执行等各种功能。 但是，CBASS-TREE在跟踪ARM体系结构的指令方面存在局限性，因为DBI功能依赖于IDA Pro多处理器反汇编程序和调试程序提供的专用工具，它提供了广泛的分析功能，此外专用工具存在问题 GDB也有[10]。问题在Sect中引入。

因此，作为第一步，我们开发了一种动态二进制检测工具ARM-Tracer，用于ARM体系结构，在嵌入式系统中占据了大部分市场份额。我们的目标是从目标流程中提取每个用户级指令的上下文信息，以便进一步分析。 要做到这一点，应该解决由多线程环境引起的原子操作和信号处理等一些挑战[11,12]。详细信息将在下一节中介绍。

本文的结构如下。第2节介绍了我们的工具ARM-Tracer，Sect的设计和实现。图3显示了实验结果，Sect4 brie fl y涉及相关工作，最后是Sect5结束了论文。

2设计和实施

ARM-Tracer旨在用于基于ARM的32位Linux。它通过执行任务代码从目标进程中提取每个用户级指令的上下文信息，该任意代码的作用是每次执行指令时获取信息。

在运行目标进程期间，有两种不同的方法可以实现用户级DBI来执行arbi-trary代码。一个是在同一个过程中执行它。它将目标程序的本机指令转换为中间语言，并使用任意代码定位它们，以便在代码缓存中进行分析[13]。另一个是在一个单独的进程中完成它，这与调试器的附加和控制功能的方式相同。它们都有优点和缺点，但我们选择了第二个，尽管它会因上下文切换而产生更多开销。 它比第一个更容易实现并且更稳定[14,15]。它甚至可以在导致崩溃的程序上实现稳定的分析; 因此，它适用于安全分析。它可以通过系统调用ptrace实现，它不仅允许读取和写入调试对象进程的内存，而且还允许从调试对象进程获取信号。

在ARM上实现DBI作为单独的流程模型时，应该考虑四个方面-单步，原子操作，内核段和多线程信号。它们详细描述如下。

2.1单步

为了获得来自目标进程的每条指令的上下文信息，它应该在每次执行指令时获得对目标进程的控制。在实现单步功能方面，它完全取决于目标架构的硬件支持。在x86的情况下，它可以通过x86支持的陷阱flA简单地实现。当进程设置为陷阱位时，它会发出一个信号，在执行单个指令后立即将其传送到调试器，然后调试器可以在调试对象停止时执行任何操作。因此，在x86架构中实现单步功能是一项轻松的任务。另一方面，ARM不提供此类硬件支持。因此，ARM上的单步功能只能由软件实现，无需硬件支持。

为此，我们可以在没有硬件支持的环境中使用调试断点机制（DBM）进行单步[14]。DBM用断点指令替换下一个程序计数器（PC）的原始指令，以便在调试对象遇到断点时发出信号并给调试器一个控制。注意ARM有两种类型的执行状态-ARM 和拇指。它们一起使用并在一个过程中频繁地来回切换。两者都具有不同的指令大小-32位用于ARM，16位用于Thumb-这意味着用于获取的操作码大小根据当前执行状态确定。换句话说，ARM体系结构有两个不同的断点指令，而x86只有一个断点指令。因此，在设置适当的断点指令以获得控制之前，我们应该知道位于下一个PC的指令的执行状态。

表1 直接影响下一台PC的指令列表

|  |  |  |
| --- | --- | --- |
| ARM(32bit) | Thumb（16bit） | Thumb2（16/32bit） |
| B Label | B Label | B Label |
| BL Label | BL Label | BL Label |
| BX Reg | BX Reg | BX Reg |
| BLX Label | BLX Label | BLX Label |
| BLX Reg | BLX Reg | BLX Reg |
| RFE Reg | CBZ Reg，Label | TBB[RegA,RegB] |
| ADD PC,Reg，PC | CBNZ Reg，Label | TBH[RegA,RegB,LSL#1] |
| SUB PC,Reg，PC | ADD PC,SP,PC | RFE Reg |
| SUB PC,LR,#imm | MOV PC,Reg | SUBS PC,LR,#imm |
| MOVS PC,LR | LDR PC,[Reg] | LDR PC, [Reg] |
| LDR PC,[Reg] | LDM Reg，RegList | LDM Reg，RegList |
| LDM Reg，RegList |  |  |

