面向 Linux 系统的嵌入式设备陷门模板化框架

赵利军 董莎莎 张沙石
(陆军工程大学军事理论创新与作战实验中心 江苏 徐州 221000)

摘 要 不同嵌入式设备间存在差异,且陷门的设计受设备的软硬件结构特点及启动过程的差异影响较大。因此,几乎不可能设计出一个固定不变的代码适用于任何设备的陷门。为解决上述问题,研究嵌入式设备的启动过程,建立涵盖不同启动流程异同点,搭建陷门模板化设计框架。为设计具有一定程度通用性的陷门提供了依据。

关键词 通用性 陷门 启动模型 模板化框架

中图分类号 TP309.1 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2019.06.041

TRAPDOOR TEMPLATE FRAMEWORK FOR LINUX SYSTEM ON EMBEDDED DEVICE

Zhao Lijun Dong Shasha Zhang Shashi
(*Military Theoretical Innovation and Operational Experiment Center, Army Engineering University of PLA, Xuzhou 221000, Jiangsu, China*)

Abstract There are differences among different embedded devices, and the design of trapdoor is greatly influenced by the differences of hardware and software structure and start-up process. So it is almost impossible to design a trapdoor with fixed code, which is suitable for any device. To solve the above problems, this paper studied the start-up process of embedded devices, established a trapdoor template design framework covering the similarities and differences of different start-up processes, and built a trapdoor template design framework. It provides a basis for the design of trapdoor with a certain degree of versatility.

Keywords Generality Trapdoor Startup model Template framework

0 引 言

随着嵌入式设备的广泛使用,如何攻克嵌入式设备成为当前网络空间作战中的关键环节,为了能继续持有攻击主机的访问特权,恶意程序必须较好地隐藏自己不被检测工具发现。作为一种高级的隐藏技术,近几年 PC 机上的陷门攻击技术——Bootkit 技术,得到了广泛使用和发展。Bootkit 技术是一种更高级的 Rootkit 技术,两者的不同之处在于 Bootkit 将存储位置从文件系统变为硬件存储,并且在操作系统加载之前启动。Rootkit 修改操作系统内核,重装系统可将其清除。Bootkit 在加载操作系统之前启动,因此独立于任何操作系统。与传统的 Rootkit 技术主要是在系统启动时提升权限不同,Bootkit 通过篡改内核及操作系统引导过程隐藏自身。

2007 年研究人员 IceLord(网名)发布的公开资料是可以找到的第一个 Windows 环境下的 BIOS Rootkit[1]。由此,针对 BIOS 的陷门设计从一个技术构想变成了程序实体。同一年,Nitin Kumar 和 Vipin Kumar 在 Black Hat 大会上呈现了一种名为 Vbootkit 的 Bootkit 攻击技术,并成功突破 Windows Vista 系统。Vbootkit 能够篡改引导向量获取执行权限,绕过启动管理器的签名机制。通过 Hook Windows Vista 系统启动文件入侵操作系统内核[2]。在 2008 年,Wenbin Zhen 等[3]发布了 Tophet.a,一款基于篡改 OSLoader 的 Bootkit 技术。该技术可使攻击者获取系统控制权并通过篡改 boot.ini 文件中的语法格式将控制权转换到攻击者的驱动。同年的 Black Hat 大会上,C. Miller 通过嗅探

