Tenda AX12 路由器 0-Day 栈溢出漏洞挖掘方法

郑 炜[1], 许晴晴[1], 李 奇[1], 陈 翔[2], 孙家泽[3]

[1]西北工业大学 软件学院 西安 中国 710100
[2]南通大学 信息科学技术学院 南通 中国 226019
[3]西安邮电大学 计算机学院 西安 中国 710121

摘要 随着 5G 技术对物联网发展的加速, 预计到 2025 年将会有约 250 亿台物联网设备连接到人们的生活。其中承担物联网设备网络管理角色的路由器使用量非常大, 但是路由器存在众多安全问题, 通过路由器设备进行攻击, 可以非法获取用户信息。为了维护网络安全, 提前发现路由器的漏洞具有重要的研究意义。本文以 Tenda AX12 路由器为研究对象, 从固件入手对其进行 0-Day 栈溢出漏洞挖掘研究, 并提出了基于危险函数追踪的逆向分析漏洞挖掘方法。首先从危险函数中分析函数所在前端的对应位置, 将前后端对应; 然后对固件中的 Web 服务进行分析, 对其中可能发生栈溢出的 httpd 二进制代码进行危险函数分析, 该方法使用反汇编代码对危险函数的普通形式和展开形式进行定位, 并对危险函数进行参数分析和动态检测; 接着通过搭建仿真模拟机在模拟机上运行该服务的二进制文件, 并在 Web 前端页面对潜在漏洞位置进行数据包捕捉; 最后根据前期分析的危险函数参数情况对包进行改写并发送, 以此来触发漏洞, 验证漏洞的存在性, 同时验证该危险函数是否发生栈溢出。为了更真实地确定漏洞存在, 我们又在真实设备上验证漏洞的真实存在性和可利用性。实验结果表明了该漏洞的挖掘检测方法的有效性, 我们分别在不同型号的路由器上挖掘到 4 个 0-Day 漏洞, 并且经过与 SaTC 工具进行对比实验结果表明该检测方法能够更准确的定位到出现漏洞的函数位置。

关键词 物联网; 路由器; 危险函数 strcpy; 0-Day 栈溢出漏洞; SaTC
中图法分类号 TP309.1 **DOI 号** 10.19363/J.cnki.cn10-1380/tn.2024.05.11

Tenda AX12 Router 0-Day Stack Overflow Vulnerability Mining Method

ZHENG Wei[1], XU Qingqing[1], LI Qi[1], CHEN Xiang[2], SUN Jiaze[3]

[1]School of Software Engineering, Northwestern Polytechnical University, Xi'an 710100, China
[2]School of Information Science and Technology, Nantong University, Nantong 226019, China
[3]School of Computer Science, Xi'an University of Posts and Telecommunications, Xi'an 710121, China

Abstract As 5G technology accelerates the development of the Internet of Things, it is expected that there will be about 25 billion IoT devices connected to people's lives by 2025. Among them, the routers that play the role of network management of IoT devices are used a lot, but there are many security problems in the routers. Attacks through router devices can illegally obtain user information. In order to maintain network security, it is of great research significance to discover the vulnerabilities of routers in advance. This paper takes the Tenda AX12 router as the research object, we start from the firmware, the 0-Day stack overflow vulnerability mining research is carried out, and propose a reverse analysis vulnerability mining method based on dangerous function tracing. First, analyze the corresponding position of the front end of the function from the dangerous function, and correspond the front end and the back end; then analyze the Web service in the firmware, and analyze the dangerous function of the httpd binary code in which stack overflow may occur. The common form and expanded form of the dangerous function are located, and the parameter analysis and dynamic detection of the dangerous function are carried out; then the binary file of the service is run on the emulator by building a simulation machine, and the potential vulnerability location is packaged on the Web front-end page. Capture; finally, rewrite and send the packet according to the dangerous function parameters analyzed in the previous stage, so as to trigger the vulnerability, verify the existence of the vulnerability, and verify whether the dangerous function has a stack overflow. In order to more realistically determine the existence of the vulnerability, we verify the real existence and exploitability of the vulnerability on real devices. The experimental results show the effectiveness of the mining and detection method of this vulnerability. We have mined four 0-Day vulnerabilities on different types of routers, and compared with the SaTC tool, the experimen-

通讯作者: 郑炜, 博士, 博导, 副教授, Email: wzheng@nwpu.edu.cn。
本课题得到国家自然科学基金专项项目(国家自然重点基金)课题(No. 62141208)、国家重点研发计划课题 (No. 2020YFC0833105Z1)的资助、国家自然科学基金项目(No. 62272387)、陕西省科技厅重点研发计划项目(No. 2023-YBGY-030)、西安市重点产业链核心技术攻关项目(No. 23ZDCYJSGG0028-2022)。
收稿日期: 2022-09-07; 修改日期: 2022-11-30; 定稿日期: 2024-01-30

tal results show that the detection method can more accurately locate the location of the function where the vulnerability occurs.

Key words IOT; router; dangerous function strcpy; 0-Day stack overflow vulnerability; SaTC

1 引言

近年来, 随着 5G 技术的问世, 智能家居、智能路由器等物联网设备逐步进入人们的视野。在物联网设备中, 以路由器设备数量最多。作为网络中的核心基础设备, 路由器的安全问题也愈发引起安全人员的重视。一方面是作为连接用户与互联网的桥梁, 这些设备常常处于“永远在线”状态。另一方面随着智能路由器和智能家居的兴起, 路由器拥有更多功能的同时, 也带来更多的安全漏洞。

近些年来, 研究人员针对物联网设备做了大量的研究, 包括基于 Fuzzing 的网络协议漏洞检测、思科 IOS(Internetwork Operating System, IOS)安全分析、思科 IOS 漏洞利用等。然而, 仍有一些问题尚未得到解决。首先, 以往研究主要集中在利用漏洞或进行路由器调试上, 但对漏洞检测的研究仍然有限。其次, 虽然 Fuzzing 已被广泛用于检测网络协议的安全性, 但由于路由器的特殊性, 已有的 Fuzzing 框架不能直接用于测试路由器协议。

安全漏洞产生的原因之一在于用户对于路由器的安全意识薄弱, 只考虑产品的网速、外观等因素而忽略了路由器的安全问题, 例如对于 WiFi 密码设置和管理员密码的设置过于简单随意, 这种弱口令的账户密码给别有用心的人提供了可趁之机。一旦账户信息泄漏, 攻击者可以远程登录路由器, 修改路由器的配置, 进行流量拦截和篡改, 从而达到盗取用户重要信息的目的。路由器厂商没有做好安全防御也是原因之一, 路由器厂商对于漏洞的处理态度消极, 固件的版本更新缓慢, 造成漏洞被攻击者恶意使用从而危害社会。市面上有 90%的路由器存在后门, 后门即在生产过程中用来研发、测试产品时远程对产品进行调试的入口, 方便生产的同时也给黑客提供了利用的机会。路由器 WiFi 信息的自动保存和连接设置也为路由器的安全带来了隐患。将不同型号的设备短时间内交换并设置相同的账户密码, 手机等设备不需要重新登录就能自动连接 WiFi, 这使得攻击者可以在毫无察觉的情况下得到路由器的管理员权限, 这对于保密机构和公司的重要场合来说存在机密泄漏的危险。

路由器位于网络空间底层, 互联多种异构网络, 负责网络中数据传递路径的计算和查表转发工作, 是网络空间中最为重要的核心基础设备之一。因此, 路由器的安全对于网络本身的安全乃至整个国家的信息安全都具有极其重要的意义。面向路由器等物联网设备的二进制程序的漏洞检测技术有多种, 本节从 3 个方面总结研究现状分析: 路由器漏洞的挖掘技术、静态检测二进制程序漏洞和动态检测二进制程序漏洞。

对于路由器漏洞挖掘, Chen 等人[1]提出了一种新颖的静态污点检查解决方法 SaTC, 以有效地检测嵌入式设备提供的 Web 服务中的安全漏洞。Web 界面上的字符串文字通常在前端文件和后端二进制文件之间共享, 以对用户输入进行编码。因此, 安全分析师从前端提取这些常用关键字, 并使用它们来定位后端中指向输入条目的参考点; 接着基于有针对性的数据流分析来查明对不受信任的用户输入的危险用途, 通过运行程序检测出潜在漏洞的函数数据流。

基于静态分析的技术不需要实际运行程序, 利用模拟程序执行流程来遍历代码区域, 因此可以在程序执行前发现漏洞。该技术能够对程序路径进行全面分析, 路径覆盖率较高, 且漏报率较低[2-3]。Redini 等人[4]提出了一种静态分析方法 Karonte, Karonte 能够通过建模和跟踪多二进制交互来分析嵌入式设备固件, 通过在二进制文件之间传播污染信息以检测不安全的交互来识别漏洞。Gao 等人[5]针对二进制程序缺乏程序语义, 很难判断内存访问是否超出其边界的问题, 尝试通过还原其内存布局来定位漏洞。他们提出的方法对于同一个程序, 分别输入一组执行成功和执行异常的数据, 然后回收这两种执行的内存。如果将执行恢复成功的内存作为缓冲区边界, 则可以更容易判断异常执行中的缓冲区溢出。他们测试了 20 个存在缓冲区溢出漏洞的程序。结果表明该方法在时间成本和漏洞定位精度方面优于一些静态程序分析工具。

动态分析技术通过具体执行目标程序, 分析程序执行时表现出的状态特征和相关信息以发现程序中存在的缺陷[6-9]。该技术可以获取程序的运行时信息, 但无法覆盖程序的所有路径, 因此误报率较低, 漏报率较高, 并且程序运行也会产生较大的系统性能开销。一些研究人员[10-13]使用模糊技术来检测物联网设备中的漏洞。SRFuzzer[14]是用于测试物理 SOHO(小型办公室/家庭办公室)路由器的自动模糊

框架, 它需要首先从正在运行的设备捕获大量的 Web 请求, 然后可以对用户输入的语义进行建模以生成测试用例。Wang 等人[15]基于遇到更多和复杂的基本块的执行路径可能有更高的机会包含安全漏洞这一观察, 提出了一种辅助二进制程序静态分析的混合模糊测试方法。该方法的基本思想是根据相关执行路径的复杂性对种子输入进行优先级排序。为此, 他们利用静态分析来评估每个基本块的复杂性, 并采用硬件跟踪机制来动态提取执行路径以计算种子输入的权重。该方法的主要优点是可以通过使用硬件跟踪和混合模糊测试来有效地测试二进制程序。

Karonte 工具利用静态分析技术来执行多二进制污点分析。然而, 研究人员只关注后端二进制文件, 而忽略了存储在前端文件中的用户输入上下文, 这会导致大量的漏报。二进制文件中定位缓冲区溢出的方法能将成功回收的内存作为缓冲区边界, 并用于定位异常执行中的缓冲区溢出。但也存在一些缺点, 比如只检查代码行数较少的程序。如果行数过多, 单次访问分析指令和恢复内存布局需要很长时间, 并产生很多误报, 并且在测试用例方面, 它并不能涵盖所有用例情况。因此, 无论是静态分析技术还是动态分析, 即使扫描出大量可疑漏洞, 仍需要安全分析人员完成漏洞确认以及漏洞触发位置的“最后一公里”问题。

