文章编号:1007-5321(2020)01-0046-08 **DOI**:10.13190/j.jbupt.2019-085

嵌入式固件脆弱哈希函数自动识别与破解方法

张国栋[1,2], 应 欢[3], 杨寿国[1,2], 石志强[1,2], 李霁远[4]

(1. 中国科学院 信息工程研究所, 北京 100093; 2. 中国科学院大学 网络空间安全学院, 北京 100049;
3. 中国电力科学研究院有限公司, 北京 100192; 4. 国网浙江省电力有限公司电力科学研究院, 杭州 310014)

摘要: 针对现有固件脆弱哈希函数识别误报率高、定位不准确、破解难度大等问题,提出一种嵌入式固件脆弱哈希函数自动识别与破解方法,基于机器学习模型和结构化匹配的脆弱哈希函数识别与定位技术以及基于 VEX 中间表达式(VEX IR)符号执行的 Z3 约束求解器(Z3 SMT)的求解方法,构建了从固件二进制文件的脆弱哈希函数的识别与定位到破解的完整自动化分析流程. 实验结果表明,所提方法对多种架构和不同编译优化选项下编译的二进制文件的脆弱哈希函数的识别与定位的准确率高达 98%,对类似于 BKDR 哈希函数(BKDRHash)结构的脆弱哈希函数能够准确定位,并快速破解出多个碰撞值.

关 键 词: 嵌入式设备固件; 特征提取; 机器学习; 脆弱哈希函数识别与破解; 约束求解

中图分类号: TP309.1 **文献标志码**: A

Automatic Identification and Cracking Method for Vulnerable Hash Functions of Embedded Firmwares

ZHANG Guo-dong[1,2], YING Huan[3], YANG Shou-guo[1,2], SHI Zhi-qiang[1,2], LI Ji-yuan[4]

(1. Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China;
2. School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China;
3. China Electric Power Research Institute, Beijing 100192, China;
4. State Grid Zhejiang Electric Power Research Institute, Hangzhou 310014, China)

Abstract: There exist some problems for the existing firmware vulnerable Hash functions mining technology, for the reason that the identification error rate is high, the positioning is not accurate, the cracking is difficult and so on. To solve these problems, a method that uses vulnerable Hash functions identification and positioning technique based on machine learning model and a structured matching method is proposed. Meantime, constraint solution of Z3 satisfiability modulo theories (Z3 SMT) based on VEX intermediate representation (VEX IR) and symbol execution techniques for an automatic identification and cracking method or vulnerable Hash functions of embedded firmwares are proposed. A complete automated analysis process is constructed for the vulnerable Hash functions in the firmware binaries from being identified and positioned to being cracked. Experiments show that the method can identify and position the vulnerable Hash functions in the binary files which compiled by multiple architectures and compiler optimization options with the accuracy rate as high as 98%, vulnerable Hash functions with a structure similar to the BKDRHash Hash function structure can be accurately positioned and quickly cracked out of

收稿日期: 2019-05-21
基金项目: 国家电网有限公司总部科技项目"电网嵌入式终端漏洞挖掘与攻击检测关键技术研究"(52110418001K)
作者简介: 张国栋(1991—), 男, 硕士生.
通信作者: 石志强(1970—), 男, 正研级高级工程师, 博士生导师, E-mail:shizhiqiang@iie.ac.cn.

many collision values.

Key words: embedded device firmware; feature extraction; machine learning; identification and cracking of vulnerable Hash functions; constraint solution

固件(firmware)是管理高层次软件和底层硬件之间交互的一种软件,是运行在嵌入式设备中的可执行二进制程序,用于初始化硬件、启动操作系统和管理计算机平台资源. 固件应用在非常广泛的智能设备电子产品中,例如路由器、交换机、打印机、网络摄像头、无人机,甚至工业机器人和心脏起搏器都能见到它的身影.

由于嵌入式设备的运行要求实效性,并且其计算能力有限,哈希函数为了设备的性能优化,使用了一些极为脆弱的哈希函数或对标准哈希函数做了简化,导致其存在安全缺陷. 固件的脆弱哈希函数存在的漏洞常成为“黑客”攻击者进行设备攻击和相关安全专家进行科学研究的主要入口点之一. 例如,编号为 CVE-2010-2967 的 VxWorks 加密算法漏洞就是 VxWorks6.9 之前版本的系统中 loginLib 的 loginDefaultEncrypt()脆弱哈希函数存在密码碰撞问题,导致密码哈希总数达 22 万个,攻击者可以通过构建密码字典暴力破解 telnet、ftp、rlogin 等会话,从而获取系统控制权限,给网络空间安全增加了危害. 因此,开展嵌入式设备固件的脆弱哈希函数的识别与破解研究也变得至关重要.