ARMis的一步就是以这种方式执行的。首先，我们应该分析当前的操作码以确定下一个PC和下一个指令的执行状态。第二种是备份下一条指令，并根据执行状态用适当的断点指令替换它，然后继续调试对象。它会立即使调试对象点击断点。第三种是用保存的指令恢复指令以保证原始执行，并从第一次重复执行。

此过程的挑战是根据当前操作码的分析确定下一台PC。通常，下一台PC在ARM状态下增加4个字节，在Thumb状态下增加2个字节，但Thumb有一个名为Thumb2的扩展指令集。Thumb2指令集由16位和32位指令组成。此外，还有一些指令可以直接更改下一台PC。我们根据ARM编译器用户指南[16]和GDB源代码审计确定了指令。表1显示了由指令集分类的指令。因此，需要在操作码级别仔细分析这些指令以确定下一台PC。

此外，还应考虑条件位。与x86不同，ARM状态上的所有指令都有条件位，这是指令的四个最重要的位。它们的执行取决于当前程序状态寄存器（CPSR）CPSR保存有关最近执行的最终操作的信息。操作结果设置为CPSR的四个最重要的位，如表2所示。无论条件如何，都会获取每条指令，然后在执行之前将指令的条件位与CPSR的条件位进行比较。如果它不符合CPSR的指定条件，则指令由具有无操作（NOP）指令的处理器替换，然后执行。另一方面，Thumb状态的指令没有条件位。但是，Thumb状态有一个例外情况。Thumb2指令集的If-Then（IT）指令使以下四条指令成为条件。

表2 当前程序状态寄存器的条件标记表

|  |  |  |  |
| --- | --- | --- | --- |
| Bit | Name | Defination |  |
| [31] | N | Overflow flag； | 1=overflow in last operation  0=no overflow |
| [30] | Z | Zero flag； | 1=result of 0 in last operation  0=no zero result |
| [29] | C | Carry/borrow flag； | 1=carry or borrow in last operation  0=no carry or borrow |
| [28] | V | Negative/less than flag； | 1=result negative or less than in last operation  0=result positive or greater than |

当用断点指令替换下一台PC上的指令时，我们应该知道执行状态。仅通过Branch Exchange（BX）或Branch with Link和Exchange（BLX）指令更改状态。 状态取决于BX（BLX）的操作数的最不重要位。如果操作数的位为1，则表示下一条指令的状态为Thumb。因此，应为Thumb状态设置断点。另一方面，ARM状态的位为0。

2.2原子操作

ARM体系结构不提供单个原子指令，而Intel x86则提供单一原子指令。 相反，ARM提供原子操作的指令序列。该序列由两个指令组成 - LDREX（加载独占）和STREX（存储独占）。该序列用于原子地更新内存。例如，表3显示了使用该序列的互斥机制。序列重复进行内存检查，直到解锁为止。

当使用DBM单步执行序列时，由于指令的特性，它可能导致不必要的跟踪，这有时导致序列的有限循环。重要的是不必跟踪序列。因此，应该通过跳过序列来解决它，而不是跟踪序列的每个指令以减少跟踪时间并保证不影响原始流。

我们为处理序列做了一个处理程序。找到STREX指令后，将调用该处理程序。然后，处理程序寻找使序列重复的分支指令。最后，处理程序在下一个分支指令处设置断点以转义循环。

表3 原子操作的指令序列

|  |  |
| --- | --- |
| MOV rl，#0x1 | ;load the‘lock taken’value try |
| LDREX r0，[LockAddr] | ;load the lock value |
| CMP r0，#0 | ;is thelock free? |
| STREXEQ r0，r1，[LockAddr] | ;try and claim the lock |
| CMPEQ r0，#0 | ;did this succeed? |
| BNE try | ;no try again |
| … | ;yes we have the lock |

2.3内核段

存在可由用户级代码访问的专用内核段EVT（异常向量表）驻留在段中。 EVT包含异常处理程序的跳转表。当DBM设置的断点指令由处理器执行时，它会导致异常，使处理器将PC更改为EVT，最终调用适当的处理程序 异常处理程序完成后，PC会自动更改为异常发生。用户级代码不直接执行异常处理。它由处理器处理，因此不需要处理与异常相关的指令。

