面向Cisco IOS-XE的Web命令注入漏洞检测

何杰, 蔡瑞杰, 尹小康, 陆炫廷, 刘胜利

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220100113

面向 Cisco IOS-XE 的 Web 命令注入漏洞检测

何 杰 蔡瑞杰 尹小康 陆炫廷 刘胜利
数学工程与先进计算国家重点实验室 郑州 450001
(polaris201909@qq.com)

摘 要 思科公司的新型操作系统 Cisco IOS-XE 广泛部署于 Cisco 路由器、交换机等平台,但系统的 Web 管理服务中存在通过命令注入实现权限逃逸的安全漏洞,使网络安全面临严重威胁。近年来,模糊测试常被用于检测嵌入式设备的安全漏洞,然而目前没有针对 Cisco IOS-XE 系统 Web 管理服务的模糊测试框架,由于 IOS-XE 特有的系统架构和命令模式,现有 IoT 模糊测试方法在 IOS-XE 上的检测效果不佳。为此,提出了一个针对 Cisco IOS-XE 系统 Web 管理服务的模糊测试框架 CRFuzzer,用于检测命令注入漏洞。CRFuzzer 结合 Web 前端请求和后端程序分析以优化种子生成,基于命令注入漏洞的特征发现脆弱代码以缩小测试范围。为了评估 CRFuzzer 的漏洞检测效果,在实体路由器 ISR 4000 系列和云路由器 CSR 1000v 上对 31 个不同版本共 124 个固件进行了测试,共检测出 11 个命令注入漏洞,其中 2 个为未公开漏洞。

关键词:Cisco IOS-XE;Web 服务;命令注入;漏洞检测;模糊测试
中图法分类号 TP393

Detection of Web Command Injection Vulnerability for Cisco IOS-XE

HE Jie,CAI Ruijie,YIN Xiaokang,LU Xuanting and LIU Shengli
State Key Laboratory of Mathematical Engineering and Advanced Computing,Zhengzhou 450001,China

Abstract Cisco's new operating system,Cisco IOS-XE,is widely deployed on platforms such as Cisco routers and switches. However,there are vulnerabilities in the system's Web management interface to allow permission escalation through command injection. Network security is facing serious threats. In recent years,fuzzing is usually used to detect security vulnerabilities in embedded devices,but there is currently no fuzzing framework for Cisco IOS-XE,and current fuzzing methods for IoT have poor performance due to the unique system architecture and command mode of IOS-XE. To solve the problems mentioned above,this paper proposes a novel fuzzing framework CRFuzzer for the Web management service in Cisco IOS-XE system to detect command injection vulnerabilities. CRFuzzer combines front-end requests and back-end scripts analysis to optimize seed generation,and locates vulnerable code based on characteristics of command injection to narrow the scope of testing. In order to evaluate the vulnerability detection performance of CRFuzzer,124 firmwares of 31 different versions are tested on the physical router ISR 4000 series and the cloud router CSR 1000v,and a total of 11 command injection vulnerabilities are detected,and 2 of them are undisclosed vulnerabilities.

Keywords Cisco IOS-XE,Web service,Command injection,Vulnerability detection,Fuzzing

1 引言

代码注入是 Web 应用中常见的安全漏洞,也是网络安全面临的主要威胁之一,在 OWASP[1] 于 2021 年发布的十大网络安全威胁中排名第三。其中,命令注入是攻击者利用应用程序对不可信数据的不恰当处理,在输入中插入恶意构造的系统命令,从而导致计划外的执行行为[2],造成拒绝服务(Denial of Service,DoS)或权限逃逸等。命令注入漏洞不仅存在于托管在服务器上的 Web 应用程序中,也存在于网络核心设备及物联网设备的 Web 管理服务中。

Cisco IOS-XE(下文中简称 IOS-XE)是 Cisco 公司的新型操作系统(IOS-XE,IOS-XR 和 NX-OS)之一,被广泛应用于 Cisco 企业级交换机、聚合服务路由器、分支路由器、工业级路由器、以太网交换机和云虚拟路由器等平台[3]。其中,Web UI 作为用户查看路由设备信息和修改设备配置的交互接口,同样存在命令注入漏洞。攻击者利用这些安全漏洞,可以实现权限逃逸,以 Read-Only 权限登录的用户,可以提升至管理员权限,进而获取敏感数据信息,或篡改路由设备的配置。攻击者甚至能够实现更高级别的逃逸,进入设备底层的 Linux 系统,以 root 权限执行任意命令,如读取或篡改系统文件、

到稿日期:2022-01-12 返修日期:2022-07-07
基金项目:科技委基础加强项目(2019-JCJQ-ZD-113)
This work was supported by the Foundation Strengthening Key Project of Science & Technology Commission(2019-JCJQ-ZD-113).
通信作者:刘胜利(dr_liushengli@163.com)

下载并运行恶意程序等，实现对路由设备的完全控制。

然而，目前没有针对 IOS-XE 的漏洞检测方法，由于 IOS-XE 特有的系统架构和命令模式，现有针对传统 Cisco IOS 或 IoT 设备的漏洞检测方法在 IOS-XE 上效果不佳。Cisco IOS 的漏洞检测[4-6]极度依赖于 Dynamips 对固件进行模拟运行，而目前没有模拟器支持 IOS-XE 系统。针对 IoT 的漏洞检测方法中，模糊测试被认为是发现漏洞最简单且高效的方法。但现有的 IoT 模糊测试方法，要么仅通过分析前端[7-8]或参考官方及第三方 API 文档[9]来生成初始种子，容易造成漏报，且没有明确的测试目标范围，漏洞检测效率较低；要么依赖于 QEMU 等模拟器对设备进行仿真[10-12]，无法适用于 IOS-XE。