收稿日期:2018-11-06。**赵利军**,讲师,主研领域:软件开发,信息安全。**董莎莎**,讲师。**张沙石**,助教。

MacBook 电池中内建的电量控制器经 I2C 收到的数据,对电池的嵌入式微控制器固件进行修改,可以使电池过度发热甚至起火,并能够在获得微控制器的控制权后对系统进行配置[4]。Lee 等[5]提出了劫持 ARM Linux 嵌入式设备的方法,该方法借鉴通用计算机下的 Linux 系统常用劫持方案,将系统调用表或者系统调用函数进行修改,并指向自己的恶意代码,从而实现控制流的转移,达到劫持目的。在文献[6]中阐述了破解 Netgear NTV300 电视机顶盒的方法。文中作者 dump 下 Netgear NTV300 中的固件映像,通过 Binwalk 解析固件组成、使用的压缩算法等,并使用 IDA 的静态分析功能,最终实现了对该设备的漏洞注入和远程控制。2013 年,Ang Cui 等[7]介绍了惠普 HP-RFU(远程固件升级)激光打印机固件修改漏洞,通过打印含有特殊命令的文档来隐蔽地修改打印机的固件,进而获得该打印机的控制权,并采用静态分析方法对其中使用的第三方库分析发现超过 80.4% 的固件程序存在 zlib 漏洞。文献[8]对惠普 RFU 漏洞进行了验证,攻击者通过 MS Office Word、Adobe PostScript 等标准的打印文档向任意打印机中注入恶意代码和命令,使用这种攻击途径可以向打印机发送修改后的固件。2015 年,李成林[9]针对目前主流杀毒软件对 KiFastCallEntry 挂钩提出了躲避方案,提出了改进型 SSDT 挂钩,该方法不再直接修改 SSDT 分发表里的函数入口地址,而是在函数体的内部进行修改。2016 年,冯培钧等[10]在“一种新型 Linux 内核级 Rootkit 设计与实现”一文中设计并实现了一种新型 Linux 内核级 Rootkit,该 Rootkit 能够实现后门提权、进程隐藏及文件隐藏等功能,并能绕过当前主流的 Rootkit 检测工具的检测。2017 年,李扬等[11]提出一种基于硬件虚拟化的内核 Rootkit 技术,该技术利用 Intel VT-x 硬件虚拟化技术将客户系统(Guest OS)迁移到 VMM 之上运行实现 Rootkit。

但是,对于一个新的嵌入式设备,多数的陷门设计技术都难以在有限的时间内设计出来,设计的陷门通用性也较差,无法部署到不同目标设备上。为此,本文通过研究嵌入式设备的启动过程,建立涵盖不同启动流程异同点的、陷门模板化设计框架,为设计具有一定程度通用性的陷门提供依据。

1 陷门模板化技术

嵌入式设备的软硬件结构特点及启动过程的差异性严重影响陷门的设计效率,人们可以针对不同设备,进行分析和设计符合需求的陷门。但本文尝试引入陷门设计的模板化框架,旨在为提炼出一种具有一定程度通用性的、适用于主流嵌入式设备的陷门设计提供方便。

1.1 嵌入式设备启动流程分析

首先,我们对各种不同的嵌入式设备启动流程进行分析,归纳它们的相同点与差异性,建立统一的启动流程。

存储芯片不同会造成 Bootloader 引导的差异;Bootloader 不同会造成内核启动流程的差异;不同内核加载过程也存在较大差异;不同文件系统挂载的机制也有区别。启动流程分析主要是对这些差异性进行总结和归纳。

Bootloader 的启动过程可以是单个阶段的,也可划分为多个阶段,嵌入式设备下一般为二阶段的启动过程,分 Stage1 和 Stage2 两部分启动。二阶段的 Bootloader 启动流程如图 1 所示。

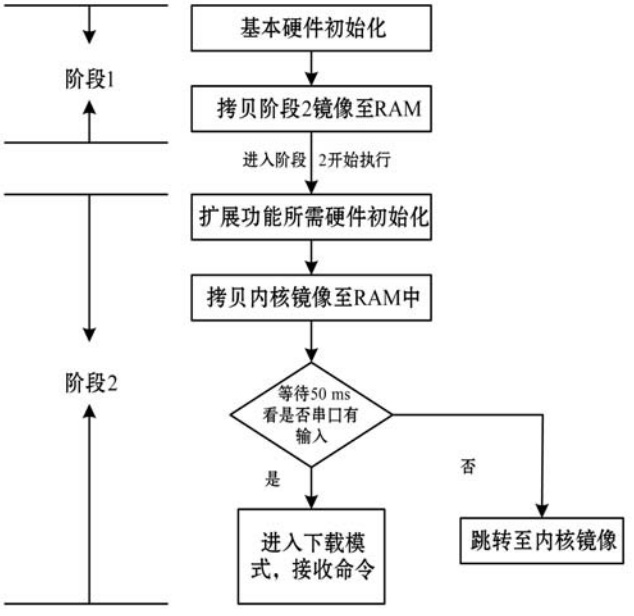

图 1 Bootloader 启动流程

导致 Bootloader 启动方式差异的因素还有内核挂载文件系统机制的差异。以 Linux 内核为例,Linux 内核中包含两种挂载早期根文件系统的机制,即 initrd 机制和 initramfs 机制。

initrd 是一个功能完备的小型根文件系统,是一种启动早期用户空间处理流程的老式方法,它通常包含一些指令,用于在系统引导完成之前加载一些特定的设备驱动程序。为了使用 initrd 的功能,大多数架构的引导加载程序会将 initrd 镜像传递给内核。常见的场景是,引导加载程序先将压缩过的内核镜像加载到内存中,接着将 initrd 镜像加载到另一段可用内存中。如图 2 所示。

