第 69 卷 第 6 期
2023 年 12 月

武汉大学学报(理学版)
J. Wuhan Univ. (Nat. Sci. Ed.)

Vol. 69 No. 6
Dec. 2023,690~698

DOI: 10.14188/j.1671-8836.2022.0206

一种基于Unicorn的UEFI DXE驱动模拟执行方法

操方涛,傅建明†,李子川

空天信息安全与可信计算教育部重点实验室,武汉大学 国家网络安全学院,湖北 武汉 430072

收稿日期:2022-09-29 † 通信联系人 E-mail:jmfu@whu.edu.cn
基金项目:国家自然科学基金(61972297,62172308,62172144)
作者简介:操方涛,男,硕士生,现从事二进制程序漏洞挖掘方面的研究。E-mail:fangtao.cao@whu.edu.cn

摘 要:UEFI(uniform extensible firmware interface,统一可扩展固件接口)标准近年来已被广泛应用于计算机系统。针对UEFI固件的动态分析方法严重受限于运行环境的扩展性的问题,提出了一种高效的UEFI固件模拟执行方法DxeEmulator。该方法可以自动解析固件文件系统,完善DXE驱动运行所需的数据依赖和代码依赖,并按照驱动之间的依赖关系调度驱动。在6个品牌生产商的665个固件上测试了DxeEmulator的效果,相比现有工作基本块覆盖数量有大幅提升。基于DxeEmulator构造了一个漏洞挖掘模块,在数据集中共发现了12个缓冲区溢出漏洞,其中包含9个0-day漏洞。

关 键 词:UEFI模拟执行;Unicorn;UEFI固件解析;协议依赖;驱动调度;漏洞检测

中图分类号:TP368 **文献标志码**:A **文章编号**:1671-8836(2023)06-0690-09

A Unicorn-Based UEFI DXE Driver Emulation Method

CAO Fangtao, FU Jianming†, LI Zichuan

Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, Hubei, China

Abstract: UEFI (uniform extensible firmware interface) specifications have found extensive deployment in computer systems. The poor scalability of the running environment severely limits dynamic analysis methods on UEFI firmware. To solve this challenge, we introduce DxeEmulator, a high-efficiency UEFI emulator. DxeEmulator is capable parsing the firmware file system, identifying the data dependencies and code dependencies, and automatically dispatching drivers. We conducted an evaluated DxeEmulator using a real-world dataset of 665 firmware images across 6 vendors. The results demonstrate that DxeEmulator achieves a notably improved coverage compared to existing work. Additionally, we implemented a vulnerability detection module based on DxeEmulator, uncovering 12 buffer overflow vulnerabilities, including 9 0-day vulnerabilities.

Key words: UEFI (uniform extensible firmware interface) emulation; Unicorn; UEFI firmware parsing; protocol dependency; driver scheduling; vulnerabilities detection

0 引 言

UEFI[1](uniform extensible firmware interface,统一可扩展固件接口)是Intel等生产商在BIOS(basic input output system,基本输入输出系统)之上提出的接口标准。UEFI具有开发便捷、性能高、扩展性强等特点,目前已广泛应用于计算机系统,同时也引入了更多的攻击面。

随着应用层、操作系统层的保护措施日臻完善,攻击者逐渐开始利用位于操作系统底层且缺乏保护的UEFI进行渗透,针对UEFI固件的攻击事件频发[2~5]。如果UEFI固件被攻破,UEFI启动之后阶段的安全性都无法得到保障。因此,UEFI固件安全关乎整个计算机系统的安全。

引用格式:操方涛,傅建明,李子川. 一种基于Unicorn的UEFI DXE驱动模拟执行方法[J]. 武汉大学学报(理学版),2023,69(6):690-698. DOI:10.14188/j.1671-8836.2022.0206.

CAO Fangtao, FU Jianming, LI Zichuan. A Unicorn-Based UEFI DXE Driver Emulation Method [J]. *J Wuhan Univ (Nat Sci Ed)*, 2023, 69(6): 690-698. DOI:10.14188/j.1671-8836.2022.0206(Ch).

UEFI的工作过程分为7个阶段[1,6]:SEC(security phase,安全验证)、PEI(pre-efi initialize phase,预efi初始化)、DXE(driver execution environment,驱动执行环境)、BDS(boot device selection,启动设备选择)、TSL(transient system load,操作系统加载前期)、RT(run time,运行时)和AL(after life,系统灾难恢复)。其中DXE是UEFI启动过程中的核心阶段,该阶段包含了各种驱动程序。这些驱动程序会读取NVRAM(non-volatile random-access memory,非易失性随机访问内存)和PCD(platform configuration database,平台配置数据库)中储存的配置信息,用于注册设备驱动服务和初始化主板上的芯片组。DXE驱动的功能复杂且提供了大量数据接口,这导致DXE驱动更容易出现漏洞,DXE驱动漏洞已逐渐成为UEFI固件漏洞的主要来源。

DXE驱动支持模块化开发,不同模块之间通过协议进行交互。驱动既可以是协议的提供者,也可以是协议的使用者,一个驱动只有在其所需的协议都被安装后才能运行。静态分析方法难以准确构造控制流和数据流,无法处理驱动之间的协议依赖关系。因此,动态分析方法更适用于模块化设计的DXE驱动。但是动态分析方法受限于执行环境的可扩展性,往往无法应用于大规模分析。

