软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2024,35(8):3577−3590 [doi: 10.13328/j.cnki.jos.007116]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

UEFI 的启发式逆向分析与模糊测试方法 *

林欣康[1,2], 顾匡愚[1,2], 赵 磊[1,2]

[1](空天信息安全与可信计算教育部重点实验室(武汉大学), 湖北 武汉 430072)

[2](武汉大学 国家网络安全学院, 湖北 武汉 430072)

通信作者: 赵磊, E-mail: leizhao@whu.edu.cn

摘 要: 统一可扩展固件接口(unified extensible firmware interface, UEFI)作为新一代固件接口标准, 广泛应用于现代计算机系统, 但其漏洞可能引发严重安全威胁. 为了减少 UEFI 漏洞引发的安全问题, 需要进行漏洞检测. 而第三方安全测试场景下的模糊测试是检测的主要手段. 但符号信息的缺失影响了测试效率. 提出了一种启发式的 UEFI 逆向分析方法, 恢复固件中的符号信息, 改进模糊测试并实现了原型系统 ReUEFuzzer. 通过对来自 4 个厂商的 525 个 EFI 文件进行测试, 证明了逆向分析方法的有效性. ReUEFuzzer 可以提升函数测试覆盖率, 并在测试过程中发现了一个零日漏洞, 已报告给国家信息安全漏洞共享平台以及公共漏洞和暴露系统. 实验证明, 该方法在 UEFI 漏洞检测方面具有有效性, 可以为 UEFI 安全提供一定的保障.

关键词: 统一可扩展固件接口; 逆向工程; 模糊测试; 静态程序分析; 固件安全

中图法分类号: TP311

中文引用格式: 林欣康, 顾匡愚, 赵磊. UEFI 的启发式逆向分析与模糊测试方法. 软件学报, 2024, 35(8): 3577−3590. http://www.jos.org.cn/1000-9825/7116.htm

英文引用格式: Lin XK, Gu KY, Zhao L. UEFI Fuzz Testing Method Based on Heuristic Reverse Analysis. Ruan Jian Xue Bao/ Journal of Software, 2024, 35(8): 3577−3590 (in Chinese). http://www.jos.org.cn/1000-9825/7116.htm

UEFI Fuzz Testing Method Based on Heuristic Reverse Analysis

LIN Xin-Kang[1,2], GU Kuang-Yu[1,2], ZHAO Lei[1,2]

[1](Key Laboratory of Aerospace Information Security and Trusted Computing (Wuhan University), Ministry of Education, Wuhan 430072, China)

[2](School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China)

Abstract: As a next-generation firmware interface standard, the unified extensible firmware interface (UEFI) has been widely used in modern computer systems. However, UEFI vulnerabilities have also brought serious security threats. To avoid security problems caused by UEFI vulnerabilities as much as possible, vulnerability detection is needed, in which, fuzzing under third-party security testing scenarios is mainly used. Nevertheless, the absence of symbolic information affects the efficiency of testing. This study proposes a heuristic UEFI reverse analysis method, which recovers the symbolic information within the firmware, improves fuzz testing, and implements a prototype system, ReUEFuzzer. Through testing 525 EFI files from four manufacturers, the effectiveness of the reverse analysis method is demonstrated. ReUEFuzzer can enhance the function test coverage and has identified an unknown vulnerability during the testing process, which has been reported to China National Vulnerability Database and the Common Vulnerabilities and Exposures (CVE) system. Empirical evidence shows that the method presented in this paper is valid for UEFI vulnerability detection and can provide a certain degree of security guarantee for UEFI.

* 基金项目: 国家自然科学基金(62172305)

本文由“系统与网络软件安全”专题特约编辑向剑文教授、陈厅教授、王浩宇教授、罗夏朴教授、杨珉教授推荐.

林欣康和顾匡愚为共同第一作者.

收稿时间: 2023-09-10; 修改时间: 2023-10-30; 采用时间: 2023-12-15; jos 在线出版时间: 2024-01-05

CNKI 网络首发时间: 2024-03-09

Key words: unified extensible firmware interface (UEFI); reverse engineering; fuzz testing; static program analysis; firmware security

统一可扩展固件接口(unified extensible firmware interface, UEFI)[1]是一种用于加载和启动操作系统, 并向操作系统提供丰富的运行时服务的固件接口. 由于其灵活性和可扩展性, UEFI 已经被广泛应用于现代计算机系统中, 以取代传统的基本输入/输出系统(basic input/output system, BIOS)[2]. 然而, 随着 UEFI 的广泛应用, 其安全性也面临着越来越多的威胁. 根据 CVE 官方网站的数据[3], 仅在 2022 年, 就有 134 个与 UEFI 相关的漏洞被披露. 这些漏洞中, 71%的 CVSS 评分[4]高于 7.0 分, 属于高危漏洞. 这些漏洞可能导致绕过操作系统安全措施的控制权劫持、任意位置的敏感数据泄露、恶意程序(如 Bootkit[5])的植入等一系列高危的安全威胁. 因此, 针对UEFI的漏洞挖掘显得尤为重要, 通过主动发现这些问题并及时修补, 可以有效地预防这些安全威胁.

目前, 对于 UEFI 的漏洞挖掘主要有两种方法, 分别为动态模糊测试和静态污点分析. 在动态模糊测试方法中, 考虑到固件通常无法直接运行, 因此一般采用模拟执行的方式来运行 UEFI 固件. 然后, 再利用动态插桩等方法收集测试过程产生的覆盖率信息以及程序状态来引导模糊测试器生成有效的测试用例对 UEFI 进行测试. 而现有的静态污点分析方法不具备通用性, 在现有的代表性工作中仅针对 SMM 提权漏洞, 这是因为该类型的漏洞在具有保护的 SMRAM 区域中对外部函数出现逃逸引用时就会发生, 因此只需要检测其中的数据流及控制流依赖即可发现该漏洞, 而其他类型的漏洞一般不具有该特征.

考虑到 UEFI 漏洞挖掘通用性的需求, 本文利用动态模糊测试方法对 UEFI 进行漏洞挖掘. 由于在实际应用场景中, UEFI 的符号信息通常难以获得, 这些符号信息具体包括: (1) 基础服务信息, 即全局服务函数以及协议的符号信息; (2) 函数签名信息, 即函数名以及对应的参数数据结构. 缺失符号信息导致对 UEFI 进行模糊测试时会出现如下的问题: UEFI 功能难以正确模拟、攻击平面难以定位以及变异过程盲目, 最终导致模糊测试的覆盖率低下甚至无法测试. 例如: 当缺失基础服务信息时, 会导致模拟器无法模拟全局服务函数和协议, 从而使得 UEFI 在模拟时,无法正常使用这些基础服务, 最终使得运行到相关代码时导致 UEFI 模拟失败. 此外, 一部分的服务函数还与外部输入有关, 定位攻击平面需要确定这些服务函数和协议的位置. 而缺失函数签名信息时, 由于 UEFI 大量使用复杂数据结构, 例如多层嵌套结构体在没有符号信息的辅助下进行模拟, 可能会因为指针解析、校验和检查错误等原因产生崩溃, 使得 UEFI 模糊测试的变异过程变得盲目, 从而导致测试时 UEFI 代码覆盖率有限.