但是，有一些功能可以帮助除EVT之外的共享段中的用户空间。这些函数用于为用户空间提供一些需要内核支持的操作，这是由于未实现的本机功能（如内存屏障）。因此，函数可以由用户级代码直接调用，但问题是不允许用户级代码修改内核段中的指令。由于需要用断点指令替换现有指令以稳定跟踪指令，因此应仔细处理段中的功能。为此，我们模拟了细分中的功能。如果下一台PC指向段中的特定功能，则ARM-Tracer会模拟该功能并在用户空间的返回地址处设置断点指令。它可以保证无缝跟踪。

2.4多线程信号

多线程环境中存在问题。这是关于信号处理。由于代码段由进程中的所有线程共享，因此即使断点设置为跟踪特定线程，每个线程也可以命中断点。这在使用DBM的环境中是不可避免的。因此，它应该完全设计用于处理由线程引起的SIGSEGV，SIGSTOP，SIGTRAP，SIGILL和EXIT等信号。它应该能够区分信号来自哪个目标线程和其他线程，然后根据线程类型执行不同的作业。

图1显示了ARM-Tracer的整个流程图。 它旨在跟踪用户选择的特定线程。 首先，它会停止所有线程。 它选择目标线程，并获取目标线程的寄存器上下文。 然后，它通过分析具有寄存器上下文的当前PC的操作码来确定下一个PC。当找到原子操作的序列时，处理程序处理它。下一步是在下一台PC上设置断点，以通知ARM-Tracer线程到达断点。要捕获信号，请使用等待系统调用。等待系统调用逐个从信号队列中获取信号。这部分重要的是可以有其他线程以及目标线程发出的信号。因此，需要根据目标或其他线程适当地处理信号，如下所述。

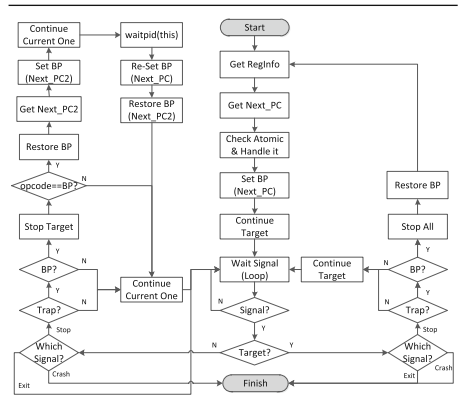


图1 ARM-Tracer的整个流程图

目标线程当目标线程发送信号时，必须首先查看信号的类型。在SIGTRAP的情况下，它检查信号是否由断点产生。如果不是，则表示信号是由线程创建或进程fork产生的。但是，如果信号结果是断点，它会停止所有线程，以避免在使用原始断点恢复断点之前由其他线程引起的副作用。恢复之后，重复确定下一台设置断点的PC的阶段，直到目标线程完成或崩溃为止。

其他线程当其他线程发送信号时，只有断点引起的SIGTRAP才被认为是重要的。仅通过立即继续当前线程来处理其他信号。如果断点被识别，它首先停止目标线程以防止目标线程通过设置断点的当前点。然后，它检查当前PC上的操作码以查看它是否再次是断点指令。这是因为操作码可以由其他线程预先用原始操作码恢复。在这种情况下，当前线程只是继续。当操作码仍然被识别为断点时，ARM-Tracer会用原始的恢复它。这里重要的是刚刚恢复的原始指令应该在当前线程通过后再次用断点替换，这样可以保证跟踪目标线程没有错过任何一条指令。为此，需要再次确定下一个PC以将断点设置为第二个点，并继续当前线程。同时，它等待当前的第二个点达到断点。一旦当前的断点到达临时设置的断点，它就会在第一点重置断点以捕获目标线程，然后在第二点恢复当前断点与原始断点。最后，继续当前的线程。所有这些步骤都可以保证跟踪目标线程，而不会遗漏与其他线程协调的单个指令。

3实验和结果

我们在真实设备上进行了两次实验验证ARM-Tracer，该设备是智能手机Nexus4-ARMv7，四核1.5GHz，2GB DDR2，Android4.3和Linux内核3.4.0。首先，我们通过跟踪BusyBox的各种UNIX实用程序来测试单线程环境中的DBI功能。其次，我们已将我们的工具应用于具有多个线程的Android应用程序[17]。 我们的工具应该将跟踪日志保存到包含每个指令的反汇编和寄存器上下文的文件中。在拆卸时，它采用了Capstone的优势，这是一个轻量级的多平台，多架构拆卸框架[18]。