为了解决上述问题, 本文提出了使用危险函数逆向追踪的方法来发现路由器的固件系统中 Web 服务的多个二进制文件中存在的漏洞。本文以 Tenda AX12 路由器为研究对象, 对其进行 0-Day 漏洞栈溢出漏洞研究, 主要从研究对象的固件入手, 对固件进行文件系统提取, 分析文件系统的系统架构确定仿真模拟机的镜像, 为搭建仿真环境提供前提。本文重点挖掘的危险函数是 strcpy 函数, 通过研究发现该函数将参数复制到另一个参数中时容易发生栈溢出, 这种情况特别容易发生在前端与后端交互的时候, 通过将用户传入的数据复制到内存中, 在未经过滤的情况下就发生了缓冲区溢出。本方法对文件系统中的 Web 服务进行分析, 重点选取 httpd 服务进行分析, 针对以上问题我们提出了基于追踪危险函数的逆向分析方法, 对文件中可能发生栈溢出的 httpd 二进制代码进行危险函数分析, 该方法使用反汇编代码对危险函数的普通形式和展开形式进行定位, 然后对危险函数进行参数分析和动态检测, 验证该危险函数是否发生溢出。具体来说: 首先从危险函数中分析函数所在前端的对应位置, 将前后端对应, 然后通过搭建仿真模拟机在模拟机上运行该服务的二进制文件, 并在 Web 前端页面对潜在漏洞位置使用抓包工具进行网络包捕捉, 之后根据前期分析的危险函数参数情况对包进行改写并发送, 以此来触发漏洞, 验证漏洞的存在性, 为了更真实地确定漏洞存在, 本方法在真实设备上验证漏洞的真实存在性和可利用性。实验结果表明了论文所提方法的有效性, 此外, 通过与 SaTC 工具进行比对实验, 结果表明该方法更能准确地定位到漏洞的位置。

论文主要贡献总结如下:

- 提出了一种新的基于逆向分析危险函数的方法来挖掘路由器 Web 服务中二进制文件存在的漏洞。
- 基于提出的方法, 针对 Tenda AX12 路由器对固件系统中 Web 服务的漏洞进行了挖掘, 并将方法应用到不同型号的路由器上, 在四个固件样本上发现了 4 个 0-Day 栈溢出漏洞, 并且由于漏洞严重的安全影响被信息安全平台分配了 CNVD/CVE, 从而验证了我们提出的方法。

2 相关技术概述

目前广泛应用的漏洞挖掘技术主要分为三大类: 一是静态代码审计; 二是模糊测试[16-18]; 三是通过深度学习[19]的方法。

基于固件的漏洞挖掘多针对后门类漏洞和污点类漏洞。后门漏洞是固件中常见的漏洞, Costin[20]首次提出了大规模自动化地分析嵌入式设备的固件, 自动解压并处理固件, 使用模糊哈希的方式匹配固件中存在的弱密钥。通过关联分析的方式在不同维度查找不同固件镜像中间的相似性。Thomas[21]使用静态数据对比分析来检测设备固件中的后门漏洞。HumIDIFy[22]方法中使用了一种半监督学习的分类器通过半自动化的方式识别固件中的二进制以及二进制具有的功能, 通过与预期的 profile 相比较检测出固件中的后门。针对污点性漏洞, 近几年使用的方法有: 将二进制转换为中间描述, 生成函数的数据流图并以此来追踪路径上的污点数据[23]; 通过敏感源跟踪设备状态和用户信息的网络连接之间信息流动查找敏感数据流[24]; 结合协议解析器识别技术和基于二进制程序依赖图发可任意粗弱点推断技术来加速嵌入式固件中的污点类漏洞[25]; 利用仿真器以及静态污点追踪来定位并利用固件漏洞[26]; 利用动态模糊测试工具, 对被测设备进行污点分析, 识别物联网应用程序中常用 硬编码的字符串、用户输入和系统 API, 对记录的协议字段作为参数的函数进

行变异[27], 该工具是通过数据增广的方式即对测试用例进行变异生成更多的用例, 试图触发设备中的漏洞。以上方法中涉及到漏洞挖掘的基础方法静态分析以及深度学习相关的方法。本文通过静态代码审计和动态运行程序相结合的方法, 将重点放在研究路由器交互设备漏洞的挖掘上, 通过搭建真实的运行环境, 分析固件和协议漏洞、体系架构等, 实现对网络安全漏洞的扫描分析, 指出了漏洞挖掘从固件分析到漏洞利用的整个流程框架, 解决了漏洞挖掘过程中路径爆炸的问题, 同时也为漏洞挖掘整个流程的研究提供了一些参考。

静态代码审计是指在不运行程序的前提下, 对程序的源代码或者是二进制代码进行的分析。通过分析二进制代码可以对程序的内部结构有一个大体的了解, 方便找到可能存在的漏洞位置。本研究使用 IDA Pro 来实施静态分析, 它提供了丰富的静态分析手段, 能够识别各种不同体系架构的二进制文件格式。除此之外可以将其与动态执行程序相结合判断潜在漏洞的真实性, 搭建一个支持程序运行的环境是推动实验进一步动态执行程序的前提条件。以下我们针对搭建运行环境以及分析体系架构等方面的技术做了进一步介绍。

2.1 路由器固件及其文件系统

2.1.1 路由器固件

固件(Firmware)是运行在物联网设备上的核心软件之一[28]。因此路由器固件相当于固化在只读存储器中的一个软件, 不同厂家的路由器品牌其内的硬件、固件和固件中的操作系统都不尽相同, 路由器设备作为一种嵌入式设备, 包含文件系统在内的核心内容均以固件的形式存储在闪存(Flash)中[29]。固件中一般为二进制文件, 主要包括固件头部, 引导程序, Linux 内核, 文件系统等。其核心内容是文件系统, 包括了路由器的配置文件, 提供 Web 服务的二进制文件, 如 Tenda 系列路由器的 httpd 二进制文件。

2.1.2 路由器文件系统

路由器文件系统通常作为固件的一部分, 是操作系统的重要组成部分和操作运行的基础[30]。路由器文件系统属于一种嵌入式文件系统, 不同的路由器使用的文件系统格式不尽相同, 根据路由器所使用的文件系统格式, 文件系统被打包并组装到固件中。在众多的嵌入式文件中物联网设备更关注 Flash 文件系统, 其中 Linux 固件中 Cramfs 和 Squashfs 是常见的两种文件系统。

Squashfs 是一个只读格式的文件系统, 具有超高的压缩率, 其压缩率最高可达 34%。当系统启动后, 会将文件系统保存在一个压缩过的文件系统文件中, 这个文件可以使用换回的形式挂载并对其中的文件进行访问, 当进程需要某些文件时, 仅将对应部分的压缩文件解压缩。

2.2 路由器的指令架构

很多小型的 SOHO 路由器都是基于 Linux 系统开发的。和普通的 Linux 系统相比, 路由器的 Linux 有两个特点: 一是指令架构, 路由器是一种嵌入式系统, 多采用 MIPS 和 ARM 这两种指令架构; 二是路由器的 shell 是基于 BusyBox 的。

2.2.1 MIPS 指令架构

MIPS 指令架构由 MIPS 公司所创, 属于 RISC(Reduced Instruction Set Computer, RISC)体系, 是一种普遍应用于小型设备的处理器架构, 其应用领域覆盖游戏机、路由器、激光打印机、掌上电脑等。使用 MIPS 指令架构的 Linux 系统称为 MIPS Linux。MIPS32 架构是一种基于固定长度的定期编码指令集, 并采用导入/存储数据模型。经改进, 这种架构可支持高级语言的优化执行。在路由器中, 我们经常使用的一种 MIPS 架构就是 MIPS32。本次实验的研究对象使用的系统架构即为 MIPS 架构。

2.2.2 MIPS 大端格式和小端格式

系统架构的格式判断对于后续仿真实验环境的确定极其重要。只有了解系统架构的存储格式, 才能准确还原路由器的固件系统。数据在存储器中按照字节存放, 处理器也是按照字节访问存储器中的指令或数据。如果需要读出一个字, 即 4 字节, 如 *men*[*n*]、*men*[*n*+1]、*men*[*n*+2]、*men*[*n*+3]这 4 个字节, 那么最终交给处理器的有两种结果, 具体如下。

{*men*[*n*],*men*[*n*+1],*men*[*n*+2],*men*[*n*+3]}

{*men*[*n*+3],*men*[*n*+2],*men*[*n*+1],*men*[*n*]}

前者称为大端格式(Big-Endian), 也称 MSB (Most Significant Byte); 后者称为小端格式(Little-Endian), 也称 LSB(Least Significant Byte)。在大端格式下, 数据的高位保存在存储器的低地址中, 而数据的低位保存在存储器的高地址中[31]。0x12345678 在两种模式下的存储情况如图 1 所示。

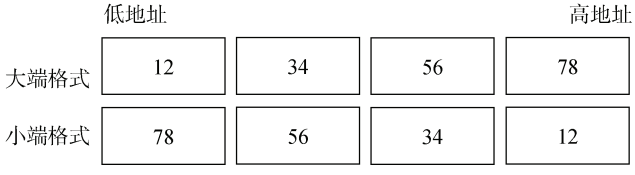

图 1 MIPS 大小端格式的数据存储

Figure 1 Data storage in MIPS big-endian format

2.3 仿真路由器 Web 服务的 QEMU

获取文件系统并了解指令架构的前提下, 本文采用 QEMU 仿真技术模拟路由器的 Web 服务。

2.3.1 QEMU 工作原理

QEMU 是纯软件实现的虚拟化模拟器, 它可以模拟 x86、ARM、ARM64、MIPS、PowerPC 等架构的硬件设备, 最常见的是模拟出一台能够独立运行操作系统的虚拟机, 虚拟机认为自己在和硬件交互, 但实际是和 QEMU 模拟出来的硬件交互, QEMU 将这些指令转译给真正的硬件。

由于 QEMU 是纯软件实现的, 所有的指令都要经过 QEMU, 性能非常低, 因此在生产环境中, 通过配合 KVM 来完成虚拟化工作, KVM 是硬件辅助的虚拟化技术, 主要负责比较繁琐的 CPU 和内存虚拟化, QEMU 负责 I/O 虚拟化, 两者合作各自发挥自身的优势。

从本质上看, 模拟出的每个虚拟机对应 host 上的一个 QEMU 进程, 而虚拟机的执行线程(如 CPU 线程、I/O 线程等)对应 QEMU 进程的一个线程。在第四节的实验中 QEMU 就是运行在 Ubuntu18 上的, 而 MIPS32 就相当于图 2 中的客户机系统。