为了解决DXE驱动分析中静态分析不准确而动态分析扩展性差的难题,本文实现了一种UEFI DXE驱动模拟执行工具DxeEmulator,用于辅助研究人员分析UEFI固件。本文的主要工作如下:

1) 实现了一种UEFI固件解析工具提取UEFI固件中的变量数据和驱动文件,用于解决DXE驱动运行中的数据依赖问题。

2) 提出了一种三阶段的DXE驱动文件动态调度方法,用于解决DXE驱动运行中的代码依赖问题。并实现了一个全自动的UEFI DXE驱动模拟执行系统DxeEmulator,可用于大规模的测试和漏洞挖掘。据我们所知,DxeEmulator是第一个考虑了DXE驱动之间依赖关系的模拟执行工具。

3) 收集了6个品牌生产商的665个UEFI固件,并在该数据集上评估了DxeEmulator的效果。

1　相关研究

目前,针对UEFI的分析方法可以分为静态分析和动态分析两大类。动态分析方法会借助模拟器或者实体机运行固件程序。

1.1　静态分析

UEFI静态分析方法一般借助数据流分析和控制流分析还原程序行为。这类方法缺乏真实环境信息,往往会表现出较高的误报。Alex等[7]实现了UEFI二进制程序静态分析工具efiXplorer,该工具能够识别UEFI中的服务函数调用,并能够识别缓冲区溢出漏洞。但efiXplorer仅通过识别函数调用和参数信息检测缓冲区溢出,无法准确判断危险行为调用点是否可达,误报率非常高。Oleksandr等[8]在开发板上运行UEFI固件,并在SMRAM(system management interrupt,系统管理中断)被建立起来之后通过ITP(in-target probe,Intel JTAG调试工具)dump SMRAM和SMI(system management interrupt,系统管理中断)Handler的入口点,然后通过符号执行查找SMI处理函数中的漏洞。但这种方法仅能应用于开发板能运行的特定UEFI固件。Yin等[9]总结了UEFI固件中的SMM(system management mode,系统管理模式)特权提升漏洞的根本原因,并提供了静态分析工具SPENDER,用于识别可造成SMM特权提升的“Escaped Reference”,该方法通过解析UEFI驱动程序中的协议,构造驱动之间的调用图并进行静态污点分析,最终发现了36个特权提升漏洞。但该方法无法准确还原驱动中的各种协议结构体,这影响了结果的准确性。

1.2　动态分析

动态分析方法借助模拟执行环境或真实硬件获取DXE驱动运行时信息,能够补充静态分析缺失的环境信息。但这类方法受限于运行环境的扩展性,往往只能分析特定固件。曾令静[10]对篡改UEFI固件中的变量带来的威胁进行了分析总结,并提出了一种通过检查变量属性和数据避免UEFI变量被篡改的策略,作者在EDK2[11]的NT32模块上验证后发现这些策略能有效对抗各种UEFI变量篡改攻击,但该方法仅能应用于EDK2标准固件。郭东奥[12]对EDK2源码进行了静态分析,从中提取协议函数和这些函数的参数类型,然后将源码编译为驱动文件,利用TriforceAFL[13]Fuzz这些驱动文件中的协议函数以检测漏洞。该方法需要获取驱动程序的源码,且只能分析TriforceAFL能够模拟执行的固件。Yang等[14]利用simics平台运行EDK2中的固件,然后使用AFL Fuzz固件中的SMI处理函数,并在固件中发现了两个未检查地址范围的指针解引用,但该方法只能应用于smics专用的固件,无法扩展到真实设备的固件分析。

1.3 UEFI模拟执行

为了解决动态分析受限于运行环境的问题,研究人员提出了许多模拟执行方案,但这些方案仍旧无法满足安全研究的需求。EDK2[11]中提供的模拟执行模块EmulatorPkg可以执行EDK2编译的UEFI固件,并且可以运行开发者自己编写的驱动,但该模块只支持调试版的驱动文件,无法扩展到其他UEFI固件上。QEMU[15]支持完整UEFI固件的模拟执行,并对一些基本的硬件进行了抽象,可以满足部分硬件访问需求,但QEMU仅支持模拟执行EDK2编译出的部分固件。efi_dxe_emulator[16]是一个基于Unicorn[17]的模拟执行工具,它实现了DXE阶段用到的各种服务,并支持运行和调试单个驱动文件,但efi_dxe_emulator目前已经停止维护,无法直接运行。Qiling[18]是一个基于Unicorn的模拟执行工具,它的服务实现完善,对不依赖其他协议的驱动运行效果较好,但Qiling不能进行固件的解析,它无法解决驱动文件执行过程中的数据依赖问题,也不能解析驱动之间的依赖关系。

2 存在的问题

尽管研究人员已经提出了一些UEFI模拟执行方案[15~18],但由于无法满足固件运行中的数据依赖和代码依赖,这些方案都无法有效扩展到其他固件上。本文在分析UEFI执行过程后,总结出了DXE驱动模拟执行时存在的4个问题。