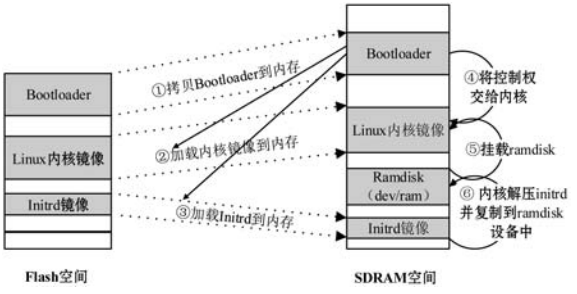

图 2　initrd 机制启动流程

在这个过程中,引导加载程序负责在将控制权转交给内核之前,将 initrd 镜像的地址传递给内核。具体的机制取决于架构、引导加载程序和平台的实现。然而,内核必须知道 initrd 镜像的位置才能够加载它。当内核引导时,它首先会检测内核命令行中是否包含 root = 参数并指定一个 ramdisk(比如 root =/dev/ram0)。然后,将这个压缩的二进制文件从内存中的指定位置复制到一个合适的内核 ramdisk 中,并挂载它作为根文件系统。与 PC 机上 Linux 内核会卸载 initrd,尝试挂载另一个文件系统作为其根文件系统不同,这类系统中唯一的根文件系统就是 ramdisk,在整个系统初始化完成后,initrd 就会成为最终的根文件系统。

与 initrd 是一种基于 RAM 的块设备不同,initramfs 是一种基于 RAM 的内存文件系统。initramfs 在编译内核的同时被编译,并与内核连接成一个文件。在引导阶段它与内核同时被 Bootloader 加载到 RAM 中。而 initrd 是另外单独编译生成的,是一个独立的文件,它由 Bootloader 单独加载到 RAM 中内核空间外的地址。其启动过程如图 3 所示。

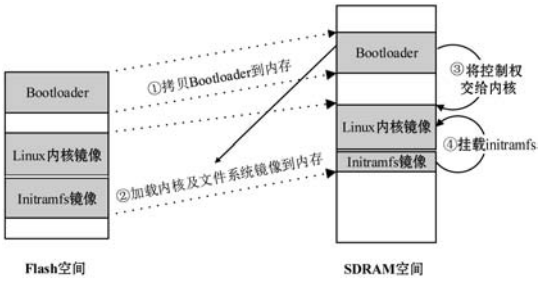

图 3　initramfs 机制启动流程

不同机制下根文件系统的挂载方式不同,如图 4 所示。

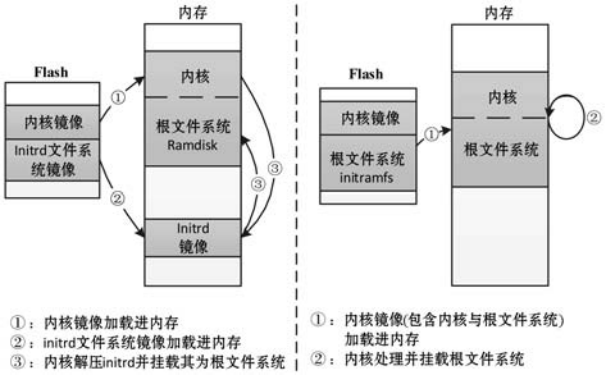

图 4　根文件系统挂载方式示意图

根据上述分析的 Bootloader 启动流程和内核挂载文件系统机制的差异,对启动流程的共性和特性进行归纳,得到 Linux 内核嵌入式设备启动流程的一般化描述,如图 5 所示。

