

基于软硬件协同形式验证的固件漏洞分析技术

张朋辉, 田曦, 楼康威

(国防科学技术大学电子科学与工程学院, 湖南 长沙 410003)

摘要: 为了系统高效地分析固件中潜在的安全隐患, 提出了一种基于行为时序逻辑 TLA 的软硬件协同形式验证方法。通过对固件工作过程中的软硬件交互机制进行形式建模分析, 在动态调整攻击模型的基础上, 发现了固件更新过程中存在的安全漏洞, 并通过实验证实了该漏洞的存在, 从而证明了形式验证方法的可靠性。

关键词: 固件安全; TLA; 形式验证; 漏洞分析; 软硬件协同

中图分类号: TP309

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2016.00071

Firmware vulnerability analysis based on formal verification of software and hardware

ZHANG Peng-hui, TIAN-Xi, LOU Kang-wei

(School of Electronic Science and Engineering, National University of Defense Technology, Changsha 410003, China)

Abstract: In order to analyze the potential vulnerabilities in the firmware systematically and effectively, a formal verification method based on TLA, in a collaborated form of software and hardware was proposed. With this method, the interaction mechanism of software and hardware in the computer boot process was modeled and analyzed. By adjusting the attack model, a secure vulnerability in the update process of the firmware was found, and its existence by an experiment, which proved the reliability of formal verification was demonstrated.

Key words: firmware security, TLA, formal verification, vulnerability analysis, software and hardware collaboration

1 引言

固件 (firmware) 是计算机上电后最先执行的代码, 主要负责系统硬件平台的初始化及加载操作系统, 并最终将控制权传递给它。早期的固件版本叫作基本输入输出系统 (BIOS, basic input output system), 由于其开发效率低, 功能扩展性差以及更新机制不完善等缺点, 已经逐渐被支持图形界面和鼠标操作的统一可扩展固件接口 (UEFI, unified extensible firmware interface) 所取代^[1]。UEFI 在充分集成 BIOS 功能的基础上添加

了许多新特性, 如完全使用 C 语言开发、更新机制简单快速以及方便跨平台移植等。固件常驻在主板上的 Flash 芯片中, 即使在重装操作系统后依然不受影响。由于固件对计算机的安全至关重要, 固件厂商以及芯片公司均提供了专门的机制来保护其内容免受干扰, 但是当遇到需要修复固件漏洞或者添加一些新功能的情况时, 又必须有一种可靠的方案来对其进行更新, 这种机制一定程度上给固件内容的安全带来了风险。一旦固件的代码受到感染, 系统启动过程中的其他组件也会受到安全威胁, 因此, Brossard 等^[2]指出, 只

收稿日期: 2016-05-22; 修回日期: 2016-06-27。通信作者: 张朋辉, zph.111@163.com

基金项目: 国家高技术研究发展计划基金资助项目 (No.2015AA2557)

Foundation Item: The National High Technology Research and Development Program (No.2015AA2557)

要取得了固件代码的控制权, 任意控制计算机的行为就成为了可能。

针对固件安全性的研究目前主要基于软件漏洞分析方法, 大多局限在代码层面, 通过使用软件安全研究中常用的手工测试、Fuzzing 测试技术^[3,4]、二进制比对技术^[5]以及动态分析技术^[6]等, 尝试从固件的源代码中发现漏洞。这些方法一方面工作量较大, 需要研究人员具有较强的软件开发和测试能力, 另一方面可持续性不足, 无法形成一个系统高效的检测方法。除此以外, 由于纯粹从代码方面考虑, 这些方法遵循的是传统代码漏洞挖掘方法的原则, 无法结合硬件设备的运行来验证代码行为, 更多针对的是源代码中是否存在恶意代码进行检测, 没有关注固件在运行过程中同硬件架构的交互细节, 即软硬件协同细节, 进而也不能对硬件的行为和软硬件交互进行建模分析, 检测不出实际运行过程中可能产生的一些漏洞。

形式验证的方法能够很好地弥补上述传统方法的不足, 通过使用数学语言对软件或者硬件内部模块抽象建模, 可以非常直观地研究其内部运行机制。另外, 形式语言可以对已有的软硬件模型综合分析, 随着建模细节的增多, 形式语言不仅可以对软硬件模块内部的运行进行检验, 还可以对模块之间的信息流进行分析验证。然而, 当前形式验证的方法大多集中在软件^[7,8]和协议^[9]的安全性设计及验证中, 少部分面向硬件的研究也仅仅侧重门级逻辑的检验和功能设计的正确性验证等^[10,11], 没有或很少涉及固件中软硬件协同工作的底层细节。

为此, 本文提出了基于 TLA (temporal logical of actions) 行为时序逻辑的软硬件协同形式验证分析方法, 以固件运行过程中的软硬件协同工作机制为主要内容进行安全研究。不同于传统的单一从硬件或软件出发进行安全验证, 本文以硬件工作流程为基础, 结合软件对硬件的调用过程综合分析了计算机开机过程中可能存在的风险。首先, 通过 TLA+语言对固件的架构体系和软硬件执行过程进行形式建模分析, 给出了固件驱动硬件的动作以及硬件对固件的反馈判断信息等形式模型; 然后, 结合攻击模型 Attack 全面分析软硬件协同工作过程中可能存在的安全风险, 通过分

析模型检测器 TLC (TLA+ checker) 所提供的错误路径, 检验了固件对硬件行为的控制是否存在被恶意修改的可能及其给计算机安全带来的影响; 最后, 通过实验证明了这种方法的可行性。