2.1 UEFI固件解析

现有的DXE驱动模拟执行方法都没有提供固件解析模块,这些模拟执行工具运行过程中需要借助其他解析工具解析固件。UEFITool[19]对固件文件系统的各种结构的覆盖最全面,但该工具仅提供了解压功能,而没有对提取的变量数据和可执行文件进行归类;uefi-firmware-parser[20]支持提取UEFI固件中的可执行文件,但对NVRAM变量等关键信息的提取支持不完善;uefi_retool[21]可以提取UEFI固件中的可执行文件和文件名信息,但舍弃了变量数据、文件类型等其他信息。这些工具都不能提供UEFI固件模拟执行所需的全部信息。

2.2 外设访问

UEFI作为连接硬件和操作系统的桥梁,会和硬件进行大量交互。因此,DXE驱动中包含了大量读写外设的指令。但不同主板上的硬件各不相同,不同UEFI固件中的外设访问行为也不固定,无法用统一的方法表示。外设行为可能会影响执行流程,直接忽视外设访问可能导致程序陷入等待外设响应的死循环或直接崩溃。

2.3 DXE驱动调度

DXE驱动之间存在依赖关系,DXE驱动只有在其依赖的协议均被加载到内存中后才能成功运行。一个示例如图1所示,UsbRtDxe依赖的两个协议分别在Uhcd和PcdDxe中安装,而Uhcd同样依赖于PcdDxe安装的协议。这种依赖关系在UEFI开发中由开发者指定并被编译到固件中。但固件中的依赖信息仅包含协议的GUID,通过协议的GUID无法直接对应到驱动文件。efiXplorer[7]和uefi_retool[21]通过静态分析跟踪协议安装函数,识别每个驱动文件安装了哪些协议,但这两个工具存在以下缺陷:1) 二者均通过追踪传入的系统表识别协议安装函数,但静态分析无法准确识别所有函数;2) 二者都会识别DXE驱动中的所有服务调用,但识别出的调用位置在实际运行中可能并不可达,会导致误报。

2.4 DXE阶段的服务

DXE驱动在执行过程中会调用各种服务,这些服务的功能包括内存管理、协议管理、变量获取、时钟获取等。这些服务依赖于PEI阶段初始化的资源和特定的内存布局,而在模拟执行DXE驱动时跳过

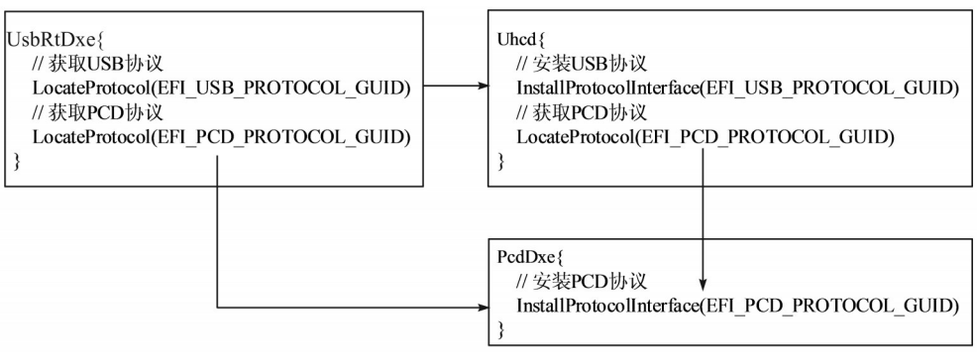

图1 驱动依赖关系示例

Fig. 1 An example of dependencies among drivers

了前置阶段,因此内存中并没有这些服务。缺乏服务将导致驱动程序无法正常运行。

以上问题中与数据相关的问题(2.1和2.2节)产生的原因可能是缺失了UEFI运行环境中的各种全局数据,导致运行过程中产生非法地址访问或者控制流异常;与代码相关的问题(2.3和2.4节)产生的原因可能是缺失了运行时所需的函数或协议,导致模拟执行中出现控制流异常。

3　设计方案

为了应对上述问题,辅助研究人员对UEFI固件进行安全分析,本文提出了一种按照驱动依赖关系调度执行顺序的UEFI固件模拟执行方法,并基于CPU模拟器Unicorn,实现了系统原型DxeEmulator,其工作流程如图2所示。

为了解决数据依赖问题,DxeEmulator从固件中提取出NVRAM和PCD变量数据,并在模拟执行时将变量数据提供给DXE驱动;同时,DxeEmulator实时监控DXE驱动的外设访问行为,提供外设返回值满足程序继续执行的条件。为了解决代码依赖问题,DxeEmulator根据驱动文件、协议依赖关系和Apriori配置文件调度DXE驱动的执行顺序;对于各种服务函数,DxeEmulator还提供了它们的高级实现。

3.1　解析UEFI固件

为了提取DXE驱动运行所需的各种信息,本文在uefi-firmware-parser[20]的基础上,结合UEFITool[19]和EDK2[11]中的相关结构体实现了一种全面的UEFI固件关键信息提取工具。提取的文件和信息包括:

1) DXE驱动文件,位于文件系统中各个文件的PE32节中[22];

2) NVRAM变量,包含操作系统启动选项和其他运行时的关键数据;

