

## 基于语义冲突的硬编码后门检测方法

胡安祥<sup>1</sup>, 肖达<sup>2</sup>, 郭世臣<sup>1</sup>, 刘胜利<sup>1</sup>

(1. 数学工程与先进计算国家重点实验室, 河南 郑州 450001; 2. 郑州工业应用技术学院信息工程学院, 河南 郑州 451100)

**摘要:** 路由器安全问题主要聚焦于内存型漏洞的挖掘与利用, 对后门的检测与发现的研究较少。硬编码后门是较常见的后门之一, 设置简单方便, 仅仅需要少量代码就能实现, 然而却难以被发现, 往往造成严重的危害和损失。硬编码后门的触发过程离不开字符串比较函数, 因此硬编码后门的检测借助于字符串比较函数, 主要分为静态分析方法和符号执行方法。前者自动化程度较高, 但存在较高的误报率, 检测效果不佳; 后者准确率高, 但无法自动化大规模检测固件, 面临着路径爆炸甚至无法约束求解的问题。针对上述问题, 在静态分析的基础上, 结合污点分析的思想, 提出了基于语义冲突的硬编码后门检测方法——Stect。Stect 从常用的字符串比较函数出发, 结合 MIPS 和 ARM 体系结构的特点, 利用函数调用关系、控制流图和分支选择依赖的字符串, 提取出具有相同起点和终点的路径集合, 如果验证成功的路径集合中的字符串具有语义冲突, 则判定路由器固件中存在硬编码后门。为了评估 Stect 对路由器硬编码后门的检测效果, 对收集的 1 074 个设备固件进行了测试, 并与其他后门检测方法进行了对比。实验结果表明, 相比现有的后门检测方法 Costin 和 Stringer, Stect 具有更好的检测效果: 从数据集中成功检测出 8 个固件后门口令, 召回率达到 88.89%。

**关键词:** 路由器固件; 硬编码后门; 字符串比较函数; 语义冲突

**中图分类号:** TP393

**文献标志码:** A

**DOI:** 10.11959/j.issn.2096-109x.20223015

## Hard-coded backdoor detection method based on semantic conflict

HU Anxiang<sup>1</sup>, XIAO Da<sup>2</sup>, GUO Shichen<sup>1</sup>, LIU Shengli<sup>1</sup>

1. State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China

2. School of Information Engineering, Zhengzhou University of Industry Technology, Zhengzhou 451100, China

**Abstract:** The current router security issues focus on the mining and utilization of memory-type vulnerabilities, but there is low interest in detecting backdoors. Hard-coded backdoor is one of the most common backdoors, which is simple and convenient to set up and can be implemented with only a small amount of code. However, it is difficult to be discovered and often causes serious safety hazard and economic loss. The triggering process of hard-coded

收稿日期: 2022-02-22; 修回日期: 2022-05-11

通信作者: 刘胜利, mr.shengliliu@gmail.com

基金项目: 科技委基础加强项目 (2019-JCJQ-ZD-113)

**Foundation Item:** Science & Technology Commission Foundation Strengthening Project (2019-JCJQ-ZD-113)

**引用格式:** 胡安祥, 肖达, 郭世臣, 等. 基于语义冲突的硬编码后门检测方法[J]. 网络与信息安全学报, 2023, 9(1): 150-157.

**Citation Format:** HU A X, XIAO D, GUO S C, et al. Hard-coded backdoor detection method based on semantic conflict[J]. Chinese Journal of Network and Information Security, 2023, 9(1): 150-157.

backdoor is inseparable from string comparison functions. Therefore, the detection of hard-coded backdoors relies on string comparison functions, which are mainly divided into static analysis method and symbolic execution method. The former has a high degree of automation, but has a high false positive rate and poor detection results. The latter has a high accuracy rate, but cannot automate large-scale detection of firmware, and faces the problem of path explosion or even unable to constrain solution. Aiming at the above problems, a hard-coded backdoor detection algorithm based on string text semantic conflict (Stect) was proposed since static analysis and the think of stain analysis. Stect started from the commonly used string comparison functions, combined with the characteristics of MIPS and ARM architectures, and extracted a set of paths with the same start and end nodes using function call relationships, control flow graphs, and branching selection dependent strings. If the strings in the successfully verified set of paths have semantic conflict, it means that there is a hard-coded backdoor in the router firmware. In order to evaluate the detection effect of Stect, 1 074 collected device images were tested and compared with other backdoor detection methods. Experimental results show that Stect has a better detection effect compared with existing backdoor detection methods including Costin and Stringer: 8 hard-coded backdoor images detected from image data set, and the recall rate reached 88.89%.