目前针对固件二进制文件的哈希函数识别与破解的研究还很浅显,国内外的研究主要体现在漏洞函数识别[1]、函数漏洞挖掘领域,主要做法包括源代码级函数分析、同源漏洞函数关联[2]和基于逆向工程的二进制代码漏洞的静态分析技术[3]等. 常见的哈希函数破解方法有彩虹表法、生日攻击法、相等子串法、中间相遇法、差分攻击法、先行攻击法等. 经调研得知,将上述哈希函数的识别和破解方法相结合并运用到固件二进制文件的研究中,将会产生识别误报率高、定位不准确、破解效率低等一系列问题.

为了解决上述问题,提出一种嵌入式固件脆弱哈希函数自动识别与破解方法,能够有效地识别与定位出固件的二进制文件中的脆弱哈希函数,对识别与定位出的一些极为脆弱的哈希函数能够有效地进行逆向破解. 主要方法包括:1)为了解决研究对象固件的来源问题,采用 Python Scrapy 框架技术对国内外 65 个知名智能设备厂商进行固件爬取,共收集有效固件 46 078 个,构建了一套完整的固件库,提供了充足的固件研究对象;2)为了实现不同架构、不同编译优化选项下编译的多个二进制文件的脆弱哈希函数的准确识别与定位,提出一种基于机器学习模型和结构化匹配的脆弱哈希函数识别与定位方法;3)为了有效地解决固件反汇编差异和脆弱哈希函数破解难度大的问题,采用了一种基于 VEX 中间表达式(VEX IR, VEX intermediate representation)符号执行的 Z3 约束求解器(Z3 SMT, Z3 satisfiability modulo theories)的求解方法. 所提方法形成了一个从固件二进制文件的脆弱哈希函数的识别与定位到破解的完整自动化技术流程,主要技术流程如图 1 所示.

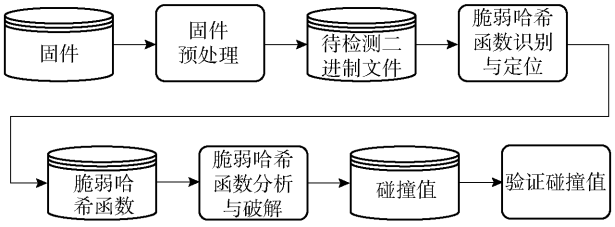

图 1　固件脆弱哈希函数自动识别与破解流程示意图

1　固件脆弱哈希函数识别与定位

1.1　嵌入式设备固件的预处理

固件的预处理是为了获取待研究的固件中可执行反汇编的二进制文件,为后续嵌入式设备固件脆弱哈希函数的识别和定位以及分析与破解提供研究对象支持.

嵌入式设备固件预处理流程如图 2 所示,主要包括:固件爬取、固件存储、固件解码、可执行反汇编二进制文件过滤. 固件预处理的目的是为了获得待检测的固件二进制文件. 具体做法包括以下几部分.

1) 固件爬取. 研究和分析国内外嵌入式设备厂商固件网页特征,针对预选的厂商开发基于 Python Scrapy 框架的固件爬虫,爬取的固件信息包括固件名、厂商名、设备类别、产品型号、固件版本号、固件下载链接、固件描述、固件发行日期等.

2) 固件存储. 搭建 MongoDB 数据库对爬取的固件信息进行存储. 编写固件下载脚本,针对

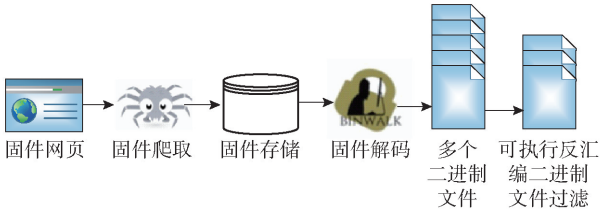

图2 嵌入式设备固件预处理流程示意图

固件信息库中的固件下载链接进行固件自动化、批量化下载,将下载的多个固件存储到服务器指定位置.