TLA 最初是由 Lamport 设计, 用来描述和推断操作系统的状态^[12]。TLA+是基于此逻辑扩展出的一套完整性描述语言。借助这一语言, 可以用较简单抽象的形式模型来描述一些复杂的软硬件架构, 同时结合模型检测工具 TLC, 能够在诸如软件安全分析、协议安全分析、硬件设计完整性验证以及软硬件体系漏洞挖掘等方面极大地提高研究效率, 这一特点使其在目前的研究工作中得到了越来越广泛的使用。TLC 是形式语言 TLA+的模型检测工具, 用于检查 TLA+规范中定义的性质能否被满足, 还可以用于在 TLA+规范中查找错误或者是在定义的系统模型中挖掘安全隐患^[13]。实际运行的时候, TLC 工具可以将 TLA+规范中所有独立可达的状态统计出来。

2 行为时序逻辑 TLA 以及其 TLC 工具

一个 TLA+模型通常由许多模块 (module) 组成, 模块中包含模型中用到的所有变量、状态、行为 (behavior)、动作 (action)、状态谓词 (predicate)、各种操作符以及可选的持续性约束条件等。描述对象系统所具有的性质可以通过抽象的变量或者常量来表示, 状态就是一个系统中所有变量的一组特定取值, 变量取值的变化表示系统从一种状态进入了另一种状态。行为由系统中一系列连续发生的状态组成, 本文通过定义各种可能的行为来表示一个系统的规范。TLA+可以对状态的转变做出描述, 新旧 2 种状态下变量的表示方法不一样。变量的新状态会在表示符号的右上角加一个 “'”, 如果用 x 来表示某种系统属性, 那么下一状态下其表示方法为 x' 。如果一个 TLA+公式中, 既含有形如 x 的变量, 又含有形如 x' 的变量, 则表征的是一种新旧状态之间的转换关系, 那么这样的公式就是一个动作, 最常见的动作就是状态转换关系 (next state relation)。状态谓词是指不带 “'” 的时序公式, 如初始状态谓词 (initial predicate)。TLA+的操作符有很多, 理论上基础数学中用于集合、量化、关系等的操作

符都可以使用,比如“ $\subset$ ”、“ $\forall$ ”和“ $>$ ”等。除此以外,在表征系统状态演进时,TLA+还引入了一些新的操作符,最重要的就是“ $\Box$ ”和“ $\Diamond$ ”。“ $\Box$ ”表示“always”,即对任意状态 P ,时序公式“ $\Box P$ ”表示状态 P 会一直持续下去。“ $\Diamond$ ”表示“eventually”,即对任意状态 P ,时序公式“ $\Diamond P$ ”表示状态 P 最终总会出现。需要注意的是,如果 P 表示的是一个动作,有时也会用这种表示方法来表明系统特性,这时需要改变书写方式,记为 $\Box\langle P \rangle_{vars}$ 或者 $\Diamond\langle P \rangle_{vars}$,下标表示在动作 P 中变量 $vars$ 可以保持不变。持续性制约条件描述的是系统中确定会发生的事,是在任意一个时刻都不会违背的系统特性,分为较弱的制约条件(WF)以及较强的制约条件(SF)。例如

$$SF_{vars}(A) \triangleq \Box\Diamond ENABLED\langle A \rangle_{vars} \Rightarrow \Box\Diamond\langle A \rangle_{vars} \quad (1)$$

作为持续性条件时,表示在以后的状态变化中,只要动作 A 的激活条件能够满足,则动作 A 在系统的行为中就必然会出现。如果 A 不断地被激活,就会有无限多的 A 出现在系统中。

TLA+中行为的真假通过时序公式来表示,时序公式 F 会给行为 σ 赋一个布尔值,记作 $\sigma \models F$ 。当且仅当上式为真时,认为行为 σ 满足公式 F 。通常按式(2)定义一个系统

$$Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge Liveness \quad (2)$$

其中, $Init$ 是系统的初始状态谓词,表征了系统最开始的一些性质。 $\Box[Next]_{vars}$ 表示系统的状态转换关系, $Next$ 表示一个转换动作,下标 $vars$ 表示在 2 个相邻的状态之间,部分变量的值可以保持不变。 $Liveness$ 是持续性限制条件,表述了系统在以后的状态转变中确定会出现的情况。这 3 部分组合在一起,就是对一个系统的完整定义。

在使用 TLC 检测器验证模型之前,可以选择使用初始谓词和状态转换关系来描述系统规范,也可以使用表述系统行为的时序公式来描述,如图 1 所示。

此外,执行模型检测的一些附加检测条件也可以进行选择,可以分别对规范中的系统性质(property)、类型不变量(typeInvariant)以及是否存在死循环(deadlock)等进行检测,如图 2 所示。

图1 规范行为的描述方式

图2 TLC 检测内容

死循环检测可以对系统规范的状态转换流程做出检测,验证系统是否存在执行隐患。类型不变量检测主要是确保系统中某些属性不被更改,可以检测系统抵御恶意篡改风险的能力。系统性质检测侧重系统的功能验证,可以检查规范中定义或者描述的功能最终能否正常实现。三者可以同时选择,对系统规范做出比较全面的检测,也可以指定检测某一项,针对系统规范的某一方面做出更有针对性的验证。当 TLC 工具在验证规范的过程中发现有漏洞隐患存在时,就会以错误路径的方式自动化给出一条反例,指出错误路径中所列出的一系列状态转换,最终可以导致对系统规范待验证性质的违背。TLC 工具寻找错误路径是自动进行的,而且只提供一个状态转换最少的路径。