**Keywords:** router firmware, hard-coded backdoor, string comparison functions, semantic conflict

## 0 引言

路由器是互联网的核心交换设备,作为网络互联的基础,其安全问题一直是网络空间安全领域的研究重点。路由器安全问题主要有两方面:一是路由器自身存在的后门或安全漏洞,被攻击者挖掘和利用;二是路由器缺乏有效安全的配置管理,容易遭到攻击、入侵或植入后门<sup>[1]</sup>。相比其他漏洞挖掘方法,模糊测试有着自动化程度高、误报率低等优势,无须对目标程序的源代码或二进制程序进行分析,已经成为较常用的漏洞挖掘方法<sup>[2]</sup>。然而,模糊测试仍然有其局限性:无法检测出逻辑错误和访问控制漏洞(即后门漏洞)。因此很少有对后门漏洞的检测与分析。

后门是指一种用于秘密绕过软件、计算机系统、密码机制等正常认证流程,获取计算机系统的访问权限或经密码系统加密后的明文的程序方法<sup>[3]</sup>。常见的后门之一是硬编码后门,硬编码后门的来源可以分为两种:一是部分厂商出于对管理的需要在程序中植入的硬编码口令;二是攻击者对固件进行篡改植入的后门口令。硬编码后门设置方便,仅需要少量的代码就可实现,然而不容易被发现,尤其是当使用混乱的字符串时,更难被逆向工程师发现,如在 D-Link 路由器中发现的口令“xmlset\_roodkcableoj28840ybtide”<sup>[4]</sup>。不论硬编码后门的来源是什么,攻击者都可以利用

后门口令控制设备、劫持流量或传播病毒。

近年来,对硬编码后门的检测方法主要有静态分析和符号执行两种方法。忽朝俭等<sup>[5]</sup>提出了一种基于嵌入式固件的库函数识别方法,用于无文件系统固件的后门检测。Costin 等<sup>[6]</sup>提出大规模自动化分析嵌入式固件,使用模糊哈希的方式匹配固件中可能存在的弱密钥,通过关联分析的方式在 4 种不同维度查找固件的相似性,成功在 693 个固件中检测出 38 个未知漏洞。Thomas 等<sup>[7]</sup>提出基于静态数据比较分析的方法 Stringer 检测商用设备固件中的硬编码后门。文献[8]使用静态程序分析生成固件的程序依赖图,获取一个从入口点到特权程序点的程序路径,再通过符号执行判断路径中是否具有确定性的约束,如果存在,就认定其为后门。当前的静态分析方法自动化程度较高,但有着误报率高、检测效果不佳的问题;符号执行方法虽然准确率高,但无法大规模自动化检测固件,存在路径爆炸、耗时长的缺陷。

硬编码后门往往出现在登录验证流程中。从登录开始到验证结束可能经过不同的分支路径,但校验成功的路径的字符串语义是相同的。在有些路由器上,当用户权限从 level 1 提升至 level 15 时,有两种不同的登录校验方式:需要输入{用户名,密码};只需要输入{密码}。虽然两种校验方式的分支路径不同,然而对固件逆向分析后发现:前者通过的字符串为{“username:”,“password:”},

后者通过的字符串为{“user=default”, “secret:.”}, 两条不同的路径经过的校验字符串却有着相同的语义。然而使用后门口令校验时, 往往会有着截然不同的字符串: 文献[4]中口令为“xmlset\_roodkcableoj28840y btide”, 文献[9]中口令为{“zyfw”, “PrOw!aN\_ fXp”}。因此, 可以通过字符串语义冲突来约束检测结果, 降低误报率。

为解决硬编码后门检测问题, 本文在文献[7]的基础上, 结合污点分析的思想, 将函数控制流中不同的路径根据返回值划分为校验成功和校验失败两个路径集合, 只对校验成功的路径集合进行语义冲突分析, 提出了基于语义冲突的硬编码后门检测方法——Stect。Stect 从常用的字符串比较函数出发, 结合 MIPS 和 ARM 指令集的特点, 利用函数的调用关系、控制流图 (CFG, control flow graph) 和分支选择依赖的字符串, 识别固件中的硬编码后门。所提方法既可以自动化大规模检测固件, 又降低了误报率。本文的主要贡献如下。