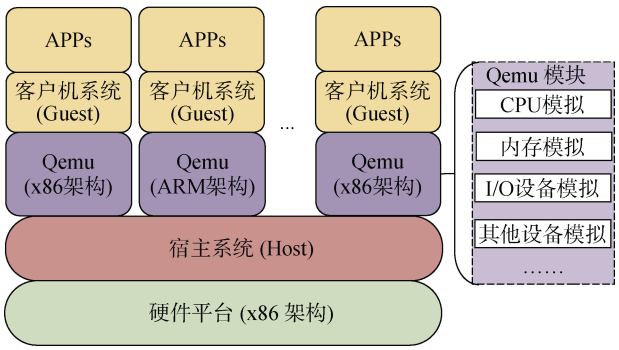

图 2 虚拟机的层次结构

Figure 2 Virtual machine hierarchy

QEMU 软件虚拟化实现的思路是采用二进制指令翻译技术, 主要是提取 guest 代码, 然后将其翻译成 TCG 中间代码, 最后再将中间代码翻译成 host 指定架构的代码, 如 x86 体系就翻译成其支持的代码形式, MIPS 架构同理。

2.3.2 QEMU 的仿真方式

QEMU 的仿真方式有 User mode 和 System mode 两种模拟模式, 为了模拟整个计算机系统, 我们选用 System mode 模式, 它可以模拟中央处理器及其他周边设备, 同时可以在一部主机上虚拟数部不同虚拟计算机, 我们可以为 QEMU 虚拟机指定运行的内核或者虚拟硬盘等文件, 利用其他 VMM(Xen, KVM)来使用硬件提供的虚拟化支持, 创建接近于主机性能的全功能虚拟机, 为路由器仿真提供更完善的环境。

2.3.3 QEMU 的网桥通讯

QEMU 提供 4 种网络通信方法, 它们分别是 User mode stack、socket、tap/bridge、VDE。本节重点介绍 QEMU 虚拟机的 tap/bridge 通讯模式。网桥可以看作将若干个网卡连接起来的网桥, 一个网络接口的数据可以通过网桥转发给另一个接口, 实现网络通信。在创建完 QEMU 虚拟后, 一般在 host 里会多一张 tap 网卡。

tun/tap 网卡工作模式。普通的物理网卡是通过物理链路来收发数据, 一端连接物理链路, 一端连接内核协议栈。数据通过物理链路进入到达内核协议栈做进一步的处理, 对于一些错误的数据包, 协议栈可以选择丢弃; 对于不属于本机的数据包, 协议栈可以选择转发; 对于属于本机的数据, 协议栈就会通过 Socket API 告知上层正在等待的应用程序。而 tun/tap 是通过/dev/net/tun 来收发数据, 一端连着/dev/net/tun, 一端连着协议栈。tun 网卡: 工作在三层网络层, 能处理 IP 数据包并支持路由功能。工作在二层链路层, 能处理 mac 层数据包, 支持 mac 层广播, 可以与物理网卡做桥接, 主要用在虚拟机通讯。tap 网卡主要是两部分组成, 字符设备驱动和虚拟网卡驱动。字符设备驱动负责和用户进程交互, 把数据写入/dev/net/tun, 然后通知用户进程从/dev/net/tun 里面拿数据实现数据交互。虚拟网卡驱动负责和网络协议栈的数据交互。tap 网卡工作模式如图 3 所示。

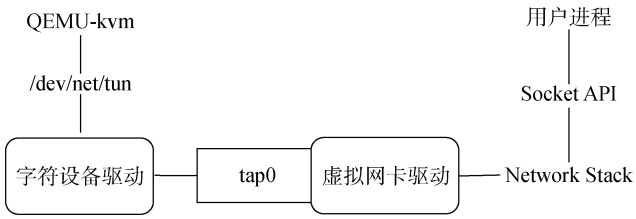

图 3 tap 网卡的工作模式

Figure 3 The working mode of tap network card

2.4 路由器的栈溢出漏洞

缓冲区(不做特殊说明时, 以下所指皆为栈空间缓冲区)是用于在内存中存储数据的内存区域。如 C 语言代码“char *dst*[20];”就是在内存中申请 20 字节用于存放字符型数据的缓冲区。

简单地说, 缓冲区溢出就是在大缓冲区里的数据向小缓冲区复制的过程中, 由于没有检查小缓冲区的边界或者检查不合格, 导致小缓冲区明显不足

以接收整个大缓冲区的数据, 超出的部分覆盖了与小缓冲区相邻的内存区中的其他数据而引发的内存问题。缓冲区溢出是一种非常普遍且危险的漏洞, 利用缓冲区溢出攻击, 可以造成程序运行失败、系统宕机、重新启动等后果。更为严重的是, 利用缓冲区溢出攻击执行非授权指令, 可以取得系统特权, 进而进行各种非法操作。

在 MIPS 体系结构中, 函数分为叶子函数和非叶子函数两种。叶子函数是指函数内部不分配栈空间, 该函数 A 也不再调用其他函数, 非叶子函数表示此函数 A 调用了另外的函数 B, 两者的区别在于此函数是否调用其他函数。

MIPS 函数的调用过程与 x86 不同。在 x86 体系结构下, 函数 A 调用函数 B 时, 总是先将函数 A 的地址压入堆栈, 在函数 B 执行完毕后返回 A 函数时, 再从堆栈中弹出返回函数 A 的地址, 然后返回函数继续执行。MIPS32 架构下的函数调用指令不会把返回地址存入堆栈, 而是直接存入寄存器$RA 中, 此时需要对在 MIPS32 架构下缓冲区溢出能否被利用进行分析。

当非叶子函数调用危险函数时, 非叶子函数的地址首先会保存在寄存器中, 进入非叶子函数后将 main 函数的返回地址保存在该非叶子函数的堆栈中, 继续执行非叶子函数并将保存在堆栈中的 main 函数的返回地址写入到寄存器中。如果非叶子函数调用了输出函数, 函数返回时会先从堆栈中取出返回地址并将其放入寄存器中, 此时若非叶子函数的局部变量中存在缓冲区溢出漏洞, 就可能导致堆栈上返回的 main 函数的返回地址被覆盖, 导致返回地址被修改, 该非叶子函数取出的函数返回地址可以是由攻击者精心构造的数据, 进而缓冲区溢出被利用。

只要函数中存在非叶子函数, 并且有缓冲区溢出漏洞, 就可以覆盖某一个函数的返回地址, 从而劫持程序执行流程; 而在叶子函数中, 如果存在可以溢出大量数据的情况, 就存在通过覆盖父函数中的返回地址利用缓冲区溢出漏洞的可能性。

2.5 SaTC 工作原理分析

SaTC 是 2021 年 Chen 等人[1]提出的一种新的静态污点检测解决方案, 用来有效的检测由嵌入式设备提供的 Web 服务中的安全漏洞。SaTC 在固件的 Web 前端界面提取常用的关键字并在后端找到它们的引用, 并使用他们来定位后端中指向输入条目的参考点。SaTC 从参考点开始, 使用输入敏感的污点分析来发现后端的漏洞。如图 4 所示。

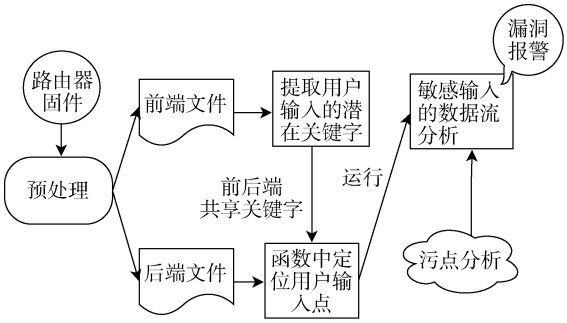

图 4 SaTC 工作原理

Figure 4 SaTC working principle

SaTC 在预处理阶段对固件解包, 将文件类型识别为前端文件和后端程序; 分析前端文件并利用典型模式提取用户输入的潜在关键字; 识别后端的边界二进制文件, 根据用户输入的关键字调用不同的处理函数, 从这些函数中找到定位检索用户输入的点; 通过用户输入所依赖的共享关键字, 从一个程序传递到另一个程序尝试寻找用户输入相关的隐式入口, 这有助于追踪二进制文件的隐式数据依赖关系; 使用输入敏感的污染分析来跟踪不可信数据的使用情况。使用粗粒度的污点传播、输入引导的路径选择和跟踪合并技术优化来使传统的污点分析在嵌入式系统上更高效; 当 SaTC 发现用户输入在任何预定义的接收器中使用时, 例如作为系统调用的参数, 它会收集路径约束并判断可达性。如果在输入约束较弱的情况下可以到达接收器, 则 SaTC 会发出潜在漏洞警报。

3 整体框架设计

本文通过逆向分析固件的源代码获取漏洞。本节从预处理分析、逆向分析危险函数、仿真环境搭建、触发潜在漏洞四个阶段介绍本文方法的整体框架, 本文最为关键的部分是使用逆向分析的方法发现危险函数, 进而分析潜在的漏洞。方法框架图如图 5。

3.1 预处理分析

本文采用静态分析的方法对嵌入式系统进行分析, 即在嵌入式系统不实际运行的情况下, 通过对固件文件反向解析, 分析内存和代码调用之间的关系, 从而挖掘嵌入式系统可能存在的漏洞和后门, 对已识别的漏洞进行定位。

3.1.1 固件获取

路由器固件包含了该路由器中所有的可执行程序及配置文件信息, 这些信息对进行路由器漏洞的分析和挖掘都至关重要。路由器固件的获取可以从

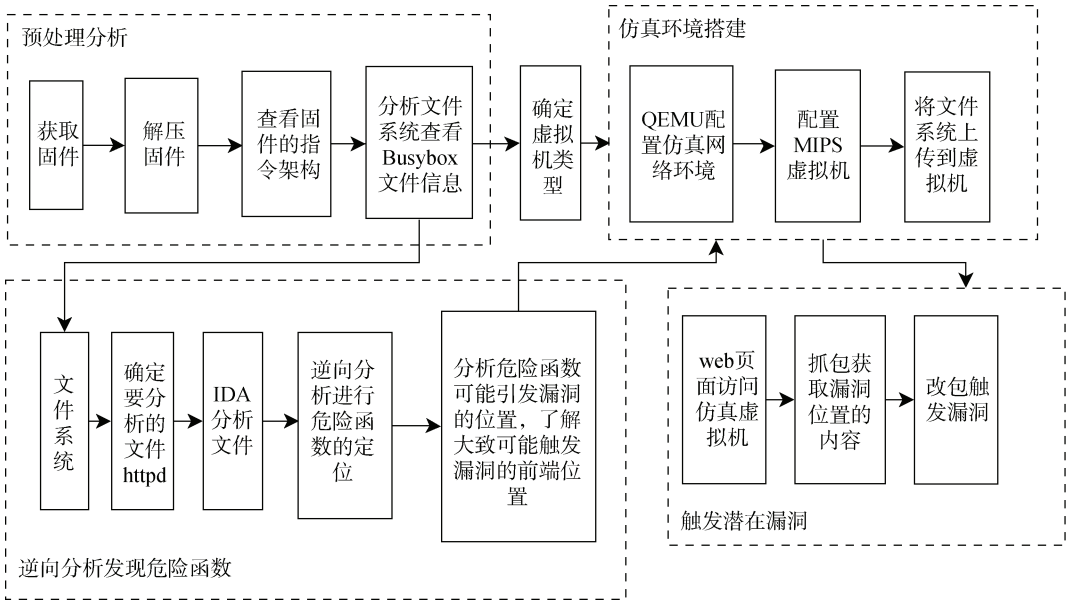

图 5　方法框架图

Figure 5　Method framework

官网下载，这个方法最为简便，但有些厂商现今已经不提供官网路由器固件版本下载，针对这种情况可以根据需要，通过硬件接入进行手工提取或在固件升级界面通过 shell 命令传输的方式获得固件。