3) 固件解码. 编写固脚本程序调用 Binwalk 等固件分析工具进行批量化固件解码.

4) 可执行反汇编二进制文件过滤. 对解码后产生的多个二进制文件进行筛选,得到可执行反汇编的二进制文件,作为后续的研究对象.

1.2 固件脆弱哈希函数识别与定位方法

固件脆弱哈希函数识别与定位方法流程如图3所示,主要步骤包括:1)收集固件的脆弱哈希函数和其他函数的源码,将源码编译成多个不同架构、不同编译优化选项下的可执行反汇编二进制文件;2)通过编写交互式反汇编器专业版(IDA Pro, interactive disassembler professional)工具的 IDAPython 脚本插件,对编译的二进制文件中的函数进行研究,提取区别于脆弱哈希函数与其他函数的多项特征,并将提取的特征进行数值化处理,同时标记哈希函数为正样本、其他函数为负样本;3)通过 Python 的 scikit-learn 机器学习库的 classifier 分类器,使用 Logistic Regression 逻辑回归方法对特征数据进行训练和测试,构建分类模型;4)通过设置的评价指标判断模型是否符合要求,如若不符合需重新进行特征提取和模型训练,反复试验,直至得到符合要求的分类模型;5)设置相似度阈值,将固件预处理的待检测二进制文件的函数特征信息与保存的模型进行基于结构化匹配的函数关联处理,识别和定位出嵌入式设备固件中的脆弱哈希函数.

识别和定位出固件的脆弱哈希函数,为后续的脆弱哈希函数的分析与破解的研究工作提供前提.

1.3 脆弱哈希函数特征提取及数值化处理

函数囊括在二进制文件中,要识别一种类别的函数,通常根据不同类别函数比较明显的特征进行区别. 实现脆弱哈希函数特征提取需要解决3个关键性问题:1)提取的脆弱哈希函数特征具备不受不同架构和编译优化选项的影响或受其影响较小; 2)提取的特征能够较为明显地区分脆弱哈希函数和其他函数; 3)提取的函数特征信息需要进行数值化处理.

