计算机科学
COMPUTER SCIENCE

基于IR2Vec模型的跨架构密码算法识别

赵晨霞，舒辉，沙子涵

引用本文

赵晨霞，舒辉，沙子涵.基于IR2Vec模型的跨架构密码算法识别[J]．计算机科学, 2023, 50(6A): 220100255-7.

ZHAO Chenxia, SHU Hui, SHA Zihan. Cross-architecture Cryptographic Algorithm Recognition Based on IR2Vec [J]. Computer Science, 2023, 50(6A): 220100255-7.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220100255

基于 IR2Vec 模型的跨架构密码算法识别

赵晨霞[1,2]　舒　辉[2]　沙子涵[2]
1 郑州大学网络空间安全学院　郑州 450001
2 信息工程大学数学工程与先进计算国家重点实验室　郑州 450001
(sidefacezcx@sina.com)

摘　要　在信息安全领域,加密技术被用来保障信息的安全性,在可执行文件中识别密码算法对于保护信息安全有着重要意义。现有密码算法识别技术大多只能针对单一架构,在跨架构场景下识别能力较差,因此,提出了 IR2Vec 模型,着力解决跨架构下的密码算法识别问题。该模型首先基于 LLVM 衔接不同的前端和后端的特性来解决跨架构的问题,利用 LLVM-RetDec 将可执行文件反编译成中间表示语言,然后改进 PV-DM 模型将中间表示语言语义向量化,通过求取向量的余弦距离来判断语义相似性。收集多种密码算法来建立密码算法库,将待检测目标可执行文件分别与密码算法库中的文件进行一一对比,取相似度最高的为识别结果。实验结果表明,该技术能够有效识别出可执行文件中的密码算法,该模型可同时支持 X86,ARM 和 MIPS 3 种架构,Clang 和 GCC 两种编译器,以及 O0,O1,O2 和 O3 这 4 种优化选项的二进制文件交叉识别。

关键词:相似性识别;跨架构;密码算法;LLVM
中图法分类号　TP393

Cross-architecture Cryptographic Algorithm Recognition Based on IR2Vec

ZHAO Chenxia[1,2],SHU Hui[2] and SHA Zihan[2]
1 School of Cyber Science and Engineering,Zhengzhou University,Zhengzhou 450001,China
2 State Key Laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450001,China

Abstract　In the field of information security,encryption technology is used to ensure the security of information. Identifying cryptographic algorithm in executable file is of great significance to protect information security. Most of the existing cryptographic algorithm recognition technologies can only target a single architecture and have poor recognition ability in cross-architecture scenarios. Therefore,this paper proposes IR2Vec model to solve the problem of cryptographic algorithm recognition in cross-architecture. Firstly,the model solves the cross-architecture problem based on the characteristics of LLVM connecting different front-end and back-end. The executable file is decompiled into the intermediate representation language by LLVM-RetDec,and then the PV-DM model is improved to quantify the semantics of the intermediate representation language,and the semantic similarity is judged by calculating the cosine distance of the vector. Collecting a variety of cryptographic algorithms to establish the cryptographic algorithm library,comparing the executable files of the target to be detected with the files in the cryptographic algorithm library one by one,and taking the one with the highest similarity as the recognition result. Experimental results show that the technology can effectively identify the cryptographic algorithm in the executable file. The model can support the cross recognition of binary files of X86,ARM and MIPS,Clang and GCC compilers and O0,O1,O2 and O3 optimization options.

Keywords　Similarity recognition,Cross-architecture,Cryptography algorithm,LLVM

1　引言

随着物联网技术的快速发展,为了保障数据安全性,加密技术的应用范围从单一架构的桌面程序扩展到了 X86,ARM,MIPS 等多架构下的物联网设备应用。在跨架构[1]可执行文件中识别出所用密码算法种类对于密码脆弱性分析、密码误用分析等具有基础支撑作用。

目前关于密码算法识别的技术主要基于静态特征、信息熵、控制结构、语义分析等方法。这些方法虽然各有优势,但其整体识别效果并不理想,目前的研究成果暂未发现哪一种单模型可同时满足跨架构、抗优化和跨编译器的需求。

本文提出了一种基于中间语言的语义相似性来识别二进制文件中密码算法的方法。实验结果表明利用中间语言可以有效解决跨架构识别的问题,并具备跨编译器、抗优化的密码算法识别能力。

本文主要贡献包括以下 3 个方面:

(1)提出了基于 LLVM-RetDec 反汇编生成的中间表示(intermediate representation,IR)文件进行语义分析,达到了在 X86,ARM 和 MIPS 多架构下跨架构识别的目的。

(2)根据不同架构下反编译生成的 IR 文件特点以及密码

基金项目:国家重点研发计划(2019QY1305)
This work was supported by the National Key R&D Program of China(2019QY1305).
通信作者:舒辉(shuhui123@126.com)

算法本身的特性，提出了一套跨架构 IR 文件预处理策略，解决了不同架构下二进制文件反编译的 IR 文件格式不统一的问题。

(3)在 PV-DM 模型的基础上进行改进，提出了 IR2Vec 模型。该模型将 IR 文件的指令语义嵌入，为跨架构密码算法识别提供了新的技术途径，并具有跨编译器和抗优化识别的能力。

2 相关工作

传统的密码算法识别和二进制代码相似性分析主要有以下几种方法。1)字符特征检测[2]虽然能够精准检测出密码算法的种类，但无法应对不含有字符特征或者变换后的密码算法程序。2)基于信息熵的检测[3-6]只能检测出是否存在密码算法，但无法识别具体的密码算法种类。3)基于控制结构的检测方法[7-13]过于依赖控制流图(Control Flow Graph，CFG)和数据流图(Data Flow Graph，DFG)，对抗混淆和优化的能力较差，精确度不高，很容易造成误判。例如 Genius[14]系统利用机器学习的谱聚类算法[15]来对函数的 CFG 进行分类生成码本，再进行编码，将相似函数的搜索问题转化为特征编码的搜索问题。Xu 等[16]在 Gemini 系统中基于 Structure2vec[17]提出利用图神经网络为二进制函数的 CFG 生成嵌入向量，然后通过余弦距离来度量函数间的相似性，但它没有解决混淆和跨编译器的问题。(4)基于语义分析[18-19]的研究方法是当前的研究热点。2019 年 Ding 等通过改进 PV-DM 实现了汇编函数语义嵌入，提出了 Asm2vec 模型[20]，通过计算包含语义信息的函数向量间的余弦距离以达到二进制文件相似性分析的目的，具有抗混淆和跨编译优化的特性，但是不支持跨架构检测。GeneDiff 模型[21]借助动态分析框架 Valgrind 的中间语言 VEX IR 并改进 PV-DM 模型来生成语义嵌入向量，进而求得函数间的相似度。SAFE 模型[22]利用自然语言处理中的 Word2vec 模型来实现汇编语言的指令嵌入，再利用循环神经网络(RNN)来捕获指令序列的上下文关系，最后引入自注意力机制。SAFE 只能根据特定指令架构的数据集训练形成单架构模型 i2v，严格意义上说，它不具备单模型的跨架构能力。

3 密码算法识别

建立包含各类密码算法的密码向量库，各密码算法分别由 GCC 和 Clang 编译器以 4 种不同的优化层次(O0-O3)[23]分别在 X86，ARM 以及 MIPS 这 3 种交叉编译链生成的二进制文件反编译成 IR 文件，并经过预处理，通过 IR2Vec 模型训练学习生成对应的向量。待检测的二进制文件同样经过反编译和预处理，通过 IR2Vec 模型生成相应的向量。最后通过将待检测文件生成的向量分别与密码向量库中的向量根据距离公式求取两个向量的距离来得到两个样本的相似度。划定一个阈值，当有多个相似度大于该阈值时，则认为相似度值最大的那个密码算法是目标样本的密码算法种类；如果密码向量库中没有样本的相似度值大于阈值，则认为识别失败或该目标样本中不包含任何密码算法。密码算法识别过程的整体框架如图 1 所示。

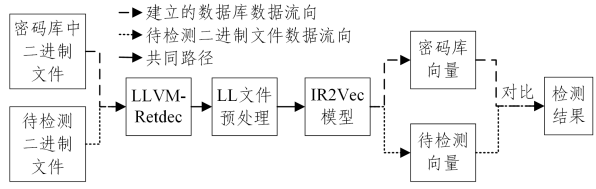


图 1 整体框架

Fig. 1 Overall framework

3.1 中间语言的预处理

由于 X86，ARM 和 MIPS 架构下的二进制文件本身就较源码损失了部分信息，且还增加了编译器优化和指令架构的特性，所以二进制文件通过 LLVM-Retdec 反编译成的 IR 文件在格式上也有差别，需要进行如图 2 所示的预处理，以适用于 IR2Vec 模型。