本文的方法可以用于不同设备的漏洞挖掘研究，这里以 Tenda AX12 路由器 22.03.01.21_cn 版本的固件为例。使用 Binwalk 进行固件分析，该工具脚本自动化的特点可以帮助我们提取文件系统，协助我们进行固件分析。我们获取到的固件是一个以.bin 为后缀的文件，使用系统自带的 file 命令或者 binwalk 命令对该固件进行系统信息查看，使用 binwalk 命令可以得到固件的版本信息、大小以及偏移项等信息。如图 6 所示，从图中可以得出文件系统是 Squashfs filesystem，使用的 MIPS 架构是 little-endian 小端。

```
root@xqq-virtual-machine:/home/xqq/Desktop/AX12# file AX12.bin
AX12.bin: u-boot legacy uImage, OpenWrt fullimage, Linux/MIPS, Multi-File Image
(Not compressed), 11285390 bytes, Mon Aug 23 09:12:00 2021, Load Address: 0x0000
0000, Entry Point: 0x00000000, Header CRC: 0xEA84EB74, Data CRC: 0xFD32E297
root@xqq-virtual-machine:/home/xqq/Desktop/AX12# binwalk AX12.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             uImage header, header size: 64 bytes, header CRC:
0xEA84EB74, created: 2021-08-23 09:12:00, image size: 11285390 bytes, Data Addre
ss: 0x0, Entry Point: 0x0, data CRC: 0xFD32E297, OS: Linux, CPU: MIPS, image typ
e: Multi-File Image, compression type: none, image name: "OpenWrt fullimage"
72            0x48            uImage header, header size: 64 bytes, header CRC:
0x4730172C, created: 2021-08-23 09:11:35, image size: 2080384 bytes, Data Addres
s: 0xA0020000, Entry Point: 0xA0020000, data CRC: 0xFBB469AB, OS: Linux, CPU: MI
PS, image type: OS Kernel Image, compression type: lzma, image name: "MIPS UGW L
inux-4.9.206"
136           0x88            LZMA compressed data, properties: 0x6D, dictionary
 size: 8388608 bytes, uncompressed size: 7074804 bytes
2097224       0x200048        Squashfs filesystem, little endian, version 4.0, c
ompression:xz, size: 9188230 bytes, 2663 inodes, blocksize: 262144 bytes, create
d: 2021-08-23 09:11:57
```

图 6　Binwalk 固件信息输出

Figure 6　Binwalk firmware output

3.1.2　提取固件文件系统及信息分析

在 AX12.bin 所在目录下按照预定义配置文件中的提取方法从固件中提取探测到的文件及系统，根据 magic 签名的扫描结果进行递归提取。从解压的包中找到 Tenda AX12 路由器文件系统所在目录 squashfs-root。

为了保证未来仿真环境的准确性，我们进入固件的文件系统分析 BusyBox 的 ELF 文件头信息，再

次确定文件系统 MIPS 的指令架构, 头文件的分析为后续仿真虚拟机确定虚拟机内核和镜像提供了参考。从 BusyBox 读取到的信息显示该系统使用的 MIPS 大端, 如图 7 所示。观察两次得到的结果并不相同, 这导致无法确定 MIPS 架构真正的使用情况, 通过测试确定该固件的文件系统使用的是 MIPS 大端, 此时若直接使用 Binwalk 解析出来的信息确定仿真环境的架构, 会导致仿真失败, 因此需要多次综合分析信息, 确保为仿真提供一个正确的系统架构信息。

```
root@xqq-virtual-machine:/home/xqq/Desktop/AX12/_AX12.bin.extracted/squashfs-roo
t/bin# readelf -h busybox
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 01 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, big endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       1
  Type:                              EXEC (Executable file)
  Machine:                           MIPS R3000
  Version:                           0x1
  Entry point address:               0x403ab0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          0 (bytes into file)
  Flags:                             0x70001005, noreorder, cpic, o32, mips32r2
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           0 (bytes)
  Number of section headers:         0
  Section header string table index: 0
readelf: Error: Reading 466997 bytes extends past end of file for dynamic string
 table
```

图 7 BusyBox 输出信息

Figure 7 BusyBox output

3.2 栈溢出漏洞挖掘

3.2.1 Tenda AX12 仿真 Web 服务路由寻找方法

路由是指路由器从一个接口上收到数据包, 根据数据包的目的地址进行定向并转发到另一个接口的过程。在 Web 开发中, 路由是指根据 url 分配到对应的处理程序和模块, 找到路由函数就可以分析路由器所定义的 Web 服务。常见的漏洞应用有 http 服务、upnp 服务、MQTT 服务等其他服务和协议, 其中 http 服务是路由器和 Web 服务产生漏洞较多的, 这里以查找 Web 服务的 httpd 文件为例展开分析。

本文重点分析 Web 服务中的 httpd 二进制文件, httpd 文件是 Apache 超文本传输协议(HTTP)服务器的主程序, 是一个可以独立运行的程序。启动 Tenda AX12 仿真机的 Web 服务, 在浏览器中访问路由器的 Web 页面, 采用 Burpsuite 工具抓包的方式来寻找路由器的路由函数是逆向分析的第一步。熟悉路由器的 Web 页面有助于分析可能出现潜在漏洞的功能点, 使用抓包工具对该点进行抓包分析。根据抓包得到的 get 和 post 请求分析关键参数信息, 找到参数对应后端路由表中的函数, 通过关键参数信息将前后端对应。

根据得到的参数信息, 使用 IDA 工具进行动态调试, 对参数信息进行全局字符串搜索。IDA 具有追踪程序处理过程中的输入数据的功能, 通过追踪指令并执行流程获得程序的全局观, 呈现出逻辑完整的代码结构, 同时它也具有高质量的支持 MIPS 指令的反汇编代码能力, 提供 C 的伪代码形式, 和代码审计形成良好的配合。使用 IDA 搜索对应的参数信息, 定位到函数调用的位置, 并以此为基础向上逆向分析函数的交叉引用, 直到找到可能存在潜在漏洞的位置。

3.2.2 基于追踪危险函数 strcpy 的逆向分析

本文选取了 httpd 文件作为研究对象, httpd 文件是固件中的一个二进制文件, 文件经过解析将汇编语言以 C 语言的形式展示出来, 便于分析文件中的函数。C/C++ 语言中自带很多库函数, 这些库函数中在实现时出于效率的考虑, 没有加入边界检查代码, 在进行程序设计时, 这类函数使用频率很高, 很容易出现缓冲区溢出的情况, 这类库函数被称为危险函数[32]。这些危险函数通常指的是比较容易调用的内存拷贝、字符串拷贝、格式化输入、输出流以及内存初始化类型的函数[33]。软件开发工程师在代码编写过程中如果使用不当, 就很容易造成缓冲区溢出。通过分析, 可以把所有可能引起缓冲区溢出的危险函数分为三类, 分别是输入类函数、输出类函数和字符串处理类函数, 如表 1 展示了部分常用的危险函数, 这些函数都是可能存在问题的字符串处理操作, 由于函数未进行自变量检查, 从而引起缓冲

区溢出问题。并不是所有程序调用这些函数都会出现缓冲区溢出，只有不恰当的使用这些函数才会导致缓冲区溢出。例如 strcpy(*a*,*b*)，该函数是将变量 *b* 复制到变量 *a* 中，若 *b* 是从外界载入的，即由用户输入的变量，或者由前端向后端传入的变量，这种类型的变量如果在前后端都做了过滤，出现漏洞的可能性很小，但如果在前后端未经过滤处理，用户在不知道代码内部逻辑的情况下输入非法数据，极有可能造成潜在漏洞。该部分漏洞的产生有两种情况，一是程序员在前后端都未做过滤处理，即没有判断是否越界，此时用户输入过量的数据必将导致数据越界、缓冲区溢出；二是程序前后端是分离的，前端程序员做了过滤处理，但是后端程序员忽略了该点，此时用户在前端不能输入过量的数据，但是对于程序工作者和网络安全工作者来说可以通过后台输入过量的数据导致系统崩溃，从网络安全的角度来说，这无疑为黑客提供了机会。

表 1　常见的危险函数分类

Table 1　Common hazard function classification

类别	输入类函数	输出类函数	字符串处理操作
函数名称	fget()	printf()	strcpy()
	get()	sprintf()	strcat()
	scanf()	vsprintf()	strncpy()
	sscanf()		strecpy()
	vscanf()		strccpy()
	getc()		strncat()
	fscanf()		

本文重点关注 strcpy 危险函数，研究发现 strcpy 函数为非叶子函数，strcpy 函数是将源字符串复制到缓冲区中，没有指定要复制字符的具体数目，因此复制字符的数目直接取决于源字符串中的数目，如果源字符串恰好来自用户输入，且没有专门限制其大小的时候就有可能引起缓冲区溢出。利用 strcpy 的函数性质，从后端逆向查找定位到 Web 界面，搜集前端使用该函数对用户输入进行复制的函数，从而进一步筛选出引起缓冲区溢出的函数。

正向分析需要从主函数逐步分析，由于数据量较大会造成路径爆炸，需要逐步分析到内层函数确定是否存在潜在漏洞。该方法将危险函数作为分析文件的输入条目，多个危险函数构建输入条目候选列表，通过输入条目搜索的方式从程序中分析函数粒度的控制流程图。通过选取候选输入列表中的输入项，对待挖掘文件中的函数进行逐个扫描，根据文件中程序指令向上和向下的跳转指令指示，记录调用危险函数的函数控制流图。同一个危险函数存在多条控制流程路径，分别记录多条控制流程中传入危险函数的参数数据，形成每条有效路径的数据流图。有效路径的选择根据函数控制流图的参数传入情况与危险函数传入参数是否有相关的数据流向来确定。例如，函数 A、B、C 的函数调用图如图 8，函数 C 为危险函数，函数 A 调用了函数 B，函数 B 调用了函数 C，有效路径中函数 C 的实参与函数 B 的形参数据相关，形参通过参数运算或者赋值运算后的新的变量或参数传递给函数C，此时函数B到函数 C 的函数控制流图是一条有效路径，如此循环直到找到函数入口，输出一条有效路径。无效路径可能出现的情况为函数 C 传入的参数与函数 B 传入的参数是不相关数据，即没有相关数据流向，此时我们认为该路径不是一条有效路径，通过舍弃该条路径来降低无效路径的分析时间，提高分析的效率。