通过对脆弱哈希函数和其他函数的多项特征统计分析,可以发现不同架构、不同编译优化选项下编

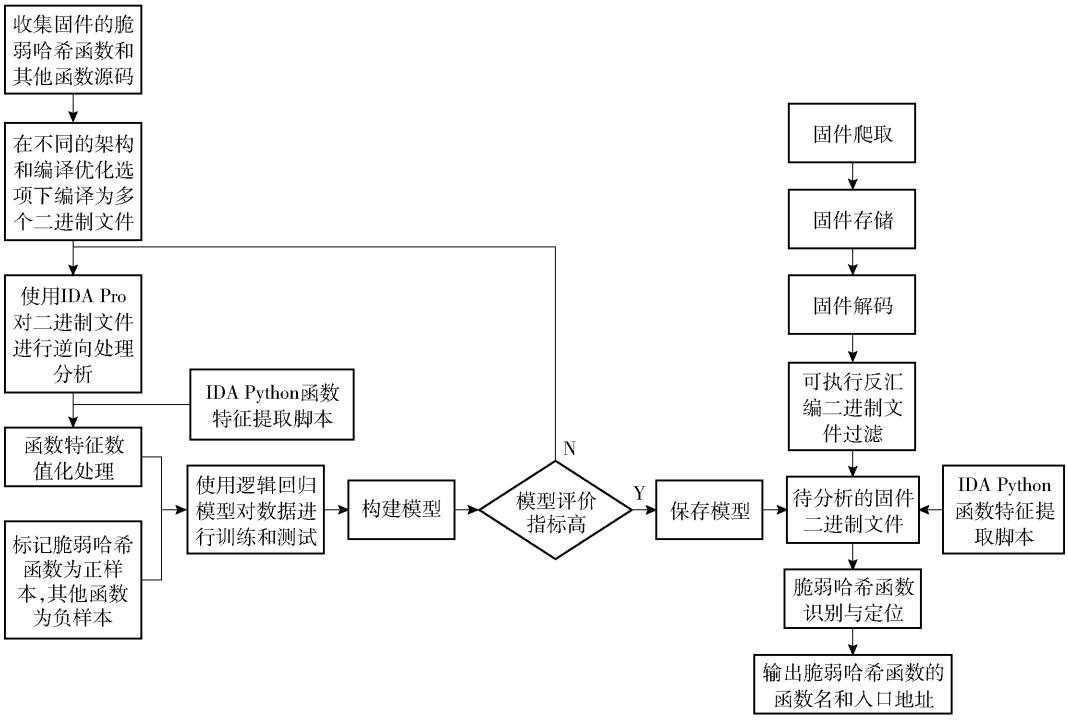

图3 固件的脆弱哈希函数识别与定位方法流程

译的二进制文件中的脆弱哈希函数与其他函数存在较为明显的差异，主要包括函数名、指令数量、指令类型数量、跳转指令数量、调用次数、异或数量、栈大小、代码基本块的数量和控制流图(CFG，control flow graph)中是否存在循环等多维特征. 对特征进行数值化处理，采取对特征信息进行标签标记，存在记为 1，不存在记为 0，其具体数量采用数值累加的方法.

脆弱哈希函数特征提取与数值化处理过程包括：将目标二进制文件导入 IDA Pro，使用脚本插件进行函数特征提取，对二进制文件中的所有函数进行遍历. 具体到某一函数时，采取对上文所述的 9 个特征进行研究和分析，并将提取的特征信息进行数值化处理，转化为对数据分析和挖掘. 设置脆弱哈希函数为正样本并标记为 1，其他函数为负样本并标记为 0.

1.4　基于逻辑回归的模型构建

模型构建过程主要包括：对函数特征进行提取，并设计转化为数值特征. 对是否为哈希函数进行打标签，对特征进行二值化处理. 调用 Python 的 scikit-learn 机器学习库的 Logistic Regression 模块对 1.3 节的数值化处理的函数特征值进行训练与测试. 由于是监督学习，训练样本集中至少包含特征向量和标签 2 个信息，而测试样本集中的每个样本只包含特征信息，通过计算预测标签信息，对比预测值和实际值，计算准确率. 如果准确率很低，需要重新提取脆弱哈希函数特征或采集数据. 如果准确率高，说明此模型具有一定的可用性. 而后使用构建的可靠模型对编译的二进制文件进行脆弱哈希函数识别与定位，使用设置的评价体系对模型进行评估. 根据评估结果，如果建立的模型能较好地区分编译的二进制文件中的脆弱哈希函数和其他函数时，将此训练好的模型保存，用于开展后续的固件的脆弱哈希函数识别与定位工作.

1.5　脆弱哈希函数的识别与定位

将待检测固件中二进制文件与构建的模型采用基于结构化匹配技术进行固件脆弱哈希函数识别与定位，对二进制文件中的所有函数进行研究，逐个地将每个函数与保存的数学分类模型进行函数关联和结构化匹配，设置匹配脆弱哈希函数相似度阈值，判断函数是否为脆弱哈希函数，如果为脆弱哈希函数，则输出其函数名和函数入口地址.

2　固件脆弱哈希函数分析与破解

2.1　固件脆弱哈希函数分析与破解方法

在识别与定位出脆弱哈希函数的函数名和函数入口地址后，接下来的工作是对识别出的脆弱哈希函数进行深层次分析，并尝试逆向暴力破解.

实现对多种类型固件下二进制文件的脆弱哈希函数进行分析和破解，需要解决 2 个关键问题：1)由于多编程语言的差异性、编译版本的差异性、多架构等因素影响，导致嵌入式设备固件的编译存在较大的差异性，即使是同一段代码在不同架构和编译优化选项下编译的固件二进制代码也存在很大的差异；2)针对脆弱哈希函数进行逆向破解，需要设置逆向求解的方法，同时由于无法知晓原始输入数据(明文)的内容，所以逆向破解的方向也很多样，这将会造成破解难度大、破解效率低下.

针对上述问题，设置了一种基于 VEX IR 符号执行[4]的 Z3 SMT[5]约束求解方法，其工作重点在于：1)对二进制文件中脆弱哈希函数的代码进行结构划分和提取；2)将划分的各模块机器码或汇编代码转化为中间表达式 VEX IR 语句表示；3)构建基于符号执行的求解表达式；4)添加求解约束条件，对脆弱哈希函数进行逆向暴力破解，得到脆弱哈希函数原始数据的碰撞值，并验证碰撞值.