3 软硬件协同工作过程及其 TLA 建模

从计算机上电到完全进入操作系统是一个十分复杂的过程,在这个过程中,计算机首先会初始化固件的运行环境,包括 CMOS 芯片、处理器、

芯片组、随机存储器 RAM 等。TLA+形式模型具有较强的针对性和可扩充性。利用这一特点,可以将上述部件内部的相关操作抽象为某一个具体的状态,而不用将其所有的工作机制都写入规范^[14]。例如,大多数计算机在上电之后执行的第一条指令都会是 JMP 命令,用来将指令指针指向 Flash 芯片从而执行固件代码,本文不会描述这种相关性不大的底层细节,由于研究的重心是计算机固件及其同硬件的交互,因此,针对其他部件的一些工作原理本文进行了有保留的抽象。

3.1 软硬件协同工作过程

如图3所示,本文定义了从计算机上电开始到操作系统完全启动过程中软硬件协同工作的几个关键步骤。计算机刚上电时,固件代码执行的第一个工作通常是初始化 CPU 及检查当前微码(Microcode)文件寄存器中的内容是否支持当前处理器的具体型号。不匹配的处理器在实际工作过程中往往会产生许多问题,如发热量过大、系统频繁出现故障等,微码文件的扩充通常需要通

过固件更新来实现。接下来是芯片组的初始化,为后续执行过程准备相关寄存器等硬件资源。之后固件会分析 CMOS 中保存的信息,对重启原因进行检查,判断上次系统结束运行时是正常关机操作,还是其他原因导致异常关机。如果是正常开机,就按照正常开机流程直接执行后续步骤,反之,则有相应的分支处理流程,这个阶段叫做软重启(soft reboot)判断。软重启判断结束之后,如果确定执行的是软重启,则进行相应的软重启原因排查。导致系统执行软重启的原因很多,系统更新、固件更新、新硬件的发现和驱动安装以及某些底层软件的更新都有可能引起操作系统执行软重启。由于研究的对象是计算机固件,因此本文将操作系统更新以及后续操作抽象为一个状态,将硬件改动和驱动程序更新同样也抽象为一个状态。如果确定软重启的原因是固件更新,系统还会进一步执行判断,检查执行更新操作的条件是否满足,最主要的是对固件存储芯片 Flash 进行可写标志检查和对固件更新文件执行合法性

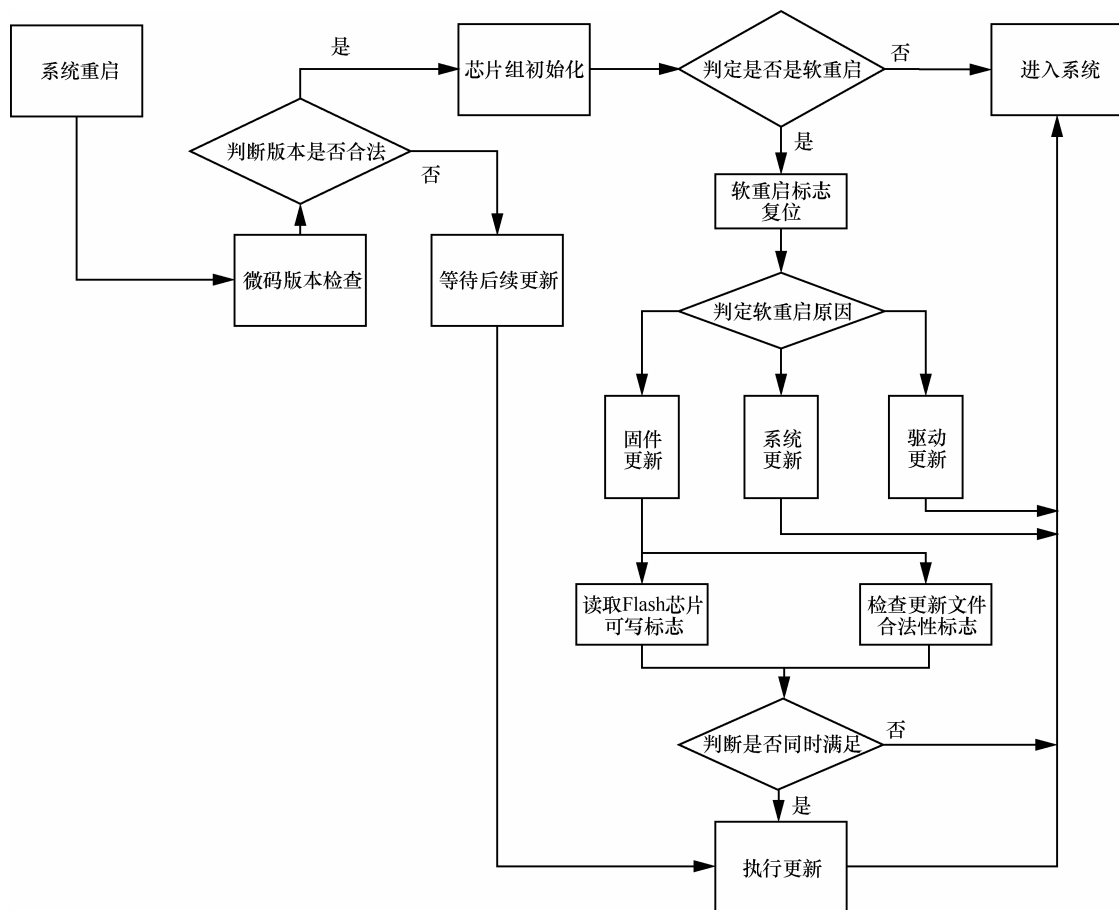


图3 软硬件协同工作流程

检查^[15]。前者是针对更新操作的合法性，后者是针对更新文件的安全性。执行更新需要这 2 个条件同时满足，任意一个条件不满足时都会导致固件更新失败，直接进入操作系统。