本文应用模糊测试技术对 IOS-XE 进行安全漏洞检测。为了设计一种有效的模糊测试方法，需要解决如下难点：

(1)缺乏反馈信息。目前没有工具或方法可以在不利用漏洞的情况下，监测 IOS-XE 系统内部的执行信息，导致模糊测试无法基于代码覆盖率等反馈信息来优化变异策略。

(2)严格的参数格式。Web 服务的前端和后端程序都对参数格式有严格的语法要求，大部分随机变异生成的测试用例会被服务端直接拒绝。此外，后端处理请求的函数中，可能会为一些参数附加特殊字符(如引号)，使其不可执行。

(3)存在隐藏 API。由于跨平台的 IOS-XE 系统之间存在大量的代码复用，或部分功能需要额外配置，使得 IOS-XE 系统中存在一些 API，前端 Web UI 中没有相应的配置接口或配置接口不可用。若仅从前端分析生成初始种子，容易造成漏报。

为解决以上问题，本文提出了一个针对 IOS-XE 系统 Web 管理服务的模糊测试框架 CRFuzzer，用于检测命令注入漏洞。与现有模糊测试方法不同，CRFuzzer 结合 Web 前后端分析优化种子生成策略，通过分析前端请求来解析参数的属性，分析后端程序来发现隐藏的 API，进而完善种子的生成。同时，CRFuzzer 基于漏洞特征筛选出可能存在命令注入漏洞的脆弱代码，缩小测试范围。此外，CRFuzzer 根据常量值参数为单个后端端点生成多个初始种子，以尽可能地提高模糊测试的代码覆盖率。本文的主要贡献总结如下：

(1)基于 IOS-XE 系统 Web 服务的工作机制，提出了一种结合 Web 前后端分析来优化种子生成的方法，提高了模糊测试的漏洞检测能力。

(2)分析了 IOS-XE 系统中两种类型的命令注入漏洞，提出了一种基于漏洞特征的脆弱代码筛选方法，缩小了测试范围，提高了模糊测试的效率。

(3)提出并实现了针对 IOS-XE 系统 Web 管理服务的模糊测试框架 CRFuzzer，用于检测命令注入漏洞。在实体路由器 ISR 4000 系列及云虚拟路由器 CSR 1000v 上，对 31 个不同版本共 124 个固件进行了测试，共计检测出 11 个命令注入漏洞，其中 2 个为未公开漏洞。

2 背景与相关工作

2.1 Cisco IOS-XE Web UI 通信架构

IOS-XE 基于 Linux 构建，传统的 IOS 作为一个独立进程 IOSd 运行于 Linux 内核之上，而 Web UI 的服务端 Nginx 同样作为一个独立进程运行。服务端请求处理采用 Lua-Nginx[13]的模式，后端由 Lua 程序进行请求数据解析、命令构建和响应数据过滤。WebUI 的通信架构如图 1 所示。

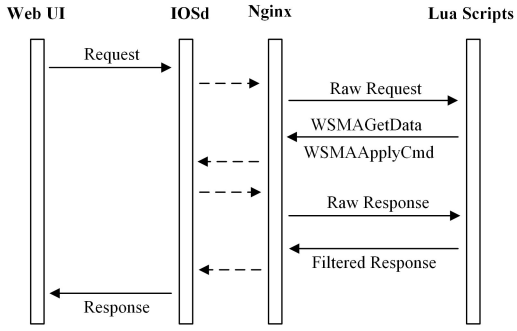


图 1 Cisco IOS-XE Web UI 的通信架构

Fig. 1 Communication architecture of Cisco IOS-XE Web UI

用户使用浏览器登录 IOS-XE 的 Web 界面，通过 Web UI 对路由设备进行信息查看和配置修改，将在前端生成的相应的请求数据发送到服务端。在 Web UI 的服务端，Nginx 将请求数据传递给 baseHandler 程序，进行用户权限校验，随后在 URLMap 中搜索相应的 uri，找到对应的请求处理程序，完成请求参数解析和命令构建。最后在底层 Linux 系统中执行命令，或调用 WSMAGetData 或 WSMAApplyCmd 函数将生成的命令传递到 IOSd 执行。

在后端的处理过程中，Lua 程序负责解析请求数据并构建相应的命令。其中，部分参数将被用于生成最终的命令，如果缺乏对此类参数的校验，或者对参数校验的方式不正确，便可能会发生命令注入。攻击者可以在参数中附加恶意构造的命令，使系统产生计划外的执行行为，以获取敏感数据信息或篡改设备配置。

2.2 Cisco IOS-XE 命令注入漏洞

与传统 Cisco IOS 的单片化结构[14]不同，IOS-XE 采用模块化设计，以 Linux 为底层系统，将部分功能模块移出 IOS，改为 Linux 上的独立进程。余下的 IOS 部分同样作为独立进程 IOSd 运行于 Linux 之上，与传统 IOS 享有相同的命令模式。根据 IOS-XE 特有的系统架构和不同的命令执行层级，本文将 IOS-XE Web 服务中的命令注入漏洞分为两类：在底层 Linux 系统中执行的底层系统型命令注入漏洞以及在 IOSd 的命令行接口(Command Line Interface，CLI)中执行的用户权限型命令注入漏洞。

(1)底层系统型。这类漏洞允许用户进入 IOS-XE 的底层 Linux 系统，在 Linux 中以 root 权限执行任意命令。攻击者利用这类漏洞，可以读取或删改底层 Linux 系统的文件，获得对路由设备的完全控制。

(2)用户权限型。这类漏洞允许 Read-Only 权限的用户在 IOSd 的 CLI 中执行管理员权限用户才能执行的命令。攻击者利用这类漏洞，可以在 Read-Only 权限下获取设备敏感数据信息，甚至修改路由设备的配置。

2.3 相关工作

网络核心设备和 IoT 设备被广泛应用于工业生产或日常生活中，在给人们带来便利的同时，设备的安全漏洞也使网络空间面临严重的安全威胁。研究人员尝试利用模糊测试、

符号执行、污点分析等多种方法对嵌入式设备进行安全分析，检测其中存在的安全漏洞。IOS-XE 与常见的嵌入式设备存在较大差异，目前没有针对 IOS-XE 的模糊测试方法，因此本文选取了较为接近的技术，即针对嵌入式设备的模糊测试，进行相关工作的介绍。根据测试对象，对嵌入式设备的模糊测试可以分为基于模拟仿真和针对实体设备两种研究路线。