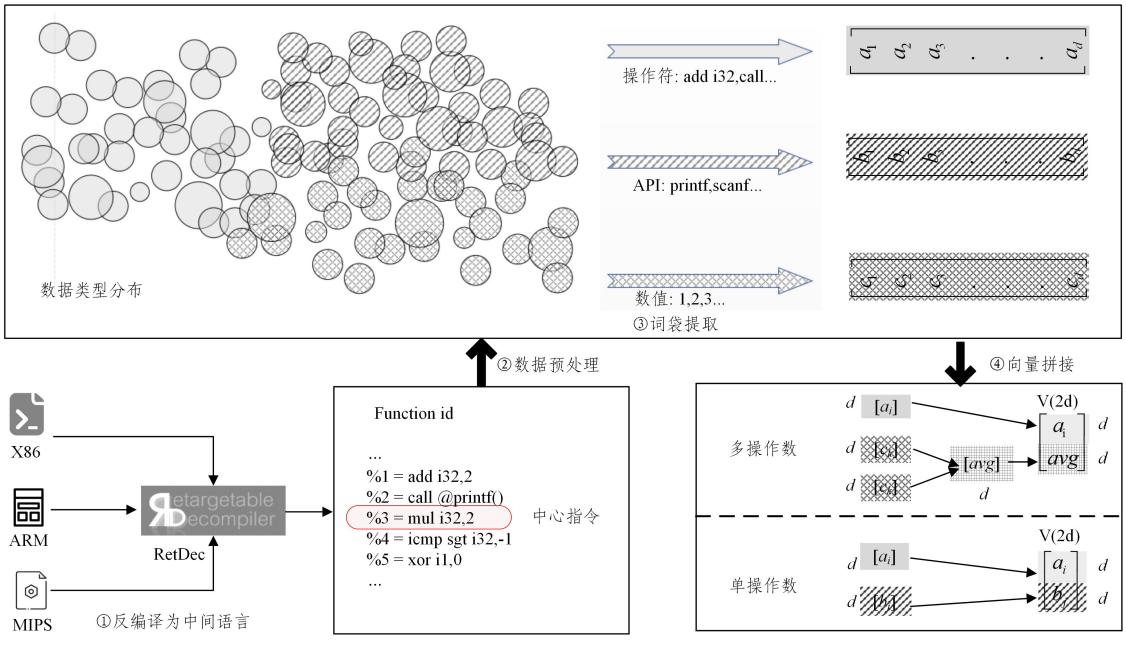


图 2 数据处理

Fig. 2 Data processing

对于 IR 文件指令的分析，因每个操作都有对应的数据类型，故将操作符和数据类型视为一个操作符整体，主要保留操作符、API 函数名以及数值 3 种指令 token。指令的操作符不同，其后所拼接的操作数也有所不同。主要分为多操作数和单操作数，具体方法见 2.2.2 节。

反编译生成的 IR 文件中函数体可读性较强，基本与源码中的函数体一一对应。为了减少程序间的跳转，便于滑动窗口的移动，当程序中出现函数调用时，将函数体替换调用指令。通过函数内联的方法改变控制流图以减少函数间跳转的复杂性。为避免内联后的函数代码过长，本文设置了函数调用深度的上限，当函数内部调用深度超过 10，则只取前 10 层的调用。对于子函数为 API 函数时，在指令处理中，将 API 函数名称作为 token 纳入词袋中，不再作内联处理。具体预处理步骤如图 3 所示。

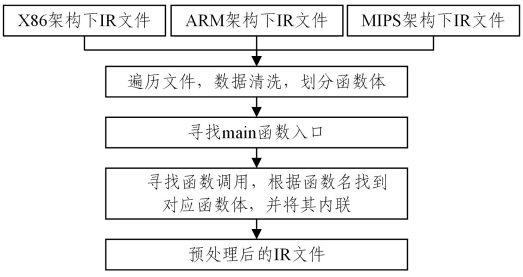


图 3　IR 文件预处理流程图

Fig. 3　IR file preprocessing flowchart

3.2　中间语言表示学习

本节提出了中间语言表示学习来解决跨架构相似性识别的问题。

IR2Vec 模型是基于 doc2vec 的 PV-DM 模型改进的，同时也参考了 Asm2Vec 模型设计的思想，将整个密码算法程序视作句子，指令视作单词。以上下文的指令来预测中心词的指令，如图 4 所示。