3) PCD数据库,不同平台用到的配置参数;

4) Apriori配置文件信息,UEFI固件中预设的驱动执行顺序,基础DXE驱动列表,该配置文件中的驱动必须在其他驱动之前执行。

3.2　模拟外设访问

DXE驱动访问外设时,需要在外设寄存器/外设内存中写入参数,等待外设处理完毕后再读取输出。本文为了避免缺乏外设导致的程序崩溃,监控并处理了外设访问。

x64 UEFI固件的外设访问机制包括PMIO(port-mapped I/O,端口映射I/O)和MMIO(memory-mapped I/O,内存映射I/O)[23]。PMIO是将外设寄存器映射到指定的I/O地址空间,通过特定机器指令访问外设,x64程序会使用in/out指令读取这一类外设。MMIO则是将外设映射到物理地址空间中,然后直接使用内存操作指令处理外设数据,由硬件负责将对应操作传递到外设中。

针对PMIO外设访问,DxeEmulator监控in/out指令的使用并返回随机值。针对MMIO外设访问,DxeEmulator只关注PCI/PCIe(peripheral component interconnect express)设备。PCI设备通过PCI配置空间地址端口和数据端口处理外设数据。

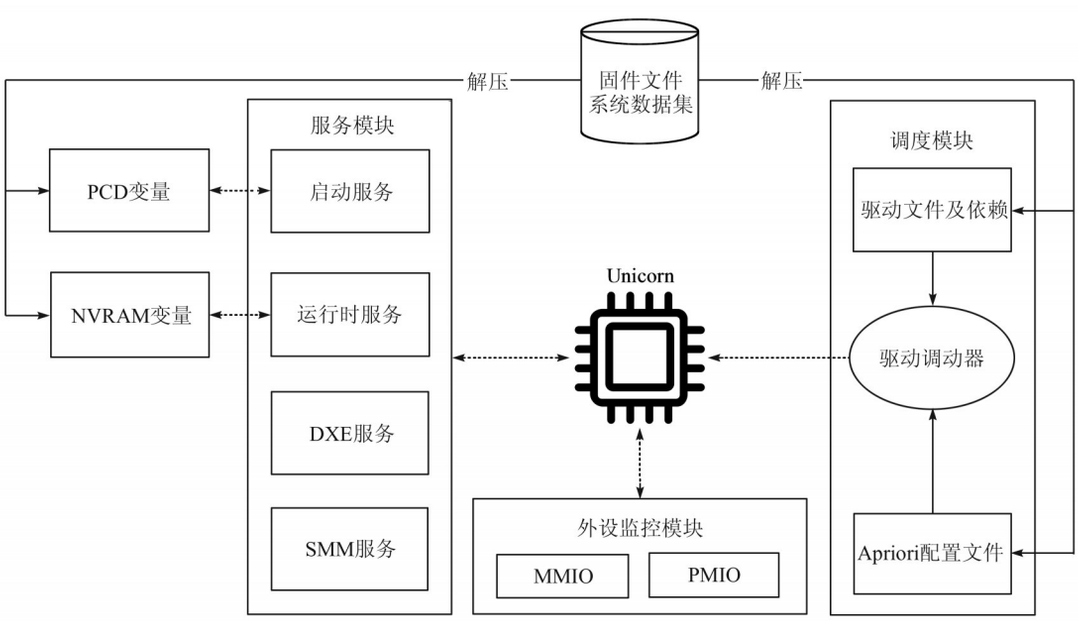

图2　DxeEmulator工作流程

Fig. 2　DxeEmulator workflow

DxeEmulator会捕获PCI配置空间地址端口的信息并计算目标地址。当程序读数据端口时,DxeEmulator会读取对应地址数据并写回数据端口。通过PCIe接口访问的地址则会在编译时被直接硬编码到DXE驱动中,DxeEmulator映射了PCIe标准中的内存段,保证了PCIe的MMIO访问可以成功进行。

3.3 调度DXE驱动文件

为了满足DXE驱动之间的依赖关系,需要按依赖的先后顺序加载运行DXE驱动程序。EDK2[11]中的调度方法如算法1所示。该算法会维护可调度驱动队列mScheduledQueue和驱动列表mDiscoverList并循环调度驱动,每轮循环都加载mScheduledQueue中的所有驱动,然后将mDiscoverList中所有依赖的协议已经被满足的驱动添加到mScheduledQueue中。

但EDK2的调度方法只适用于完整的UEFI系统,DxeEmulator在模拟执行DXE驱动时缺失了前置环境,无法直接应用此调度方法。为此,本文提出了一种三阶段驱动调度方法,每个阶段的工作如下。

3.3.1 基本调度

该阶段还没有任何驱动被执行,DxeEmulator从Apriori列表中的驱动开始运行,并扩展到其他依赖这些驱动安装的协议的驱动。DxeEmulator将调度队列设置为Apriori配置文件中的驱动列表,然后执行算法1。该阶段结束后,未被调度的驱动依赖都无法被满足。