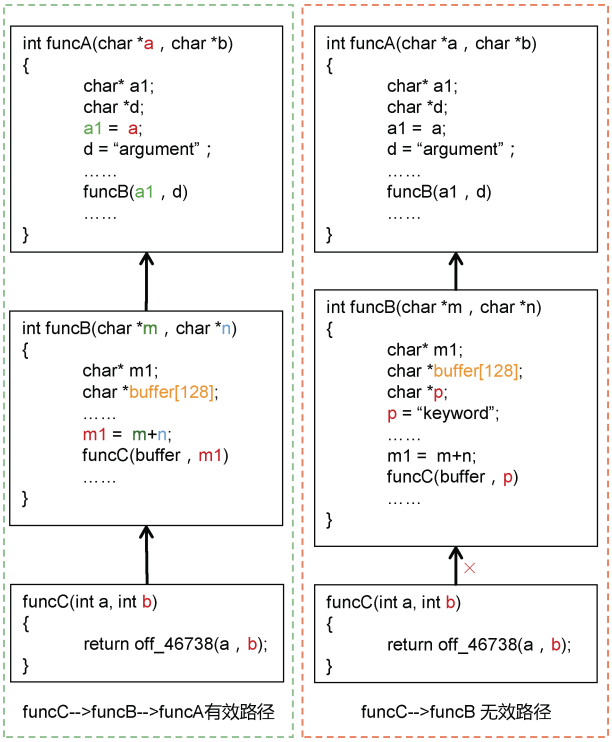

图 8　有效路径和无效路径

Figure 8　Valid and invalid paths

以本文研究的危险函数 strcpy 为例，图 9 展示了 strcpy 函数在 httpd 文件中的调用情况局部图(由于图片文件较大，附图仅展示 strcpy 的函数调用框架图，见附录)，程序中包含 200 个函数直接或间接的调用了该函数，其中有 75 个函数直接调用了危险函数 strcpy，仅调用 strcpy 已经有上百条路径。使用正向分析从程序的入口查找危险函数的控制流图体系庞大，会造成路径爆炸且效率很低。相比于正向分析，

本论文使用逆向分析可以只针对特定的危险函数 strcpy, 从最内层向外层进行深度搜索, 可以减少函数的分析数量, 同时缩短时间提高搜索效率, 便于更快的定位到潜在漏洞。

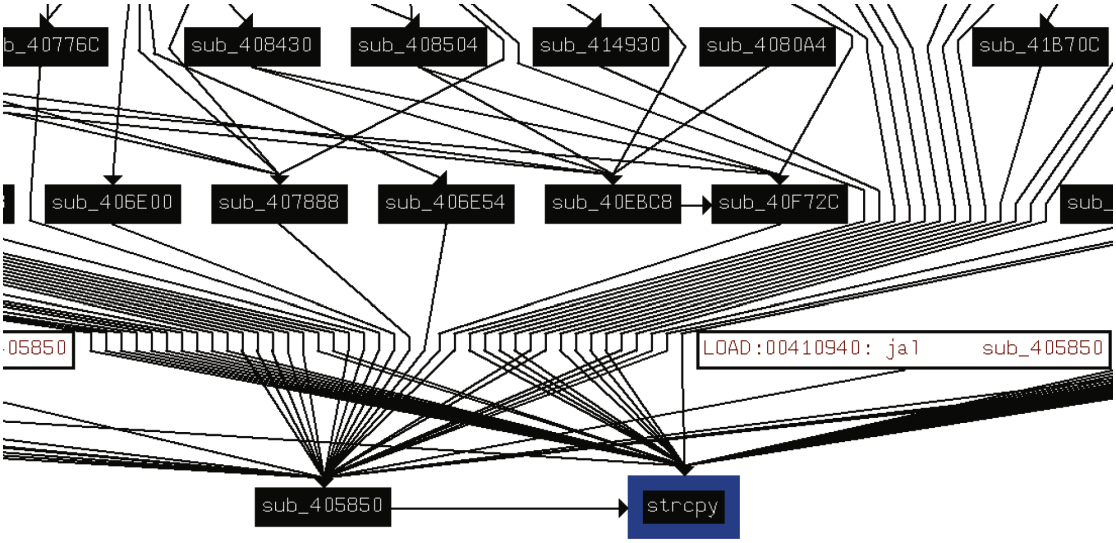

图 9 strcpy 在 httpd 文件中的调用关系局部图

Figure 9 Partial diagram of the calling relationship of strcpy in the httpd file

经过路径选择得到有效路径及有效路径的数据流向图, 根据函数的控制流图、数据流图分析从函数入口到危险函数是否存在可控数据, 即用户与平台交互的数据, 若存在, 对不同类型的危险函数设置不同的分析方法, 分析该数据经过数据处理最后传入到危险函数时, 是否合理使用了危险函数。若不存在则将该函数控制流的路径设置为无效路径, 不再对其进行分析。对于字符串复制以及内存拷贝类函数, 我们分析危险函数调用前是否对可控数据进行缓冲区长度校验, 缺失对参数长度的校验将会被认为是可疑漏洞。对于命令执行类函数, 我们认为只要该函数存在可控数据输入就认为它是可疑漏洞。本文使用以上方法针对捕捉到的关键参数 SetMacFilterCfg 字符串, 在 IDA 工具中进行危险函数 strcpy 的函数搜索, 对调用 strcpy 函数的非叶子函数进行逆向分析, 判断函数调用是否具有栈溢出的可能性。

3.3 QEMU 仿真 Tenda AX12 路由器

确保能够识别、提取、分析和运行设备中的代码是漏洞挖掘重要的前提之一, 路由器的模拟环境和在线真实环境为漏洞动态挖掘提供了可信环境, 为触发漏洞提供了更直观的测试环境。

3.3.1 QEMU 网络环境

为保证虚拟机能够与外界通信, Qemu 为虚拟机提供网络设备, 关闭宿主机网卡接口, 为其添加一个网桥并在网桥上添加接口, 分别启动网桥接口和网卡接口, 从动态主机配置协议(DHCP)服务器获取网桥的 IP 地址, 通过查看网络配置确保网桥创建成功。

本文采用 tap 通信方式与外界联网, 在宿主机中创建并配置一个 tap 设备, 并将该设备作为新建网桥的另一个接口与虚拟机连接, 即可实现网络连通。

3.3.2 MIPS 虚拟机环境

由固件提取的文件系统的信息分析得出的结论确定 MIPS 指令架构, 由此确定 Tenda AX12 路由器的 MIPS 架构是 MIPS 大端机, 安装对应的架构内核文件和磁盘镜像。启动 MIPS 虚拟机并指定网卡接口, 将 QEMU 虚拟机接入网络。

多终端窗口下操作将固件解析的文件系统 squashfs-root 上传到 MIPS 虚拟机, 确保成功上传后在 MIPS 虚拟机启动路由器的 Web 服务, proc 目录下存储进程所需的文件, dev 目录存储相关设备, 因此将这两个目录挂载到 chroot 环境下并启动 shell。启动路由器的 Web 服务, 并进入 httpd 目录下, 成功进入 MIPS32 Linux 的 shell 即 BusyBox, 表示仿真路由器模拟结束, 在浏览器中访问 IP, 端口默认为 80 的 Tenda AX12 Web 服务, 检测 Tenda AX12 路由器仿真结果。

3.4 漏洞攻击

本小节主要概述如何对潜在漏洞进行攻击。本研究经过对文件系统的 httpd 二进制文件分析得到后端文件中关联的前端页面中的潜在漏洞位置, 为实验搭建必要的测试环境, 方便漏洞的检测攻击。通过

在Firefox浏览器输入仿真路由器的IP地址进入前端界面, 采用 Burpsuite 的代理工具对界面中已确定关键字处进行操作同时执行抓包操作, 根据前期基于危险函数的逆向分析得出的结果, 对捕捉到的数据包进行代码分析, 确定逆向分析阶段得到的关键字与捕捉到的数据包中的关键字相同, 并对数据包中的关键字内容进行改包操作。对于缓冲区溢出类型的漏洞, 多数情况下没有对输入的字符串长度进行过滤, 有些虽然在前端做了过滤, 但是大部分程序使用抓包工具可以绕过此类型的过滤, 进入后端对代码进行破坏, 因此使用 Burpsuite 工具对捕捉到的包中相应的关键字内容进行修改, 破坏危险函数的正确使用规则就可以实现漏洞利用, 从而造成拒绝服务。

4 实验及结果分析

4.1 实验环境

实验环境的搭建是实验得以进行的基础。本节介绍了实验所需的仿真环境和漏洞挖掘实验环境工具。

4.1.1 仿真环境

实验在 Linux 系统的 Ubuntu18 上进行, 在该环境上对路由器模拟仿真, Tenda AX12 路由器仿真实验所需的工具如表 2 所示。

表 2 Tenda AX12 路由器仿真实验工具

Table 2 Tanda AX12 route simulation experiment tool

	仿真环境平台/版本	简称
路由器型号及固件版本	Tenda AX12 v22.03.01.21_cn	Tenda AX12 路由器
实验平台	Ubuntu 18.04 LTS	Ubuntu18
文件系统提取工具	Binwalk v2.1.1	Binwalk
虚拟机仿真路由器	Qemu v2.11.1	Qemu
MIPS 虚拟机	Big endian	MIPS 大端格式机
Web 浏览器	Firefox v91.7.1.esr(64 位)	火狐浏览器

安装 Binwalk 工具并安装 sasquashfs、jefferson、ubi_reader、yaffshiv、unstuff 等镜像以提取 Squashfs、JFFS2、UBIFS、YAFFS、StuffIt 文件系统。Binwalk 本身只能对设备的固件进行组成格式的分析和设备文件系统的提取, 因此需要安装压缩算法模块, 解密已知加密固件映像的库, 图像模块依赖包, 反汇编程序以及固件提取组。

4.1.2 漏洞挖掘实验环境

经过文件系统分析处理后对 Tenda AX12 路由器进行漏洞挖掘, 实验环境如表 3。

表 3 Tenda AX12 路由器栈溢出漏洞挖掘工具

Table 3 Tenda AX12 route stack overflow vulnerability mining tool

	工具及版本	简称
Tenda AX12 仿真机	Tenda AX12 v22.03.01.21_cn	Tenda AX12 仿真机
Tenda AX12 httpd 文件	httpd	httpd
逆向分析工具	IDA_Pro v7.5	IDA
抓包工具	Burpsuite_pro v2021.6	Burpsuite
SaTC 运行环境	Ubuntu18.04 LTS	Ubuntu18
	Docker v20.10.14	Docker
Web 浏览器	Firefox v91.7.1.esr(64 位)	火狐浏览器
Web 浏览器代理插件	FoxyProxy 标准版	FP 代理插件