在基于模拟仿真的方法中，RPFuzzer[15] 基于修改后的 Dynamips 对路由器固件进行模拟运行，而 FIRM-AFL[10]，FirmFuzz[11] 和 FIRMADYNE[12] 基于 QEMU 对 IoT 设备进行仿真，并在其中嵌入模糊测试工具。在固件缺少 Boot-Loader 的情况下，FIoT[16] 根据调用流图将固件代码进行切片，然后对切片后的代码进行动态执行并应用模糊测试。Re-Link[17] 将模糊测试与动态数据流追踪相结合，根据请求对设备上存储数据的依赖，检测需要多步请求才能触发的命令注入漏洞。上述方法使用模拟器模拟运行固件，主要目的在于获取系统运行时的 CPU 及内存等状态信息，以便进行异常监测。同时，这些状态信息可以作为反馈，引导变异策略。然而，研究人员很难完全模拟各种 CPU 架构，Dynamips 或 QEMU 等模拟器也不支持对 IOS-XE 进行模拟运行。

在针对实体设备的方法中，主要工作体现在优化种子生成和变异策略方面。SRFuzzer[7] 是针对实体 SOHO 路由器的自动化模糊测试框架，通过捕获设备运行时的 Web 请求来生成初始种子，对请求中的参数进行语义分析和属性标记以优化变异策略。IoTFuzzer[8] 则通过分析用于连接和控制设备的 App 来获取初始种子，在变异规则中主要针对堆栈溢出、整数溢出或越界访问、空指针或数据类型错误 3 种内存错误漏洞。Snipuzz[9] 的初始种子来源于官方或第三方 API 文档或测试用例，Snipuzz 提出了一种 message 的元素属性推断机制，基于 response 归类推断 message 中各字节之间的关系。上述方法均建立在固件及系统运行时状态无法获取的前提下，仅从前端提取 API 信息，用于生成初始种子，忽略了对后端程序的分析，无法解决本文所提到的隐藏 API 问题。

部分方法结合了对后端程序的分析，Jiang 等[18] 将程序静态分析与动态模糊测试相结合，从固件解包得到的程序中提取字符串信息，用于生成初始种子，该方法仍然依赖于 QEMU 对固件的模拟运行。Chen 等[19] 指出前后端程序变量通常共享相同的 keyword，据此可以建立前端输入和后端程序的关联性，但在后续工作中使用的是静态污点分析方法。

此外，上文所列举的模糊测试方法的测试目标没有导向性，多为“地毯式”测试，漏洞检测效率较低。Dowser[20] 通过静态分析方法，查找涉及循环、数组访问、指针访问等与溢出型漏洞紧密相关的操作，由此筛选可能存在溢出漏洞的脆弱代码段。后续仅对筛选出的代码区域进行测试，有效缩小了模糊测试的目标范围。在设备固件可获取、可逆向的条件下，这种思想同样可以应用于对嵌入式设备的模糊测试。

最后，当前没有针对 IOS-XE 系统 Web 管理服务的模糊测试框架，现有方法没有结合 IOS-XE 命令注入漏洞的特征，无法检测出 Read-Only 权限提升到管理员权限这一类型的漏洞。

3　方法设计

本文提出了一个针对 IOS-XE 系统 Web 管理服务的模糊测试框架 CRFuzzer，用于检测命令注入漏洞。该框架的系统流程图如图 2 所示，主要包括种子生成、测试用例生成、模拟请求和异常监测与测试环境恢复 4 个模块。

(1)种子生成(Seed Generator)：为模糊测试生成初始输入。CRFuzzer 分别从前端 Web 请求和后端 Lua 程序中提取 API 信息，并结合命令注入漏洞特征发现可能存在漏洞的脆弱代码段，最终生成模糊测试的初始种子。

(2)测试用例生成(Mutator)：对种子进行参数属性标记，并应用变异规则，生成测试用例。CRFuzzer 分别为 IOS-XE 系统中存在的两种命令注入漏洞设计变异规则与优化策略，对请求中的可变参数进行变异。

(3)模拟请求(Request Emulator)：模拟 Web UI 向路由设备发起请求，并接收响应。CRFuzzer 模拟 IOS-XE 系统 Web UI 的登录操作、GET 请求和 POST 请求。其中，GET 请求在登录后可以直接发起，而 POST 请求需要先获取 CSRF-Token。

(4)异常监测与测试环境恢复(Monitor)：监测路由设备状态，并在每次测试后还原测试环境。CRFuzzer 利用 IOS-XE 系统 Web 服务的 CLI Command API 查看或修改路由器配置信息，以判断命令注入是否成功，并将设备配置恢复到初始状态。

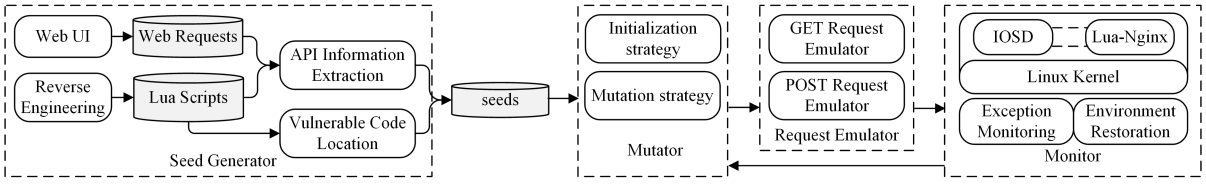


图 2　CRFuzzer 系统结构

Fig. 2　System architecture of CRFuzzer

3.1　种子生成

3.1.1　API 信息提取

在生成初始种子时，从前端请求和后端程序提取 API 信息各具优缺点。

(1)从前端请求中可以提取到数据格式较为准确的 API 信息及各个参数的属性。但是，考虑到隐藏 API 的存在，仅从前端请求中生成初始种子，容易造成漏报。

(2)从后端 Lua 程序中可以提取到更为完整的 API 信息，但是从源码中还原 API 参数格式较为困难，目前也没有公开的工具。此外，除常量字符串以外，较难直接识别出各个参数的属性。