3.1实验1

我们使用BusyBox提供的实用程序进行了一些测试，以验证我们工具的DBI功能。BusyBox将各种常见UNIXutilities的小版本组合到一个可执行文件中，为嵌入式系统提供了一个相当完整的环境[19]。表4显示了跟踪BusyBox 1.23.1的一些实用程序的结果。我们测试了许多实用程序，但由于缺少空间，它只代表其中一些实用程序。

它们中的每一个都在一个线程中执行。我们可以看到每个实用程序中发现大约30个助记符。我们还使用GDB逐步跟踪它们进行比较。因此，我们确认结果与我们的工具创建的日志相同，这意味着我们的工具使用单步执行调试断点机制的DBI功能可靠使用。

3.2实验2

我们将我们的工具应用于anAndroid应用程序-PolarisOf fi ce 6.0.1-这是一个广泛使用的应用程序，用于读取，编辑和共享各种文档。在这种情况下，我们使用输入文件执行测试，使目标应用程序崩溃。将分析通过模糊目标应用程序生成的崩溃，以确定可利用性作为我们未来的工作。Android应用程序基本上创建了许多线程。此外，目标应用程序还会创建其他线程。其中一个负责处理输入文件。要在解析和处理输入文件时从导致崩溃的线程中提取跟踪日志以进行分析，我们需要识别试图打开输入文件的所有线程的特定线程。一旦捕获了输入文件的时刻，ARM-Tracer就开始指令跟踪，直到目标崩溃为止。这里重要的是它应该保证其他线程按预期运行而不会出现任何错误，否则会导致副作用。

表4 跟踪BusyBox的一些UNIX实用程序的结果

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cal | | Date | | Id | | Pwd | | |
| 1 | CMP | 12585 | CMP | 2378 | CMP | 2455 | LDR | 911 |
| 2 | LDR | 11960 | LDR | 1866 | LDR | 2215 | CMP | 883 |
| 3 | MOV | 10178 | ADD | 1486 | B | 1667 | ADD | 778 |
| 4 | ADD | 7761 | B | 1407 | ADD | 1437 | B | 658 |
| 5 | B | 7339 | MOV | 1369 | MOV | 1286 | MOV | 592 |
| 6 | BL | 3182 | STR | 701 | STR | 795 | STR | 406 |
| 7 | SUB | 2933 | SUB | 477 | SUB | 327 | SUB | 230 |
| 8 | STR | 2748 | BL | 387 | BL | 277 | STM | 171 |
| 9 | BX | 2408 | SUBS | 288 | STM | 247 | BL | 166 |
| 10 | SUBS | 2141 | TXT | 281 | PUSH | 181 | TST | 113 |
| 11 | TST | 1089 | BX | 276 | ORR | 176 | PUSH | 102 |
| 12 | POP | 819 | STM | 217 | POP | 174 | POP | 96 |
| 13 | ORR | 803 | ORR | 207 | SUBS | 169 | SUBS | 91 |
| 14 | STM | 791 | AND | 141 | TST | 162 | BX | 85 |
| 15 | PUSH | 692 | POP | 126 | AND | 157 | AND | 77 |
| 16 | AND | 646 | PUSH | 123 | LSL | 116 | ORR | 57 |
| 17 | MVN | 433 | EOR | 96 | BX | 106 | MVN | 57 |
| 18 | RSB | 397 | LSL | 85 | RSB | 83 | EOR | 48 |
| 19 | LSL | 387 | CLZ | 75 | MVN | 66 | RSB | 40 |
| 20 | CLZ | 313 | MVN | 73 | LSR | 64 | BLX | 24 |
| 21 | CMN | 310 | RSB | 65 | EOR | 37 | LSL | 22 |
| 22 | EOR | 215 | LSR | 63 | BIC | 33 | LSR | 20 |
| 23 | LSR | 209 | SRS | 62 | CLZ | 30 | CMN | 19 |
| 24 | SRS | 174 | MUL | 38 | SRS | 29 | MLA | 18 |
| 25 | MUL | 170 | CMN | 26 | CMN | 25 | LDM | 16 |
| 26 | BIC | 165 | BIC | 20 | BLX | 20 | CLZ | 15 |
| 27 | SVC | 150 | LDM | 20 | LDM | 19 | BIC | 12 |
| 28 | TEQ | 122 | MLA | 19 | SVC | 17 | SVC | 11 |
| 29 | ADC | 114 | BLX | 19 | MLA | 14 | ASR | 4 |
| 30 | MLA | 98 | SVC | 18 | MUL | 5 | MUL | 2 |
| 31 | LDM | 96 | ASR | 12 | ASR | 4 | TEQ | 1 |
| 32 | UMULL | 57 | TEQ | 9 | TEQ | 3 | SRS | 1 |
| 33 | ASR | 42 | ADC | 6 |  |  |  |  |
| 34 | BLX | 30 | UMULL | 3 |  |  |  |  |
| 35 | SMLAL | 19 | SMLAL | 1 |  |  |  |  |
| Total instr | 71576 |  | 12440 |  | 12396 |  | 5726 |  |
| Taken time | 25s |  | 4s |  | 4s |  | 2s |  |