3.2 软硬件协同工作过程建模

通过前文对计算机固件更新过程的描述，本文从中抽象出了几个关键的动作，包括判断微码文件合法性的 `CheckMicrocode`，判断本次启动是否为软重启的 `IsSoftReboot` 以及判断软重启原因的 `IsFirmwareUpdate`、`IsOSUpdate` 和 `IsDriverUpdate` 等，还有检查更新条件是否满足的 `UpdateCheck`，最后是固件更新执行结束后反映系统状态的 `EnterOS` 等。

在对固件系统特性进行抽象的过程中，用到了几个变量，这些变量同样也是系统运行过程中各组件间实现通信的媒介。在描述微码文件合法性判定时，用 `Microcode_verification_sign` 作为标志位同 CPU 型号进行比较。在描述软重启判定条件时，用 `Soft_reboot_sign` 表示判定标志位。同样地，用 `Reboot_reason_sign` 表示软重启的原因。在判定固件更新是否执行时，用 `Write_flash_sign` 变量和 `Update_verification_sign` 变量分别表示 Flash 芯片的可写标志位和固件更新文件的安全性检验标志位。用 `Firmware_security_data` 表示固件更新文件中经过改动后的安全相关内容，可以看作是固件版本，`System_security_data` 表征系统中当前固件的版本情况。没有更新时，令这 2 个变量取值相等，表示固件文件的版本和系统中存在的固件版本相同。用这 2 个变量的不同作为判断系统中是否存在待更新固件文件的依据，进而，可以通过判定这 2 个变量的值来观察系统当前的状态，判断系统是否成功执行了更新。为了更加抽象地表示固件内容和固件更新的执行情况，本文引入了 3 个常量 `OldFirm`、`NewFirm`、`AtkFirm`。用 `OldFirm` 表示更新之前的固件版本、`NewFirm` 表示正常更新过程中更新文件的固件版本、`AtkFirm` 表示经过篡改的固件版本。

系统的初始状态定义为

$$Init \triangleq TypeInvariant \wedge Varset \quad (3)$$

由于 TLA+语言中没有明确定义数据类型，在给变量赋值的时候只会检查形式上是否有语法错误，并不会对赋值的合法性做出检查，因此，用

`TypeInvariant` 将变量类型以系统性质的形式固定下来，具体形式如下。

$$\begin{aligned} TypeInvariant \triangleq Varset \in [& Microcode_verification_sign: \{0,1\} \\ & Soft_reboot_sign: \{0,1\}, Reboot_reason_sign: \{0,1,2,3\}, \\ & Write_flash_sign: \{0,1\}, Update_verification_sign: \{0,1\} \\ & Firmware_security_data: \{OldFirm, NewFirm, AtkFirm\} \\ & System_security_data: \{OldFirm, NewFirm, AtkFirm\}] \end{aligned} \quad (4)$$

式(4)一方面可以将变量的取值类型固化，防止出现逻辑错误，另一方面可以便于变量的添加和引用等操作。这样的类型不变定义叫做一个记录(record)，之后对变量进行操作时，只需要通过类似 `Varset.Soft_reboot_sign` 的形式引用就可以了。

状态转换关系需要包含各个阶段的 TLA+公式，每一个阶段都需要详细定义，包括其触发激活条件、判断语句以及对状态变量值采取的相关操作，然后，采用逻辑操作符“ $\vee$ ”来连接每一种可能的操作^[16]。

$$\begin{aligned} Next \triangleq & \vee CheckMicrocode \vee IsSoftReboot \vee \\ & IsFirmwareUpdate \vee UpdateCheck \vee \\ & IsOSUpdate \vee IsDriverUpdate \vee EnterOS \end{aligned} \quad (5)$$

式(5)将固件系统从开机上电到完全进入操作系统的步骤用 TLA+语言做了抽象，其含义是，在任意时刻，上述几个动作都有可能发生。TLA+定义的只是系统中可能发生的所有动作，在定义状态转换关系 `Next` 的时候并不会规定各个动作发生的先后顺序，能够表明动作之间执行顺序的语句只会在各个动作的详细定义中，以触发激活条件的形式给出。这样 TLC 工具在对规范做完整性验证的时候，会根据列出的动作激活条件以及下一步的具体操作，生成一个完整的状态执行流程。

在描述持续性条件时，考虑到实际系统运行过程中会有其他不可测情况的出现，如各种分支情况，本文采用弱持续性条件作为约束

$$\begin{aligned} WF_{Varset}(Next) \triangleq & \Box(\Box ENABLED\langle Next \rangle_{Varset} \\ & \Rightarrow \Diamond\langle Next \rangle_{Varset}) \end{aligned} \quad (6)$$

式(6)表明只有动作 `Next` 被永久激活，即 `Next` 动作的激活条件总能被满足时，动作 `Next` 才肯定会出现，而非强持续条件所描述的，只要 `Next` 动作的激活条件出现，`Next` 就一定会执行。通过这

种定义,在模型实际执行的过程中加入了对容错的支持。

最后,综合上面的工作,给出了系统规范的完整定义式

$$Spec \triangleq Init \wedge \Box [Next]_{Varset} \wedge \Box \Diamond WF_{Varset}(Next) \quad (7)$$