1) 提出了一种新的硬编码后门检测方法——Stect。Stect 实现流程主要分为 3 个阶段: 固件预处理、函数 CFG 转为有向拓扑图和查找语义冲突的路径。

2) 只检测验证成功的分支路径是否有语义冲突, 保留语义冲突的路径, 以提高 Stect 的精度, 减少误报率。

3) 使用网络上收集的 1 074 个路由器固件对其进行评估。从 9 个后门固件中成功识别出 8 个后门口令, 召回率达到 88.89%, 在路由器的硬编码后门检测方面优于现有的 Stringer 和 Costin 后门检测方法。

## 1 研究背景和相关工作

Muench<sup>[10]</sup>按操作系统类型将嵌入式设备分为 3 类: 基于通用操作系统的设备; 基于嵌入式操作系统的设备; 不具备操作系统的设备。大部分路由器使用基于 Linux 改进的系统, 属于第一类设备, 有着较为完整的文件系统, 代码量级大。Chen 等<sup>[11]</sup>利用网络“爬虫”收集了 23 035 个固件镜像, 对其体系结构进行了统计, MIPS 和 ARM 体系结构总共约占 91.9%, 如图 1 所示。

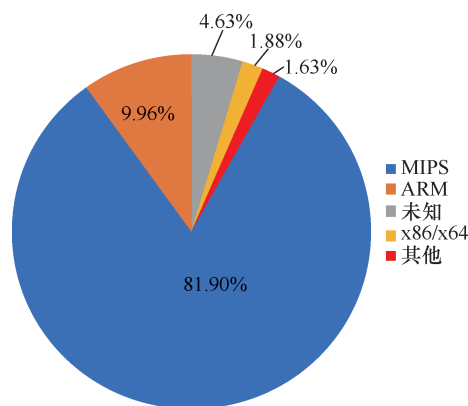


图1 路由器体系结构统计  
Figure 1 Router architecture statistics

不同的体系结构有着不同的指令集和通用寄存器, 因此每个体系结构都需要相应的处理流程, 才能将 CFG 转为有向拓扑图。本文方法的检测对象为 MIPS 和 ARM 体系结构的路由器固件。

### 1.1 MIPS 和 ARM 体系结构

在函数 CFG 转为有向拓扑图之前, 需要先熟悉 MIPS 和 ARM 体系结构和指令集的特点。

MIPS 和 ARM 体系结构都是采取精简指令集的处理架构, 使用固定长度的指令集。无论是 MIPS 还是 ARM, 都不允许 CPU 直接访问内存单元, 只能使用寄存器作为中继, 内存和寄存器通过 load/store 指令交互。

MIPS 体系结构有 32 个通用寄存器, 在汇编程序中既可以用 \$0~\$31 表示, 也可以用寄存器的名字表示。

ARM 体系结构中的寄存器数量取决于 ARM 版本。除了基于 ARMv6-M 和 ARMv7-M 的处理器外, 一般有 30 个通用寄存器, 其中包括 SP 和 LR 寄存器。ARM 处理器在用户模式下有 16 个通用寄存器, 在汇编程序中用寄存器的名字表示。

调用函数时, MIPS 使用寄存器 \$a0~\$a3 传递参数, 返回值存放在寄存器 \$v0~\$v1 中; 而 ARM 使用寄存器 R0~R2 传递参数, 返回值存放在寄存器 R0 中。虽然两者都使用五级流水线技术来提高运算性能, 但 MIPS 还额外引入分支延迟槽技术进一步提高程序运行速度。

### 1.2 硬编码后门特征

硬编码后门往往出现在登录验证流程中, 其原理如下: 在系统正常登录流程中, 使用硬编码密钥就能通过分支旁路绕过登录验证, 从而登录

系统或者得到系统控制权限。因此后门口令所在函数具有以下两个特征。

### (1) 存在字符串比较函数

无论是正常登录流程，还是后门口令执行流程，为保证只有管理员或信任用户才能登录系统，字符串比较函数（简称  $f_{\text{checkStr}}$  函数）都不可或缺。登录验证流程如图 2 所示。在正常登录流程中， $f_{\text{checkStr}}$  函数保证只有管理员才能登录系统；而在硬编码后门执行流程中， $f_{\text{checkStr}}$  函数确保只有攻击者才能控制系统。