CRFuzzer 将两种方式相结合，分别从前端请求和后端

Lua 程序中生成初始种子，优化种子生成策略。CRFuzzer 从后端程序中提取出较为完整的 API 信息，借助从前端捕获的请求尽可能地进行参数属性标记，以便于后续对参数进行变异。

(1)从前端请求提取 API

与一些现有的 IoT 模糊测试方法相同，CRFuzzer 从前端请求中提取 API 信息。在提取过程中，CRFuzzer 解析并存储请求的头部和参数信息。头部中的 URL 及其结构信息(如 Content-Type、Cookie 结构、X-Csrf-Token 等)将用于请求模拟器的构建，而参数则是命令注入漏洞触发的必要条件，将用于后续的变异操作。

IOS-XE 的 Web 服务中也存在 CONF-READ 通信模式[7]。如，在 User Administration 功能中，Add 或 Delete 均以 POST 方法发起请求，执行增加或删除用户的操作，为 CONF 请求。而在执行完配置修改后，会以 GET 方式发起请求，获取当前所有用户信息，为 READ 请求。但 CRFuzzer 针对的两种命令注入漏洞，其触发只需要单个请求，不涉及多个请求形成的序列操作，因此 CRFuzzer 对不含参数的请求不做考虑。

(2)从后端 Lua 程序提取 API

Lua-Nginx 集成了标准 Nginx 内核、Lua 解释器(例如 LuaJIT 或标准 Lua)，以及各种编写良好的 Lua 库，被应用于很多主流嵌入式设备[21]。IOS-XE 的 Web 服务端也采用 Lua-Nginx 模式，请求数据的解析和命令构建是由 Lua 程序完成的。Lua 程序接收到请求数据，在用户权限校验通过后，获取请求中的参数信息，然后调用相关函数进行命令构建。参数解析和命令构建示例程序如图 3 左半部分所示。

```
1. if method == "POST" then
2.     local returnValue = {}
3.     local dataTable = cjson.decode(ngx.var.request_body)
4.     local selectedPort = dataTable["selectedIFName"]
5.     local configCDP = dataTable["configCDP"]
6.     local opType = dataTable["getCDP"]
7.     if opType.type == "getCDP" then
8.         getCDPDetails()
9.     end
10.    if configCDP.enableCDP == true then
11.        ......
12.    else if selectedPort.selectedName ~= "" then
13.        ......
14.    end
15. end
```

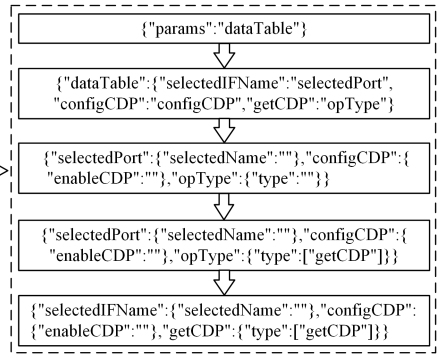


图 3 从后端提取参数示例

Fig. 3 Example of parameter extraction from back-end

程序中 dataTable 为前端请求中传递的参数，数据格式为 Lua 语言中的 table，CRFuzzer 根据函数对 dataTable 中数据的引用，来识别请求中的具体参数，生成相应的初始种子。其中，参数可能存在多层嵌套，从前端收集的种子中可以观察到，嵌套关系最多只有两层，因此 CRFuzzer 仅考虑至多两层嵌套关系的参数形式。参数识别的流程如下：

(1)识别程序对请求参数的解析，获得初始参数名称；

(2)根据对参数元素的访问方式，识别第一层参数，记录参数名与对应的变量名称；

(3)根据第二步中提取到的变量名称，以相同的方法识别第二层参数；

(4)识别程序中的固定字符串；

(5)根据步骤(2)－步骤(4)的记录，生成 API 信息。

从后端提取 API 信息的示例如图 3 所示，其中，根据参数名称识别参数所含元素的方法如算法 1 所示。

算法 1 参数元素识别

输入：参数名称列表 paramList，Lua 程序 luaFile

输出：参数所含元素及其对应的变量名，形式为字典 paramDict

```
1. paramDict←null
2. for key in paramList do
3.  paramDict[key]←null
4.  for line in luaFile do
5.    paramName←match_param_by_reference(key,line)
6.    nickName←get_param_nickname(paramName,line)
7.    if isNull(nickName)then
8.      paramDict[key][paramName]←null
9.    else
10.       paramDict[key][paramName]←nickName
11.   end if
12.  end for
13. end for
14. return paramDict
```

在 API 信息的提取过程中，部分参数的值是常量字符串，这类参数如果随意赋值，将生成无效测试用例。此外，常量字符串通常用于分支条件判断，常量字符串的不同取值对应不同的执行路径。因此，CRFuzzer 为常量字符串的每个取值生成一个初始种子，使其尽可能覆盖更多的代码执行路径。

3.1.2 脆弱代码发现

现有针对嵌入式设备的模糊测试技术偏向于黑盒测试，对所提取到的 API 接口进行“地毯式”测试，漏洞检测效率不高。CRFuzzer 基于两种命令注入漏洞的特征，根据给定的规则对后端程序进行脆弱代码发现，缩小模糊测试的目标范围。

IOS-XE 的 Web 管理服务中，后端由 Lua 程序对请求数据进行解析，构建指定的命令并执行。如果此命令中包含用户输入的参数，而 Lua 程序对该参数缺乏校验或校验方式错误，则攻击者便可以通过在请求的参数中注入恶意命令，来实现计划外的执行行为。因此，命令注入漏洞具备两个基本特征：1)相应的请求含有用户输入的参数；2)用户输入的参数被用于命令构建并被调用执行。