和

$$Spec \Rightarrow \Box TypeInvariant \wedge \Box \Diamond [EnterOS]_{Varset} \quad (8)$$

其中,式(7)是基于TLA+形式语言给出的一个固件系统完整定义,涵盖了第3.1节中描述的系统架构和软硬件工作过程。式(8)则是将系统规范的特性以性质的形式给出,其含义是,无论在何种情况下,系统的运行始终不会违背类型不变定义,所有的变量都会在给定的初始范围内取值。另外,随着状态的推移,系统最终总会停留在EnterOS上,因为这个动作意味着固件执行阶段的结束,计算机最终进入操作系统启动阶段。

4 攻击模型的建立以及TLC验证结论

本文的第3节对固件本身及其同硬件的交互过程做出了形式化建模,按照第3.1节中介绍的工作过程进行了详细的分步描述,得到了一个固件执行系统规范,给出了初始状态谓词、状态转换关系以及持续性条件的详细定义。在此基础上,本文对固件更新过程的安全性进行了研究。

4.1 攻击模型 Attack

系统规范的漏洞检查并不等同于对其进行正确性验证,因为正确性验证所做的工作是检查模型能否严格按照规范中定义的流程执行,检验的是系统中定义的性质最终能否实现。而漏洞检查则不同,这种情况下,一般会假设系统规范在理想环境下是正确的,关注的重点是系统在受到攻

击的时候会不会存在安全隐患。因此,要做漏洞检查分析,还需要将攻击者也用模型化的方法加入系统规范中^[17]。

在软硬件结合的背景下,给出漏洞的定义。漏洞是一种特殊的状态元素(配置寄存器)和控制信号的组合,这种组合会导致对信息安全目标的违背或者会导致绕开信息安全的保护机制。具体到固件和硬件的协同工作过程中,漏洞就是固件发出某种非正常的操作指令,可以控制计算机某些寄存器的配置,进而影响硬件特定行为方式的判定,导致固件中某些重要的数据或工作方式存在被恶意修改的可能。

用TLA+语言定义攻击者模型的时候,由于形式语言的抽象性,可以将多种攻击能力同时合并到攻击模型中,然后检查系统的安全性。显然在攻击能力足够强的时候系统必然会出现一些安全问题,但是这种情况下产生的许多问题都是在非正常假设条件下所得到的结果,即攻击者的能力可以任意配置且没有任何限制,因此,生成的错误路径并非一定具有实际可行性,所以,合理地选择攻击模型的特权以及能力就显得尤为重要。

一个比较有效的研究方法是按照从强到弱动态调节攻击者能力,当攻击者能力过强,甚至已经与实际情况相违背时,TLC验证工具必然提供一些错误路径,对这些路径需要加以判断,如果确定是不可行的,可以动态地修改规范中攻击模型的权限及能力,然后加载到TLC工具中重新进行验证,直到发现实际可行的漏洞,如图4所示。

基于这一思想,首先,确定了一个最强的攻击模型,即该模型拥有直接修改固件内容的权限,但此时TLC工具给出的错误路径经分析后并不

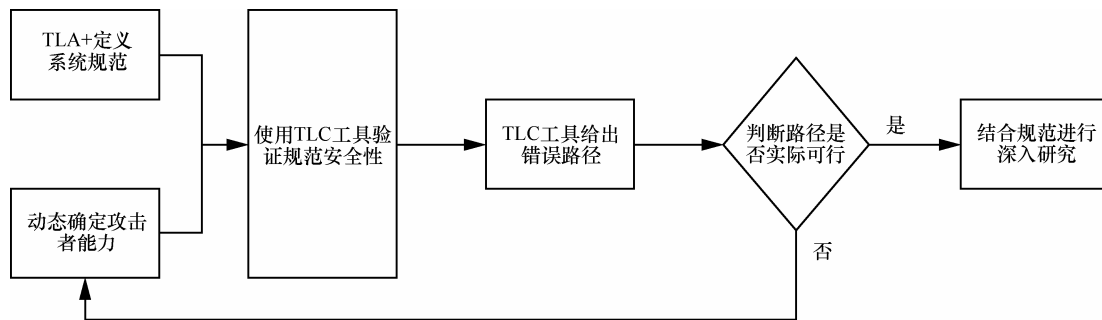


图4 研究方法改进过程

能被实际使用，因为 Flash 芯片的修改还涉及到一些判定机制，想要修改其内容必须符合之前所述的几个条件；之后，在攻击模型中添加了一个判定条件，发现 TLC 同样不会提供错误路径，这是因为在这种条件下攻击模型没有足够的权限对 Flash 芯片进行修改，并非规范中不存在漏洞。通过动态调整，最终确定的攻击者模型公式如下。

$$\begin{aligned} Attack \triangleq Varset' = \\ [Varset \text{ EXCEPT } !Write_flash_sign = 1, \\ !Update_verification_sign = 1, \\ !Firmware_security_data = AtkFirm] \quad (9) \end{aligned}$$

式(9)表明，本文所定义的攻击者，能够更改 Flash 芯片的可写标志位，使攻击者有可能修改固件存储芯片 Flash 中的内容，满足了固件执行更新判定过程中的第 1 个条件。同时，该模型能够满足更新文件合法性验证的要求，即满足了第 2 个条件。另外，要求攻击者能够修改计算机中的固件内容，为了区别计算机中原有的固件版本 OldFirm 和合法更新文件中的版本 NewFirm，标记修改后的新固件为 AtkFirm。