固件脆弱哈希函数分析与破解方法流程如图 4 所示，主要步骤包括：1)使用 angr 工具的 cle 加载目标二进制文件，提取已被识别与定位出的脆弱哈希函数的控制流图 CFG[6]；2)基于深度优先搜索算法和拓扑排序算法将脆弱哈希函数代码拆分为初始化 Block、循环体 Block、末尾结束 Block 三个基本块，并得到每个基本块入口地址和跳转地址；3)使用 angr 的 pyvex 将各模块的机器码或汇编代码转化成对应的中间表达式 VEX IR 语句；4)采用符号执行将变量值转化为符号值，本文中使用 $b_0, b_1, \cdots, b_{i-1}, b_i$ 分别代替输入字符串的第 1，第 2，…，第 $i-1$，第 i 位的内容，对前一步的循环次数进行展开，执行 n 次循环；5)构建 Z3 表达式，并确定函数中存在的循环次数，研究发现，循环的次数实际就是哈希函数输入值的长度；6)添加逆向求解的约束条件，求解加密信息的原始输入数据或其碰撞值；7)验证碰撞值是否正确，对比原始输入数据和碰撞值两者的脆弱哈希函数加密后的输出值是否相等，相等则破解成功，不等则验证下一个碰撞值.

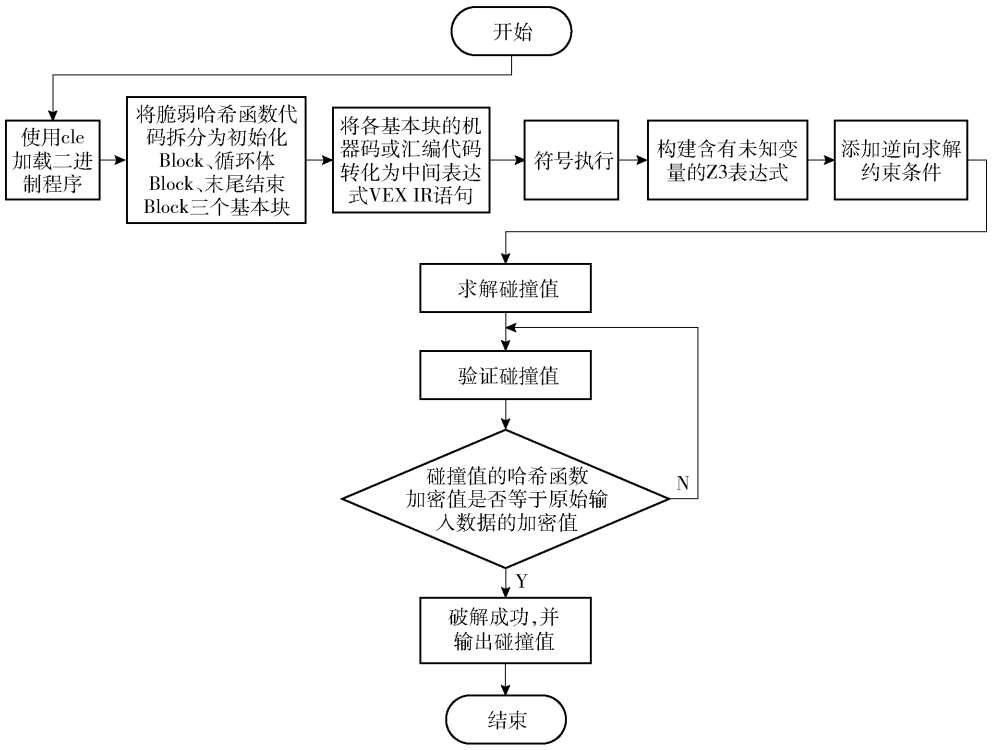

图4 固件的脆弱哈希函数分析与破解方法流程

2.2 脆弱哈希函数的结构划分和提取

二进制文件通过 IDA Pro 反汇编后的汇编代码中通常存在 call、jump、jcond、return 指令,这些指令会使模块代码间存在跳转,造成函数地址空间代码块的分块. 在机器语言中,通常间接转移指令引起的跳转被视为一个分界线,把一个完整的函数结构分为几个不同的代码块.

脆弱哈希函数的结构划分和提取采用 angr 的 cle 加载二进制文件,并将各函数的基本块拆分为 3 部分:初始化 Block、循环体 Block、末尾结束 Block. 利用深度优先搜索算法生成目标图的拓扑排序表,利用拓扑排序表得出函数 3 个基本块的地址.