针对上述问题, 本文提出一种逆向分析UEFI的方法, 即通过对二进制固件反汇编代码的分析来还原上述两种符号信息: 使用服务函数访问模式逆向分析的方法还原基础服务信息; 使用断言匹配、二进制代码比对和函数调用关系的逆向分析方法还原函数签名信息. 最后, 本工作在逆向分析方法的基础上设计并实现了UEFI 的模糊测试原型系统 ReUEFuzzer. 为验证其有效性, 本工作在随机选取 EDK2(EFI development kit 2)[6]中的 EFI 进行基础服务信息还原时, 在随机选择的样本中含有 617 个全局服务函数, 其中 595 个成功还原,还原成功率为 96.43%. 在随机选取固件解包得到的 EFI 进行函数签名信息还原时, 在随机选择的样本中含有 1 645 个函数, 其中 1 315 个成功还原出相应的符号、数据结构等信息, 还原成功率为 79.94%. 这表明本文提出的逆向分析方法能够较好地还原 UEFI 的符号信息. 之后, 我们进行了对比实验, 将 ReUEFuzzer 与缺失符号信息的模糊测试器进行对比. 实验结果表明: 在第三方测试场景下, ReUEFuzzer 能够更加有效地测试复杂函数并且提高代码覆盖率. 此外, 利用 ReUEFuzzer, 我们还发现了一个零日漏洞, 并被 SuperMicro[7]公司的团队确认, 同时, 将其上报国家信息安全漏洞共享平台[8]以及公共漏洞和披露系统[3].

1 背 景

1.1 UEFI漏洞挖掘

UEFI[1]是一种在计算机硬件和操作系统之间提供软件接口的系统规范, 其起源于 Intel[9]公司的 EFI(可扩展固件接口, extensible firmware interface)项目. UEFI 旨在取代传统的 BIOS(基本输入输出系统, basic input/output system), 为现代计算机提供更高效、灵活和安全的启动和运行环境. 目前, 关于 UEFI 漏洞挖掘的工作

较少, 主要有两种技术路线, 分别采用动态模糊测试方法和静态污点分析方法.

Yang 等人[10]提出一种动态模糊测试方法, 该方法基于 Simics 仿真器[11]对 UEFI 固件代码进行灰盒模糊测试. 但是目前, 该方法仅适用于厂商内部对 UEFI 进行测试, 因为该方法需要测试人员拥有关于待测 UEFI 固件的源码、文档等信息用来定位测试的关键位置, 并且具备编译固件的能力, 使其能够在 Simics 仿真器上运行. 因此, 在无法获取源码和文档的第三方安全测试的场景下, 该方法无法有效地应用. Yin 等人[12]提出一种静态污点分析方法, 他们发现, 尽管已经部署了硬件隔离和漏洞缓解机制, 攻击者仍然可以利用 SMI 处理程序漏洞绕过现有的保护机制, 并实现 SMM 提权攻击. 他们指出, SMI 处理程序中存在逃逸引用,可能引入 SMM 权限提升漏洞, 并提出一种静态分析框架 SPENDER 用于检测 SMM 权限提升漏洞. 然而, 该方法仅仅针对 SMM 权限提升漏洞, 难以进行扩展. 因此, 目前对于第三方安全测试的场景而言, 仍然缺失一种全面有效的动态分析方法来进行 UEFI 固件的模糊测试. 为此, 需要进一步研究和开发基于仿真器的动态分析方法, 以在不需要源码和文档等信息的情况下, 对 UEFI 固件进行全面有效的模糊测试.

1.2 基于模拟执行的模糊测试

模糊测试[13−17]是一种重要的软件测试技术, 通过为目标程序提供随机输入, 并监控程序异常行为(例如崩溃)来发现程序中的错误. 由于固件和普通程序不同, 固件通常无法在测试环境下直接执行, 因此在固件的模糊测试中, 广泛使用模拟执行技术进行辅助. 目前, 基于模拟执行的模糊测试工作已经有一些具有代表性的成果.

QEMU[18]是一种支持多种指令集和系统模拟的仿真平台, 在固件安全性分析的仿真平台设计中已被广泛应用. Chen 等人提出的 Firmadyne[19]基于 QEMU 模拟器, 并且首次对固件镜像进行了大规模的动态分析. 在后续的工作中, FirmAE[20]优化了 Firmadyne 的模拟过程, 而 FirmAFL[21]在 Firmadyne 的基础上, 还结合了 QEMU 的全系统仿真模式和用户仿真模式, 利用用户模式开销较低的特性来提升模糊测试的吞吐量. 这些工作主要针对具有操作系统抽象的嵌入式固件, 但是无法测试不具有操作系统抽象的 UEFI 固件. 而 QEMUSTM32[22]项目将 QEMU 的仿真对象扩展到了 STM32 芯片上, 并证明: 当有完整的硬件文档时, 通过对硬件的仿真可以完全对不具有操作系统抽象的固件镜像进行仿真. 它也可以和模糊测试器相结合来进行模糊测试, 但是会依赖在进行 UEFI 第三方测试时无法获取的文档, 因此无法将该方法应用于 UEFI 第三方测试.

此外, 一些工作通过真实硬件与模拟器结合的方法来解决模拟执行的问题. Avatar[23]旨在通过提供更好的硬件组件支持来实现嵌入式固件的动态程序分析, 它通过构建一个包括处理器仿真器和真实硬件的混合执行环境, 使得 Avatar 可以利用仿真器执行和分析指令, 同时将 I/O 操作通道传导到物理硬件上. PROSPECT[24]在 QEMU 中创建了一个虚拟字符设备, 用于拦截内核中与字符设备进行通信的系统调用, 并将其转发到目标嵌入式系统上的正确字符设备, 从而在模拟器中正确处理与字符设备交互的操作. 该系统可在目标设备上执行, 并将执行结果反馈给分析系统. 在随后的扩展工作中[25], Kammerstetter 改进了系统的可伸缩性, 通过缓存预期的外设行为, 并在外设行为偏离预期时重置缓存, 从而近似固件状态. 这些工作能够更好地模拟固件, 但是需要依赖真实硬件, 而且只能对于具有操作系统抽象的固件镜像有效.

与上述测试对象相比, UEFI 本身结构复杂, 没有操作系统抽象, 使用特有的全局变量共享方式和协议机制来进行模块间通信. 同时, 在第三方测试场景下也无法使用源码和文档. 因此, UEFI 难以使用上述模拟方法进行模糊测试.

2 UEFI 启发式逆向分析方法设计

2.1 逆向分析方法总体架构

为了还原基础服务信息以及函数签名信息, 本文提出了如图 1 所示的逆向分析方法总体框架, 根据这两种信息各自的特点, 分别使用不同的方法来还原.

- 针对基础服务信息的还原, 本文提出一种基于服务函数访问模式的逆向分析方法. 我们发现在 UEFI

中的一些服务函数具有固定的访问模式, 通过访问模式可以还原出这些服务函数相关的符号信息, 并且还可以利用这些函数还原协议和 GUID 等信息;

- 针对函数签名信息的还原, 本文提出 3 种方法: 第 1 种方法基于断言匹配, 我们发现 UEFI 固件中普遍存在一些断言信息, 通过这些信息与 UEFI 标准中函数的特点进行匹配, 从而能够推断上下文,逆向分析出函数名和数据结构; 第 2 种方法基于二进制代码相似性比对, 我们利用 UEFI 固件与现有的开源实现之间的相似性, 对比 UEFI 固件和开源实现, 推断出固件内部复杂的数据结构信息; 第 3 种方法基于函数调用关系, 我们分析固件的函数调用图以查看传递的参数信息, 找出未被前面方法覆盖的函数签名信息, 并验证已经还原出的函数名与数据结构, 从而使得对 UEFI 固件的逆向分析结果更加全面. 并且为了整合不同方法得到的结果, 我们通过一种基于优先级的整合策略对上述几种方法推断得到的函数签名信息进行综合得到最终的推断结果.