4.2 TLC 检测结果

为了查找固件系统规范中的漏洞，强调其内容不可随意更改，结合前述攻击者模型，本文在规范中加入了一个针对固件中同安全性内容相关的系统性质，强调更新后的固件版本不能为 AtkFirm，即

$$Property \triangleq Varset.System_security_data \neq AtkFirm \quad (10)$$

这样，TLC 工具在执行验证的时候，就会基于此性质自动化地寻找错误路径，结果如图 5 所示。

图 5 中的错误路径一共包含 6 个状态，详细列出了系统规范中可能存在安全风险的状态转换方式。结合系统规范和攻击者模型分析该错误路径可知，TLC 工具提供了一个潜在的软硬件协同工作漏洞，即如果攻击者具备修改 Flash 芯片可写标志的能力，同时自身又满足更新文件安全性的要求，那么这种情况下计算机固件的更新过程存在安全隐患。

5 错误路径实现及分析

根据 TLA 模型以及 TLC 验证工具得到的结果，本文选取计算机的固件更新文件作为攻击载体。之所以选择固件更新文件，是因为它本身就已经满足前述 2 个判定条件的要求。第 3 个条件要求攻击者能够加载区别于系统中现有内容的新固件文件，可以通过修改固件更新文件中的内容，然后执行更新来实现。在实际研究过程中发现，在固件更新过程中，并不能够保证更新文件中所有模块的合法性，其验证过程更多是侧重厂商号、版本号以及对硬件设备兼容性上，即使存在对更新文件完整性的验证，也很难对封装在文件内部的一些模块进行深入检查，其中最重要的就是 PCI Expansion Rom^[18]。

5.1 PCI Expansion ROM 文件

PCI Expansion ROM 是 PCI 规范中定义的一个文件^[19]，允许设备通过该 ROM 文件的代码执行设备初始化操作。通常一个 Expansion ROM 可以包含几个不同的 image 文件，以便适配不同的主板架构和处理器架构。每一个 image 都是从 512 byte 边界处开始，而且必须包含 PCI Expansion ROM 头文件。x86/x64 架构平台的 PCI 头文件格式如表 1 所示。

Error-Trace	
Name	Value
▲ ▲ <Initial predicate>	State (num = 1)
▸ ▮ Varset	[System_security_data -> 1, Microcode_verification_sign -> 1, Soft_reboot_sign -> 1, Reboot_reason_sign -> 1, Write_flash_sign -> 0, Update_verification_sign -> 0, Firmware_security_data -> 2]
▲ ▲ <Action line 24, col 17 to lin	State (num = 2)
▸ ▮ Varset	[System_security_data -> 1, Microcode_verification_sign -> 1, Soft_reboot_sign -> 0, Reboot_reason_sign -> 1, Write_flash_sign -> 0, Update_verification_sign -> 0, Firmware_security_data -> 2]
▲ ▲ <Action line 27, col 21 to lin	State (num = 3)
▸ ▮ Varset	[System_security_data -> 1, Microcode_verification_sign -> 1, Soft_reboot_sign -> 0, Reboot_reason_sign -> 0, Write_flash_sign -> 0, Update_verification_sign -> 0, Firmware_security_data -> 2]
▲ ▲ <Action line 16, col 11 to lin	State (num = 4)
▸ ▮ Varset	[System_security_data -> 1, Microcode_verification_sign -> 1, Soft_reboot_sign -> 0, Reboot_reason_sign -> 0, Write_flash_sign -> 1, Update_verification_sign -> 1, Firmware_security_data -> 3]
▲ ▲ <Action line 31, col 16 to lin	State (num = 5)
▸ ▮ Varset	[System_security_data -> 1, Microcode_verification_sign -> 1, Soft_reboot_sign -> 0, Reboot_reason_sign -> 0, Write_flash_sign -> 0, Update_verification_sign -> 0, Firmware_security_data -> 3]
▲ ▲ <Action line 37, col 19 to lin	State (num = 6)
▸ ▮ Varset	[System_security_data -> 3, Microcode_verification_sign -> 1, Soft_reboot_sign -> 0, Reboot_reason_sign -> 0, Write_flash_sign -> 0, Update_verification_sign -> 0, Firmware_security_data -> 3]

图 5 错误路径

表 1 PCI Expansion ROM 头文件格式

偏移	长度	数值	功能描述
0h	1	55h	ROM 文件签名第 1 个字节
1h	1	AAh	ROM 文件签名第 2 个字节
2h	1	xx	代码文件大小, 单位是 512 byte
3h	3	xx	INIT 函数的入口, 固件执行一个 FAR CALL 跳转到这里
6h~17h	12	xx	保留
18h~19h	2	xx	指向 PCI 数据结构

前 2 个字节是 PCI Expansion ROM 文件的签名, 固定为 0xAA55, 用于 ROM 文件的检索。其中, 偏移 03h 处是设备初始化函数的入口, 固件在初始化 PCI 设备时会跳转到这里将执行权交给 PCI Expansion ROM。偏移 18h~19h 指向 PCI 的数据结构 (data structure), PCI 的数据结构必须位于第一个 64 kB 段内, 里面包含了设备码 (device ID) 和厂商码 (vendor ID) 以及代码类型等内容。

在检测到 PCI 设备需要相应的 ROM 文件协助初始化之后, 固件用 2 个步骤搜索其对应的 PCI Expansion ROM 文件。首先, 检测 PCI 设备在配置空间里是否有可执行的 ROM 文件, 如果有, 就将该文件映射到地址空间的一个空闲区域; 然后, 通过检测双字节 “AA55h” 签名来搜索 ROM