3.3.2 架构协议依赖调度

未执行的驱动中有一部分依赖于和架构紧密相关的一系列协议mArchProtocols。mArchProtocols会被PEI阶段的程序安装至内存,但由于前置环境的缺失导致这些协议未被安装,依赖这类协议的驱动无法进入调度队列。DxeEmulator会在基础调度阶段结束后在可用协议列表中添加mArchProtocols中未被安装的协议,然后执行算法1。

3.3.3 随机调度

经过了前两个阶段后还未执行的驱动依赖于未被安装的协议,但仍可以尝试执行这些驱动。因为:1)开发人员编写程序时可能添加了未使用的依赖;2)依赖协议的调用代码可能位于不可达的路径上;3)即使未满足依赖会导致崩溃,驱动可能会在崩溃之前安装其他驱动依赖的协议。DxeEmulator在这一阶段每次随机选择一个未加载的驱动添加到调度队列中,然后执行算法1,直到所有的驱动都被尝试加载。

算法1 驱动调度算法

输入 目标固件的所有驱动文件以及协议、依赖项信息

输出 目标固件的所有驱动的执行顺序和安装的所有协议

```
EPT ← mArchProtocols + 其他默认协议   // Export Protocol Table,可用的导出函数的集合
mScheduledQueue ← Apriori配置文件     // 满足依赖的待调度队列
mDiscoverList ←{所有DXE驱动程序}  // 驱动列表
WHILE (mScheduledQueue不为空)
  WHILE (mScheduledQueue不为空)
    driver ← mScheduledQueue. pop() // 取出队列第一个元素
    load driver image       // 将driver二进制文件加载到内存
    start driver            // 将控制权交给driver入口开始执行
    IF (run error) THEN record driver     // 记录执行出错的协议
    EPT += driver安装的协议
  FOR driver ← mDiscoverList[0 to last]
  DO
    IF (driver已经被调度) THEN continue
    // 没有Import Protocol Table需要满足mArchProtocols都已被加载
    IF (driver没有IPT文件) THEN
      IF (mArchProtocols中所有协议均已被加载) THEN mScheduledQueue. put(driver)
    // 判断驱动的Import Protocol Table中的Protocol是否已全部被加载
    ELSEIF (EPT 包含driver的IPT ) THEN mScheduledQueue. put(driver)
  END
记录mDiscoverList中所有未被加载的驱动
```

3.4 实现DXE依赖的服务

DXE阶段依赖的服务有四种:启动服务,运行时服务,DXE服务和SMM服务。本文参考了Qiling[18]的工作实现了这些服务,以解决驱动对服务的代码依赖问题。

具体来说,本文的工作包括:人工实现所有服务,并设置函数参数数量与每个参数的类型;在Unicorn中申请内存存放所有服务函数表;hook表中的所有函数指针。在模拟执行过程中,当驱动调用了一个被hook的函数时,DxeEmulator会根据参

数传递规则读取函数参数并传入实现的服务中,经服务处理后将返回数据写入内存或返回值寄存器。

3.5 系统实现

本文基于CPU模拟器Unicorn实现了DxeEmulator。整个系统的实现包含6 653行python代码,其中539行用于固件解压和关键信息提取,另外6 114行用于模拟执行。

4 实验设计与评估

4.1 实验设置

4.1.1 数据集来源

本文编写爬虫收集了6家主流主板品牌生产商的UEFI固件,并在解析后的固件文件中测试了DxeEmulator的运行效果。获取到的固件统计信息如表1所示,数据集中的所有样本的架构均为x64。

4.1.2 实验环境

实验环境的CPU为Intel i7-8700K CPU @ 3.70 GHz,内存为32 GB,操作系统为Windows 10,编程语言为python 3.8。

4.1.3 实验设计

为了全面地评估DxeEmulator的运行效果,本文设计了实验回答以下问题:Q1本文提出的针对数据依赖的解决方案能否提高固件模拟执行的效果(4.2),Q2本文提出的三阶段调度方法能否提高固件模拟执行的效果(4.3),Q3 DxeEmulator相较于现有工具是否有提升(4.4),Q4 DxeEmulator的运行效率能否支持大规模测试(4.5),Q5 DxeEmulator能否用于安全分析(4.6)。

4.2 数据依赖效果分析

为了验证DxeEmulator的数据依赖解决方案是否提升了模拟执行效果,我们在每个品牌生产商中随机选择了10个固件进行实验。分别记录了数据依赖模块启用/停用情况下运行的基本块数量,实验结果见表2的第3、4列。从结果可以看出,在所有品牌的固件中,添加数据依赖模块后均能提升基本块覆盖数量。这说明DxeEmulator的数据依赖解决方案确实能够提升运行效果。

4.3 代码依赖效果分析

为了验证DxeEmulator的三阶段调度方法能否提高固件模拟执行的效果,我们进行了分阶段的实验,分别统计了执行到每个调度阶段时的成功运行的驱动数量。该实验使用了6个品牌中的18个固件,图3记录了各厂商的固件的实验结果。根据实验结果,运行至随机调度阶段相比于只使用基本调度算法平均可以多运行137.5%的DXE驱动,而提升效果最好的BIOSTAR可以多运行243.1%的DXE驱动,说明本文提出的三阶段调度方法能有效解决模拟执行中UEFI DXE驱动的代码依赖问题。