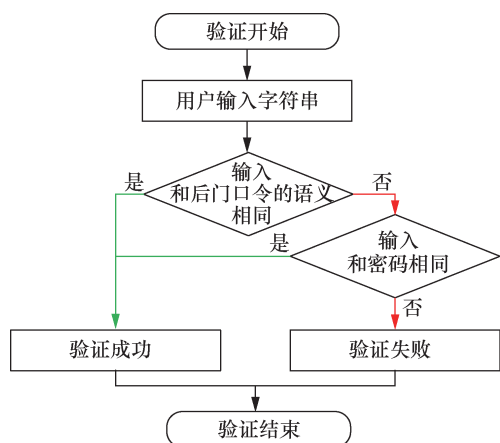


图 2 登录验证流程  
Figure 2 Login verification process

### (2) 验证成功的路径校验字符串语义相同

将函数的 CFG 转为有向图，视基本块（block）为顶点，视块与块之间的跳转关系为有向边。当不同的分支路径交汇于同一终点时，验证失败的路径的校验字符串语义各有不同，但校验成功的分支校验字符串语义相同。如果后者的校验字符串语义有冲突，那么很可能有硬编码后门。如图 3 所示（图中绿线代表验证成功，红线代表验证失败），顶点 7 为验证失败的终点，顶点 8 为验证成功的终点。

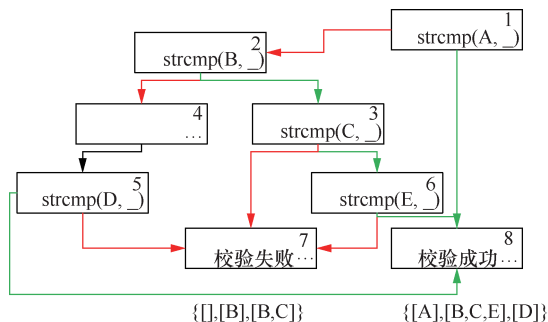


图 3 简化的函数控制流程图  
Figure 3 Simplified function control flow graph

## 2 硬编码后门检测方法 Stect

Stect 后门检测分为以下 5 步：① 固件预处理；② 根据  $f_{\text{checkStr}}$  函数定位可疑函数；③ 可疑函数 CFG 转为有向拓扑图；④ 在有向图中查找验证成功的分支路径；⑤ 使用语义冲突约束求解，识别出后门口令。具体流程如图 4 所示。

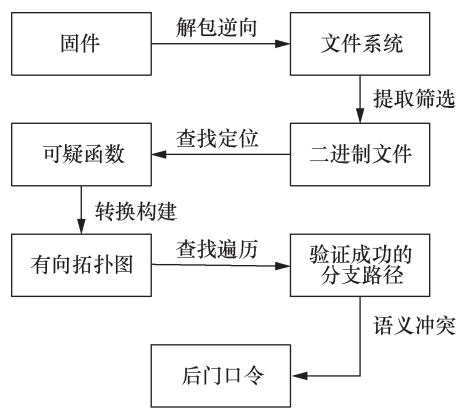


图 4 Stect 方法流程  
Figure 4 Flow chart of Stect

### 2.1 固件解包逆向

路由器固件作为商业发行的软件，往往不会公开源代码或开发文档。因此在检测路由器硬编码后门时，需要先对固件解包分析，再对二进制程序逆向分析。常用的解包工具有 binwalk<sup>[12]</sup>、Firm-Mod-Kit、BAT<sup>[13]</sup>等，其中 binwalk 是较常用的固件解包工具，操作简单，不仅支持自动化脚本，还可以自定义签名、提取规则和集成插件模块；逆向分析工具包括 IDA Pro<sup>[14]</sup>、Ghidra、angr<sup>[15]</sup>等。其中，IDA Pro 是交互式的反汇编器，不仅可以产生二进制文件的汇编代码，而且具有识别函数基本块、获取函数交叉引用、描述函数流图等功能。

### 2.2 语义冲突

路由器的操作系统几乎是基于 Linux 系统改进的，Linux 内核系统主要由 C 语言和汇编语言编程实现，其中 C 语言占比 98.4%。C 语言中常用的  $f_{\text{checkStr}}$  函数有 strstr()、stristr()、strcmp()、strncmp()、strcasecmp()、strncasecmp()、wcscmp() 和 memcmp()，参数形如 (char \*s1, char \*s2 [, size\_t n])，当返回值为 0 时，表明字符串 s1 等于 s2。

函数 CFG 转为有向拓扑图时，视 block 为顶点，视块与块之间的跳转关系为有向边。当调用

$f_{\text{checkStr}}$  函数时, 会产生两个分支: 一个分支表示验证成功, 另一个分支表示验证失败。分支选择取决于输入的字符串  $s1$  和硬编码字符串  $s2$  是否相等, 字符串  $s2$  被称为  $f_{\text{checkStr}}$  函数的校验字符串。在有向图中找到这样的路径集合: 集中路径有相同的起点和终点。对于验证成功的终点, 如果存在一条经过后门口令的路径, 那么这条路径的校验字符串与其他路径的校验字符串语义有冲突。