IR2vec 的处理与自然语言处理不同之处不仅在于表达形式和语法有很大不同，其执行的顺序也有很大区别：自然语言是顺序执行，没有跳转，而代码的执行是局部顺序执行，整体会根据需求进行基本块的跳转或者函数间的调用。IR2Vec 模型相比 Asm2Vec 模型最重要的改进在于：1）两者基于的目标程序语言不同，Asm2Vec 是面向汇编代码进行语义提取，而 IR2Vec 是面向 LLVM 中间语言代码进行语义提取；2）单次训练可支持的指令架构种类数量不同，Asm2vec 一次训练的模型仅支持单一架构的汇编代码，而 IR2Vec 可同时支持多种指令架构下的二进制程序；3）IR2Vec 考虑了密码算法程序 IR 指令的特点，为不同类型的 IR 指令制定不同处理策略，使获取的语义更准确。

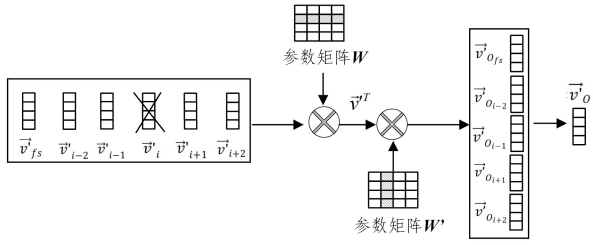


图 4　单次模型预测处理过程

Fig. 4　Single model prediction process

3.2.1　基本块

图 5 为 RC4 加密函数的源码与中间语言层面的对应基本块控制流图，分为：

（1）入口基本块：以入口基本块为程序执行起始点，分配内存以及初始化。

（2）for 循环条件判断基本块：判断循环条件是否满足，来决定跳转到循环体基本块或结束基本块。

（3）for 循环体基本块：执行循环内取余、异或等一系列操作，最后跳转到 for 循环条件判断基本块，以判断是否满足下一轮的循环条件。

（4）结束基本块：执行 ret 命令，返回函数计算结果。

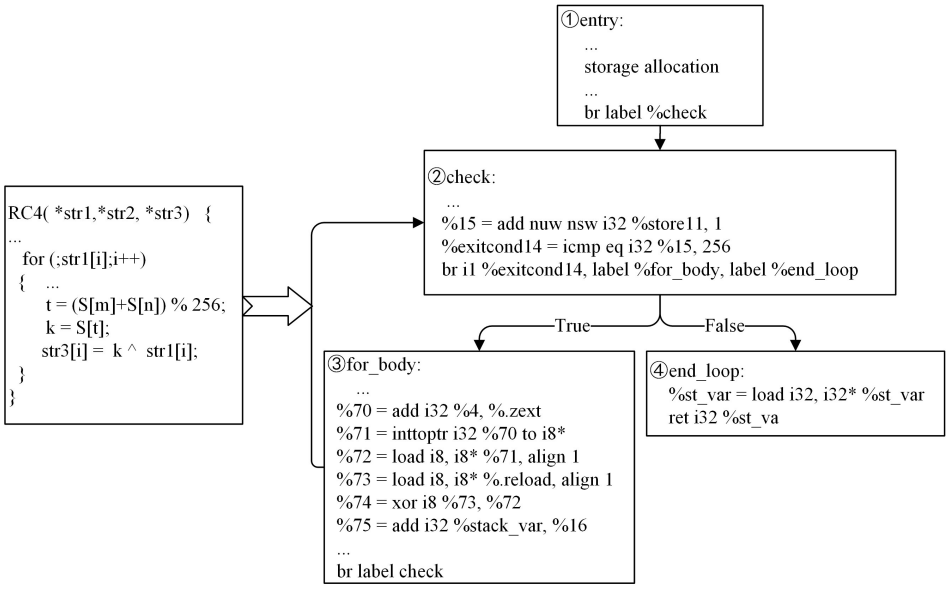


图 5　基本块控制流图

Fig. 5　Basic block control flow diagram

所有基本块都以 br 跳转指令或者 ret 返回指令结束。每一条完整的执行路径都是以结束基本块结尾。

对于上述示例的采样指令序列，有两条路径：

（1）①→②→③→②→③…→②→④：以入口基本块开始

遍历，途径 for 循环条件判断基本块，若结果为 true，进入 for 循环体基本块，再执行 for 循环条件判断基本块，直到结果为 false，进入结束基本块。