4.4 模拟执行效果对比

为了验证DxeEmulator的模拟执行效果相较于现有工作是否有提升,在4.2节的数据集上做了DxeEmulator和Qiling[18]的效果对比实验。Qiling的各种异常处理工作导致部分运行异常会被忽略,无法直接比较二者的成功率。所以本文基于访问的基本块数量进行了模拟执行效果比较,实验结果见表2。

通过实验对比可以发现,DxeEmulator相较于Qiling基本块访问数量有大幅提升;即便不考虑数据依赖模块,DxeEmulator也能取得更好的效果,这种提升在6个品牌的固件中都有体现。而在Intel的固件中基本块访问数量提升了196.62%,这是因为Intel的固件中存在大量操作PCI区域内存的代码,而Qiling未提供外设访问功能,无法解决外设带来的数据依赖问题。实验结果说明DxeEmulator相较现有工具能够大幅提升DXE驱动模拟执行的成功率。

4.5 执行效率

为了验证DxeEmulator模拟执行的效率,本文统计了模拟执行中的时间消耗,结果如表1所示。

表1 数据集来源与成功率

Table 1 Dataset source and success rate

品牌	固件总数	DXE驱动数量	运行成功数量	成功率/%	驱动平均运行时间/ms
Acer(宏碁)	138	26 155	19 778	75.62	43.3
ASRock(华擎)	111	34 694	18 550	53.47	33.9
ASUS(华硕)	60	11 644	7 209	61.91	30.4
BIOSTAR(映泰)	93	18 873	13 211	70.00	50.7
Intel(英特尔)	136	4 242	3 242	76.43	40.9
MSI(微星)	127	26 894	20 011	74.41	51.2

由于模拟执行失败包含的情况比较复杂,我们只统计了运行成功的模块所需的时间。DxeEmulator模拟所有品牌的驱动时间消耗都低于51.2 ms,即便统计超时的情况,模拟执行一个完整固件平均也仅需1 min。这足以说明DxeEmulator可以有效应用于大规模固件分析。

表2 基本块运行数量统计

Table 2 Statistics of running basic blocks

品牌	Qiling	无数据依赖的DxeEmulator (相较于Qiling的提升比例)	DxeEmulator (相较于Qiling的提升比例)
Acer(宏碁)	64 270	89 791 (39.71%)	92 645 (44.15%)
ASRock(华擎)	136 288	136 179 (−0.07%)	142 087 (4.25%)
ASUS(华硕)	113 764	219 471 (92.92%)	228 917 (101.22%)
BIOSTAR(映泰)	96 271	113 504 (17.90%)	116 911 (21.44%)
Intel(英特尔)	85 671	239 330 (179.36%)	254 114 (196.62%)
MSI(微星)	87 790	122 979 (40.08%)	133 481 (52.05%)

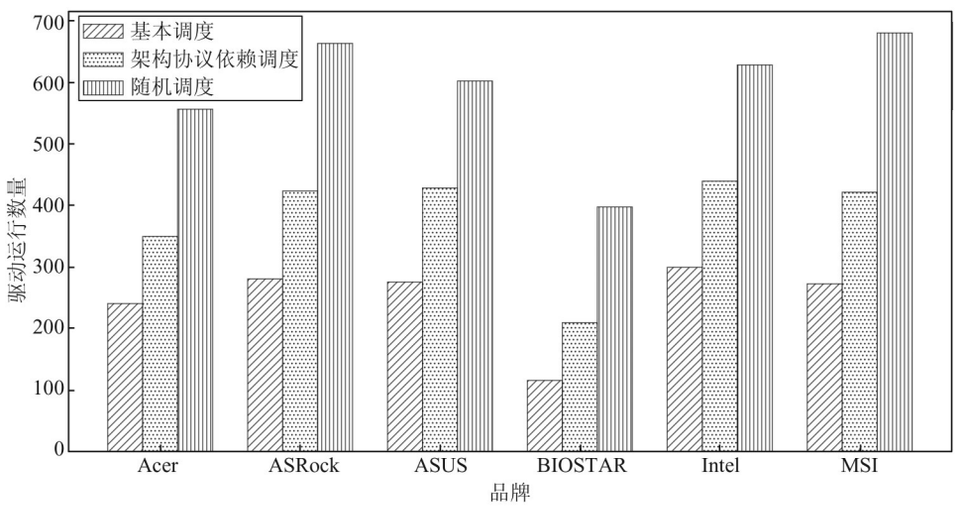

图3 不同阶段驱动运行数量

Fig. 3 Driver count at different stages

4.6 漏洞挖掘应用

为了验证DxeEmulator在UEFI固件分析中的实际应用效果,本文在DxeEmulator的基础上构建了一个漏洞检测模块,用于检测UEFI DXE驱动程序中特有的GetVariable函数导致的缓冲区溢出漏洞。具体来说,该函数会读取固件中的变量并写入缓冲区,如果固件读取变量时没有对变量长度进行检查,则可能利用篡改后的数据覆盖缓冲区中的数据。