为筛选出有意义的字符串, 如果字符串中含有  $\{\text{'\%s'}, \text{'\%d'}, \text{'\%t'}, \text{'\%v'}, \text{'\%a'}, \text{'\%b'}, \text{'\%f'}\}$  这一类无法输入的字符或者字符串长度小于 4 时, 视其为空字符串; 为快速判断字符串是否有语义冲突, 视莱文斯坦比是 0.75 的字符串为语义冲突阈值, 低于 0.75 的字符串则被视为有语义冲突。

### 2.3 Stect 方法

Stect 方法用于检测固件中的后门口令。输入为从网络中收集的路由器固件数据集, 输出为语义冲突的路径集合。Stect 方法具体实现如下。

#### 步骤 1 数据预处理

从网络中收集到设备固件后, 使用 binwalk 固件分析工具解包固件, 提取文件系统的二进制文件。

#### 步骤 2 定位可疑函数

使用 IDA Pro 反汇编工具对二进制文件进行逆向分析, 检测文件中是否存在  $f_{\text{checkStr}}$  函数, 如果存在, 则找到交叉引用  $f_{\text{checkStr}}$  函数的地址, 根据地址找到调用  $f_{\text{checkStr}}$  函数的函数 (简称可疑函数); 如果不存在, 则检测下个文件。

#### 步骤 3 构建有向拓扑图

根据步骤 2 中定位的可疑函数 CFG, 将 block 视为顶点  $node$ , 块与块之间的跳转关系视为有向边  $edge$ , 转换构建为有向图  $G$ , 定义为  $G = \{\text{nodes}, \text{edges}\}$ ,  $\text{node} = \{\text{block.startEA}, \text{flag}\}$ ,  $\text{edge} = \{\text{node}_{\text{curr}}, \text{node}_{\text{next}}, \text{str}\}$ 。

对于没有调用  $f_{\text{checkStr}}$  函数的 block, 顶点  $node$  的 flag 等于 0; 对于调用  $f_{\text{checkStr}}$  函数的 block, 顶点  $node$  的 flag 等于 1。在  $\text{node}_{\text{curr}}$  中, 结合函数调用使用的寄存器找到校验字符串  $\text{str}$ , 通过跳转指令映射  $\text{str}$  和  $\text{node}_{\text{next}}$ , 生成  $\text{edge}$ 。具体算法实现如下。

#### 算法 1 函数 $f$ 转有向图 $G$

输入 函数  $f$  和调用  $f_{\text{checkStr}}$  地址集  $\text{refsList}$

输出 有向图  $G$

```

1)  $\text{idx} \leftarrow 0, n \leftarrow \text{refsList.length}$ 
2)  $\text{nodes} \leftarrow [], \text{edges} \leftarrow []$  //保存有向图  $G$  的顶点和有向边
3) foreach block in  $\text{ida.FlowChart}(f)$  do //对每个基本块进行分析处理
4)    $\text{str} \leftarrow \text{null}, \text{nextEA} \leftarrow 0$ 
5)    $\text{start} \leftarrow \text{block.startEA}$  //基本块起始地址
6)    $\text{end} \leftarrow \text{block.endEA}$  //基本块结束地址
7)   if  $\text{idx} < n$  and  $\text{refsList}[\text{idx}] < \text{end}$  then
8)      $\text{str}, \text{nextEA} \leftarrow \text{getFrom}(\text{block})$  //从基本块中获取字符串和对应的跳转地址
9)      $\text{idx} \leftarrow \text{idx} + 1$ 
10)  end if
11) if  $\text{str} \neq \text{null}$  then //添加顶点, 如果顶点中字符串不为空, 则设置 flag 为 1
12)    $\text{nodes.append}([\text{start}, 1])$ 
13) else if  $\text{nodes.append}([\text{start}, 0])$ 
14) end if
15) foreach child in  $\text{block.succs}()$  do //获取基本块的所有跳转地址
16)    $\text{next} \leftarrow \text{child.startEA}$ 
17)   if  $\text{next} = \text{nextEA}$  then //获取有向边, 并映射字符串和有向边
18)      $\text{edges.append}([\text{start}, \text{next}, \text{str}])$ 
19)   else if  $\text{edges.append}([\text{start}, \text{next}, \text{null}])$ 
20)   end if
21) end for
22) end for
23)  $G \leftarrow \{\text{nodes}, \text{edges}\}$ 
24) Return  $G$  //返回有向图  $G$ 