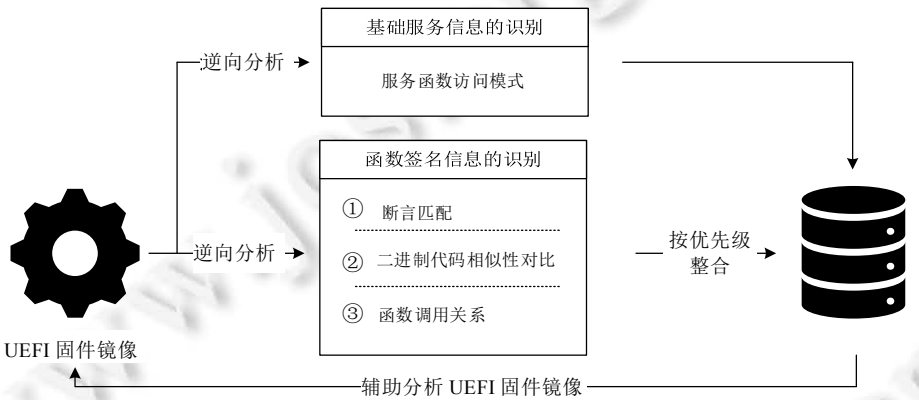

图 1 逆向分析方法总体框架

2.2 基础服务信息的识别

本文提出一种基于服务函数访问模式的逆向分析方法, 用于还原 UEFI 固件中关于全局服务表的全局变量、全局服务函数以及协议的 GUID 等基础服务信息. 根据 UEFI 规范 UEFI 中要设置大量的服务函数, 这些函数以函数指针的形式存放在一些重要的全局变量指向的表结构中, 例如 gST(全局系统表), gBS(启动服务表)和 gRT(运行时服务表). UEFI 规范规定了系统表传递给各模块的函数原型以及各模块使用系统表中各函数的调用方法, 因此, UEFI 模块对于全局变量、数据结构具有固定的访问模式, 这些访问模式在汇编层面表现为易于识别的特征, 并且这些访问模式在不同的指令集下都是类似的. 因此, 我们可以通过分析这些访问模式来还原基础服务信息.

如图 2 所示, 我们总结了 4 种访问模式, 用于还原 UEFI 固件中的全局服务函数、存放相关指针的全局变量等信息. 我们依次使用这些访问模式, 逐步对全局服务函数相关的信息进行推断, 每一步推断中, 使用之前推断的结果与当前访问模式相结合, 从而推断出更多的信息. 下面我们将依次介绍这 4 种访问模式以及如何利用它们来推断符号信息.

- 访问模式①: 大多数 UEFI 模块入口点函数的参数为 EFI_HANDLE 类型的 ImageHandle 和 EFI_SYSTEM_TABLE*类型的 SystemTable. 对于访问模式①, 我们在利用该模式进行推断时, 首先定位 UEFI 固件的模块入口点, 然后根据模块入口点的参数传递, 我们可以推断出 ImageHandle 变量以及 SystemTable 变量. 其中, System Table 是十分重要的数据结构, 它包含了许多关键信息, 例如系统的启动时间戳、内存分配器、协议栈等. 其中, System Table 中的表项存储了全局服务表的指针, 包括 gBS (boot services)和 gRT (runtime services). 这些全局服务表提供了许多基本的系统功能, 例如内存分配、文件读写、控制台输出等;
- 访问模式②: UEFI 模块通过 SystemTable 加上特定偏移来获取全局服务表指针. 因此对于访问模式②,

我们通过搜索针对 SystemTable 特定偏移的查表操作, 推断出例如 gBS (boot services)和 gRT (runtime services)等全局服务表的指针, 以及负责存储它们的全局变量. 以 gBS 与 gRT 为例, 对于 32 位的 UEFI 固件, gBS 相对于 SystemTable 的偏移为 0x3c, gRT 的偏移为 0x40; 对于 64 位的 UEFI 固件, gBS 的偏移为 0x58, gRT 的偏移为 0x60. 因此, 我们可以在固件代码中全局匹配对于这种含有特定偏移的移动操作, 通过这种模式, 能够匹配到全局变量和全局服务表的映射, 从而还原出有关全局服务表的符号信息;

- 访问模式③: 在模块使用全局服务函数时, 使用全局服务表加上偏移的方式来计算并读取对应的函数指针, 对于不同 UEFI 固件中的同一种服务函数, 这个偏移是不变的. 因为 UEFI 标准规定了服务表的结构和其中所有函数的偏移. 因此对于访问模式③, 我们通过搜索汇编代码中使用全局服务表与偏移来读取内存的代码块, 并根据其中使用的全局服务表的指针以及使用的偏移大小来找到该调用对应的服务函数;
- 访问模式④: 服务函数中的部分函数传递的参数包含 GUID, 这些函数包括 LocateProtocol, InstallProtocolInterface, InstallMultipleProtocolInterfaces. 以 LocateProtocol 为例, 该函数是用来定位已经被安装的协议, 协议是一种通过预定义的接口规范来提供服务的机制. UEFI 规范定义了一系列的协议, 包括但不限于 Boot 服务、Driver Binding 协议、Device Path 协议等. 每个协议都有自己的 GUID 用于标识和访问, 同时也有一系列预定义的接口函数来实现相应的服务. UEFI 中的协议机制为操作系统和硬件提供了一种通用的接口, 使得操作系统可以通过协议来访问硬件设备的服务, 从而实现更加灵活的硬件控制和驱动. 协议和对应的 GUID 会以 LocateProtocol 参数的形式传入到该函数中, 因此, 我们可以通过该函数的参数推断得到协议以及它对应的 GUID 的信息.

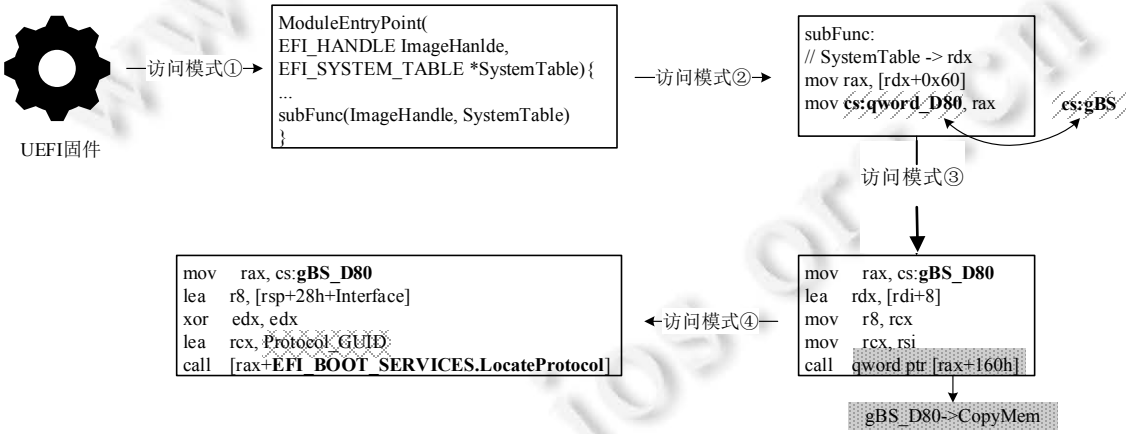

图 2 访问模式总览

2.3 函数签名信息的识别

在逆向还原 UEFI 中的函数签名信息时, 我们主要依赖 UEFI 固件与 UEFI 开源实现之间的相似性. 这是因为 UEFI 必须遵循 UEFI 规范[26], 而开源实现则是 UEFI 规范的参考实现, 因此, 大多数厂商都会在开源实现的基础上进行二次开发. 我们调查了总市场占比 87%的主流厂商的 UEFI 固件, 发现都存在这一现象. 基于此, 我们提出了 3 种方法来还原函数签名信息, 分别为基于断言匹配的逆向分析方法、基于二进制代码相似性匹配的逆向分析方法和基于函数调用关系的逆向分析方法.