(2)①→②→④：以入口基本块开始遍历，途径 for 循环条件判断基本块，若结果为 false，进入结束基本块。

3.2.2 指令

以上述源码和对应的 IR 文件为例，对应的指令主要分为加载存储类指令、计算类指令、判断类指令、跳转类指令、类型转换类指令、结束类指令。IR 文件共有 64 种指令。以预处理之前的 ARM 架构下的 AES 密码算法为例，各类指令占比统计如表 1 所列。

表 1 指令占比统计

Table 1 Statistics of instruction proportion

(单位：%)

指令类型	所占比例
加载存储类(alloca，store，load)	55.750
计算类(add，mul，shl，sdiv，udiv，xor 等)	17.130
判断类(icmp sgt，icmp slt，icmp eq 等)	2.500
跳转类(br，call，switch)	12.625
类型转换类(ptrtoint，inttoptr，trunc，zext 等)	11.875
结束类(ret)	0.125

根据密码算法的语义特点，在加密和解密的核心过程中都包含有大量的计算、循环和判断。AES 算法的加密过程包括 10 轮的轮密钥加、字节代换、行移位、列混合、轮密钥加，解密过程包括轮密钥加、逆行移位、逆字节代换、逆列混合。所以密码算法的核心语义在计算类和判断类指令的表示中最为丰富，其次是跳转类指令。加载存储类和类型转换类的指令对于整体加解密过程的语义理解重要度不够，所以在预处理阶段将加载存储类指令按照相应数据的关系进行简化，去除类型转换类的指令。将跳转类指令中的 call 指令利用函数内联的方式将调用自定义函数的 call 指令消除。

(1)计算类指令

计算类指令本质是包含有操作符和操作数的数值计算，其向量表示由操作符和操作数拼接而成。d 代表向量维度，表示有 d 维向量特征。将操作符和操作数都视为 token，将操作数和操作符分别映射为向量 $\vec{v}_{\text{operand}} \in R^d$ 和 $\vec{v}_{\text{operator}} \in R^d$；指令映射为 $\vec{v}'_i \in R^{2\times d}$，由操作符 token 的向量 $\vec{v}_{\text{operator}}$ 和操作数 token 的平均向量 $\vec{v}_{\text{operand}}$ 拼接而成。

$$\vec{v}'(in)=\vec{v}_{\text{operator}}(in) \parallel \frac{1}{|operand(in)|}\sum_{t}^{operand(in)}\vec{v}_{\text{operand}_t} \tag{1}$$

其中，$\vec{v}_{\text{operator}}$ 代表操作符向量，$\vec{v}_{\text{operand}_t}$ 代表操作数向量，$\parallel$ 代表向量的拼接，$operand(in)$ 代表当前指令的操作数集合，$|operand(in)|$ 代表当前指令的操作数个数。

(2)判断类指令

IR 文件中的判断类指令其本质都是 icmp，只是判断对象的类型不同，判断是通过对比来实现，结果为 bool 类型。

计算类指令和判断类指令都可归结为赋值语句，将操作符 operator 和操作数 operand 计算得到的结果赋值给生成的变量。其本质都是数值计算，其向量表示可以由操作符和操作数拼接而成，故同式(1)。

(3)跳转类指令

br 指令为跳转类指令，一般分为单分支跳转和多分支跳转，单分支跳转一般为程序的循序执行，不涉及条件的判断。多分支跳转一般前一条指令为比较指令，其比较结果具有不确定性，根据比较结果的值来决定跳转到哪一个基本块。如果为 true，则跳转到第一个分支；如果为 false，则跳转到第二个分支。br 指令用来确定控制流的流向，并不生成实际向量。

判断类指令一般与 br 指令一起使用，由判断类指令的结果来决定 br 指令的跳转地址。

call 指令为函数调用指令，被调用的指令分为 API 和自定义函数两种情况，对于 API(如 printf 和 malloc 等)调用，模型不做过多深入处理。将 API 函数名看作一个 token，与其他 token 一起训练和更新。对于自定义函数，由于函数体内具有一定的计算过程，无法直接确定其向量，因此为了简化调用跳转，在首次遍历 main 函数时，会将自定义函数进行内联(实现细节可见 3.1)。所以不对调用自定义函数的指令进行处理，而是直接处理自定义函数内部的指令。

$$\vec{v}'(in)=\vec{v}_{\text{operator}}(in) \parallel \vec{v}_{API}(in) \tag{2}$$