文件, 通过设备码和厂商码进行匹配, 如果找到, 就通过一个 CALL 指令跳转到这里执行, 如果没找到, 则跳过此设备, 返回固件的主线程继续执行^[20]。

5.2 实验结果及分析

在分析了固件的启动流程以及 TLA+模型之后, 本文在 x86 平台下针对常见的 Award 固件展开了研究。首先, 使用官方备份和更新工具提取出 Flash 芯片中的固件文件; 然后, 使用 CBROM 工具提取出了其中的 PCI 网卡模块, 如图 6 所示。

图 6 中 AR815x.lom 文件就是此固件文件中内置的网卡 PCI Expansion ROM 文件, 其具体信息如图 7 所示。

22. PCI ROM[B] 0AE00h(43.50K) 0676Dh(25.86K) AR815x.lom

图 6 提取 Award 固件中网卡模块的 PCI Expansion ROM 文件

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00000000h:	55	AA	80	E8	C8	12	CB	48	3E	03	00	00	00	00	00	00
00000010h:	00	00	00	00	00	00	20	00	40	00	70	00	8D	64	24	00
00000020h:	55	4E	44	49	16	6C	00	00	01	02	BE	0C	00	08	60	4E
00000030h:	6A	33	50	43	49	52	8D	A4	24	00	00	00	00	8D	49	00
00000040h:	50	43	49	52	68	19	83	10	18	00	18	00	00	02	00	00
00000050h:	80	00	01	02	00	80	00	00	62	10	63	10	60	20	73	10
00000060h:	62	20	83	10	00	00	8D	A4	24	00	00	00	00	8D	49	00
00000070h:	24	50	6E	50	01	02	00	00	00	AE	00	00	00	00	BF	00
00000080h:	F1	00	02	00	00	E4	00	00	00	00	7B	0C	00	00	00	00
00000090h:	0D	0A	43	6F	70	79	72	69	67	68	74	20	28	43	29	20
000000a0h:	31	39	39	37	2D	32	30	30	30	20	20	49	6E	74	65	6C
000000b0h:	20	43	6F	72	70	6F	72	61	74	69	6F	6E	0D	0A	00	49
000000c0h:	6E	74	65	6C	20	43	6F	72	70	6F	72	61	74	69	6F	6E
000000d0h:	00	49	6E	74	65	6C	20	55	4E	44	49	2C	20	50	58	45
000000e0h:	2D	32	2E	30	20	28	62	75	69	6C	64	20	30	38	33	29
000000f0h:	00	41	74	68	65	72	6F	73	20	42	6F	6F	74	20	41	67
00000100h:	65	6E	74	00	0D	0A	46	6F	72	20	41	74	68	65	72	6F
00000110h:	73	20	50	43	49	45	20	45	74	68	65	72	6E	65	74	20
00000120h:	43	6F	6E	74	72	6F	6C	6C	65	72	20	76	32	2E	30	2E
00000130h:	32	2E	37	28	31	31	2F	30	32	2F	31	30	29	0D	0A	00
00000140h:	28	62	75	69	6C	64	20	30	38	33	29	00	00	00	00	00

图 7 网卡模块的 PCI Expansion ROM 文件信息

由 PCI Expansion ROM 的规范可知, 图 7 中 0018h 和 0019h 偏移指向 PCI 的数据结构, 即偏移地址为 0x0040。PCI 数据结构前 4 个字节为标识符 PCIR, 偏移 04h~05h 即 0x0044h~0x0045h 为厂商的标识码, 即 Vendor ID 为 0x1969; 偏移 06h~07h 为设备标识码, 即 Device ID 为 0x1083。

通过编译, 将一个开源的引导程序按照该网卡的设备信息封装成 PCI Expansion ROM 的格式, 使用 CBROM 工具将新的 ROM 文件导入到固件的更新文件中, 替换原来的网卡 ROM 文件。然后, 通过正常的固件更新操作, 将修改后的更新文件注入到系统固件中, 成功地劫持了开机引导过程, 使计算机上电后在执行到 PCI 设备初始化的时候, 操作系统便不经固件, 而是由修改之后的 PCI Expansion ROM 进行引导, 如图 8 所示。

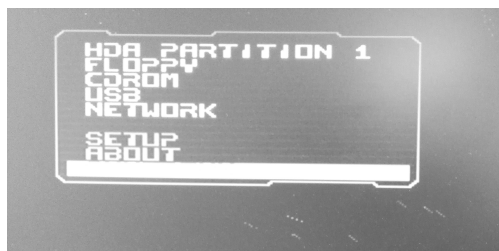


图 8 网卡 PCI Expansion ROM 引导进入操作系统

在实际的更新执行过程中, 固件系统本身并没有检测所有封装模块的安全性, 在跳转到 PCI Expansion ROM 文件执行设备初始化的时候, 也很难对代码的行为做出合法性检验, 从而给固件自身甚至是操作系统的运行环境都带来了安全威胁。本次实验通过修改合法的固件更新文件得到了一个符合攻击模型形式规范的实际攻击载体, 然后按照正常的更新流程替换了网卡的 ROM 文件, 这样系统固件在执行遍历 PCI 设备操作的时候, 发现网卡设备对应的 PCI Expansion ROM 文件, 于是就按照规范定义的操作将这部分代码复制到内存中执行。由于此时固件尚未将控制权交给操作系统, 因此, 这时候执行的代码都拥有绝对的高权限^[21]。