图 4 给出了底层系统型脆弱代码，在底层系统型命令注入漏洞中，构建的命令将在底层 Linux 系统中被 execute 函数调用执行。通过“;”“&”等 shell 命令连接符号，在用户输入

的参数中附加构造的命令，可以实现在底层 Linux 系统中以 root 权限执行任意命令。

```
1.if uri == "/webui/rest/underlyingOS" and method == "POST" then
2.    if (utils.isPostAuthorized()) then
3.        local data = cjson.decode(ngx.var.request_body)
4.        local testdata = data["testdata"]
5.        local filename = data["filename"]
6.        local command = 'echo "' .. testdata .. '" > /tmp/' .. filename
7.        os.execute(command)
8.    …
9.end
```

图 4　底层系统型脆弱代码示例

Fig. 4　Example of underlying OS vulnerable code

而如果构建的命令为 IOS-XE 的 CLI 类型，则调用 WSMAGetData 或 WSMAApplyCmd 函数将命令传递到 IOSd 执行，以获取原始的响应结果。其中，WSMAGetData 将在特权模式（Privileged EXEC Mode）下执行命令，而 WSMAApplyCmd 将在全局配置模式（Global Configuration Mode）下执行命令，两者都需要具备管理员权限。如图 5 所示，如果后端程序对用户输入的参数缺乏校验或校验方式错误，攻击者便可以将恶意命令附加在参数中，获取路由设备敏感数据信息，甚至篡改路由器配置。

```
1.if uri == "/webui/rest/userPriv" and method == "GET" then
2.    if (utils.isGetAuthorized()) then
3.        local interfaceName = utils.getRequestParameters()
4.        local response = {}
5.        response.interfaceName = interfaceName
6.        local ipsecDetails = xml.eval(wsmaModule.WSMAGetDat(user,
                                "show ip interface " .. interfaceName, ""))
7.    ...
8.end
```

图 5　用户权限型脆弱代码示例

Fig. 5　Example of user privilege vulnerable code

在命令构建前，baseHandler 会对用户权限进行初步校验，其中 GET 请求允许所有用户发起，而 POST 请求仅允许管理员用户执行。在已经具备管理员权限的条件下，进行命令注入以获取敏感数据信息或修改路由设备配置，这种注入并不涉及权限逃逸，不具备实际意义。因此，用户权限型命令注入漏洞的请求只能通过 GET 方法发起。

基于以上两种命令注入漏洞的基本特征，对后端 Lua 程序进行筛选，获取可能存在安全漏洞的脆弱代码段，将其作为 CRFuzzer 后续测试的目标。

3.2　测试用例生成

3.2.1　种子初始化策略

由于嵌入式设备对用户输入的参数都有严格的格式要求，在对种子中的参数进行变异时，如果采用随机变异，生成的大部分测试用例在服务端参数格式校验时便会出错，导致服务端直接拒绝该请求。例如，如果将一个字符串传递给整型的参数，或是对一个常量字符串参数进行变异，都将生成无效测试用例。

为了减少无效测试用例的生成，在应用变异策略之前，需要先标记种子的参数属性。在进行种子变异时，将不再考虑常量字符串的变异，而其他参数将根据其标记的属性进行相关变异，不进行跨属性变异操作。CRFuzzer 基于 3.1 节中发现的脆弱代码对应的 API 信息，从种子集合中挑选出相应的初始种子，并对这些种子进行请求测试，标记参数属性。

对于基于前端请求生成的初始种子，其参数格式比较准确，参数属性的识别与标记相对简单。首先识别文本类型，如字符串、数字等，然后，对其中标记为字符串的参数进行数次差异较大的变异，根据反馈结果判断其是否为常量字符串。

对于基于后端程序分析生成的初始种子，由于程序分析过程中较难自动化标记参数属性，因此生成时初始标记为空字符。CRFuzzer 首先依据参数名称进行常理化推断，例如参数 ipaddress 的属性将会被标记为 IP 地址，然后依次对剩下的空字符进行属性变异以生成新的种子，变异的属性主要包括可变字符串、数字等，并通过模拟客户端生成请求数据发送到路由设备，根据反馈的结果信息推断参数的实际属性。

3.2.2　种子变异规则

（1）底层系统型命令注入漏洞

IOS-XE 的文件系统中，bootflash 目录存储于底层 Linux 的根目录下，可以在 Linux 中访问 bootflash 中的文件。为方便后续监测注入的命令是否成功执行，CRFuzzer 选择在种子的可变字符串后附加一条命令，效果为在 bootflash 中创建文件。IOS-XE 可能会为参数附加特殊字符（如引号、转义符等），使其不可执行，因此 CRFuzzer 在变异时需要考虑对特殊字符的补齐，使注入的命令可以成功执行。

（2）用户权限型命令注入漏洞

CRFuzzer 致力于发现可以实现用户权限逃逸的命令注入漏洞，使 Read-Only 权限的用户能够以管理员权限执行命令，获取敏感数据或修改设备配置。为此，本文针对用户权限型命令注入漏洞设计了两种变异规则。

1）查看配置。这种注入方式较为简单，尝试获取管理员的用户名和密码。但由于后端的 Lua 程序可能会对原始响应进行过滤，使得注入的命令即便成功执行，最终客户端收到的响应数据中也没有路由设备的配置信息。

2）修改配置。注入的命令为添加管理员用户。这种命令注入方式涉及特权模式和全局配置模式的切换，需要附加 conf t 命令，适用的固件版本具有局限性。

（3）变异优化

通过种子变异以生成测试用例时，只需要保证 Lua 程序能够执行到相应的分支，而不要求原始命令能够正确执行。因此，CRFuzzer 尽可能使前缀命令执行错误，而直接执行所注入的命令，一方面可以减少对路由设备配置的修改，便于恢复测试环境；另一方面，可以节约测试用例的执行时间，例如涉及 install 命令时，CRFuzzer 在变异过程中会将 package 的路径设置为错误路径，避免进行 package 的传输与安装。