4.1.3 漏洞利用实验环境

找到潜在漏洞以后对漏洞进行利用, 检测漏洞是否可以被利用, 进一步触发漏洞。实验环境如表 4 所示。

表 4 Tenda AX12 栈溢出漏洞利用所需工具

Table 4 Tenda AX12 stack overflow exploit tool

	工具及版本	简称
Tenda AX12 模拟机/真机	Tenda AX12 v22.03.01.21_cn	Tenda AX12 模拟机/真机
Windows10 下的火狐浏览器	Firefox v91.7.1(64 位)	火狐浏览器
火狐浏览器代理插件	FoxyProxy 标准版	FP 代理插件
抓包分析工具	Burpsuite_pro v2021.6	Burpsuite
POC 编写工具	python v3.7.8	python3

4.2 实验与结果分析

本文使用逆向分析的方法成功挖掘到 strcpy 危险函数引起的栈溢出漏洞, 并利用该漏洞对仿真机进行了攻击, 造成了仿真路由器的拒绝访问, 证实了漏洞的存在性和可利用性, 同时在真机上进行了漏洞利用的实验, 得出了同样的结论。该漏洞已获得 CNVD 和 CVE 平台的认证。

4.2.1 漏洞挖掘实现

我们首先对需要路由器进行预处理分析, 从固件中解析出文件信息, 为仿真环境的搭建做好充分的准备, 同时从文件系统中使用逆向分析的方法对危险函数进行逆向追踪, 从二进制文件中分析危险函数存在前端页面的位置, 得到漏洞所在前端页面的位置, 通过仿真机运行 Web 服务, 对漏洞进行触发利用。

在逆向分析危险函数时, 使用 IDA 分析 httpd 二进制文件, 以本文研究的 strcpy 危险函数为例, 程序中包含 200 个函数直接或间接的调用了该函数, 其中有 75 个函数直接调用了危险函数 strcpy, 本文根据前期分析捕捉到的关键字 setMacFilterCfg 进入其所在的函数中, 在该函数中搜索 strcpy 危险函数的交叉引用函数, 可以得到 75 个函数调用了此危险函数。经过解析 75 个函数中 strcpy 函数的调用情况可以发现, 部分 strcpy 函数是将固定的字符串或变量复制到一个变量中, 该部分函数很少有机会造成栈溢出漏洞, 但仍有一部分函数将用户输入的变量复制到一个固定内存大小的变量中。

本实验使用 IDA 工具通过搜索 strcpy 函数的交叉引用找到函数 sub_423160, 继续分析交叉引用找到函数 sub_424334 调用了函数 sub_423160, sub_423160 函数的第二个参数 *a*2 对应于 sub_424334 函数调用时的第二个参数 *v*3,如下图 10 所示。

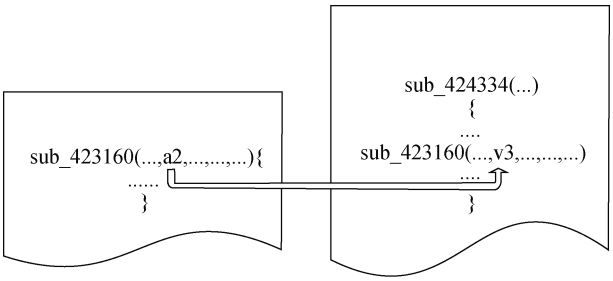

图 10　*a*2 和 *v*3 的对应关系

Figure 10　The correspondence between *a*2 and *v*3

分析 sub_424334 函数溯源 *v*3 的变量调用情况可知, *v*3 用于存储用户输入的 devicelist, 即黑名单的设备名称和 MAC 地址。用户将输入后的 devicelist 拼接字符串传递给 *v*3, *v*3 对应 *a*2, *a*2 通过危险函数 strcpy 复制到 sub_423160 函数定义的 *v*15 上, 如图 11 所示, 然而 *v*15 只申请了 160 个字节, 假设用户输入很长的字符串就有会导致栈溢出漏洞。用户输入

```
1  int __fastcall sub_423160(int a1, const char *a2, _DWORD *a3, int a4, int a5)
2  {
3    unsigned __int8 v9; // $s3
4    int v11; // $v0
5    int v12; // $s4
6    int v13; // $s2
7    int v14; // $a2
8    char v15[160]; // [sp+2Ch] [-A8h] BYREF
9
10   if ( strcmp(a1, "black") )
11   {
12     v9 = 1;
13     if ( strcmp(a1, "white") )
14     {
15       puts("filter_mode Error!");
16       return -1;
17     }
18   }
19   else
20   {
21     v9 = 0;
22   }
23   memset(v15, 0, sizeof(v15));
24   v11 = strchr(a2, 13);
25   if ( v11 )
26   {
27     *(_BYTE *)v11 = 0;
28     v12 = v11 + 1;
29     printf(
30       "%s[%s:%s:%d] %sparase rule: name == %s, mac == %s\n\x1B[0m",
31       "\x1B[0;33m",
32       (const char *)&dword_449F94,
33       "parse_macfilter_rule",
34       284,
35       "\x1B[0;32m",
36       a2,
37       (const char *)(v11 + 1));
38     to_lower_str(v12);
39     strcpy(&v15[32], a2);
40     strcpy(v15, v12);
41   }
```

图 11　*a*2 为用户输入的数据, 第 8 行定义 *v*15 变量的长度为 160 个字节, 第 39 行使用 strcpy 函数将 *a*2 变量复制到变量 *v*15,并且没有对 *a*2 的长度进行过滤

Figure 11　*a*2 is the data input by the user. Line 8 defines the length of the *v*15 variable as 160 bytes. Line 39 uses the strcpy function to copy the *a*2 variable to the variable *v*15, and the length of *a*2 is not filtered.

的参数流向如图 12 所示。同时我们可以知道 sub_423160 函数是非叶子函数, 可以确定这样的栈溢出漏洞可以达到劫持程序的执行流程的目的。

4.2.2 漏洞攻击利用

后端函数存在的潜在漏洞确定后, 需要熟悉它所对应的 Web 前端的业务逻辑。在 Web 前端找到触发漏洞的位置进行抓包, 本文在设置黑白名单功能处进行抓包, 接下来对捕获的请求包进行恶意构建, 由上文可知漏洞的触发只需要用户输入的变量字符串长度超出 160 字节, 因此, 我们将捕获到的请求包中 devicelist 的关键字赋值为大于 160 个字节的字符串, 并发送请求。此时刷新前端页面显示连接失败, 无法连接到服务器, 同时后端界面也报出 Segmentation fault 的错误信息, 如图 13 所示。由此可见栈溢出漏洞被触发, 使得 Web 服务器崩溃, 达到了拒绝服务的目的。

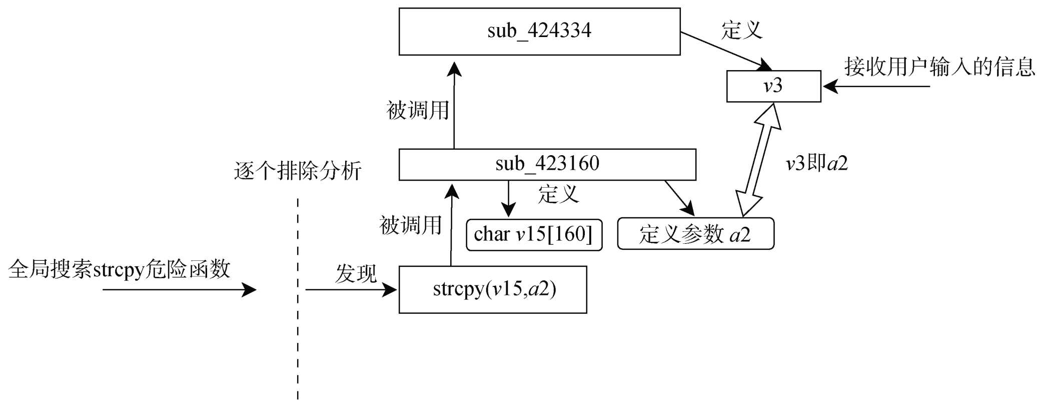

图 12 追踪危险函数的逆向分析过程

Figure 12 Tracing the reverse analysis process of the hazard function

```
[cgi:formSetMacFilterCfg:511] get mac filter mode: ! 0!
[cgi:parse_macfilter_rule:284] parase rule: name == 1234aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa, mac == A2:A3:A4:A5:A6:A7
Segmentation fault
```

图 13 Web 服务后台报错

Figure 13 Web service background error

将该实验作用于 Tenda AX12 的真机上进行攻击, 同样得到了 Web 服务器崩溃的结果, 证明 Tenda AX12 路由器的栈溢出漏洞在真机上实验成功。

在验证漏洞的真实性后, 漏洞发现者需要提供 POC(Proof of Concept)代码, 它是一段可以重现漏洞的代码, 也可以是一个能够触发程序崩溃的恶意文件, 也可以是漏洞发掘者用来实现特殊功能的 Python 脚本。因此 POC 有很多展现形式, 目的是为了更方便的触发漏洞或者还原整个攻击过程。

本文以 Python3 语言进行 POC 的开发, 通过安装 request 模块来发送 http 中的 POST 请求, 以安装 pwn 模块借助 cyclic()函数自动生成大量的恶意字符, 与其他正常语句拼接形成攻击语句。

以下为 Tenda AX12 模拟机编写的 POC 如图 14 所示:

经过测试可以达到以上攻击效果, 即提供路由器的 Web 服务的 httpd 崩溃, 对于 Tenda AX12 来说需要在 QEMU 启动的 MIPS 虚拟机的 BusyBox 命令窗口重启 httpd 服务; 而对于 Tenda AX12 真机来说需要重启路由器真机后面的恢复按钮重启 httpd 服务, Web 页面将进入重新配置路由器的界面。

4.2.3 基于 SaTC 的对比实验

Chen 等人[1]设计的 SaTC 包含三个组件: 用于从前端文件中收集关键字的输入关键字提取器、用于在后端二进制文件中定位输入条目的输入条目识别器以及用于有效检测漏洞的输入敏感污点引擎。SaTC 的原型是基于 Ghidra 和 Karonte 实现的, 它支持解析多种类型的前端文件, 包括 JavaScript、HTML 和 XML 文件, 并且可以分析广泛使用的架构中的后端, 例如 x86、ARM 和 MIPS。

处理用户输入的后端函数通常与相应的前端文件共享一个相似的关键字: 在前端, 用户输入用关键字标记并编码在数据包中; 在后端, 相同或相似的关键字用于从数据包中提取用户输入。因此,

```python
#coding=utf-8
import requests
from pwn import *

url='http://192.168.136.153/goform/setMacFilterCfg'
payload=cyclic(230)
header={
'Host': '192.168.136.153',
'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:91.0) Gecko/20100101 Fire
fox/91.0',
'Accept':'*/*',
'Accept-Language':'zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2',
'Accept-Encoding':'gzip, deflate',
'Content-Type':'application/x-www-form-urlencoded; charset=UTF-8',
'X-Requested-With':'XMLHttpRequest',
'Origin':'http://192.168.136.153',
'Connection':'close',
'Referer':'http://192.168.136.153/mac_filter.html?random=0.008005922687726486&',
'Cookie':'password=b6ae3a4e44146ebb29dda4e51e5deac5ooobcx'}

data1 ='macFilterType=black&deviceList=%s\r\rA2:A3:A4:A5:A6:A7'%payload
ret = requests.post(url = url ,headers = header,data = data1, verify=False)
```