表5 Polaris Of F ce Viewer的跟踪崩溃样本的结果; 前25条说明

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Crash#1 | | | Crash#2 | | Crash#3 | | Crash#4 | |
| 1 | LDR | 167211 | LDR | 226551 | LDR | 1181296 | LDR | 1720569 |
| 2 | B | 97006 | MOV | 137710 | B | 919524 | B | 1306681 |
| 3 | MOV | 93594 | B | 131991 | MOV | 810170 | MOV | 1152527 |
| 4 | CMP | 78411 | STR | 116576 | CMP | 679755 | CMP | 996965 |
| 5 | ADD | 76122 | CMP | 105656 | STR | 607233 | STR | 864062 |
| 6 | STR | 61001 | ADD | 101129 | ADD | 576831 | ADD | 796701 |
| 7 | BL | 33635 | BL | 46437 | BL | 206183 | BL | 293650 |
| 8 | ORR | 20999 | POP | 26863 | POP | 163935 | POP | 245785 |
| 9 | BX | 19566 | BX | 25767 | PUSH | 145101 | PUSH | 216025 |
| 10 | POP | 18917 | PUSH | 25439 | BX | 113769 | BX | 158419 |
| 11 | PUSH | 17724 | ORR | 23696 | SUB | 99135 | SUB | 144901 |
| 12 | SUBS | 13057 | SUBS | 16346 | ORR | 97837 | LSL | 124347 |
| 13 | PLD | 12127 | SUB | 15985 | LSL | 86641 | ORR | 121853 |
| 14 | SUB | 11940 | PLD | 13354 | SUBS | 72901 | SUBS | 104935 |
| 15 | LSL | 11136 | MOVT | 13277 | PLD | 71866 | PLD | 103131 |
| 16 | MUL | 10169 | LSL | 12569 | AND | 69218 | AND | 100374 |
| 17 | STM | 9785 | STM | 11015 | VMOV | 69176 | RSB | 81407 |
| 18 | LDM | 7300 | MUL | 10181 | CBZ | 59162 | MOVT | 77406 |
| 19 | MOVT | 6654 | LDM | 8503 | RSB | 58712 | VMOV | 72203 |
| 20 | RSB | 5812 | RSB | 8015 | MOVT | 55333 | LSR | 72054 |
| 21 | AND | 4982 | AND | 7527 | LSR | 49573 | TST | 65325 |
| 22 | LSR | 3386 | BIC | 3523 | TST | 45297 | CBZ | 59765 |
| 23 | TST | 2917 | TST | 3359 | STM | 41020 | STM | 55251 |
| 24 | BIC | 2623 | LSR | 3223 | VLDR | 40206 | BIC | 54626 |
| 25 | SRS | 1922 | UXT | 2153 | BIC | 35048 | BLX | 53776 |

当使用GDB-version 7.10.1跟踪特定线程时，这是2015年12月[20]的最新版本，当偶尔发生线程切换时，它经常会错过目标线程的某些指令。因此，GDB支持一种称为调度程序锁定的特殊选项。该选项允许通过避免线程切换来一致地跟踪特定线程，但它不保证运行其他线程，这可能在多线程环境中导致副作用。