3.3　异常监测与测试环境恢复

3.3.1　异常监测

（1）底层系统型命令注入漏洞

IOS-XE 的 Web UI 以管理员用户登录时，可以执行配置查看命令。在生成的测试用例中，注入的命令为在 bootflash 中创建文件。因此，判断命令是否注入成功，只需要以管理员用户登录，向 CLI Command API 发送一个查看 bootflash 存储内容的命令，检查返回结果中是否含有所创建的文件名。

(2)用户权限型命令注入漏洞

根据用户权限型命令注入漏洞相应的变异规则,CRFuzzer分别对响应数据和设备配置进行监测。若测试用例的响应数据中包含预先配置的管理员用户及其密码,则表明成功获取了敏感数据。以管理员用户登录,向CLI Command API发送查看路由器配置的命令,若返回结果包含所添加的管理员用户和密码,则表明修改设备配置成功。

3.3.2 测试环境恢复

为了证明测试用例中注入的命令被成功执行,需要在每次发起请求之前,保证路由器的各项配置一致。本文根据命令注入漏洞的类型,分别设计了轻量级的测试环境恢复方案。

(1)底层系统型命令注入漏洞

IOS-XE的Web UI以管理员用户登录时,可以执行修改设备配置的命令。在测试用例中,注入的命令为在bootflash中创建文件,并不会触发设备崩溃或重启,而通过对变异规则的优化,前缀命令无法成功执行,因此不会改变路由器的配置。因此,只需要利用CLI Command API删除bootflash中所创建的文件。

(2)用户权限型命令注入漏洞

此类漏洞测试时也不会触发设备崩溃或重启,且GET请求多为查看信息的命令,不会改变设备的配置。因此,CRFuzzer仅考虑注入的命令对设备的影响。若注入的命令为查看敏感数据,则不需要进行测试环境恢复;若注入的命令为添加管理员用户,则利用CLI Command API删除所添加的用户。

4 实验评估

本节从API信息提取与脆弱代码发现、对命令注入漏洞的检测能力以及检测效率方面评估CRFuzzer,并与现有模糊测试方法SRFuzzer进行对比。此外,由于SRFuzzer在命令注入漏洞方面仅检测对Linux系统的注入,本文为SRFuzzer加入针对用户权限型命令注入漏洞的变异规则,并增加对特殊字符的补齐策略,形成SRFuzzer+用于对比实验。

4.1 实验设置

实验所用设备为ISR 4000系列实体路由器(包括ISR 4221,ISR 4321和ISR 4331),以及云虚拟路由器CSR 1000v,4个设备的固件均使用16.5.x—16.11.x这31个版本,共计124个不同的固件。预先在路由器中配置用户名与密码的用户,均为admin的管理员用户以及用户名与密码均为guest、权限为Read-Only的用户。

4.2 实验结果

4.2.1 API信息提取与脆弱代码发现

本实验以16.5.1,16.6.1,16.7.1,16.8.1,16.9.1,16.10.1和16.11.1这7个版本的固件为例,分别进行前后端API信息提取和基于漏洞特征的脆弱代码发现。实验中仅考虑含有参数的API,且将一条API信息定义为一个URL和一种请求方式组成的二元组,形如〈URL,method〉。实验结果如图6所示。从图6可以看出,前后端提取出的API信息存在数量上的差异,后端多于前端。这表明在IOS-XE系统的Web管理服务中存在隐藏API。查看前后端提取出的API信息可以发现,前端所缺少的API主要分为如下两种:1)前端没有相应的功能接口,如Threat Defense,VPN,Application Visibility和Custom Application等,测试的固件不支持相应的功能;2)前端无法直接使用相应的功能接口,如Python Sandbox和troubleshooting audit,前者需要在路由器上预先配置Guestshell,而后者需要思科公司账号在线认证。但其后端同样存在相应的请求处理程序,可以通过模拟发送请求的方式,调用对应的API并执行相关代码。

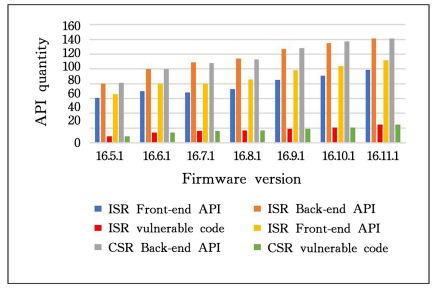


图6 API信息与脆弱代码数量

Fig. 6 Quantity of API information and vulnerable code

如图6所示,从ISR 4000和CSR 1000v后端程序提取出的API信息基本一致,而从ISR 4000和CSR 1000v前端提取出的API信息,其数量少于后端API。这是由于跨平台的IOS-XE系统间存在大量的代码复用,ISR 4000系列的路由设备不支持Threat Defense,VPN等功能,Application Visibility和Custom Application功能只存在于部分版本的固件中。这导致ISR 4000的前端缺少上述两种隐藏API,而CSR 1000v前端则主要缺少第二种隐藏API。CRFuzzer分别从前端请求和后端程序中提取API信息,实验结果表明,CRFuzzer可以提取到更为完整的API信息,有助于生成更完整的初始种子库,覆盖更多的程序执行路径。

此外,本实验基于3.1节中底层系统型和用户权限型两种命令注入漏洞的基本特征,进行脆弱代码发现,筛选出可能存在漏洞的API。如图6所示,筛选出的脆弱代码对应的API数量远少于后端提取出的API总数。IOS-XE系统的Web UI是用户查看信息和修改配置的接口,前者主要通过GET请求,允许所有权限的用户发起,而后者则主要为POST方式,仅允许管理员用户执行。由于在参数解析和命令构建之前,会先对用户权限进行校验,因此请求方法为POST的API,不存在用户权限型命令注入漏洞。此外,仅有少量API需要在底层Linux中执行构建的命令,即仅有少量API可能存在底层系统型命令注入漏洞。因此,3.1节中提出的漏洞特征相对较弱,其主要目的是在尽可能减少漏报的前提下,过滤掉部分不存在命令注入漏洞的API。实验结果表明,尽管此方法无法准确定位漏洞位置,但可以大幅缩小模糊测试的目标范围。