2.3 脆弱哈希函数的 VEX IR 表示

在进行符号执行之前,要得到函数 3 个基本块的中间表达式 VEX IR 表示语句. 在继 2.2 节得到函数各部分基本块地址之后,就可以利用 cle 和 pyvex 工具进行各基本块的中间表达式转化.

下面以一条 mov 指令,阐述其中间表达式表示方法. 该 mov 指令是 BKDR 哈希函数(BKDRHash, BKDRHash Hash function)反汇编之后位于地址 0x400535 的指令,为 mov [rbp + var_C], 0F8C9h, 其对应的 VEX IR 表示如图 5 所示.

2.4 基于 VEX IR 的符号执行

符号执行过程中,按照哈希函数代码的执行顺序执行中间语言,在循环体中进行未知变量的替换. 符号执行时,需要得到含有未知变量的 Z3 表达式. 由于中间语言的有效语句都是等式,因此需要对等式两边进行分析. 具体过程为:1)用等号作为分隔符,得到等式两边的操作;2)提取左侧变量操作数;3)判断右侧操作类型,并提取操作数;4)判断所有提取的操作数是否已经进行变量声明,若无,使用上述 Z3 变量统一命名进行变量声明,对右侧的变量进行赋值;5)执行转化后的语句.

```
------ IMark(0x400535, 7, 0) ------
t14 = Add64(t6,0xfffffffffffffff4)
STle(t14) = 0x0000f8c9
PUT(pc) = 0x000000000040053c
```

图5 VEX IR 表示示例

得到含有未知变量的 Z3 表达式,还需要将变量值替换为符号值. 在执行循环语句过程中,通过判断语句中表达式的值是否等于参数寄存器的值进行定位输入参数并替换为符号值.

2.5 基于 Z3 SMT 的约束求解及验证碰撞值

将程序分析问题转化为约束求解问题,需要添加求解约束条件,使求解方式朝着预期的方向进行,缩小暴力破解的搜索范围和提高函数的破解速率. 笔者设置了 2 条约束条件:1)假设原始输入数据是

由一系列的数字和大小写字母组成的字符串，在添加约束条件时，将破解字符内容限制在数字和大小写字母之中；2）令符号执行之后含有未知量的 Z3 表达式的值等于原始输入数据经脆弱哈希函数加密后的输出值.

程序输出的“sat”是“satisfiable”的缩写，是一个确定命题逻辑式是否满足的问题[7]. 另外，由于原函数不是单满射函数，则可能存在多个符合条件的原始输入数据的碰撞值. 使用 Z3 的 check( ) 方法判断是否满足约束条件，如果 s. check( ) = = sat，说明至少存在一个值，然后调用 model( ) 方法进行求解碰撞值，若 s. check( ) = = unsat，则说明不存在破解值. 得到满足条件的输出解时，其值为 ASCII 码值，需要将其转化为数字或者字母进行输出. 通过调用 Python 内置函数 chr( )，将其结果转化为较为容易识别的数字或者字母，将字符按照 $b,\cdots,b_{i-1}$ 的顺序输出.

将碰撞值输入原脆弱哈希函数加密，得到的输出值与原始输入数据的加密值相等，则可以说明求解出的该碰撞值是正确的. 验证步骤如图 6 所示.