其中，$\vec{v}_{API}$ 表示 API 的向量。

(4)加载存储类指令

一个变量值从产生到使用要经历 alloca 指令进行分配内存空间、store 指令为其赋值、load 指令将其取出使用这几个步骤。真正的执行离不开其中的任何一步，但在语义分析时，因为 alloca 指令包含语义对于程序分析无意义，故跳过；当 load 指令的操作数 2 和 store 指令的操作数 2 相同时，load 指令的向量由 load 操作符的向量和 store 指令存储的值(操作数 1)向量拼接而成。如果条件不成立，则视为取出的值没有被使用，故不生成向量。

$$\vec{v}'(inload)=\vec{v}_{\text{operator}}(in_{\text{load}}) \parallel \vec{v}_{\text{operand1}}(in_{\text{store}}) \text{ when } operand_2(in_{\text{load}})=operand_2(\text{in}_{\text{store}}) \tag{3}$$

其中，in_{load} 表示 load 指令，in_{store} 表示 store 指令。

(5)结束类指令

ret 指令标志着一个函数的结束，相当于自然语言中的句号、感叹号等结束标志。遇到 ret 指令则会产生一个结束向量。$\vec{v}_{\text{ret}}(in)$ 表示 ret 操作符的向量，$\vec{v}_{\text{retvalue}}(in)$ 表示 ret 返回的值向量。

$$\vec{v}'(in)=\vec{v}_{\text{ret}}(in) \parallel \vec{v}_{\text{retvalue}}(in) \tag{4}$$

将预构建密码库中每个文件的函数 f 映射到向量 $\vec{\theta}_f \in R^{2\times d}$。$\vec{\theta}_f$ 是在训练中需要学习的函数 fs 的向量表示。$\delta(in_j, f)$ 代表当前指令 in_j 的预测向量表示，取函数 f 的向量以及前后各两条指令向量 $\vec{v}'(in_{j-2})$，$\vec{v}'(in_{j-1})$，$\vec{v}'(in_{j+1})$ 和 $\vec{v}'(in_{j+2})$ 的平均值。

$$\delta(in_j,f)=\frac{1}{5}(\vec{v}'(in_{j-2})+\vec{v}'(in_{j-1})+\vec{\theta}_f+\vec{v}'(in_{j+1})+\vec{v}'(in_{j+2})) \tag{5}$$

3.2.3 梯度更新和反向传播

梯度更新主要分为两个阶段：计算指令级梯度和计算 token 级梯度。因为指令是由 token 组成，指令向量是具有实际

语义的表示向量，而 token 向量是指令向量表示中最基本的组成单位，故准确的 token 级向量可以帮助得到更加准确的 IR 指令语义向量。

(1)指令级梯度

指令向量$\vec{v}'_t$的梯度计算如下：

$$\frac{\partial E}{\partial \vec{v}_t'}=\| t=t_c \| -\frac{\vec{v}_t'}{\sum_j^n \delta(in_j,f)}\delta(in_j,f) \tag{6}$$

其中，$\| t=t_c \|$代表样本的判断预测结果 t 是否与正确的预测样本 t_c 一致，若一致，则 $\| t=t_c \|=1$，否则 $\| t=t_c \|=0$。E 表示损失函数，通过求取正确的预测结果在所有预测结果中的概率得到。

当训练集的词袋过大时，可以通过负采样的方法来减少计算量。模型从预测结果的词袋里随机采样 k 个负样本，计算正样本在所有样本(正样本和随机采样的负样本总和)中的概率，函数向量$\vec{v}'_{fs}$的梯度计算如下：

$$\frac{\partial E}{\partial \vec{v}_{fs}'}=\frac{S}{k+1}\| t_j=t_c \| -\frac{\vec{v}_t'}{\sum_j^{k+1}\delta(in_j,f)}\times\vec{v}_t' \tag{7}$$

其中，S 为采样函数，t_c代表正样本，t_j代表负样本。

(2)token 级梯度

根据对指令梯度的求取，可以得到指令前 d 维的操作符的梯度更新如式(8)所示，后 d 维的操作符的梯度更新如式(9)所示。

$$\frac{\partial E}{\partial \vec{v}_{\text{operator}}}=\left(\frac{\partial E}{\partial \vec{v}_t'}\right)[0:d-1] \tag{8}$$

$$\frac{\partial E}{\partial \vec{v}_{\text{operand}_i}}=\frac{1}{|operand|}\left(\frac{\partial E}{\partial \vec{v}_t'}\right)[d:2d-1] \tag{9}$$