表5显示了跟踪Android应用程序崩溃样本的结果。与BusyBox的测试结果不同，超过100个唯一指令被识别。 因此，我们只代表最常出现的25条指令。我们也按组分类（见表6）。数据处理和内存访问组中的指令在每种情况下都占70％以上。我们可以看到我们的工具每秒可以跟踪超过4000条指令。该表的底线表示被调用以跳过与原子操作相关的序列的原子处理程序的数量。

4相关工作

x86体系结构中有许多相关的工作，但ARM体系结构中的工作很少，如下所示。Pin是英特尔开发的一个着名的动态二进制重写系统，于2006年移植到ARM，但目前还没有[21]。

DynamoRIO是一个动态代码处理框架，可以在运行时转换程序中代码的任何部分。它不支持ARM体系结构，但管理DynamoRIO的社区最近开始为ARM移植它。

Valgrind是一个广泛使用的框架，它提供了一些基于DBI的工具来自动检测内存泄漏等与内存相关的错误[22]。Valgrind使用模拟技术来检测用户级指令，这使得错误程序有所不同。例如，如果一个程序由于访问无法寻址的内存而崩溃，那么在Valgrind下运行时，该主题可能不会无法寻址。这是Valgrind工作方式的本质。 因此，它不适合与Valgrind进行碰撞分析[23]。

ADBI是一种Android动态二进制检测器，用于挂钩ARMbinary的一个函数来监视和修改参数，而不是用于指令跟踪[24]。

在调试器和debuggeemodel方面，GDB似乎与我们的工具非常相似，但GDB由于其复杂的数据结构而具有大量开销，除了当前的GDB之外，果断地，不能完全单步执行特定的线程。在发生线程切换时发现一些指令。GDB还支持跟踪特定线程的选项，但它有副作用，它不能保证Sect中提到的其他线程的和谐执行。因此，它可能会在多线程环境中导致意外结果。

表6 按组分类的总指令的分类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Crash#1 | | Crash#2 | Crash#3 | Crash#4 |
| Data processing | 334837（41.89%） | 456225（41.00%） | 2774614（41.46%） | 3959702（41.84%） |
| Memory access | 296033（37.04%） | 430293(38.66%) | 2250840（33.63%） | 3258189（34.42%） |
| Branch | 151610（18.97%） | 206506(18.55%) | 1334726（19.94%） | 1878030（19.84%） |
| Multiply | 10350（1.29%） | 10383(0.93%) | 12745（0.19%） | 16085（0.17%） |
| Packing | 3429（0.42%） | 4686(0.42%) | 20636（0.30%） | 36071（0.38%） |
| Coprocessor | 1112（0.14%） | 1975(0.18%) | 9070（0.14%） | 12824（0.14%） |
| Control | 53（0.01%） | 50(0.00%) | 22224（0.33%） | 24431（0.26%） |
| Miscellaneous | 24（0.00%） | 26(0.00%) | 478（0.00%） | 985（0.01%） |
| NEON/VFP | 1728（0.21%） | 2613（0.23%） | 266788（3.98%） | 277204（2.93%） |
| Total instr traced | 799.176 | 1112757 | 6692121 | 9463521 |
| Taken time | 215s | 292s | 1454s | 2365s |
| #of atomic handling | 46 | 54 | 2798 | 3982 |

5结论和未来的工作

我们开发了一个工作在ARM体系结构上的简洁轻量级DBI工具，它可以从目标进程甚至特定线程中提取每条指令的所需上下文信息。我们还考虑了多线程环境。我们通过将我们的工具应用于常见的UNIX实用程序和Android应用程序，进行了一些验证实验。作为评估的结果，我们发现它与其他线程协调正常工作而不影响我们设计的目标线程的原始上下文。我们期望它将用于进一步的研究，如嵌入式软件的安全性分析，这是我们的最终 我们计划在将来通过污点分析来执行崩溃分析以确定可利用性。对于污点分析，跟踪与数据传播相关的每条指令的能力是必不可少的。因此，我们将基于我们的DBI工具执行污点分析。

致谢这项研究得到了由科学，ICT和未来规划部（NRF-2014M3C4A7030648）资助的韩国国家研究基金会（NRF）的下一代信息计算发展计划的支持。

参考文献

1. Lee D, Kim I, Kim J, Jun HK, Kim WT, Lee S, Eom YI (2013) Light-weight kernel instrumentation framework using dynamic binary translation. J Supercomput 66(3):1613–1628