图 14　POC

Figure 14　Proof of concept

Chen 等人[1]认为使用前后端通用关键字来识别从嵌入式系统中发现漏洞的关键是利用 Web 服务的前端来定位处理用户提供的数据的后端代码。故本文采用 SaTC 工具作为对照实验。

我们将上文分析的 Web 服务器中的 httpd 文件使用 SaTC 工具进行分析，得到了潜在漏洞位置在 macFilterType 关键字处，距离我们的实验方法找到的实际的漏洞位置 devicelist 关键字非常接近(上文中提到的攻击点即为漏洞的位置)，但是 SaTC 的漏洞报告中并没有定位到实际的漏洞位置，如图 15 为借助 SaTC 工具检测出来的函数调用图。因此，SaTC 前后端通用关键字的污点检查技术的精确度对于本文的追踪危险函数的逆向分析方法的精确度还存在一定差距。

```
[Param "macFilterType"(0x00452bd0), Referenced at FUN_00424334 : 0x00424
3e4] >> 0x004245d8 >> FUN_00423160 >> 0x0042326c -> strcpy
[Param "macFilterType"(0x00452bd0), Referenced at FUN_00424334 : 0x00424
3e4] >> 0x004245d8 >> FUN_00423160 >> 0x00423278 -> strcpy
[Param "macFilterType"(0x00452bd0), Referenced at FUN_00424334 : 0x00424
3e4] >> 0x004245fc >> FUN_00423160 >> 0x0042326c -> strcpy
[Param "macFilterType"(0x00452bd0), Referenced at FUN_00424334 : 0x00424
3e4] >> 0x004245fc >> FUN_00423160 >> 0x00423278 -> strcpy
[Param "macFilterType"(0x00452bd0), Referenced at FUN_00423cf0 : 0x00423
f60] >> 0x00423f80 -> FUN_004237a0 >> 0x0042386c >> FUN_004235d4 >> 0x00
423780 -> strcpy
[Param "macFilterType"(0x00452bd0), Referenced at FUN_00423cf0 : 0x00423
f60] >> 0x00423f9c -> FUN_004237a0 >> 0x0042386c >> FUN_004235d4 >> 0x00
423780 -> strcpy
[Param "macFilterType"(0x00452bd0), Referenced at FUN_00423cf0 : 0x00423
f60] >> 0x00423fa8 -> FUN_0042399c >> 0x00423a18 >> FUN_004235d4 >> 0x00
423780 -> strcpy
[Param "macFilterType"(0x00452bd0), Referenced at FUN_0042ada8 : 0x0042a
eec] >> 0x0042af14 >> FUN_004235d4 >> 0x00423780 -> strcpy
[Param "macFilterType"(0x00452bd0), Referenced at FUN_0042ada8 : 0x0042a
eec] >> 0x0042b0fc >> FUN_0042aaec >> 0x0042ac14 -> strcpy
```

图 15　SaTC 检测 macFilterType 函数调用图

Figure 15　SaTC detects macFilterType function call graph

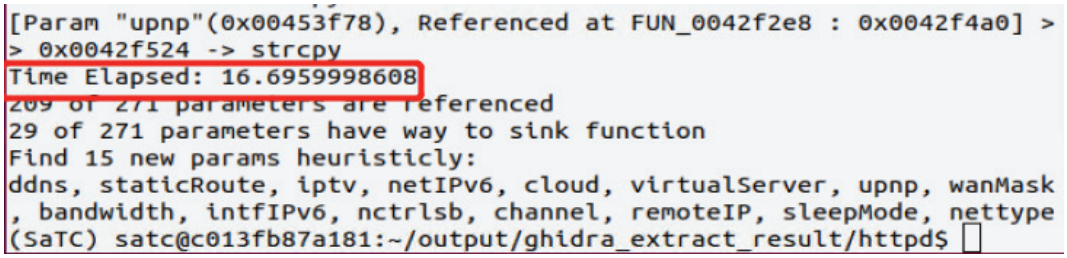

图 16　SaTC 分析 httpd 文件结果图
Figure 16　SaTC analysis httpd file result graph

通过将本文方法和 SaTC 相比较, 我们得出了以下几点:

(1) 在性能方面 SaTC 运行整个 httpd 文件分析得出结果的时间只需要 16.69 s, 如图 16 所示。当然这与使用本文的方法分析漏洞的时间存在差距, SaTC 使用 16.69 s 的时间找出了 httpd 文件中存在的可疑漏洞, 但是该工具, 对于关键字的定位不准确, 通过牺牲挖掘漏洞的完整性来达到提高分析效率的目的。

(2) SaTC 工具只能分析命令注入的漏洞, 且在分析前后端文件关键字时会收集到更多假阳性的关键字, 导致大量误报, 这些关键字一般是常数字符串、函数参数以及 id 标签的值, 本文改进了这一缺点, 对于不是用户交互的可控变量的参数, 在路径选择时对其进行剪枝, 提高分析效率。

(3) SaTC 通过前后端共享关键字来识别漏洞, 当前后端不使用共享关键字时, 该方法会将可疑漏洞检测为假阴性, 错过更多的漏洞, 降低漏洞检测的准确性。

(4) 对于本文的实验对象 Tenda AX12, SaTC 工具分析结果只能输出 319 条 strcpy 函数的函数调用关系, 解析出该文件的 271 个参数中有 29 个参数可以到达该危险函数, 但并未将参数和函数结合到一起确定具体的是由某个函数调用的某个参数引发的漏洞, 如此多的函数调用路径仍需要人工手动验证其准确性, 这无疑给工作人员增加了很大的时间成本。本文使用路径选择的方法优化路径, 选择更有可能的路径, 减少对无效路径的分析, 提高了验证的效率。

(5) SaTC 工具输出所有的调用函数关系仅仅是 strcpy 函数的调用路径, 没有对其他危险函数进行函数调用分析。本文提出的方法可以对多种危险函数进行分析, 且在分析的过程中不断对路径进行选择优化, 提高了漏洞挖掘的效率。

4.2.4　结果分析

本文的追踪危险函数的逆向分析方法可以准确的定位到漏洞存在的位置, 同时也可以利用该漏洞触发漏洞。SaTC 工具可以在较短时间内分析出更多的潜在漏洞, 但漏洞的存在性和可利用性需要进一步人工分析。本文的方法和 SaTC 方法的区别在于 SaTC 工具是搜索所有前端和后端共同的共享关键字来确定共享关键字的数据流向, 而本文的方法认为漏洞的触发大多出现在用户与设备交互的位置, 因此本文的方法是通过危险函数逆向分析的方法寻找前端用户输入与后端共享的关键字确定数据流向, 从而分析出漏洞准确的位置, 该方法极大的减少了无用关键字的搜索, 提升了效率。

物联网设备的 Web 服务通常由两个组件组成, 即前端和后端。前端向终端用户展示设备的配置和功能, 而后端解析从前端接收到的请求并执行相关的服务。用户在前端登录路由器的设置网页进行 MAC 地址过滤, 在 Web 界面输入设备名, 发送正常的 http 请求, 而攻击者通过后台控制输入不合法的设备名, 例如: 使用超出规定长度范围的设备名, 并发送恶意请求, 前端传入的 devicelist 通过图 17 中的 A 随数据流流向 B 中的 *v*3, 流向 C 中的 *v*12, 最终通过 strcpy 函数复制到参数 *v*15 中, 函数定义了 *v*15 参数长度为 160, 攻击者发送的恶意请求长度超限, 导致缓冲区溢出, 从而引起系统崩溃, 使得路由器拒绝服务。

使用本文基于危险函数的逆向追踪方法挖掘到了多个漏洞, 如表 5。四个研究对象主要是针对不同型号的路由器, 研究的漏洞文件是基于 Web 服务的 httpd 二进制文件。Tenda AX12 使用的 MIPS32 大端架构, Tenda AC 系列以及 Netgear R 系列使用的 ARM32 小端的系统架构, 以下四种漏洞的位置均表现在 httpd 文件中使用的危险函数 strcpy 引起的缓冲区溢出上, 并且造成的结果均表现为服务器拒绝服务。实验证明了本文的方法在不同品牌不同型号的路由器上是同样适用的。通过上述的验证测试工作, 表明了本文提出的漏洞探测方法对网络设备具有一定的漏洞挖掘能力, 可以用于路由设备的漏洞挖掘

图 17　恶意数据流向

Figure 17　Malicious data flow

表 5　漏洞成果

Table 5　Vulnerability achievement

研究对象	系统架构	漏洞描述	漏洞位置	是否获得 CNVD 认证	是否获得 CVE 认证
Tenda AX12 路由器	MIPS32(MSB)	HTTP 拒绝服务	sub_423160 函数栈溢出	是	是
Tenda AC9 路由器	ARM32(LE)	HTTP 拒绝服务	sub_16B4C 函数栈溢出	是	是
Tenda AC6 路由器	ARM32(LE)	HTTP 拒绝服务	sub_2E4EC 函数栈溢出	未提交	未提交
Netgear R8300 路由器	ARM32(LE)	HTTP 拒绝服务	sub_25E04 函数栈溢出	是	未提交

5　总结展望

5.1　总结

近些年来, 伴随着网络设备的产生和发展, 网络设备越来越多的出现在人们的日常生活中, 方便了人们的同时也给用户的个人财产和隐私带来了极大的挑战, 网络系统将人们紧密的联系在一起, 各种数据信息暴露的网络中, 给黑客和攻击者提供了可乘之机。为了更好的保护用户隐私, 先于黑客和攻击者更早的发现智能系统的漏洞, 本文针对路由器提出了一套漏洞挖掘的方法和流程, 同时利用该方法在 Tenda AX12 路由器上进行了实验,并且也在不同品牌不同型号的路由器上进行了实验, 都得到了有效的结果。

在 Tenda AX12 路由器模拟环节展示了一个从获取路由器到成功仿真的过程, 并为 0-Day 栈溢出漏洞的挖掘和利用奠定了基础。漏洞挖掘过程中采用追踪危险函数 strcpy 的逆向分析方法, 从 Web 服务器的 post 数据包中找到路由表函数, 逆向追踪分析定位到 Web 服务的 httpd 二进制文件中的函数 sub_423160 上, 并成功挖掘到 0-Day 栈溢出漏洞。进而对找到的漏洞进行利用, 验证漏洞的真实存在性和可利用性。该漏洞导致了路由器拒绝服务, 使得用户无法接入路由器, 本实验仅使用简单攻击利用漏洞就造成了拒绝服务, 如果该漏洞被恶意攻击者利用, 可能以此来窃取用户信息, 将给用户造成更严重的危害。

通过本文的方法我们挖掘到了 0-Day 栈溢出漏洞并上报给 CNVD 国家漏洞数据库和 CVE 平台认证, 且获得了国家官方机构的认可。证书编号为 CNVD-YCGN-202203071727, CVE-2022-28561, CNVD-YCGN-202203071626。

5.2　展望

本文的工作在路由器设备上进行了方法验证, 未来可以在更多的物联网设备上进行实验验证方法的可行性。同时, 本文涉及到的嵌入式固件设备的安全分析技术和通用 PC 差别不大, 通用 PC 的方法也适用于嵌入式固件的分析, 在嵌入式固件的仿真方

面, 使用纯软件的方式替代硬件, 便于嵌入式设备动态分析的方法仍然发展不够成熟, 对于部分固件, 软件仿真的实现并不顺利, 这严重影响对固件的动态分析。由于物联网设备不同其系统环境也不同, 尽管有一些工具可以实现固件仿真, 但是对于成功仿真物联网嵌入系统并不是一件容易的事情, 仍然需要我们努力去探索。利用本文的方法已经被证明是设备漏洞挖掘的一种有效的方法, 但是提高挖掘效率和改进测试方法仍然需要不断深入研究。未来我们期望能开发一个可以有效仿真嵌入式系统的平台, 提高仿真成功率。期望在未来能将现有的方法整合为一套系统的进行漏洞挖掘的方法应用于嵌入式设备上, 和更多现有的漏洞挖掘技术相结合, 以期提高其漏洞挖掘的效率, 为改善漏洞逐渐增高的现状提供一定的工具支持。国家网络空间安全需要每一位安全人员去守护, 希望本文能为网络安全漏洞挖掘提供一些方法参考。

参考文献

[1] Libo Chen, Yanhao Wang, Quanpu Cai. Sharing More and Checking Less: Leveraging Common Input Keywords to Detect Bugs in Embedded Systems[C]. *USENIX Association* 2021, 2021:1-17.