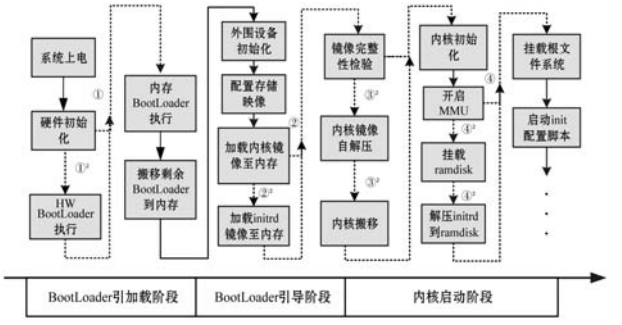

图 5　嵌入式 Linux 启动流程

图中①与①^2 代表启动过程中不同的执行路径,其中①表示不同设备之间启动过程中的共性阶段,①^2 表示不同设备之间启动过程的差异性阶段。同理,②与②^2、③与③^2、④与④^2 分别代表不同设备系统启动过程中不同的执行路径。

1.2　嵌入式设备启动模型

Linux 系统启动流程中,将涉及陷门模块设计的执行阶段划分出来,隔离成黑盒,得到粗粒度的 Linux 启动流程,如图 6 所示。

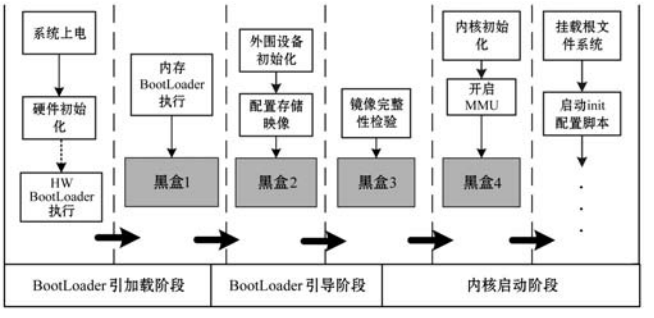

图 6　隔离涉及陷门模块设计执行阶段后的启动流程图

图 6 中,黑盒 1 和黑盒 2 属于引导代码级陷门组件模块。黑盒 3 属于内核级陷门组件模块,黑盒 4 为文件系统级陷门组件模块。其中,除黑盒 1 外,其他黑盒内部均存在路径分支,陷门的设计需要依据设备的实际执行路径进行考虑。建立的嵌入式 Linux 启动模型如图 7 所示。

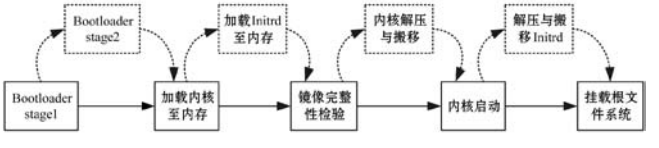

图 7　嵌入式 Linux 启动模型

由于 Bootloader Stage2、加载 initrd、内核自解压与搬移、解压与搬移 initrd 操作在某些设备启动过程中不存在,因而用虚线框表示。陷门设计时,这些阶段对应的陷门组件需要依据实际设备的启动过程和陷门需

求进行组合。

1.3 陷门模板化

高级语言中模板是根据参数类型生成函数和类的机制(有时称为“参数决定类型”)。通过使用模板,可以只设计一个类来处理多种类型的数据,而不必为每一种类型分别创建类。借鉴高级语言中模板函数、模板类等思想,提出陷门模板化技术。

陷门模板可以比作一个生成陷门攻击代码的“模子”,是对适用于某一类型设备(功能相同、结构相似)的攻击方式的抽象化概括,一个陷门模板允许根据具体的设备分析结果填写某些参数,将模板实例化成针对某个特定设备的陷门。陷门模板化设计方法能够实现部分环节的自动化、减少人工工作量,从一定程度上降低了陷门开发的难度,提高了陷门设计的效率。结合启动模型,模板化陷门设计框架如图 8 所示。