2. Henderson A, Prakash A, Yan L, Hu X, Wang X, Zhou R, Yin H (2014) Make it work, make it

right, make it fast: building a platform-neutral whole-system dynamic binary analysis platform. In:

Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp 248–258

3. Nicholas N (2004) Dynamic binary analysis and instrumentation. Dissertation, University of Cam-bridge

4. Zaddach J, Bruno L, Francillon A, Balzarotti D (2014) Avatar: a framework to support dynamic

security analysis of embedded systems ﬁrmwares. In: Proceedings of the 21st Symposium on Network and Distributed System Security

5. EnckW, Gilbert P, Han S, Tendulkar V, Chun B, Cox L, Jung J,Mcdaniel P, Sheth A (2014) TaintDroid:an information-ﬂow tracking system for realtime privacy monitoring on smartphones. ACM Trans Comput Syst 32(2):5

6. Luk CK, Cohn R, Muth R, Patil H, Klauser A, Lowney G,Wallace S, Reddi VJ, Hazelwood K (2005)Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of the 2005 ACMSIGPLAN Conference on Programming Language Design and Implementation, vol 40, pp190–200

7. Uh GR, Cohn R, Yadavalli B, Peri R, Ayyagari R (2006) Analyzing dynamic binary instrumentation overhead. In: WIBA Workshop at ASPLOS

8. Li L, Wang C (2013) Dynamic analysis and debugging of binary code for security applications. In:Proceedings of the 4th International Conference on Runtime Veriﬁcation, Springer, Berlin

9. DullienT, Porst S (2009) REIL: a platform-independent intermediate representation of disassembled code for static code analysis. In: CanSecWest

10. Mihajlovi B, Zilic Z, Gross WJ (2014) Dynamically instrumenting the QEMU emulator for linux process trace generation with the GDB debugger. ACM Trans Embed Comput Syst 13(5s), article no.167

11. LiuC,Chen J,YangW,HsuW(2014)Dynamically translating binary code formulti-threaded programs using shared code cache. J Electron Sci Technol 04:434–438

12. Hong D, Wu J, Yew P, Hsu W, Hsu C, Liu P, Wang C, Chung Y (2014) Efﬁcient and retargetable dynamic binary translation on multicores. IEEE Trans Parallel Distrib Syst 25(3):622–632

13. Payer M, Kravina E, Gross T (2013) Lightweight memory tracing. In: USENIX annual technical conference, pp 115–126

14. Paxson V (1990) A survey of support for implementing debuggers

15. Rodriguez R, Artal J, Merseguer J (2014) Performance evaluation of dynamic binary instrumentation frameworks. IEEE Latin Am Trans 12(8):1572–1580

16. ARM (2015) ARM compiler armasm user guide version 6.3.http://infocenter.arm.com/help/topic/ com.arm.doc.dui0801d/DUI0801D\_armasm\_user\_guide. Accessed 8 Dec 2015

17. Sun H, Zheng Y, Bulej L, Villazn A, Qi Z, Tma P, Binder W (2015) A programming model and

framework for comprehensive dynamic analysis on Android. In: Proceedings of the 14th International Conference on Modularity, ACM, pp 133–145

18. Capstone: lightweight multi-platform, multi-architecture disassembly framework. http://www.

capstone-engine.org. Accessed 8 Dec 2015

19. van der Kouwe E, Giuffrida C, Tanenbaum AS (2014) Evaluating distortion in fault injection experi-ments. In: 2014 IEEE15th International SymposiumonHigh-Assurance SystemsEngineering (HASE),pp 25–32

20. GDB 7.10.1. http://www.gnu.org/software/gdb/download. Accessed 8 Dec 2015

21. Kim H, Klauser A (2006) A dynamic binary instrumentation engine for the ARMarchitecture. In: Pro-

ceedings of the 2006 InternationalConference on Compilers,Architecture and Synthesis for Embedded Systems

22. Nicholas N, Seward J (2007) Valgrind: a framework for heavyweight dynamic binary instrumentation.ACM Sigplan Not 42(6):89–100

23. Valgrind FAQ(2015)On crash. http://valgrind.org/docs/manual/faq.html#faq.crashes.Accessed 8Dec 2015

24. ADBI: Android dynamic binary instrumentation. https://github.com/crmulliner/adbi. Accessed 8 Dec 2015