4.2.2 漏洞检测能力

为了验证CRFuzzer的漏洞检测能力,本文选取SRFuzzer和添加变异规则后形成的SRFuzzer+进行对比实验。实验在ISR 4000和CSR 1000v路由器上分别对16.5.1—16.11.1这31个版本的固件进行命令注入漏洞检测。

实验结果如表1所列,CRFuzzer共检测出11个命令注入漏洞,其中5个为底层系统型,可实现以root权限在底层

Linux 系统中执行任意命令；6 个为用户权限型，可实现以权限为 Read-Only 的用户登录，执行需要管理员权限的命令，获取敏感数据或修改路由器配置。这 11 个漏洞中 9 个 CVE 漏洞为已知漏洞，思科官方披露了部分漏洞信息和临界版本（首次修复或首个不受影响的版本），但未公开漏洞位置和漏洞触发方式；2 个 CNVD 漏洞为 CRFuzzer 发现的未公开漏洞。

表 1　漏洞检测效果对比

Table 1　Comparison of vulnerabilities detection results

ID	Type	SRFuzzer		SRFuzzer+		CRFuzzer	
		ISR 4000	CSR 1000v	ISR 4000	CSR 1000v	ISR 4000	CSR 1000v
CVE-2021-1435	Underlying OS	×	×	×	×	√	√
CVE-2020-3211	Underlying OS	×	×	√	√	√	√
CVE-2019-1862	Underlying OS	×	×	√	√	√	√
CVE-2019-12650	Underlying OS	×	×	×	×	√	√
CVE-2020-3224	User Privilege	×	×	√	√	√	√
CVE-2019-1753	User Privilege	×	×	×	√	√	√
CVE-2019-12651	User Privilege	×	×	√	√	√	√
CVE-2019-1754	User Privilege	×	×	√	√	√	√
CVE-2019-1755	User Privilege	×	×	×	×	√	√
CNVD-2022-03243	Underlying OS	√	√	√	√	√	√
CNVD-2022-03642	User Privilege	×	×	×	×	√	√
Total	11	1	1	6	7	11	11

SRFuzzer 不具备对用户权限型命令注入漏洞的检测能力，因为 Linux 与 IOSd 使用的是两套不同的命令。若不增加特殊字符补齐策略，5 个底层系统型命令注入漏洞中，SRFuzzer 仅能检测出 1 个。SRFuzzer+增加了针对用户权限型命令注入漏洞的变异规则，及对特殊字符的补齐策略，但仍然仅从前端请求中提取 API 信息用于生成初始种子，因此只能检测出非隐藏 API 相关的安全漏洞。SRFuzzer+在 CSR 1000v 上比在 ISR 4000 上表现更好，原因是 CSR 1000v 功能更加齐全，具有一些 ISR 4000 不支持的功能接口，可以从前端请求中提取出更多 API 信息。CRFuzzer 结合前端请求和后端程序分析，可以提取出更为完整的 API 信息，具有更好的漏洞检测能力。实验结果表明，CRFuzzer 可以有效检测出 IOS-XE 系统 Web 管理服务中的底层系统型和用户权限型两种命令注入漏洞。

4.2.3　漏洞检测效率

为了测试 CRFuzzer 的漏洞检测效率，本文记录了 SRFuzzer+和 CRFuzzer 在 ISR 4000 与 CSR 1000v 每个固件上进行测试所消耗的时间。SRFuzzer 通过“;”“&”或“‖”等 shell 字符在可变参数中注入在底层 Linux 系统中执行的命令，如果参数被用于构建在 IOSd 中执行的命令，将无法被 CLI 命令解析模块识别，直接返回 Error。由于 SRFuzzer 不会有命令执行的时间开销，因此本实验不再对比 SRFuzzer 的测试时间。以 16.5.1，16.6.1，16.7.1，16.8.1，16.9.1，16.10.1 和 16.11.1 这 7 个版本的固件为例，测试时间如图 7 所示。

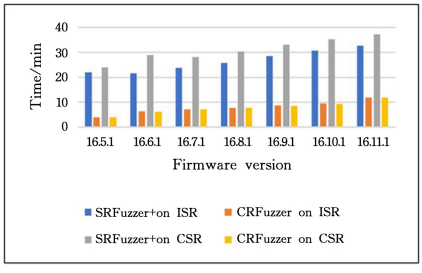


图 7　测试时间对比

Fig. 7　Comparison of testing time

CRFuzzer 在 ISR 4000 和 CSR 1000v 中提取出的 API 信息，及基于命令注入漏洞的特征筛选出的脆弱代码对应的 API 数量基本一致，在两种类型的路由设备上的测试时间也基本相同。而 SRFuzzer+从 CSR 1000v 的前端请求中能提取出更多 API 信息，因此在 CSR 1000v 上的测试时间长于 ISR 4000。

从图 7 可以看出，对于同一版本的固件，CRFuzzer 所消耗的时间短于 SRFuzzer+。CRFuzzer 结合前端请求和后端程序，提取出了更多的 API 信息，生成了更多初始种子。但 CRFuzzer 基于底层系统型和用户权限型两种命令注入漏洞的基本特征，进行脆弱代码筛选，仅对可能存在漏洞的 API 进行测试。如图 6 所示，筛选出的脆弱代码对应的 API 数量远少于 API 总数。实验结果表明，该方法可以大幅缩小模糊测试的目标范围，提高漏洞检测效率。