```

#### 步骤 4 查找相同起点的分支路径集合

对步骤 3 中有向图进行深度优先遍历查找路径和依赖字符串。对于  $N$  个顶点和  $E$  条边的图, 深度优先遍历算法的时间复杂度为  $O(N^2)$ , 为降低算法复杂度, 提高检测效率, 从标记顶点开始查找路径, 并取消遍历过的标记顶点的标记。具体算法实现如下。

#### 算法 2 查找相同起点的分支路径集合

输入 有向图  $G$

输出 相同起点的路径集合  $\text{Paths}$

```

1) Paths ← []
2) foreach node in G.nodes do//遍历有向图 G
  的所有顶点
3) if node[1] = 1 then//只对含有字符串的顶
  点开始查找路径
4)   path ← []
5)   findPath(G, node, node, path)//调用
  递归函数获取具有相同起点的路径
6) end if
7)end for
8)Return Paths//返回具有相同起点的路径集
合 Paths
9)function findPath(G, prev, curr, path)//查找
  路径的递归函数
10) curr[1] ← 0//已经遍历过的顶点，设置其
  flag 为 0
11) if G.adj(curr) = null then//到达有向图 G 的
  终点，结束递归过程
12)   path.append([prev, curr])
13)   Paths.append(path)
14)   path.remove([prev, curr])
15) return
16) else if path.append([prev, curr])//不满足
  结束条件，继续递归查找路径
17)   foreach next in G.adj(curr) do
18)     findPath(G, curr, next, path)
19)     path.remove([prev, curr])
20)   end for
21) end if
22) end function

```

#### 步骤 5 筛选语义冲突的路径集合

对步骤 4 中起点相同的路径集合进行筛选，根据分支路径的返回值划分为验证成功和验证失败的集合，只对验证成功的路径集合进行分析：将路径按相同的终点聚类为组，检测同组中的字符串是否有语义冲突。如果有语义冲突，则保存路径对和相应的字符串；如果没有，则不保存。

### 3 实验设计与结果分析

#### 3.1 实验数据集

为验证本文方法的有效性，从互联网中收集

TP-LINK、D-LINK、NETGEAR、小米、华为等网络设备厂商的设备固件，总共收集到 1 191 个固件。固件预处理过程如下：经 binwalk 工具扫描分析后，解包成功的固件有 1 131 个，从中筛选出 MIPS 和 ARM 体系结构的固件，最终得到包含 1 074 个固件的数据集。预处理阶段的固件数量统计如表 1 所示。数据集中有 9 个硬编码后门固件，其中 MIPS 固件有 7 个，ARM 固件有 2 个，后门固件详情如表 2 所示。

表 1 预处理阶段的固件数量统计  
Table 1 Statistics of firmware count in the preprocessing stage

| 预处理  | MIPS | ARM | 其他 | 总数    |
|------|------|-----|----|-------|
| 收集   | 862  | 285 | 44 | 1 191 |
| 解包   | 814  | 260 | 39 | 1 113 |
| 筛选   | 814  | 260 | 0  | 1 074 |
| 后门固件 | 7    | 2   | 0  | 9     |

表 2 硬编码后门固件详情  
Table 2 Hard-coded backdoor image details

| 厂商       | 体系结构 | CVE 编号         |
|----------|------|----------------|
| D-Link   | MIPS | CVE-2013-6026  |
| NISUTA   | MIPS | CVE-2013-7282  |
| NETGEAR  | MIPS | CVE-2016-11059 |
| D-Link   | MIPS | CVE-2021-21818 |
| D-Link   | MIPS | CVE-2021-21820 |
| TOTOLINK | MIPS | CVE-2021-35324 |
| NETGEAR  | MIPS | CVE-2021-35973 |
| Juniper  | ARM  | CVE-2015-7755  |
| Crestron | ARM  | CVE-2019-3932  |