2.3.1 基于断言匹配的逆向分析

我们提出了一种基于断言匹配的方法, 用于还原 UEFI 模块中的函数签名信息. 该方法基于如下发现, 即 UEFI 固件中通常会保留一些断言信息. 断言[27]是一种放在程序中的一阶逻辑, 用于标识验证程序开发者预期的结果. 断言的作用是验证程序的假设和前提条件是否成立, 以及检查程序的执行结果是否符合预期. 通过

添加断言, 开发人员可以减少程序中的错误和漏洞, 提高程序的可靠性和安全性. 这些断言通常采用类似于如图 3 中 sub_2F38A808 所示的形式, 其中包含断言信息和触发该断言的源代码文件路径信息. 由于断言信息通常与函数功能高度相关, 某些断言信息只会在具有相应功能的函数中被触发, 因此断言可以与UEFI标准中的函数特征进行匹配. 这种映射关系使得我们能够推断上下文, 并对UEFI中的数据结构和符号信息进行逆向分析.

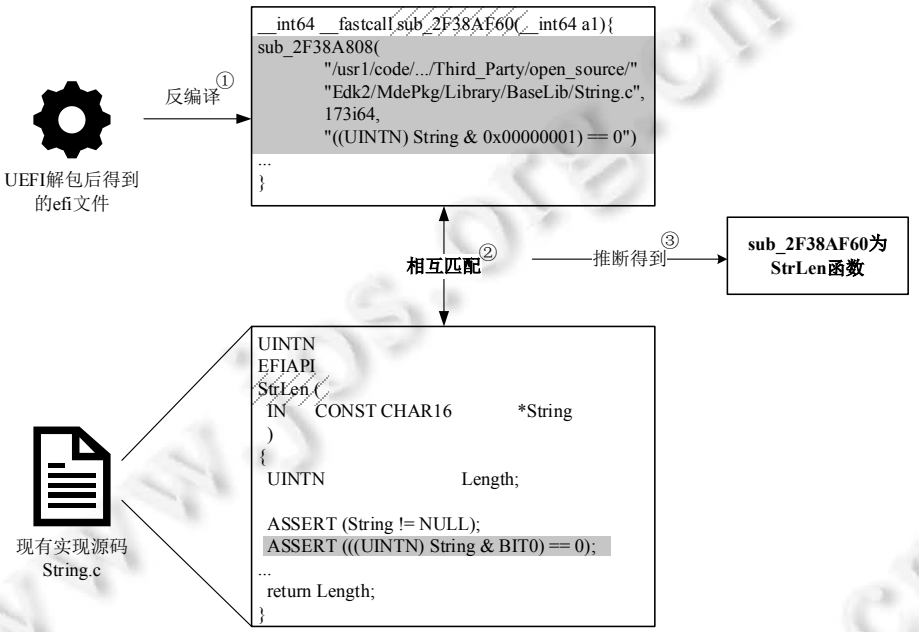

图 3 根据断言信息进行匹配推断

为了实现这种方法, 我们使用了一种自动化的流程来识别以及提取固件中的断言信息. 从这些信息中, 我们首先提取出该断言所在源文件的文件路径信息, 然后考虑到编译过程中语义信息的丢失, 我们使用字符串相似性匹配算法在源文件中查找具有相似断言信息的函数, 并使用这些符号信息来恢复 UEFI 固件中对应函数的签名信息. 以图 3 中的情况为例, 我们首先会识别到断言函数, 即图中的 ASSERT, 然后根据断言函数参数中的字符串提取出断言所在的文件路径, 即 MdePkg/Library/BaseLib/String.c, 然后利用字符串相似性匹配算法在开源实现对应的源文件中查找具有((UINTN) String & 0x00000001)==0 的断言内容, 在成功匹配后, 则可以得到具有相似断言的函数为 StrLen, 因此可以将 sub_2F38AF60 与 StrLen 函数相互匹配.

需要注意的是, 由于编译器的内联优化, 有些函数可能会为了避免函数调用而被直接嵌入到上级函数中. 在通过断言信息匹配函数时, 如果编译器优化内联了目标函数, 就无法通过断言信息将现有实现中的函数与 UEFI 固件镜像中对应断言所在的函数进行匹配. 这种情况下, 与 UEFI 固件镜像中函数相匹配的将是现有实现中函数的上层函数. 而在这种无法匹配的情况下, UEFI 固件镜像与现有实现中的函数参数数量等基础特征通常无法对应, 因此, 我们使用一种启发式的方法来缓解编译器内联优化导致的错配问题, 即: 当函数参数数量等基础特征不匹配时, 我们将这个存在断言的函数与现有实现中的上一级函数进行配对.

2.3.2 基于二进制代码相似性比对的逆向分析

在我们的研究中发现: 大多数 UEFI 固件都是基于符合 UEFI 规范的成熟实现进行开发的, 例如 EDK2. 这些固件会在其基础上进行二次开发, 以满足一些特殊的硬件要求和厂商的定制功能. 由于这些固件中的大部分代码都存在复用开源代码的情况, 只有小部分是定制化的实现, 因此通过对比开源实现和定制化固件的异同, 我们可以推断出其中复杂的数据结构与符号信息. 为了实现这一点, 我们利用二进制代码相似性比对技术(BinDiff)[28−31]来比较开源实现以及 UEFI 固件样本之间的异同, 其流程图如图 4 所示.

二进制代码相似性比对是在二进制代码分析领域中, 检测二进制代码相似性和差异性的强有力技术. 它可以比较两个二进制文件, 并确定从一个文件到另一个文件所做的更改. 对 UEFI 而言, 可以通过比较 UEFI 固件解包后得到的 EFI 文件, 来比较不同的 UEFI 固件镜像, 以识别 UEFI 固件的差异和相似之处. 通过使用二进制代码相似性比对技术, 我们可以有效地对比开源实现和定制固件中的常见数据结构和控制流, 从而恢复定制固件中的函数签名信息.

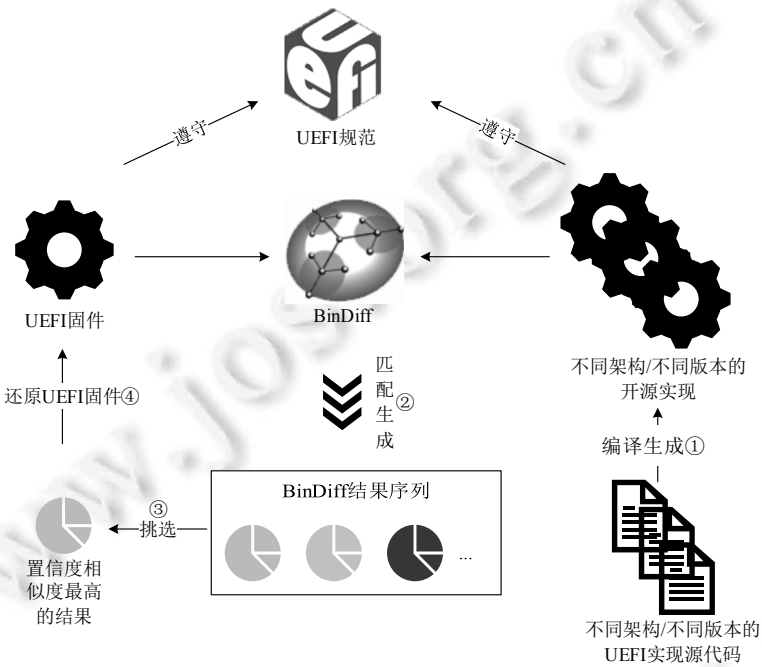

图 4 基于二进制代码相似性比对技术进行逆向还原的流程

为了尽可能地消除不同架构以及不同版本的开源实现对二进制代码相似性比对结果的影响, 在本方法中, 首先收集不同架构/不同版本的开源实现, 然后使用二进制代码相似性比对技术将这些开源实现分别与 UEFI 固件进行匹配, 该匹配过程会生成一系列的匹配结果, 包括函数名称、匹配算法、相似度、置信度等. 我们在其中挑选置信度最高的匹配结果来推断 UEFI 固件中的函数签名信息. 置信度是指在二进制文件比较过程中, 两个二进制文件相似度的可靠程度. 置信度的计算基于多种因素, 包括二进制文件的大小、结构、代码重用情况等. 在使用二进制代码相似性比对时, 高置信度通常代表更低的误报率.