本文实现的漏洞检测模块会监测驱动执行中所有对GetVariable函数的调用,并判断修改变量数据的长度能否导致超出缓冲区的数据写操作。本文在收集的数据集中应用了该检测方法,共发现了12个缓冲区溢出漏洞,其中有9个漏洞是0-day漏洞,其余3个已经在最新固件中已经被修复。具体统计数据见表3,其中包含2个影响多个生产商的漏洞。我们分析了找到的所有漏洞,并确认了这些漏洞均为真实漏洞,相关漏洞均已向生产商提交。

表3 DxeEmulator发现的缓冲区溢出漏洞

Table 3 The buffer overflow vulnerabilities found by DxeEmulator

品牌	漏洞数量	去重后漏洞数量
Acer(宏碁)	115	11
ASRock(华擎)	11	1
ASUS(华硕)	11	1
BIOSTAR(映泰)	62	1
Intel(英特尔)	6	1
MSI(微星)	94	3
总计	306	12

5 讨 论

本文的实验结果说明DxeEmulator的模拟执行

整体效果是不错的。但从表1看出,仍有一部分驱动运行失败。通过对失败的驱动进行人工分析,我们将失败原因分为三类:运行异常、运行跳过以及运行超时。

5.1　运行异常

DXE驱动运行中产生了异常,这类异常是由Unicorn模拟执行指令时抛出的。在DxeEmulator中,产生这类异常的原因包括:

1) 控制流异常,空指针被当作函数进行调用。DxeEmulator在随机调度阶段会尝试执行依赖未满足的驱动。由于依赖未满足,这些驱动可能会在读取依赖协议的函数指针时读到空指针,触发异常控制流。

2) 来自填充数据的数据流异常。DxeEmulator提供了许多初始化协议的结构体,其中可能包含与硬件和系统相关的物理地址。这些地址无法被准确初始化,被当作指针进行解引用时会导致非法地址访问。

3) 来自外设数据的数据流异常。部分外设访问的返回值会被当作地址进行解析,DxeEmulator模拟外设时除PCI/PCIe区域外均返回随机值,解引用随机地址时可能导致非法地址访问。

5.2　运行跳过

这类错误的产生原因是驱动读取了PEI阶段产生的Hob List。PEI阶段的程序可以通过Hob List向DXE驱动传递信息。但Hob List是动态生成的,无法从固件中直接提取,所以DxeEmulator没有提供读取Hob List的功能。为了避免读Hob List导致的内存非法读写,DxeEmulator在监控到此类读写操作时会跳过当前DXE驱动。在本文收集的所有固件中,运行跳过的比例都是影响成功率的最重要的原因。

5.3　运行超时

在DxeEmulator中,单个驱动运行用时正常约40 ms。为了避免驱动执行时陷入等待状态导致的系统停止运行,本文设置了一个远超正常运行时间的超时上限10 s,超时的程序将被强制跳过。经分析发现样本陷入死循环导致超时的原因为外设模拟不完善。DxeEmulator的PMIO外设模块只返回随机值,部分驱动可能会持续读取外设,直到外设返回特定值,随机值和空值都无法满足运行条件。

从分析结果可以看出,DxeEmulator的模拟运行失败原因主要为数据缺失,包括外设数据和默认数据的缺失。在今后的工作中,可以借助符号执行技术[24,25]更准确地还原外部数据,提高模拟执行中提供的数据的有效性。

6　结　语

分析人员难以对UEFI固件进行分析是提升UEFI安全性的一大阻碍,但静态分析方法缺乏准确性,动态分析方法扩展性较差。为了辅助研究人员对UEFI固件进行动态分析,本文提出了一种基于Unicorn的模拟执行方法DxeEmulator。DxeEmulator能够按照驱动之间的依赖关系动态调度驱动执行顺序,并提供模拟执行所需的变量和外设数据,解决了DXE驱动模拟执行中的代码依赖和数据依赖问题。

本文在收集到的665个固件上对DxeEmulator进行测试后发现,DxeEmulator在所有品牌的固件上均可以成功运行50%以上的DXE驱动文件,在宏碁、映泰、英特尔和微星的固件上可以成功运行70%以上的固件。在相同运行条件下,DxeEmulator相比Qiling的基本块覆盖数量有大幅提升。此外,在DxeEmulator的基础上,我们在数据集中进行了漏洞挖掘的实践,并挖掘到了12个缓冲区溢出漏洞,其中包含9个0-day。DxeEmulator使得在UEFI固件上构造或移植各种漏洞检测工具成为可能,可以在大规模UEFI固件测试中被用于辅助安全研究人员进行漏洞挖掘。

参考文献:

[1] UEFI SPECIFICATION. UEFI Platform Initialization Specification [EB/OL]. [2021-03-18]. https://uefi.org/sites/default/files/resources/UEFI_Spec_2_9_2021_03_18.pdf.

[2] LINDSEY O. MoonBounce UEFI Malware Uncovered in Targeted Attack [EB/OL]. [2022-01-20]. https://duo.com/decipher/moonbounce-uefi-malware-uncovered-in-targeted-attack.

[3] ESET I. ESET Research Discovers ESPecter, a Bootkit for Cyberespionage[EB/OL]. [2021-10-06]. https://blog.eset.ie/2021/10/06/eset-research-discovers-especter-a-bootkit-for-cyberespionage/.