结束语　本文提出了一种面向 Cisco IOS-XE 系统 Web 管理服务的模糊测试框架 CRFuzzer，用于检测命令注入漏洞。CRFuzzer 通过分析前端请求和后端程序来优化种子生成，能够发现系统中存在的隐藏 API。同时，CRFuzzer 基于两种命令注入漏洞的基本特征进行脆弱代码筛选，提高了模糊测试的效率。实验结果表明，相比 SRFuzzer，CRFuzzer 在对 IOS-XE 系统 Web UI 的命令注入漏洞检测上具有更好的效果。

在未来的工作中，将以以下两个方面展开研究：

（1）在嵌入式设备的 Web 管理服务中，还存在跨站脚本（Cross-site Scripting，XSS）、缓冲区溢出等类型的安全漏洞。后续研究将结合对这些漏洞特征的分析，完善脆弱代码筛选与种子变异策略，使得 CRFuzzer 能够检测更多类型的安全漏洞。

（2）Cisco IOS-XE 同样部署于 ASR 路由器和 Catalyst 3650/3850 等思科交换机设备，其固件封装格式和 Web 工作机制与 ISR 路由器存在差异，需要进一步研究与分析，以完善种子生成和请求模拟，扩展 CRFuzzer 的适用范围。

参 考 文 献

[1]　Open Web Application Security Project Top Ten [EB/OL]. (2013-10-30) [2021-10-01]. https://owasp.org/www-project-top-ten.

[2] STASINOPOULOS A, NTANTOGIAN C, XENAKIS C. Commix: automating evaluation and exploitation of command injection vulnerabilities in Web applications[J]. International Journal of Information Security, 2019, 18(1): 49-72.

[3] YOGESH R, NAGENDRA K N. Containers in Cisco IOS-XE, IOS-XR, and NX-OS: Orchestration and Operation[M]. Cisco Press, 2021.

[4] MUNIZ S, ORTEGA A. Fuzzing and debugging Cisco IOS[J/OL]. BlackHat Europe, 2011. https://infocon. org/cons/SyScan/SyScan 2011 Singapore/SyScan 2011 Singapore presentations/Syscan2011-CiscoIOS-Aortega-Smuniz. pdf.

[5] LI F, ZHANG L, CHEN D. Vulnerability mining of Cisco router based on fuzzing[C]// The 2014 2nd International Conference on Systems and Informatics(ICSAI 2014). IEEE, 2014: 649-653.

[6] ZHOU J X, FENG D, LI B. A fuzzing method based on dual variation strategy for Cisco IOS[C]// 2017 3rd IEEE International Conference on Computer and Communications (ICCC). IEEE, 2017: 205-209.

[7] ZHANG Y, HUO W, JIAN K, et al. SrFuzzer: An automatic fuzzing framework for physical soho router devices to discover multi-type vulnerabilities [C]// Proceedings of the 35th Annual Computer Security Applications Conference. 2019: 544-556.

[8] CHEN J, DIAO W, ZHAO Q, et al. IoTFuzzer: Discovering Memory Corruptions in IoT Through App-based Fuzzing[C]// NDSS. 2018.

[9] FENG X, SUN R, ZHU X, et al. Snipuzz: Black-box fuzzing of iot firmware via message snippet inference[C]// Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021: 337-350.

[10] ZHENG Y, DAVANIAN A, YIN H, et al. FIRM-AFL: high-throughput greybox fuzzing of iot firmware via augmented process emulation[C]// 28th {USENIX} Security Symposium ({USENIX} Security 19). 2019: 1099-1114.

[11] SRIVASTAVA P, PENG H, LI J, et al. Firmfuzz: Automated iot firmware introspection and analysis [C]// Proceedings of the 2nd International ACM Workshop on Security and Privacy for the Internet-of-Things. 2019: 15-21.

[12] CHEN D D, WOO M, BRUMLEY D, et al. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware[C]// NDSS. 2016, 1: 1. 1-8. 1.

[13] OpenResty—a dynamic web platform based on NGINX and LuaJIT[EB/OL]. (2013-08-26)[2021-12-16]. http://openresty.org/.

[14] BOLLAPRAGADA V, MURPHY C, WHITE R. Inside cisco ios software architecture[M]. Cisco Press, 2000.

[15] WANG Z, ZHANG Y, LIU Q. Rpfuzzer: A framework for discovering router protocols vulnerabilities based on fuzzing[J]. KSII Transactions on Internet and Information Systems(TIIS), 2013, 7(8): 1989-2009.

[16] ZHU L, FU X, 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(TrustCom/BigDataSE). IEEE, 2019: 248-255.

[17] YU L, WANG H, LI L, et al. Towards Automated Detection of Higher-Order Command Injection Vulnerabilities in IoT Devices: Fuzzing With Dynamic Data Flow Analysis[J]. International Journal of Digital Crime and Forensics (IJDCF), 2021, 13(6): 1-14.

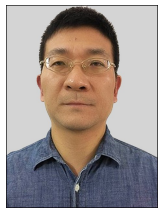
[18] JIANG Y, XIE W, TANG Y. Detecting authentication-bypass flaws in a large scale of IoT embedded web servers[C]// Proceedings of the 8th International Conference on Communication and Network Security. 2018: 56-63.

[19] CHEN L, WANG Y, CAI Q, et al. Sharing More and Checking Less: Leveraging Common Input Keywords to Detect Bugs in Embedded Systems[C]// 30th {USENIX} Security Symposium ({USENIX} Security 21). 2021.

[20] HALLER I, SLOWINSKA A, NEUGSCHWANDTNER M, et al. Dowsing for Overflows: A Guided Fuzzer to Find Buffer Boundary Violations[C]// 22nd {USENIX} Security Symposium({USENIX} Security 13). 2013: 49-64.

[21] COSTIN A. Lua code: security overview and practical approaches to static analysis[C]// 2017 IEEE Security and Privacy Workshops(SPW). IEEE, 2017: 132-142.

HE Jie, born in 1996, master. His main research interests include cyber security and embedded network device.



LIU Shengli, born in 1973, Ph.D, professor. His main research interests include network device security and network attack detection.

(责任编辑:喻藜)