2.3.3 基于函数调用关系的逆向分析

在我们之前的设计中, 采用了两种不同的方法来推断 UEFI 固件样本中存在的函数签名信息. 尽管这些方法能够推断出 UEFI 固件样本中的函数以及内部数据结构信息, 但是它们可能无法充分推断供应商定制化过程中开发的自定义组件. 这是因为这些组件可能不存在于开源实现中, 并且无法被上述提到的方法进行推断.

为了缓解这个问题, 我们提出了一种方法, 利用函数之间的关系信息(例如调用和被调用函数之间传递的参数)来推断这些未被覆盖的部分, 从而使得还原的信息更加完善. 我们的方法使用固件模块的函数调用图, 以识别函数依赖关系并揭示出前述方法未能记录的函数和数据结构. 我们还会分析函数之间传递的参数, 以推断每个函数使用的数据类型和格式. 通过这些信息与前述方法获得的结果相结合, 我们能够更加全面地推断该固件的符号信息.

2.3.4 优先级整合

在前文中, 我们介绍了 3 种不同的逆向分析方法, 以获取函数签名信息. 然而, 这些分析方法本身是相对独立的, 它们从不同的角度对 UEFI 固件进行分析, 因此, 它们得到的结果可能会相互冲突. 为了解决这个问

题, 我们还需要一个策略来整合它们的结果.

我们提出的这些针对 UEFI 的逆向分析方法具有不同的可信度, 在这些方法中, 推断过程越复杂, 就需要依赖越多的假设, 只有该方法所依赖的假设全部成立的时候, 才能够完全采信该方法推断出来的结果. 因此, 使用假设越少的方法通常是可信度更高的.

直接基于断言匹配的方法可信度最高, 因为它只是简单地进行字符串的匹配, 然后推断出上下文的信息. 其次是基于二进制代码相似性比对的方法, 因为该方法受到二进制代码相似性比对技术的局限性, 无法保证匹配成功的函数和数据结构完全可信, 所以它的可信度不及基于断言匹配的逆向分析方法. 最后是基于函数调用关系的方法, 它需要依赖前面两种方法推断得到的结论, 然后在这个基础上进行进一步的推断, 所以可以认为它的可信度是最低的.

因此, 在整合过程中, 我们只需要使用可信度作为优先级, 首先, 采信可信度较高的方法得到的结果.如果后续方法得到的结果与可信度较高的方法得到的结果冲突, 则以可信度较高的结果为准. 通过这种方式,我们可以整合不同方法得到的结果.

2.4 逆向分析方法实现

本文采用静态反编译软件 IDAPro[32]作为实现上述提到的 4 种 UEFI 逆向分析方法的工具. 为了系统地介绍这些方法的实现过程, 本节将从准备工作和具体实现两个方面进行论述.

2.4.1 获取解析固件与开源实现

我们从主流的制造商官网上下载 UEFI 固件镜像作为实验所需的样本, 这些制造商包括 Intel[9], Lenovo[33], SuperMicro[7]等. 通常, 这些制造商在其网站上提供的固件镜像有两种格式: 一种是原始的 UEFI 固件镜像, 即.bin 格式; 另一种是用于 UEFI 胶囊式固件更新的更新包, 即.CAP 格式. 这两种格式的镜像均包含了 UEFI 中各个模块的实现, 并且可以在一定程度上相互转化.

由于我们需要单独分析 UEFI 内部的子模块, 因此需要对 UEFI 的内部模块进行解析. 我们使用开源工具 UEFITool[34]来对 UEFI 固件进行格式分析, UEFITool 是一款开源的 UEFI 固件查看器和编辑器, 该工具会将 UEFI 固件中的内容解析为树结构. 该树结构包含各个模块的二进制文件以及该二进制文件的简单信息介绍. 我们的分析主要使用位于 BIOSregion 文件夹下的文件, 该文件夹下包含 UEFI 的各种功能模块.

根据我们的调查, 各大主流厂商都广泛使用基于 UEFI 标准的开源固件开发框架 EDK2 进行开发, EDK2 提供了完整的开发环境, 用于开发 UEFI 固件和 UEFI 应用程序. 因此, 我们选择 EDK2 作为基于二进制代码相似性比对方法的比较对象.

由于许多产品需要稳定性, 因此大部分 UEFI 固件镜像与版本较早的 EDK2 实现更加接近. 基于此, 我们选择了较早版本的 EDK2 源码, 而不是最新版本. 我们使用 Ubuntu 18.04 系统在 x86-64 架构下应用 GCC5 编译了 edk2-stable202111、edk2-stable202111-rc1、edk2-stable202105 和 edk2-stable202008 这 4 个版本的 EDK2 源代码, 得到了不同版本的各个模块对应的 EFI 文件. 在编译这些模块的时候, 我们保留了其符号信息, 以便指导后续的逆向分析步骤.

2.4.2 基于 IDA Pro 的逆向分析

本文基于 IDAPro[32]实现本节中详述的针对 UEFI 固件的逆向分析方法. IDAPro 是一个多平台、多处理器的反汇编程序, 它可以将机器可执行代码转化为汇编语言以及源代码, 用于调试和逆向工程.

在还原基础服务信息时, 我们采用了基于服务函数访问模式的方法, 并使用 IDAPython[35]进行实现. 我们利用 IDAPython, 按照 4 个访问模式依次对固件中的代码及数据进行匹配, 然后根据匹配结果逐步还原基础服务相关的信息, 并对相关变量的数据类型进行修改. 在对固件中的数据类型进行修改时, 在 x86-64 架构下, 我们依赖于 IDA 自带的类型库 uefi.til 以及 uefi64.til; 在 ARM 架构下, 我们将类型结构写入 C 语言格式的头文件中, 然后使用 IDA 的解析 Cheader 的功能对其进行解析, 来自动生成对应的数据类型以及结构体信息.

在还原函数签名信息时, 我们采用了 3 种方法, 这 3 种方法也使用 IDAPython 脚本进行实现. 其中, 对于基于字符串匹配方法, 我们首先进行一些预处理的工作. 我们使用 IDAPython 脚本自动化地批量处理对 UEFI

固件解包后得到的 EFI 文件, 获取 EFI 文件中的所有字符串以及其地址, 并且利用同一函数下被使用的字符串的地址紧密排布的特点对它们进行合并分组. 然后提取每组字符串中记录的源文件信息, 以搜索 EDK2 中对应的源文件, 并在源文件中对该组的其他字符串进行相似度匹配, 最终得到源文件中使用该组字符串函数的函数名和其他参数. 最后, 使用 IDAPython 脚本自动将 EFI 文件中的函数重命名并修改参数类型.