[2] Andriesse D, Chen X, Van Der Veen V, et al. An In-Depth Analysis of Disassembly on Full-Scale X86/X64 Binaries[C]. *The 25th USENIX Conference on Security Symposium*, 2016: 583-600.

[3] Deng Y, Zhang Y, Cheng L. Static integer overflow vulnerability detection in windows binary[M]. *Advances in Information and Computer Security*. Springer Berlin Heidelberg, 2013.

[4] Redini N, Machiry A, Wang R Y, et al. Karonte: Detecting Insecure Multi-Binary Interactions in Embedded Firmware[C]. *2020 IEEE Symposium on Security and Privacy*, 2020: 1544-1561.

[5] Gao T, Guo X. Buffer Overflow Vulnerability Location in Binaries Based on Abnormal Execution[C]. *2020 4th Annual International Conference on Data Science and Business Analytics*, 2020: 29-31.

[6] Cai J, Zou P, Ma J X, et al. SwordDTA: A Dynamic Taint Analysis Tool for Software Vulnerability Detection[J]. *Wuhan University Journal of Natural Sciences*, 2016, 21(1): 10-20.

[7] Wang X F, Ma H T, Jing L S. A Dynamic Marking Method for Implicit Information Flow in Dynamic Taint Analysis[C]. *The 8th International Conference on Security of Information and Networks*, 2015: 275-282.

[8] Choi Y H, Park M W, Eom J H, et al. Dynamic Binary Analyzer for Scanning Vulnerabilities with Taint Analysis[J]. *Multimedia Tools and Applications*, 2015, 74(7): 2301-2320.

[9] Erb C, Collins M, Greathouse J L. Dynamic Buffer Overflow Detection for GPGPUs[C]. *2017 IEEE/ACM International Symposium on Code Generation and Optimization*, 2017: 61-73.

[10] Jiang Y K, Xie W, Tang Y. Detecting Authentication-Bypass Flaws in a Large Scale of IoT Embedded Web Servers[C]. *The 8th International Conference on Communication and Network Security*, 2018: 56-63.

[11] Yu B, Wang P F, Yue T, et al. Poster: Fuzzing IoT Firmware via Multi-Stage Message Generation[C]. *The 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019: 2525-2527.

[12] Zheng Y W, Song Z W, Sun Y Y, et al. An Efficient Greybox Fuzzing Scheme for Linux-Based IoT Programs through Binary Static Analysis[C]. *2019 IEEE 38th International Performance Computing and Communications Conference*, 2019: 1-8.

[13] Zhu L P, Fu X T, Yao Y, et al. FIoT: Detecting the Memory Corruption in Lightweight IoT Device Firmware[C]. *2019 18th IEEE International Conference on Trust*, *Security and Privacy In Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering*, 2019: 248-255.

[14] Zhang Y, Huo W, Jian K P, et al. SRFuzzer: An Automatic Fuzzing Framework for Physical SOHO Router Devices to Discover Multi-Type Vulnerabilities[C]. *The 35th Annual Computer Security Applications Conference*, 2019: 544-556.

[15] Wang W J, Tian D H, Ma R, et al. SHFuzz: A Hybrid Fuzzing Method Assisted by Static Analysis for Binary Programs[C]. *China Communications*, 2021: 1-16.

[16] Liu B C, Shi L, Cai Z H, et al. Software Vulnerability Discovery Techniques: A Survey[C]. *2012 Fourth International Conference on Multimedia Information Networking and Security*, 2012: 152-156.

[17] Sun Haobin. Research on key technologies of Cisco router attack behavior detection [D]. Zheng State: Strategic Support Forces Information Engineering University, 2018.
(孙豪彬.Cisco 路由器攻击行为检测关键技术研究[D].郑州:战略支援部队信息工程大学, 2018.)

[18] Li F J. Research on Fuzzing-Based Vulnerability Discovery in Distributed Network[D]. Beijing: Beijing University of Posts and Telecommunications, 2015.
(李风娇. 分布式网络中基于Fuzzing的漏洞挖掘研究[D]. 北京: 北京邮电大学, 2015.)

[19] Du W, Liu G S. A Survey of Backdoor Attack in Deep Learning[J]. *Journal of Cyber Security*, 2022, 7(3): 1-16.
(杜巍, 刘功申. 深度学习中的后门攻击综述[J]. *信息安全学报*, 2022, 7(3): 1-16.)

[20] Costin A, Zaddach J, Francillon A, et al. A Large-Scale Analysis of the Security of Embedded Firmwares[C]. *The 23rd USENIX conference on Security Symposium*, 2014: 95-110.

[21] Thomas S L, Chothia T, Garcia F D. Stringer: Measuring the Importance of Static Data Comparisons to Detect Backdoors and Undocumented Functionality[C]. *European Symposium on Research in Computer Security*, 2017: 513-531.

[22] Thomas S L, Garcia F D, Chothia T. HumIDIFy: A Tool for Hidden Functionality Detection in Firmware[C]. *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2017: 279-300.

[23] Cheng K, Li Q, Wang L, et al. DTaint: Detecting the Taint-Style Vulnerability in Embedded Device Firmware[C]. *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2018: 430-441.

[24] Celik Z B, Babun L, Sikder A K, et al. Sensitive Information Tracking in Commodity IoT[C]. *The 27th USENIX Conference on Security Symposium*, 2018: 1687-1704.

[25] Zheng Y W, Song Z W, Sun Y Y, et al. An Efficient Greybox Fuzzing Scheme for Linux-Based IoT Programs through Binary Static Analysis[C]. *2019 IEEE 38th International Performance Computing and Communications Conference*, 2019: 1-8.

[26] Dai Z H, Fei Y K, Zhao B, et al. Research on the Localization of Firmware Vulnerability Based on Stain Tracking[J]. *Journal of Shandong University* (*Natural Science*), 2016, 51(9): 41-46, 52.
(戴忠华, 费永康, 赵波, 等. 基于污点跟踪的固件漏洞定位研究[J]. *山东大学学报(理学版)*, 2016, 51(9): 41-46, 52.)

[27] Chen J Y, Diao W R, Zhao Q C, et al. IoTFuzzer: Discovering Memory Corruptions in IoT through App-Based Fuzzing[C]. *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.

[28] Tobias Scharnowski, Nils Bars, Moritz Schloegel. Fuzzware: Using Precise MMIO Modeling for Effective Firmware Fuzzing[C]. *31st USENIX Security Symposium. USENIX Association*, 2022.

[29] Tan Y M, Wang Y J, Xue Z. Analysis and Detection of SOHO Router Backdoor[J]. *Communications Technology*, 2017, 50(10): 2324-2332.
(谭云木, 王轶骏, 薛质. SOHO 路由器后门分析与检测研究综述[J]. *通信技术*, 2017, 50(10): 2324-2332.)

[30] Shaohua Wu, Wei Wang, Xu Zhao. Demystifying home router 0day vulnerability mining technology[M]. BEIJING: Publishing House of Electronics Industry, 2015: 2-359.
(吴少华, 王炜, 赵旭. 揭密家用路由器 0day 漏洞挖掘技术[M]. 北京: 电子工业出版社, 2015: 2-359.)

[31] Yao Wenxiang. The Definitive Guide to ARM Cortex-M3 and Cortex-M4 (3rd Edition) [M]. Tsinghua University Press, 2015
(姚文祥. ARM Cortex-M3 与 Cortex-M4 权威指南(第 3 版)[M]. 清华大学出版社, 2015.)

[32] Wu S Z, Guo T, Dong G W, et al. Software vulnerability analysis technology[M]. Beijing: Science Press, 2014.
(吴世忠, 郭涛, 董国伟, 等. 软件漏洞分析技术[M]. 北京: 科学出版社, 2014.)

[33] Yu J J. Research and Application on Risk Function Based Buffer Overflow Detection Method[J]. *Computer Applications and Software*, 2011, 28(9): 185-187.
(于继江. 基于危险函数的缓冲区溢出检测方法的研究与实现[J]. *计算机应用与软件*, 2011, 28(9): 185-187.)

郑炜 于 2007 年在西北工业大学计算机工程专业获得博士学位。现任西北工业大学软件学院副教授, 博硕导, CCF 高级会员。研究领域为软件测试与形式化验证。研究兴趣包括: 软件安全, 智能软件测试与验证。Email: wzheng@nwpu.edu.cn

许晴晴 于 2020 年在太原工业学院数字媒体技术专业获得学士学位。现在西北工业大学软件工程专业攻读硕士学位。研究领域为软件测试、漏洞安全。研究兴趣包括: 软件测试、软件漏洞挖掘。Email: qingqing-xu@mail.nwpu.edu.cn

李奇 于 2019 年在太原理工大学安全工程专业获得学士学位。现在西北工业大学软件工程专业攻读硕士学位。研究领域为软件测试。研究兴趣包括: 软件自动化测试、漏洞挖掘。Email: 2019213622@mail.nwpu.edu.cn

陈翔 于 2011 年在南京大学计算机软件与理论获得博士学位。现任南通大学信息科学技术学院副教授, CCF 高级会员。研究领域为软件工程。研究兴趣包括: 软件测试与维护、软件仓库挖掘、经验软件工程。Email: xchencs@ntu.edu.cn

孙家泽 于 2015 年在西北大学计算机软件理论专业获得博士学位。现任西安邮电大学计算机学院教授, CCF 高级会员。研究领域为软件测试,智能优化。研究兴趣包括: 软件漏洞挖掘、智能优化算法。Email: sunjiaze@xupt.edu.cn

附　　录