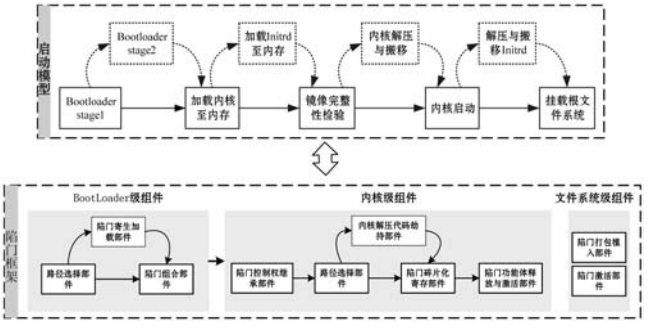

图 8 模板化陷门设计框架

整个陷门模板分为 BootLoader 级组件、内核级组件和文件系统级组件,分别对应引导阶段、内核启动阶段、文件系统挂载阶段的操作。其中文件系统级组件可单独实现陷门攻击,也可与其他两个组件配合使用。陷门模板留有拼接各组件的配置接口,可根据实际对目标设备的分析结果进行扩展。每个组件也以可配置的形式存在,可根据陷门设计需求实例化为某个特定功能的组件。

陷门功能组件中又分别存在不同部件,不同部件功能也相对独立,部件间一般需要互相配合完成特定功能。部件内部也留有“个性化”的配置接口,可根据对实际设备的分析填写相应的参数。特别说明路径选择部件主要负责根据设备启动阶段的判别信息,选择陷门设计所需的部件。

2 陷门模板设计实例

本文提出的模板化陷门设计框架适用于跨平台的陷门设计。根据陷门框架设计的陷门模板如下所示:

```
# Trapdoor Paramaters Start #
    Para0, 0x12345678
    Para1, 0x12345678
    ……
    Paran, 0x12345678
# Trapdoor Paramaters End#
# Bootloader Stage Trapdoor Start #
    ……
        # Kernel Stage Trapdoor Start #
            ……
                # Trapdoor Start #
                ……
                # Trapdoor End #
        # Kernel Stage Trapdoor End #
# Bootloader Stage Trapdoor End #
```

陷门模板代码的开头为各模块可配置的参数区。陷门代码的实体部分是一个嵌套的实现过程,与陷门代码逐级感染设备的流程想匹配。陷门最外层为最先执行的陷门代码模块,即引导级陷门代码模块。第二层为内核级陷门代码模块。最里层为陷门实际需要完成的功能模块,如网络劫持、文件下载、破坏设备等,可根据实际设备选择。每一层的执行任务完成后都会由下一层负责清理上一层陷门代码执行留下的痕迹,达到陷门隐蔽的效果。

从陷门设计框架可知,陷门结构也可能只由文件系统级陷门模块和陷门实际功能模块组成。由前面提出的可重构陷门设计方法可知陷门结构中各模块的组合以及各模块内部不同部件的组成,都可根据实际需求进行调整。

陷门功能需要汇编语言实现,因此陷门模板的实现只适用于某个特定架构。以 ARM 架构为例,阐述陷门模板具体实现。由于完整代码太长,将陷门模板的实现过程进行分块说明。

2.1 可配置参数区实现

陷门模板中的可配置参数保留区分为全局参数区和局部参数区。全局参数区主要是一些执行各模块时必须配置的参数,如 Bootloader 级模块和内核级模块必须配置内核镜像加载的起始地址及结束地址。局部参数区主要用作陷门各模块内部件执行时的参数配置,如 Bootloader 级模块陷门寄生加载部件需要配置陷门加载的地址和大小、碎片重组部件需要配置碎片数目和各碎片加载的地址及大小。可配置参数区的实现如下:

```
# 全局参数区 #
. equ Para0, 0x30008000    //内核加载到内存的起始地址
. equ Para1, 0x30008128      //内核压缩数据起始地址
. equ Para2, 0x12345678
……
```

```
. equ Paran, 0x12345678
# Bootloader Stage Trapdoor Start #
# 局部参数区 #
MSAK_Parameter:
    . word 0xefefefea                    //陷门起始地址
    . word 0xefefefeb                    //陷门结束地址
    ……
RecoverPara_MSAK:
            //陷门控制流劫持可能覆盖引某条关键指令
    . word 0xefefeff1        //将需要恢复的指令保存于此
    . word 0x00080000
    ……
```