#### 3.2 评价指标

本文采用以下指标来评价硬编码后门检测方法。

精度  $P$ ：方法判断为口令的样本中正确检测口令的比例。本文中的  $P$  为计算每个硬编码后门文件的精度值，求和后取平均值。

$$P = \frac{NT}{NT+NF} \quad (1)$$

召回率  $R$ ：方法检测出的后门固件样本占实际存在后门固件样本的比例。

$$R = \frac{TP}{TP+FN} \quad (2)$$

漏报率  $F$ : 方法未检测出的后门固件样本占实际后门固件样本的比例。

$$F = \frac{FN}{TP+FN}$$

平均耗时  $t$ : 方法检测单个二进制文件的时长。

$$t = T / M$$

覆盖率  $Cv$ : 方法检测出的二进制文件数量占二进制文件总数量的比例。

$$Cv = M / N$$

其中,  $NT$  为在硬编码后门文件中, 方法判断为口令且确实是口令的样本数量;  $NF$  为在硬编码后门文件中, 方法判断为口令而不是口令的样本数量;  $TP$  为方法检测出固件的真实口令的固件数量;  $FN$  为方法未检测出固件的真实口令的固件数

量;  $T$  为实验流程的时长;  $M$  为已经检测的二进制文件数量;  $t$  为单个二进制文件平均检测时长;  $N$  为二进制文件总数。

### 3.3 实验结果分析

为验证本文方法的有效性, 实验共采取 3 种方法对实验数据集进行后门检测, 计算方法的评估指标, 最后将所得结果进行分析对比, 3 种方法分别为基于静态数据比较分析的后门检测方法 (Stringer)、基于模糊哈希关联分析的弱密钥检测方法 (Costin) 和基于语义冲突的硬编码后门检测方法 (Stect), 分别在数据集中运行 360 h。不同后门检测方法的评估指标对比如表 3 所示, 结果表明, 相比其他两种方法, Stect 方法的检测范围更广, 效果更好。

表 3 不同后门检测方法的评估指标对比  
Table 3 Comparison of evaluation indicators of different backdoor detection methods

| 后门检测方法   | 代表性文献 | $P$    | $R$    | $F$    | $t$     | $Cv$   |
|----------|-------|--------|--------|--------|---------|--------|
| Stringer | 文献[7] | 4.56%  | 11.11% | 88.89% | 1.73s   | 37.71% |
| Costin   | 文献[6] | 11.11% | 11.11% | 88.89% | 757.89s | —      |
| Stect    | —     | 16.67% | 88.89% | 11.11% | 3.67s   | 97.87% |

对于数据集中 9 个真实设备的硬编码后门固件, Stringer 检测出 1 个, Costin 检测出 1 个, Stect 检测出 8 个。硬编码后门检测结果对比如表 4 所示。

表 4 硬编码后门检测结果对比  
Table 4 Comparison of hard-coded backdoor detection results

| CVE 编号         | Stringer | Costin | Stect |
|----------------|----------|--------|-------|
| CVE-2013-6026  | ×        | ×      | ✓     |
| CVE-2013-7282  | ×        | ✓      | ✓     |
| CVE-2016-11059 | ×        | ×      | ✓     |
| CVE-2021-21818 | ×        | ×      | ✓     |
| CVE-2021-21820 | ×        | ×      | ✓     |
| CVE-2021-35324 | ×        | ×      | ✓     |
| CVE-2021-35973 | ×        | ×      | ✓     |
| CVE-2015-7755  | ×        | ×      | ×     |
| CVE-2019-3932  | ✓        | ×      | ✓     |

注: ✓表示硬编码后门检测成功, ×表示检测失败

Stringer 方法只实现了对 ARM 固件的后门检测方法, 成功检测出 1 个 ARM 固件中的口令。

MIPS 架构采用分支延迟槽技术, 静态数据可以从内存中多次读取拼接而成, 相较 ARM 固件, MIPS 固件获取静态数据更难, 检测流程更烦琐, 分析难度更大。Costin 方法并没有给出源代码, 然而在文献[6]中给出固件分析平台, 平台不仅分析弱密钥后门, 而且检测固件漏洞和相应的 CVE 编号, 然而平台对固件分析时间太长, 因此只上传 9 个后门固件, 360 h 后从平台获取固件分析报告。结果表明, Costin 方法只能检测到已经收录的弱密钥后门, 检测时间更长。Stect 方法效果最好, 成功检测出 8 个后门口令, 召回率高达 88.89%。

评估覆盖率  $Cv$  时, Stringer 方法只能检测到 37.71%的固件, 而 Stect 方法可以检测到 97.87%的固件。即使考虑网络设备中 MIPS 和 ARM 体系结构占有率为 91.9%, Stect 方法的  $Cv$  也能达到 89.94%。