对于基于二进制代码相似性比对的方法, 我们首先使用 IDA 加载不同版本开源实现中的 EFI 文件得到 IDA 的数据库文件; 然后利用 IDA 的 BinDiff 插件[36], 分别将解包得到的 EFI 文件与不同版本开源实现中的对应文件进行匹配, 并从匹配输出挑选出置信度最高的作为最终结果, 同时从具有 DEBUG 信息的开源实现中导入对应的符号信息.

对基于函数调用关系的方法, 我们在使用 IDA 分析 EFI 文件时遍历各个函数, 将每个函数中的子调用的参数类型改为它所使用的局部变量参数的数据类型.

最后, 本文将顺序执行这几种方法的实现, 并且规定后面的方法不能覆盖前面方法得到的推断. 其推断结果将会自动保存在 IDA 的数据库中.

3 基于启发式逆向分析的模糊测试器设计与实现

基于前文提到的启发式逆向分析方法, 我们进一步设计并实现了 UEFI 的模糊测试器原型系统 ReUEFuzzer. 该原型系统的总体架构如图 5 所示.

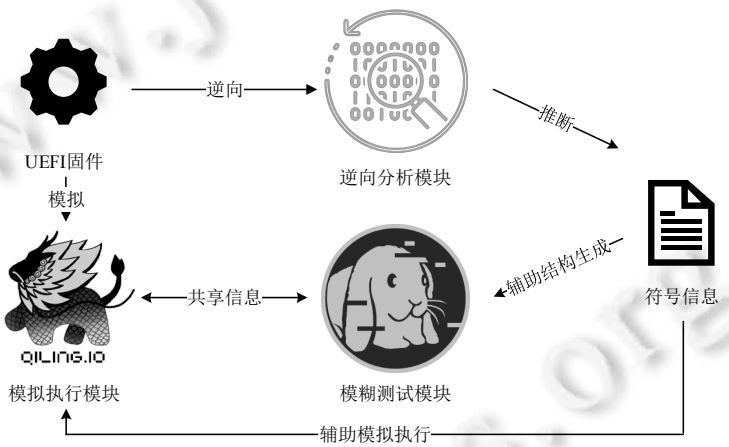

图 5 ReUEFuzzer 总体架构

由于在 UEFI 中存在许多无法通过 EFI 模块入口执行到达的跨 EFI 模块的函数调用, 我们采用直接模拟该函数的方法测试该函数并发现其中可能存在的缺陷. 因此, 在 ReUEFuzzer 中, 我们针对单个函数进行模糊测试. 在设计上, 我们参考 Yang 等人[10]的设计思路, 并使用模拟器来替代 simics 仿真器进行实现. ReUEFuzzer 由 3 个模块组成: 首先, 使用逆向分析模块还原 UEFI 固件的符号信息, 辅助模拟执行模块和模糊测试模块; 之后, 使用模拟执行模块对 UEFI 固件进行模拟, 并利用模拟器的功能对待测位置进行拦截; 最后, 使用模糊测试模块生成测试样例, 并在模拟器中将测试样例填入到合适位置进行测试. 引入符号信息使得模糊测试器可以生成结构化的测试样例. 本工作在原始的 qiling 模拟器[37]以及 AFL++[38]上新增了大约 600 行 Python 代码, 来实现针对任意函数测试的功能以及结构化参数生成的功能.

3.1 逆向分析模块

本模块旨在通过第 2 节中介绍的 4 种启发式逆向分析方法, 分别逆向还原出基础服务信息和函数签名信息. 这些获得的符号信息可以为模拟执行模块提供全局服务函数的位置, 调用的协议的 GUID 等信息, 使其能够根据这些信息对特定的函数进行模拟实现, 避免单一模块或者函数模拟时因缺失这些跨模块的信息而导致

模拟失败的问题. 此外, 该模块还可以提供一些数据结构信息, 使得模拟执行模块能够对结构中的字段进行取值约束, 避免非法取值导致指针访问非法地址等问题. 我们将利用逆向分析模块提供的符号信息, 使用 Python 编写结构体约束, 传递给模糊测试模块生成满足约束的输出, 同时, 我们将确保模拟执行模块能够正确解析结构体以进行模拟.

3.2 模拟执行模块

本工作中, 我们使用 qiling 模拟器实现该模块. qiling 模拟器[37]是基于 Unicorn[39]的轻量级二进制仿真框架, 能够对 UEFI 进行模拟. 通过修改 qiling 模拟器的源代码, 修改 EFI 文件的加载逻辑,使得模拟器初始调用从默认调用 module_entry 函数转化为可以调用任意函数, 这可以让后续测试得以从任意函数开始. 同时, 我们提前在模拟器内存中初始化全局系统表中包含的函数以及协议结构, 用来替换模拟执行时对这些函数以及协议的调用. 并且, 我们对存放全局系统表、全局服务表的全局变量进行拦截, 这些全局变量的位置由逆向分析模块提供, 使得它们有关的函数调用在仅模拟单个模块时也能够被正确处理. 同时, 在模拟器中对输入的测试样例进行处理, 通过递归解析约束, 分解结构体的树形结构, 使其能够正确解析结构化的参数, 将输入参数按约束要求布置到模拟器的内存中, 使得在执行过程中结构体能够被正确处理.

3.3 模糊测试模块

本工作中, 我们使用 AFL++[38]来实现模糊测试模块. AFL++是基于 AFL[40]框架的改进版模糊测试框架, 综合了大部分已有的改进方案. 考虑到我们使用的模拟器是基于 Unicorn 的 qiling 模拟器, 因此我们采用了 AFL++的 Unicorn 模式, 以兼容 qiling 模拟器, 并连接模拟器和模糊测试器. 该模式使用动态二进制插桩技术来收集覆盖率信息, 并引导模糊测试器进行高效的变异. 对于模糊测试器生成的测试用例, 我们可以通过回调函数将其传回到模拟器中, 并通过解析测试用例的值布置内存以及修改寄存器. 同时, 为了缓解不真实输入的生成, 我们还引入了逆向分析模块编写的结构体约束, 以生成满足结构体约束条件的测试用例, 从而使测试更具针对性, 并且在实际应用过程中主要测试攻击平面附近的函数, 以减少输入被限制情况的发生.

4 实 验

首先使用由 4 个不同厂商的 UEFI 固件并解包为 525 个 EFI 文件, 用这些 EFI 文件和 EDK2 来构造测试数据集. 接着, 我们使用本文提出的逆向分析方法来还原这些 UEFI 固件中的符号信息, 并与人工逆向的方式比较, 来说明该方法的有效性. 然后, 我们将 ReUEFuzzer 与缺失符号信息的普通模糊测试器进行对比, 用来说明 ReUEFuzzer 可以更有效地测试 UEFI 固件. 最后, 我们利用 ReUEFuzzer 对 EFI 文件进行批量测试, 并且发现了一个零日漏洞, 已经报告给厂商 SuperMicro 进行处理, 同时还提供了修复建议, 以供厂商参考. 具体见表 1.

表 1 实验环境

软/硬件	参数
操作系统	64 位 Ubuntu 20.04
CPU	Inter(R) Core (TM) i7-10700 CPU@2.90 GHz
内核版本	Linux 5.15.0-71-generic

4.1 逆向还原能力评估

为了对还原基础服务信息的能力进行评估, 我们直接针对 EDK2 编译得到的 EFI 二进制文件进行逆向还原. 因为目前的逆向分析工作中暂无对本文工作中重点关注的符号信息针对性还原的方法, 因此我们评估了该方法能够从全局系统表、全局服务表等有关的服务函数中还原的比例, 并使用该比例作为评估的指标. 本实验中, 我们使用的 EDK2 版本为 edk2-stable202111-rc1, 随机选取其中 10 个 EFI 二进制文件进行分析对比. 还原的比例见表 2.