2.2　程序栈实现

“程序栈”是保存和恢复陷门劫持控制流后破坏的设备运行时上下文信息的区域,一般位于每个陷门模块和模块部件的开头和结尾处。具体实现如下:

```
# 程序栈 #
b SelfDefinationStack
. word 0x11111111
. word 0x00001110
……
. word 0x0000111f
. word 0x00001120
SelfDefinationStack:
    str r0, [pc, #-76]
                //pc-76 为程序栈的第一个数据地址
    str r1, [pc, #-76]
    ……
    str r14, [pc, #-76]
    mrs r0, cpsr
    str r0, [pc, #-80]
    mrs r0, spsr
    str r0, [pc, #-84]
    ……
RelocationStack:
    ldr r0, RelocationStack          //获取程序栈的位置
    sub r0, pc, r0
    sub r0, r0, #0xc
    ldr r14, SelfDefinationStack
    add r14, r14, r0
    ldr r0, [r14, #-4]        //恢复保存的寄存器数据
    msr spsr, r0
    ldr r0, [r14, #-8]
    msr cpsr, r0
    ldr r14, [r14, #-12
                    ……
    ldr r0, [r14, #-68]
    ldr r1, [r14, #-64]
```

2.3　数据池实现

数据池不仅可以用来存储下一个模块的代码和数据,也可作为当前模块需要传递给下一个模块的数据保存区。数据池的实现较为简单,但是陷门模板设计中不可或缺的重要部分,具体形式如下:

```
B Lable:
Data_Buffer:
            . word 0x11111111
            . word 0x11111111
            . word 0x11111111
            ……
            . ascii
            "\xff\x5f\x2d\xe9\x00\x60\x0f\xe1\x40\
            x00\x2d\xe9\x00\x60\x4f\xe1\x40……"
Lable:  ……
```

其中第一部分为模块间传递的数据区,. ascii 为某个 shellcode 形式的陷门模块。

3　结　语

由于嵌入式设备“客制化”的生产模式,大多数情况下难以在给定的时间内设计出针对一个新的嵌入式设备的陷门,设计的陷门通用性也较差,无法部署到不同目标设备上。本文通过研究嵌入式设备的启动过程,建立涵盖不同启动流程异同点的,搭建了陷门模板化设计框架,最后通过势力说明了陷门模板的设计方法。本文提出了框架为设计具有一定程度通用性的陷门提供了依据。

参　考　文　献

[1] Lord I. BIOS Rootkit: Welcome Home, My Lord! [EB/OL]. [2007-05-11]. http://www.xfocus.net/articles/200705/918.html.

[2] Kumar N, Kumar V. Vbootkit: Compromising windows vista security[C]//BlackHat Europe, 2007.

[3] Zheng W. Advanced Bootkit-Tophet[C]//Xcon2008, November, 2008.

[4] Miller C. Battery Firmware Hacking[C]//BlackHat USA, 2011.

[5] Lee H, Kim C H, Yi J H. Experimenting with System and Libc Call Interception Attacks on ARM-based Linux Kernel [C]//Proceedings of the 2011 ACM Symposium on Applied Computing. ACM, 2011: 631-632.

[6] Craig. Jailbreaking the NeoTV[EB/OL]. [2012-10-23]. http://www.devttys0.com/2012/10/jailbreaking-the-neotv/.

(下转第 232 页)

±10 min 与 ±15 min 的预测准确率分别提高了8.9%、12.05%和12.23%;在航班客舱门关闭时,预测误差为 ±5 min, ±10 min 与 ±15 min 的预测准确率分别提高了 18.78%、15.27%和 13.29%。同时,由图 13 实验结果可见,在不同的时刻,基于级联 BP 神经网络的航班撤轮挡时刻预测模型的预测时间都保证在 0.1 s 以内,从而确保了预测结果的实时性和时效性。

4 结 语

航班撤轮挡时刻的准确预测不仅可以为航空公司评估航班 TOBT,安排航班起飞队列提供参考,也便于机场合理调配资源。为了更准确地预测撤轮挡时刻,本文通过构建级联 BP 神经网络模型,并进行过拟合分析,利用航班计划数据与航班过站保障数据,分别在航班进港前、航班入位时、航班值机结束时与航班客舱门关闭时进行航班撤轮挡时刻预测,并与现有的经验统计预测模型进行比较。通过实验验证,本文提出的基于级联 BP 神经网络的航班撤轮挡时刻预测模型的预测准确性要远好于现有的经验统计预测模型。另外,在航班进港前,预测准确率在允许容差 ±15 min 内的航班数占总航班数的 84.9%,航班入位时可以达到了 95%以上,并且,随着航班过站流程的推进,预测的准确性在不断提高。因此,本文提出的预测模型具有很好的实用性、时效性与可参考性,可以大大减少人工干预的次数,提高系统决策的效率。

参 考 文 献

[1] Kunze T, Oreschko B, Fricke H. Aircraft Turnaround Management in a Highly Automated 4D Flight Operations Environment [D]. Dresden: Technische University Dresden, 2012.