Stect 方法虽然可达到 16.67%的精度, 然而误报率还是很高。分析后发现字符串的莱文斯坦比可以有效约束检测结果, 降低误报率, 但效果并非最佳。人工分析检测失败的固件 ScreenOS

ssg5ssg20.6.3.0r19.0.bin 时, 由于固件去除了符号表, Stect 无法找到  $f_{\text{checkStr}}$  函数, 因此检测失败。使用二进制函数识别方法恢复该固件的部分  $f_{\text{checkStr}}$  函数后, Stect 再次检测分析后, 成功识别出 ssg5ssg20.6.3.0r19.0.bin 中的后门口令。

## 4 结束语

本文提出了基于语义冲突的硬编码后门检测方法——Stect。Stect 从字符串比较函数  $f_{\text{checkStr}}$  出发, 结合 MIPS 和 ARM 指令集的特点, 检测识别固件的硬编码后门。实验结果表明, Stect 方法在路由器固件数据集中取得了不错的效果。另外, 由于固件编译生成后可以去除符号表的影响, Stect 检测不到  $f_{\text{checkStr}}$  函数。下一步的工作会尝试去除符号表的固件大规模自动化识别  $f_{\text{checkStr}}$  函数。

## 参考文献:

- [1] PAGANINI P. Netgear, Linksys and many other wireless routers have a backdoor security affairs[EB].
- [2] 张雄, 李舟军. 模糊测试技术研究综述[J]. 计算机科学, 2016, 43(5): 1-26.  
ZHANG X, LI Z J. Survey of fuzz testing technology[J]. Computer Science, 2016, 43(5): 1-26.
- [3] SZOR P. The art of computer virus research and defense[J]. Choice Reviews Online, 2005, 43(3): 15-16.
- [4] HEFFNER C. Reverse engineering a d-link backdoor[EB].
- [5] 忽朝俭, 薛一波, 赵粮, 等. 无文件系统嵌入式固件后门检测[J]. 通信学报, 2013, 34(8): 140-145.  
HU C J, XUE Y B, ZHAO L, et al. Backdoor detection in embedded system firmware without file system[J]. Journal on Communications, 2013, 34(8): 140-145.
- [6] COSTIN A, ZADDACH J, FRANCILLON A, et al. A large scale analysis of the security of embedded firmwares[C]//Proceedings of the 23rd USENIX Security Symposium. 2014: 95-110.
- [7] THOMAS S L, CHOTHIA T, GARCIA F D. Stringer: measuring the importance of static data comparisons to detect backdoors and undocumented functionality[M]//Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2017.
- [8] SHOSHITAISHVILI Y, WANG R, HAUSER C, et al. Fimalice-automatic detection of authentication bypass vulnerabilities in binary firmware[C]//Proceedings 2015 Network and Distributed System Security Symposium. 2015: 8-11.
- [9] ZYXEL NETWORKS. Zyxel security advisory for hardcoded credential vulnerability | Zyxel[EB].
- [10] MUENCH M, STJOHANN J, KARGL F, et al. What you corrupt is not what you crash: Challenges in fuzzing embedded devices[C]//Network and Distributed System Security Symposium. 2018.
- [11] CHEN D D, EGELE M, WOO M, et al. Towards automated dynamic analysis for Linux-based embedded firmware[C]//Proceedings of Network and Distributed System Security Symposium. 2016.
- [12] HEFFNER C. Binwalk: Firmware analysis tool[R]. 2010.
- [13] HEMEL A, KALLEBERG K T, VERMAAS R, et al. Finding software license violations through binary code clone detection[J]. Proceedings-International Conference on Software Engineering, 2011: 63-72.
- [14] EAGLE C. IDA Pro[R]. 2012.
- [15] SHOSHITAISHVILI Y, WANG R, SALLS C, et al. SOK: (state of) the art of war: Offensive techniques in binary analysis[C]//2016 IEEE Symposium on Security and Privacy (SP). 2016: 138-157.

## [作者简介]



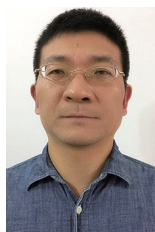
胡安祥 (1996-), 男, 江西景德镇人, 数学工程与先进计算国家重点实验室硕士生, 主要研究方向为网络空间安全和逆向工程。



肖达 (1981-), 男, 江西吉安人, 博士, 郑州工业应用技术学院副教授, 主要研究方向为网络空间安全和漏洞挖掘。



郭世臣 (1998-), 男, 河南辉县人, 数学工程与先进计算国家重点实验室硕士生, 主要研究方向为网络空间安全和逆向工程。



刘胜利 (1973-), 男, 河南周口人, 博士, 数学工程与先进计算国家重点实验室教授、博士生导师, 主要研究方向为网络空间安全。