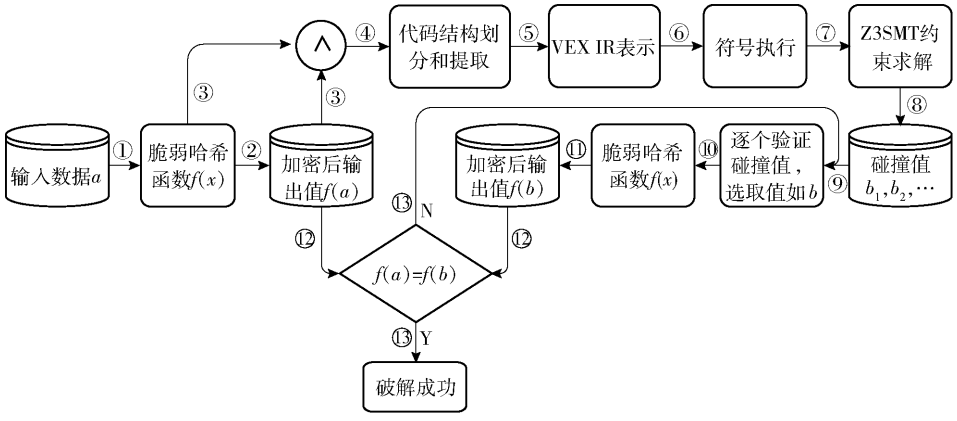

图 6　验证求解碰撞值流程

3　实验结果与分析

3.1　脆弱哈希函数识别与定位实验

1）评价指标

为了能准确评估模型的优劣，使用混淆矩阵作为检测结果对照表. 如表 1 所示，把从实验中提取的正确脆弱哈希函数个数记为 TP(true positive)，即被模型预测为正的正样本；把实验提取的非脆弱哈希函数个数记为 FP(false positive)，即被模型预测为正的负样本；把从实验中没有提取的正样本个数记为 FN(false negative)，即被模型预测为负的正样本；把从实验中没有提取的非脆弱哈希函数个数记为 TN(true negative)，即被模型预测为负的负样本.

表 1　检测结果对照表

是否被检测出	是否是脆弱哈希函数	
	脆弱哈希函数(正样本)	其他函数(负样本)
被检测出	TP 正样本判为正样本	FP 负样本判为正样本
未检测出	FN 正样本判为负样本	TN 负样本判为负样本

采用准确率 P(precision)、召回率 R(recall) 和综合评价指标 F_1(f-measure) 值作为模型好坏的衡量标准，计算公式如下：

$$P=\frac{TP}{TP+FP} \tag{1}$$

$$R=\frac{TP}{TP+FN} \tag{2}$$

$$F_1=\frac{2PR}{P+R} \tag{3}$$

准确率越高，说明构建的机器学习模型对二进制文件中的脆弱哈希函数和其他函数的区分能力越强；召回率越高，说明构建的模型对二进制文件中的脆弱哈希函数的识别与定位能力越强；综合评价指标值越高，说明构建的模型的使用价值越高，提出的算法质量越好. 实验中设置所有评价指标的容忍值为 60%.

2）实验结果与分析

收集一些典型的脆弱哈希函数(BKDHash 等) 的源码和其他一些易于分辨的函数源码，编译成不同架构、不同编译优化选项下的多个可执行的二进制文件. 进行 1.3 节的脆弱哈希函数特征提取及数值化处理，实验处理过程中的部分函数数值化处理结果如图 7 所示. 根据 1.4 节的方法构建一个模

型,在此过程中需要反复测试模型的好坏,最终确保构建的模型能够较好地用于脆弱哈希函数的函数关联和结构化匹配.

```
4433340 __ZNKSt9basic_iosIcSt11char_traitsIcEE10exceptionsEv
2 0.0 2 0.0 0.0 1 4 0
4433344 __ZNKSt9basic_iosIcSt11char_traitsIcEE3badEv
3 0.0 3 0.0 0.0 1 4 0
4605377 sub_4645C1
2 0.0 4 1.0 0.0 1 4 0
4203970 __GLOBAL__sub_I__Z11calc_HashnrPKcj
6 0.0 8 0.0 1.0 1 32 0
4433352 __ZNKSt9basic_iosIcSt11char_traitsIcEE3eofEv
4 0.0 4 0.0 0.0 1 4 0
4587160 __ZNSt15basic_streambufIcSt11char_traitsIcEED2Ev
3 0.0 3 1.0 0.0 1 4 0
4343244 __ZNKSt11__timepunctIcE8_M_am_pmEPPKc
2 0.0 8 0.0 0.0 1 8 0
4335056 __ZN9__gnu_cxx18stdio_sync_filebufIwSt11char_traitsIwE
```

图7 函数特征数值化处理结果

通过反复实验,构建的机器学习模型具有较好的效果,从表2可以看出,准确率高于98%,召回率基本高于66%,综合评价指标值大都在76%以上,远高于实验设定的容忍值. 因此,得到的实验模型对脆弱哈希函数具有良好的识别效果.

表2 评价指标计算值 %

评价指标	计算次数					平均值
	1	2	3	4	5	
P	98.7	99.0	98.7	99.0	98.5	98.8
R	66.6	75.0	83.3	66.7	54.5	69.2
F_1	76.2	81.8	80.0	80.0	66.6	76.9

而后使用建立的良好模型检测编译的二进制文件中的脆弱哈希函数,得到识别和定位出的脆弱哈希函数的函数名和函数入口地址,如图8所示,其中函数入口地址为十进制表示. 在建立的模型较好的情况下,将模型运用到固件可执行反汇编的二进制文件中,基于结构化匹配方法对固件的脆弱哈希函数进行识别与定位.