[2] Groppe M. Influences on Aircraft Target Off-Block Time Prediction Accuracy[D]. Cranfield University, 2011.

[3] Gok Y S. Scheduling of Aircraft Turnaround Operations Using Mathematical Modelling: Turkish Low-cost Airline as a Case Study[D]. Institutional Repository for Coventry University, 2014.

[4] 冯霞, 张鑫, 陈锋. 飞机过站上客过程持续时间分布[J]. 交通运输工程学报, 2017, 17(2):98 - 105.

[5] Oreschko B, Schultz M, Elflein J, et al. Significant Turnaround Process Variations due to Airport Characteristics [C]// Proceedings of International Air Transport & Operations Symposium, 2010: 263 - 270.

[6] 陈刚, 周杰, 张雪君,等. 基于 BP 与 RBF 级联神经网络的日负荷预测[J]. 电网技术, 2009(12):101 - 105.

[7] 张玄武, 郑耀, 杨波威,等. 基于级联前向网络的翼型优化设计[J]. 浙江大学学报(工学版), 2017, 51(7):1405 - 1411.

[8] Sun X, Peng X, Ren F. Detect the Emotions of the Public based on Cascade Neural Network Model[C]//Proceedings of IEEE/ACIS, International Conference on Computer and Information Science. IEEE, 2016:1 - 6.

[9] 张宇楠, 曾实. 三种模型级联的比较[J]. 原子能科学技术, 2014, 48(11):1921 - 1927.

[10] 徐涛, 赵晨旭, 卢敏. 基于因子分析的航班撤轮挡时刻预测方法[J]. 计算机工程与设计, 2017(11):3011 - 3017.

[11] Chatterji G, Zheng Y. Wheels-Off Time Prediction Using Surface Traffic Metrics[C]//Proceedings of AIAA Aviation Technology, Integration, and Operations. 2012:4167 - 4180.

[12] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A simple way to prevent neural networks from overfitting[J]. The Journal of Machine Learning Research, 2014, 15(1): 1929 - 1958.

[13] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[C]//International Conference on Neural Information Processing Systems. Curran Associates Inc. 2012:1097 - 1105.

(上接第 225 页)

[7] Cui A, Costello M, Stolfo S J. When Firmware Modifications Attack: A Case Study of Embedded Exploitation[C]//the 20th Annual Network & Distributed System Security Symposium, 2013.

[8] Bambenek. Hacking HP Printers for Fun and Profit[EB/OL]. [2011 - 11 - 29]. https://isc.sans.edu/diary/Hacking+HP+Printers+for+Fun+and+Profit/12112.

[9] 李成林. Rootkit 的改进型实现与检测技术研究[D]. 南昌:江西师范大学, 2015.

[10] 冯培钧, 张平, 陈志锋, 等. 一种新型 Linux 内核级 Rootkit 设计与实现[J]. 信息工程大学学报, 2016, 17(2): 231 - 237.

[11] 李扬, 周安民, 张磊, 等. 一种基于硬件虚拟化的 Rootkit 技术[J]. 现代计算机: 中旬刊, 2017(2): 21 - 25.

[12] 孟晨宇, 阮阳, 王佳伟, 等. Rootkit 进程隐藏与检测技术研究[J]. 软件导刊, 2016, 15(5): 188 - 190.

[13] 蒋和国, 蒋烈辉, 舒辉, 等. 基于 JTAG 仿真的 ARM Linux 设备 Bootkit 检测技术研究[J]. 计算机应用研究, 2016, 33(2): 526 - 530.

[14] 刘文祺, 范明钰, 赵永福. 隐藏关系下计算机异常干扰检测方法仿真研究[J]. 计算机仿真, 2018,35(1): 424 - 427.