(3)反向传播

模型基于反向传播来更新 $\boldsymbol{W}$ 和 $\boldsymbol{W}'$ 权重矩阵，在训练中根据梯度来决定参数更新的方向，通过学习率来控制每次更新的步长：

$$\vec{v}_i^{r+1}\leftarrow\vec{v}_i^{r}+\mu\frac{\partial E}{\partial \vec{v}} \tag{10}$$

其中，μ 代表学习率，$\vec{v}_i{}^r$代表第 r 轮训练后的指令向量。

4 测试

4.1 相关设置

模型基于 Torch 平台搭建，在 Kaggle 平台使用 GPU 进行模型训练。

模型测试环境如表 2 所列。测试数据集如表 3 所列。反编译工具为 ReDec。

表 2 测试环境配置

Table 2 Test environment configuration

Operating system	Ubuntu 20.04 LTS
Python	3.8
Torch	1.8.1

表 3 中共 13 种密码算法在 Linux-Intel 交叉编译链下使用两种不同的编译器(Clang3.8.0、GCC5.5.0)以及 4 种不同的优化层次(O0-O3)得到 X86 可执行文件，在 Linux-MIPS 以及 Linux-ARM 交叉编译链下使用 Clang3.8.0 和 4 种不同的优化层次得到 MIPS 和 ARM 的可执行文件，共 208 个不同的函数。

表 3 测试数据集

Table 3 Test datasets

Cryptography Category	Cryptography Algorithm
symmetric	AES，DES，IDEA，TwoFish，SM4，RC4
public-key	RSA，ECC
Hash	MD5，SHA1
Substitution	Affine，Caesar，Virginia

基于上述数据集进行以下 5 个实验：

实验 1 架构间(X86，ARM，MIPS)的识别

实验 2 不同编译器(GCC，Clang)的识别

实验 3 不同优化(O0-O3)的识别

实验 4 对压缩软件进行识别

实验 5 与 Asm2Vec 和 SAFE 模型进行对比

4.2 测试结果

在分析对比结果时，认为预先建立的密码库中密码与待识别的密码相似度最高的即为同一种密码算法。下面讨论在不同条件下，各实验结果的相似性对比。

实验 1 架构间识别

由图 6 可知，待识别密码算法使用 Clang 编译且不做任何优化的情况下，与对比算法为同种架构时的比较效果最为理想，除 TwoFish，AES 和 Affine 3 种密码算法的相似度没有达到 0.9，其余相似度基本都在 0.9 以上。待识别密码算法与对比算法为不同架构时的比较效果有所下降，但识别相似度仍高于 0.7。