从表 2 中可以看出, 大多数 EFI 二进制文件中的全局服务函数还原比例均超过 90%, 在 617 个全局服务

函数中, 有 595 个被成功还原, 总的还原成功率为 96.43%, 这表明本工作可以有效地还原出基础服务信息.

表 2 逆向分析方法对基础服务信息的还原情况

EFI	还原的服务函数	总共的服务函数	还原比例(%)
ArpDxe.efi	54	56	96
BdsDxe.efi	183	188	97
CapsuleRuntimeDxe.efi	17	22	77
ConPlatformDxe.efi	44	49	90
Dhcp4Dxe.efi	85	87	98
DpcDxe.efi	7	7	100
EbcDxe.efi	19	19	100
SmbiosDxe.efi	13	13	100
TcpDxe.efi	98	100	98
UdpDxe.efi	75	76	99

对于还原函数签名信息能力的评估, 我们选取了主流的个人计算机和服务器厂商 Intel、Msi、SuperMicro、Bull 的固件进行逆向分析. 对这些固件进行解包, 随机选取其中的 11 个 EFI 二进制文件, 使用逆向分析方法的实现对它们进行还原, 从而得到这些 EFI 的函数签名信息. 与人工分析的结果进行比较后, 我们将被还原的函数数目以及还原比例列在表 3 中.

表 3 逆向分析方法对函数签名信息的还原情况

厂商	UEFI	EFI	还原函数数目	总函数数目	还原比例(%)
Intel	PA0050.cap	Dxelpl.efi	56	79	71
		PeiCore.efi	95	125	76
Msi	E17H3IMS.109	PeiCore.efi	88	117	75
SuperMicro	BIOS_X12DPG-QR-1C55_20230202_1.4b_STDsp.bin	DxeCore.efi	235	285	82
		English.efi	5	8	60
		TcpDxe.efi	280	302	93
Bull	BIOS_SKD080.18.02.003.sign	TcpDxe.efi	172	202	85
		Udp4Dxe.efi	132	158	84
		CapsuleRuntime.efi	63	155	41
		ConPlatform.efi	68	73	93
		Dhcp4Dxe.efi	121	141	86

从表 3 可以看出, 我们的方法可以成功还原绝大部分函数签名信息, 包括其中的函数名以及参数数据结构. 在 11 个 EFI 模块中, 有 6 个模块的还原比例在 80%以上. 在 1 645 个函数中, 有 1 315 个函数签名信息被成功还原, 还原成功率为 79.94%. 在人工分析比对过程中, 我们发现没有被还原的函数中还存在一部分功能简单的函数(例如直接返回常数), 它们虽然没有被自动还原出来, 但是通过人工分析可以比较容易地了解它们的内部逻辑, 因此也无需还原这些功能简单的函数.

4.2 ReUEFuzzer测试能力评估

本节旨在通过与缺失符号信息的普通模糊测试器进行对比的方式评估使用符号信息辅助的 ReUEFuzzer 在 UEFI 模糊测试中的测试能力. 我们在 BIOS_SKD080.18.02.003.sign 固件中随机选取 10 个能够被我们的逆向方法进行分析的函数进行模糊测试. 在测试时, 我们将模糊测试器产生的输出当作函数参数传入函数, 并对每个函数进行两次测试. 由于该类模糊测试相关研究没有开源的工具发布, 我们将自己实现的模糊测试工具移除了符号信息作为对比基准, 称为普通模糊测试器, 以证明符号信息对于提升代码覆盖率的有效性. 两次测试的时间分别为 30 min, 对两种模糊测试器测试的可行性和覆盖率进行比较. 其中, 在使用符号信息时, 需要根据 ReUEFuzzer 的逆向分析模块得到的信息编写约束文件, 用来引导结构化的模糊测试器对函数进行测试, 对相同的数据结构而言, 这些约束文件可以复用. 在本实验中, 我们编写了大约 400 行 Python 代码, 用来生成不同的约束(见表 4).

实验结果表明, 在使用普通模糊测试器进行测试时, 10 个函数中有 6 个无法进行测试, 而 ReUEFuzzer 只有 2 个函数无法进行测试. 在都能进行模糊测试的 4 个函数中, 相同时间内, ReUEFuzzer 的路径覆盖率也比普通模糊测试器的路径覆盖率持平或有较大提升, 验证了 ReUEFuzzer 能够在单位时间内覆盖更多路径, 即

ReUEFuzzer 的时间效率比普通模糊测试器更高. 这说明基于启发式逆向方法的 UEFI 模糊测试器原型系统 ReUEFuzzer 相较于普通模糊测试器能够更加有效地对 UEFI 进行测试.

表 4 覆盖路径对比结果

模块名	函数名	推测函数名	覆盖路径	
			普通模糊测试器	ReUEFuzzer
TcpDxe.efi	sub_1348	TcpFlushPcb	0	2
TcpDxe.efi	sub_4A34	SockConnClosed	0	3
TcpDxe.efi	sub_586C	TcpSendAck	0	0
Udp4Dxe.efi	sub_2270	Udp4Transmit	1	3
Udp4Dxe.efi	sub_1A9C	Udp4CreateService	0	0
CapsuleRuntime.efi	sub_8C48	StringToBits	1	34
CapsuleRuntime.efi	sub_3D40	CompareGuid	1	1
ConPlatform.efi	sub_1068	IsGopSibling	0	1
Dhcp4Dxe.efi	sub_1C94	DhcpHandleSelect	0	4
Dhcp4Dxe.efi	sub_107C	DhcpComputeLease	6	6

进一步地, 我们分析了普通的模糊测试器无法测试的主要原因: 因为参数结构体中的二级指针无法被正确解析以及全局服务函数无法使用, 导致无法进行测试. 例如: TcpDxe.efi-sub_1348、Dhcp4Dxe.efi-sub_1C94 无法成功解析参数结构体中的二级指针而在运行时会发生崩溃; TcpDxe.efi-sub_4A34、ConPlatform.efi-sub_1068 因为无法使用 bootService 表中的函数而导致运行时发生崩溃. 对于 TcpDxe.efi-sub_586C, 因为子调用中的数据被解析成结构体, 导致出现非法内存访问, 从而使得在两种情况下都无法进行模糊测试. 对于 Udp4Dxe.efi-sub_1A9C, 因为局部变量被强制类型转化解析为结构体, 而导致出现非法内存访问的情况.

最后, 我们利用 ReUEFuzzer 对一些 UEFI 容易产生漏洞的关键位置进行测试, 例如 UEFIVariable、外设 IO、SMM 等. 我们使用批量测试的方法针对 Bull 公司的 BIOS_SKD080.18.02.003.sign 固件和 SuperMicro 公司的 BIOS_X12DPG-QR-1C55_20230202_1.4b_STDsp.bin 固件中的 371 个 EFI 文件进行批量测试, 测试中出现了 1 408 个 crashes, 由于 unicorn 动态插桩的机制, 在出现控制流劫持后会将跳转地址对应的内存解析为代码块, 从而不同的跳转地址会被识别为不同的分支, 这导致了 crashes 的冗余. 经过人工分析后, 我们发现这些 crashes 实际源于两处可疑漏洞, 并分别向对应的公司进行提交, 其中一个未知的 DXE 栈溢出漏洞被 SuperMicro 公司确认, 该漏洞对应的漏洞编号为 CNVD-2023-58048、CVE-2023-34853.