图8 脆弱哈希函数识别与定位结果

3.2 脆弱哈希函数分析与破解实验

选择脆弱哈希函数BKDHash进行分析和破解,其他脆弱哈希函数的分析与破解方法与此过程相同,只需修改实验程序中的相关参数即可.

首先,找到前文输出的脆弱哈希函数的函数名为BKDHash和函数入口地址. 其次,使用该哈希函数加密输入的明文gdzhang,得到密文为1595684327. 然后,使用2.4节和2.5节的符号执行和添加求解约束条件的方法,进行哈希函数逆向暴力破解,求解该脆弱哈希函数的碰撞值. 由图9中的碰撞值可以看出,不仅求出了其他碰撞值,而且也求解出了原始输入数据. 最后,进行验证碰撞值,将输出的所有碰撞值带回BKDHash哈希函数进行加密. 经验证可知,得到所有输出碰撞值的加密值均与原始输入值的加密值相同.

图9 求解碰撞值

针对类似于BKDHash结构的其他脆弱哈希函数,通过输入不同长度值的明文进一步实验,发现也大都能够破解出其对应的碰撞值,而且破解值中一般都包含了原始输入值,如表3所示.

至此,证明了本文逆向破解脆弱哈希函数方法的可行性以及智能设备固件漏洞挖掘的实用性.

表3 脆弱哈希函数逆向破解展示

脆弱哈希函数	明文	密文	碰撞值个数	是否含输入值
RSHash	zgd	3688467305	1	是
JSHash	zhang123	3624299274	4 545	是
BKDRHash	zhangguodong	3605804677	1 447	是
SDBMHash	guod	4064947843	1	是
FNVHash	zzz	1097062274	1	是
BPHash	Zhang1Guo2Dong3	1308341171	16 151	是
DJBHash	ZHANG666	3464747871	2	是

4　结束语

为了有效挖掘和利用智能嵌入式设备固件的哈希函数漏洞，提出了一种嵌入式设备固件的脆弱哈希函数识别与破解方法，形成了从固件脆弱哈希函数的识别与定位到脆弱哈希函数的分析与破解的完整技术流程，提高了固件哈希函数漏洞挖掘的效率和准确率，为智能设备系统的安全保驾护航.

目前的工作还存在不足，下一步工作计划包括：

1）研究同种语言在多种编译环境下编译的二进制代码的中间语言转化，将开展多种编程语言编译的二进制代码的函数识别与破解研究.

2）破解的是一些极为脆弱的哈希函数，对于更强或非专业的加密算法，如消息摘要算法 5（MD5，message-digest algorithm 5）、安全散列算法 1（SHA-1，secure hash algorithm 1）、数据加密标准算法（DES，data encryption standard）、高级加密标准算法（AES，advanced encryption standard）还不能有效地进行逆向破解，将开展针对更复杂哈希函数的识别、定位和破解的研究.

参考文献：

[1] Yaniv D, Nimrod P, Eran Y. FirmUp: precise static detection of common vulnerabilities in firmware[C]// ASPLOS' 2018. New York: ACM Press, 2018: 392-404.

[2] Xu Xiaojun, Liu Chang, Feng Qian, et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]// CCS 2017. New York: ACM Press, 2017: 363-376.

[3] Feng Chao, Zhang Xing. A static taint detection method for stack overflow vulnerabilities in binaries[C]// ICISCE 2017. New York: IEEE Press, 2017: 110-114.

[4] Yu Bo, Yang Qiang, Song Congxi. State consistency checking for non-reentrant function based on taint assisted symbol execution[C]// SPNCE 2019. Berlin: Springer, 2019: 498-508.

[5] De Moura L, Bjørner N. Z3: an efficient SMT solver[C]// TACAS 2008. Berlin: Springer, 2008: 337-340.

[6] Ming Jiang, Pan Meng, Gao Ddebin. iBinHunt: binary hunting with inter-procedural control flow [C]// ICISC 2012. Berlin: Springer, 2013: 92-109.

[7] Dennis Yurichev. 逆向工程权威指南（下册）[M]. Archer, 译. 北京：人民邮电出版社, 2017: 722-729.