6 结束语

固件位于计算机整个体系架构的底层, 是软

硬件协同工作的保证, 固件文件中潜在的安全隐患会在计算机软件和硬件工作的过程中无限放大。在对固件安全的研究过程中, 本文提出了基于软硬件协同形式验证的漏洞分析方法, 建立了完整的软硬件交互形式模型, 使用 TLC 工具在固件更新过程中发现了漏洞, 并通过实验验证了该漏洞的存在, 证明了方法的可靠性。

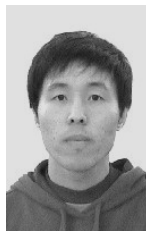
同传统的安全分析研究方法相比, 形式验证的方法效率更高, 能够很好地继承和发展之前的研究成果。这种特点不仅表现在系统模型的深度和广度上, 还体现在攻击模型的动态调整建立中。在今后的工作中, 可以逐步扩充形式模型库, 将计算机体系中重要的组件陆续加入到形式模型库中, 由浅入深, 建立更为完备的计算机系统形式模型, 在此基础上对计算机体系的安全性做出更为深入的分析。

参考文献:

- [1] ZIMMER V, ROTHMAN M, MARISSETTY S. Beyond BIOS: developing with the unified extensible firmware interface[M]. Intel Press, 2010.
- [2] BROSSARD J, DEMETRESCU F. Hardware backdooring is practical[J]. BlackHat, Las Vegas, USA, 2009, 58(1).
- [3] RONTTI T, JUUSO A M, TAKANEN A. Preventing DoS attacks in NGN networks with proactive specification-based fuzzing[J]. IEEE Communications Magazine, 2012, 50(9):164-170.
- [4] RUI MA, DAGUANG WANG, CHANGZHEN HU, et al. Test data generation for stateful network protocol fuzzing using a rule-based state machine[J]. Tsinghua Science and Technology, 2016, 21.
- [5] 周仕钧. 基于二进制补丁比对技术的漏洞特征分析与研究[D]. 昆明: 云南大学, 2013.
- [6] ZHOU S J. Holes characteristics analysis and research based on binary patch[D]. Kunming: Yunnan University, 2013.
- [7] KIM S, KIM R Y C, PARK Y B. Software vulnerability detection methodology combined with static and dynamic analysis[J]. Wireless Personal Communications, 2015:1-17.
- [8] 郑宇军, 张蓓, 薛锦云. 软件形式化开发关键部件选取的水波优化方法[J]. 软件学报, 2016(4): 933-942.
- [9] ZHENG Y J, ZHANG B, XUE J Y. Water ware optimization method of Software for multination development key components selection[J]. Journal of Software, 2016(4): 933-942.
- [10] CALINESCU R, GHEZZI C, JOHNSON K, et al. Formal verification with confidence intervals to establish quality of service properties of software systems[J]. IEEE Transactions on Reliability, 2016, 65(1):107-125.

- [9] KUMAR S, CHANDRA G, YADAV D. Formal verification of security protocol with B method[C]// IEEE International Conference on Computer and Communication Technology. c2014:161-167.
- [10] LACH J, BINGHAM S, ELKS C, et al. Accessible formal verification for safety-critical hardware design[C]//Reliability and Maintainability Symposium, IEEE Computer Society. c2006: 29-32.
- [11] CHOI H, YIM M K, LEE J Y, et al. Formal verification of an industrial system-on-a-chip[C]// IEEE International Conference on Computer Design. c2000:453-458.
- [12] LAMPORT L. Specifying systems, the TLA+ language and tools for hardware and software engineers[J]. Software Quality Professional, 2002(4):43.
- [13] CHAUDHURI K, DOLIGEZ D, LAMPORT L, et al. Verifying safety properties with the TLA+ proof system[M]//Automated Reasoning. Berlin Heidelberg: Springer, 2010:142-148.
- [14] LU T, MERZ S, WEIDENBACH C. Towards verification of the pastry protocol using TLA+ [C]//Formal Techniques for Distributed Systems Proceedings. c2011:244-258.
- [15] KALLENBERG C, BUTTERWORTH J, KOVAH X, et al. Defeating signed BIOS enforcement[J]. EkoParty, Buenos Aires, 2013.
- [16] ZHANG H, MERZ S, GU M. Specifying and verifying PLC systems with TLA+: a case study[J]. Computers & Mathematics with Applications, 2010, 60(3):695-705.
- [17] NARAYANA P, CHEN R, ZHAO Y, et al. Automatic vulnerability checking of IEEE 802.16 WiMAX Protocols through TLA+[C]// IEEE Workshop on Secure Network Protocols. c2006:44-49.
- [18] HEASMAN J. Implementing and detecting a PCI rootkit[J]. Retrieved February, 2006, 20: 3.
- [19] BUDRUK R, ANDERSON D, SOLARI E. PCI express system architecture[J]. Addison Wesley, 2003.
- [20] YIN Y. Implementation of PCI expansion ROM mechanism[J]. Computer Engineering & Applications, 2005.
- [21] WOJTCZUK R, RUTKOWSKA J. Attacking SMM memory via Intel CPU cache poisoning[J]. Invisible Things Lab, 2009.

作者简介:



张朋辉 (1989-), 男, 河南安阳人, 国防科学技术大学硕士生, 主要研究方向为专用集成电路可信性设计、硬件安全性分析。

田曦 (1971-), 男, 湖南长沙人, 博士, 国防科学技术大学副教授、硕士生导师, 主要研究方向为信息安全、可信性设计。

楼康威 (1992-), 男, 浙江金华人, 国防科学技术大学硕士生, 主要研究方向为专用集成电路可信性设计、电子信息系统信号安全分析。