5 总结与展望

本文提出了一种启发式的 UEFI 逆向分析方法, 旨在还原 UEFI 固件中的基础服务信息和函数签名信息, 并利用该方法设计了一个基于启发式逆向的模糊测试器原型系统 ReUEFuzzer, 从而更有效地对 UEFI 进行第三方安全测试. 在由 EDK2 以及来自 4 个不同厂商的、解包为 525 个 EFI 文件的 UEFI 固件构成的测试数据集上, 测试逆向分析方法的有效性, 结果发现, 该方法可以成功地还原 UEFI 的符号信息. 其中, 基础服务信息识别成功率为 96.4%, 函数签名信息识别成功率为 79.94%. 进一步地, 使用能够被还原符号信息的函数进行对比实验, 结果表明, 相比缺失符号信息的普通模糊测试器, ReUEFuzzer 能够测试更多的函数, 并且有显著的覆盖率提升. 同时, 本工作还发现了一个零日漏洞, 漏洞编号为 CNVD-2023-58048、CVE-2023-34853.这些结果表明: 本文提出的方法和系统在 UEFI 漏洞检测方面具有有效性和实用性, 可以为 UEFI 安全提供一定的保障.

下一阶段, 我们计划针对需要人工构造引导模糊测试器的模型的部分进行优化. 使用动态分析技术, 例如符号执行对待执行函数片段尝试求解, 利用其收集的约束信息, 自动推断生成模型约束, 从而减少人工参与, 节约人工成本. 进一步地, 考虑 UEFI 中各模块间的相互引用关系, 从单一模块测试进行扩展, 以达成跨模块测试的效果, 从而发现更为复杂的缺陷.

References:

[1] Wikipedia. UEFI. 2023. https://en.wikipedia.org/wiki/UEFI#

[2] Wikipedia. BIOS. 2023. https://en.wikipedia.org/wiki/BIOS#
[3] CVE. 2023. https://cve.mitre.org/index.html
[4] Wikipedia. Common vulnerability scoring system. 2023. https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures
[5] Wikipedia. Rootkit. 2023. https://en.wikipedia.org/wiki/Rootkit
[6] Tianocore/edk2: edk ii. 2023. https://github.com/tianocore/edk2
[7] Supermicro data center server, blade, data storage, ai system. 2023. https://www.supermicro.com/en/home
[8] China National Vulnerability Database. 2023 (in Chinese). https://www.cnvd.org.cn/
[9] Intel | data center solutions, iot, and pc innovation. 2023. https://www.intel.com/content/www/us/en/homepage.html
[10] Yang Z, Viktorov Y, Yang J, *et al*. UEFI firmware fuzzing with simics virtual platform. In: Proc. of the 57th ACM/IEEE Design Automation Conf. (DAC). San Francisco: IEEE, 2020. 1−6.
[11] Wikipedia. Simics. 2023. https://en.wikipedia.org/wiki/Simics #
[12] Yin J, Li M, Wu W, *et al*. Finding SMM privilege-escalation vulnerabilities in UEFI firmware with protocol-centric static analysis. In: Proc. of the 2022 IEEE Symp. on Security and Privacy (SP). San Francisco: IEEE, 2022. 1623−1637.
[13] Manes VJM, Han H, Han C, *et al*. The art, science, and engineering of fuzzing: A survey. arXiv:1812.00140v4, 2019.
[14] Rawat S, Jain V, Kumar A, *et al*. VUzzer: Application-aware evolutionary fuzzing. In: Proc. of the 2017 Network and Distributed System Security Symp. San Diego: Internet Society, 2017.
[15] Pham VT, Böhme M, Santosa AE, *et al*. Smart greybox fuzzing. arXiv:1811.09447v1, 2018.
[16] Nccgroup/TriforceLinuxSyscallFuzzer. 2023. https://github.com/nccgroup/TriforceLinuxSyscallFuzzer
[17] LibFuzzer—A library for coverage-guided fuzz testing. 2023. https://llvm.org/docs/LibFuzzer.html
[18] QEMU. 2023. https://www.qemu.org/
[19] Chen DD, Egele M, Woo M, *et al*. Towards automated dynamic analysis for linux-based embedded firmware. In: Proc. of the 2016 Network and Distributed System Security Symp. San Diego: Internet Society, 2016.
[20] Kim M, Kim D, Kim E, *et al*. FirmAE: Towards large-scale emulation of iot firmware for dynamic analysis. In: Proc. of the Annual Computer Security Applications Conf. Austin: ACM, 2020. 733−745.
[21] Zheng Y, Davanian A, Yin H, *et al*. FIRM-AFL: High-throughput greybox fuzzing of IoT firmware via augmented process emulation. In: Proc. of the 28th USENIX Security Symp. 2019. 1099−1114.
[22] Qemu STM32. 2023. http://beckus.github.io/qemu_stm32/
[23] Zaddach J, Bruno L, Francillon A, *et al*. Avatar: A framework to support dynamic security analysis of embedded systems' firmwares. In: Proc. of the 2014 Network and Distributed System Security Symp. San Diego: Internet Society, 2014.
[24] Kammerstetter M, Platzer C, Kastner W. Prospect: Peripheral proxying supported embedded code testing. In: Proc. of the 9th ACM Symp. on Information, Computer and Communications Security. Kyoto: ACM, 2014. 329−340.
[25] Kammerstetter M, Burian D, Kastner W. Embedded security testing with peripheral device caching and runtime program state approximation. In: Proc. of the 10th Int'l Conf. on Emerging Security Information, Systems and Technologies. 2016.
[26] Unified extensible firmware interface (UEFI) specification. Release 2.10. UEFI Forum, Inc., 2022
[27] Wikipedia. Assertion (software development). 2023. https://en.wikipedia.org/wiki/Assertion_(software_development)
[28] Haq IU, Caballero J. A survey of binary code similarity. arXiv:1909.11424v1, 2019.
[29] Dullien T, Rolles R. Graph-based comparison of executable objects. In: Proc. of the Actes du Symposium SSTIC05. 2005.
[30] David Y, Partush N, Yahav E. Statistical similarity of binaries. In: Proc. of the 37th ACM SIGPLAN Conf. on Programming Language Design and Implementation. Santa Barbara: ACM, 2016. 266−280.
[31] Chandramohan M, Xue Y, Xu Z, *et al*. BinGo: Cross-architecture cross-OS binary search. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Seattle: ACM, 2016. 678−689.
[32] Hex-rays. 2023. https://hex-rays.com/ida-pro/
[33] Lenovo. 2023. https://www.lenovo.com/us/en/
[34] LongSoft/UEFIT. 2023. https://github.com/LongSoft/UEFITool
[35] IDAPython/src: Idapython project for hex-ray's IDA Pro. 2023. https://github.com/idapython/src
[36] Zynamics.com/software. 2023. https://www.zynamics.com/software.html
[37] Qilingframework/qiling: A true instrumentable binary emulation framework. 2023. https://github.com/qilingframework/qiling
[38] AFLplusplus/AFLplusplus. 2023. https://github.com/AFLplusplus/AFLplusplus
[39] Unicorn-Engine. 2023. https://github.com/unicorn-engine/unicorn
[40] Zalewski M. American fuzzy lop. 2023. https://lcamtuf.coredump.cx/afl/

附中文参考文献:

[8] 国家信息安全漏洞共享平台. 2023. https://www.cnvd.org.cn/

林欣康(1999－), 男, 硕士生, CCF 学生会员, 主要研究领域为固件安全.

顾匡愚(1999－), 男, 硕士生, CCF 学生会员, 主要研究领域为固件安全.

赵磊(1985－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为软件及系统安全, 二进制程序的安全分析, 软件漏洞的自动化挖掘和分析.