[4] CATALIN C. Chinese Hacker Group Spotted Using a UEFI bootkit in the Wild [EB/OL]. [2020-10-05]. https://www. zdnet. com/article/chinese-hacker-group-spotted-using-a-uefi-bootkit-in-the-wild/.

[5] ECLYPSIUM. UEFI Attacks in the Wild [EB/OL]. [2018-10-01]. https://eclypsium.com/2018/10/01/uefi-attacks-in-the-wild/.

[6] 戴正华. UEFI原理与编程[M]. 北京:机械工业出版社,

2015: 5-12.
DAI Z H. UEFI: Principles and Programming[M]. Beijing: China Machine Press, 2015: 5-12(Ch).

[7] ALEX M, ANDREY L, YEGOR V, *et al*. efiXplorer Hunting for UEFI Firmware Vulnerabilities at Scale with Automated Static Analysis[R/OL]. [2020-12-09]. https://i.blackhat.com/eu-20/Wednesday/eu-20-Labunets-efiXplorer-Hunting-For-UEFI-Firmware-Vulnerabilities-At-Scale-With-Automated-Static-Analysis.pdf.

[8] OLEKSANDR B, JOHN L, LEE R, *et al*. Symbolic Execution for BIOS Security[C]//Proceedings of the 9th USENIX Workshop on Offensive Technologies(WOOT). Washington: USENIX Association. 2015: 1-10.

[9] YIN J W, LI M H, WU W, *et al*. Finding SMM privilege-escalation vulnerabilities in UEFI firmware with protocol-centric static analysis [C]//Proceedings of 2022 IEEE Symposium on Security and Privacy (S&P). New York: IEEE Press, 2022: 1623-1637. DOI: 10.1109/SP46214.2022.9833723.

[10] 曾令静. 基于EDK2的UEFI变量检查的研究和实现[D]. 上海: 上海交通大学, 2016.
ZENG L J. Research and implementation of UEFI variable checking based on EDK2[D].Shanghai: Shanghai Jiao Tong University, 2016 (Ch).

[11] TIANOCORE. EDK2[EB/OL]. [2022-08-29]. https://github.com/tianocore/edk2.

[12] 郭东奥. UEFI驱动程序的自动化模糊测试方法研究[D]. 南京: 南京大学,2019 .
GUO D A. Research on automatic fuzzy testing method of UEFI driver [D]. Nanjing: Nanjing University, 2019 (Ch).

[13] JENNIFER F. Project Triforce: Run AFL on Everything [EB/OL]. [2016-06-27]. https://research.nccgroup.com/2016/06/27/project-triforce-run-afl-on-everything/.

[14] YANG Z K, VIKTOROV Y, YANG J, *et al*. UEFI firmware fuzzing with simics virtual platform[C]//2020 57th ACM/IEEE Design Automation Conference (DAC). New York: IEEE Press, 2020: 1-6. DOI: 10.1109/DAC18072.2020.9218694.

[15] BELLARD F. QEMU, a fast and portable dynamic translator [C]//Proceedings of the USENIX Annual Technical Conference 2005. Anaheim: USENIX, 2005: 41-46.

[16] ASSAFCARLSBAD. Efi_Dxe_Emulator Source Code[EB/OL]. [2020-04-23]. https://github.com/assafcarlsbad/efi_dxe_emulator.

[17] NGUYEN A Q, DANG H V. Unicorn: Next Generation CPU Emulator Framework [R/OL]. [2022-02-20]. https://www.unicorn-engine.org/BHUSA2015-unicorn.pdf.

[18] QILINGFRAMEWORK. Qiling Source Code[EB/OL]. [2022-09-25]. https://github.com/qilingframework/qiling.

[19] LONGSOFT. UEFITool[EB/OL]. [2022-09-25]. https://github.com/LongSoft/UEFITool.

[20] THEOPOLIS. Uefi-Firmware-Parser Source Code[EB/OL]. [2022-09-13]. https://github.com/theopolis/uefi-firmware-parser.

[21] YEGGOR. Uefi_Retool Source Code[EB/OL]. [2022-08-04]. https://github.com/yeggor/uefi_retool.

[22] 吴伟民, 吴辉, 苏庆. UEFI固件存储系统分析[J]. 计算机工程与设计, 2017, **38**(4): 1110-1116. DOI: 10.16208/j.issn1000-7024.2017.04.049.
WU W M, WU H, SU Q. Analysis of UEFI firmware store system [J]. *Computer Engineering and Design*, 2017, **38**(4): 1110-1116. DOI: 10.16208/j. issn1000-7024.2017.04.049 (Ch).

[23] Intel. Intel I/O Controller Hub 10(ICH10) Family[EB/OL]. [2008-10-01]. https://www.intel.com/content/www/us/en/io/io-controller-hub-10-family-datasheet.html.

[24] JOHNSON E, BLAND M, ZHU Y F, *et al*. Jetset: Targeted firmware rehosting for embedded systems[C]//30th USENIX Security Symposium (USENIX Security 21). Berkeley: USENIX Association, 2021: 321-338.

[25] ZHOU W, GUAN L, LIU P, *et al*. Automatic firmware emulation through invalidity-guided knowledge inference [C]//30th USENIX Security Symposium (USENIX Security 21). Berkeley: USENIX Association, 2021: 2007-2024.

□