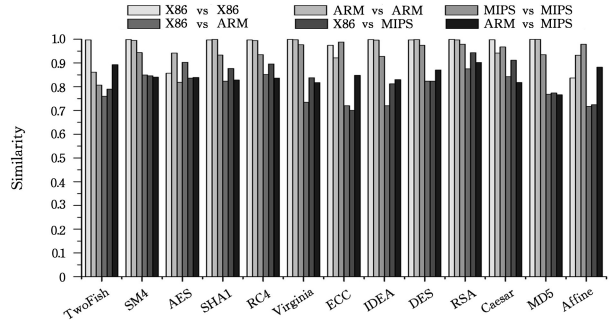


图 6 架构间对比

Fig. 6 Comparison between architectures

实验 2 不同编译器识别

由图 7 可知，待识别密码算法和对比算法分别使用 Clang 和 GCC 编译，在 X86 架构下 4 种优化层次(O0－O3)都能够被有效识别。

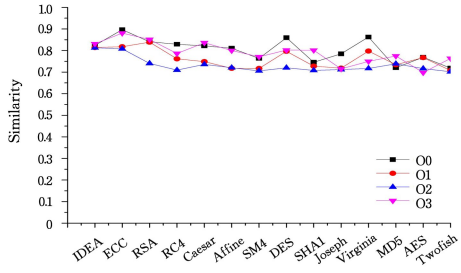


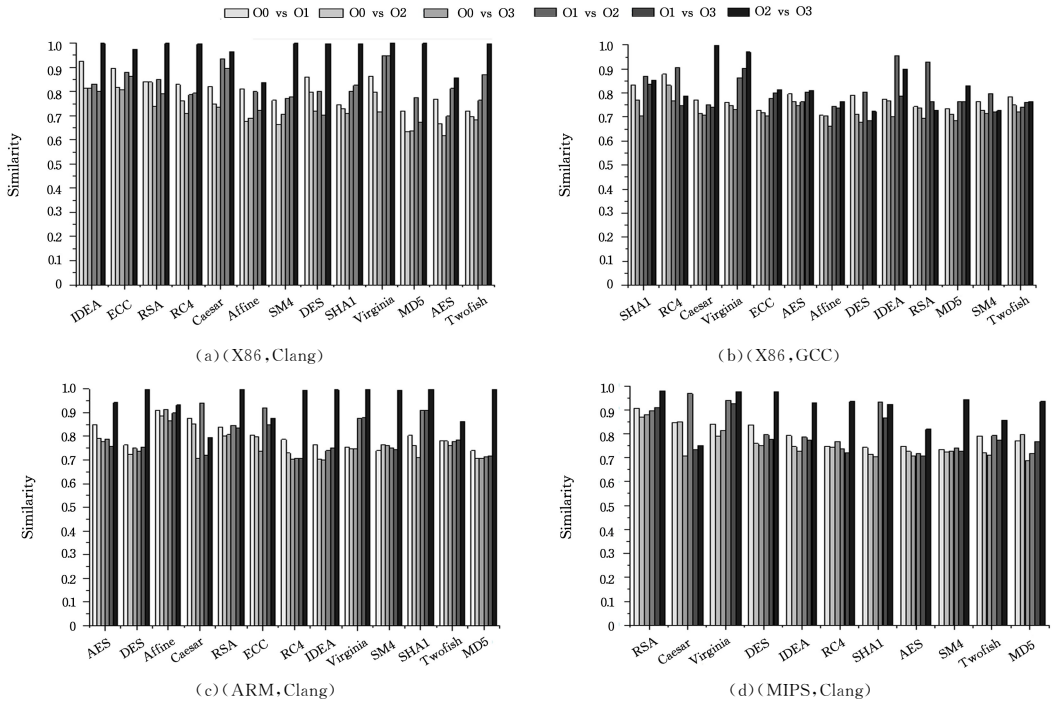
图 7 GCC 和 Clang 对比实验效果

Fig. 7 Comparison between GCC and Clang

实验 3 不同优化识别

在 X86，ARM，MIPS 3 种架构以及 Clang 和 GCC 两种编译器的两两组合(见图 8)下，待识别密码算法和对比算法分别使用不同优化层次，其相似度均可达到 0.7。其中 O2 和

O3 的识别效果最好，O0 和 O3 的识别效果最差。在 4 种组合中，X86 架构下使用 GCC 编译器（见图 8(b)）的识别相似度虽整体情况基本在 0.7 以上，但在 O2 和 O3 的对比中不如其他 3 种组合。



(a)(X86, Clang)　(b)(X86, GCC)

(c)(ARM, Clang)　(d)(MIPS, Clang)

图 8　不同优化对比

Fig. 8　Comparison of different optimizations

实验 4　对压缩软件进行识别

利用本模型和测试数据集对压缩软件 WinRAR6.0 和 FreeArc0.666 进行密码算法识别测试（见表 4），在 WinRAR6.0 中已知包含有 AES 密码算法，现识别出 AES 密码算法；在 FreeArc0.666 中已知包含有 AES 和 Twofish 密码算法，现识别出 AES 和 Twofish 密码算法。

表 4　压缩软件识别结果

Table 4　Compression software recognition results

Software	contained	identified
WinRAR6.0	AES	AES
FreeArc0.666	AES, Twofish	AES, Twofish

实验 5　与 Asm2Vec 和 SAFE 模型的对比

(1) 识别效果对比

在单一架构下的相同测试数据集上采用精确率（Precision）、召回率（Recall）、F1 值（F1-Measure）以及受试者工作特征 ROC（Receiver Operating Characteristic）作为评价指标来评估测试数据集上的 IR2Vec, Asm2Vec 和 SAFE。

表 5　IR2Vec 和 Asm2vec, SAFE 的识别效果对比

Table 5　Recognition effect comparison of IR2Vec, Asm2vec and SAFE

Model	*P*	*R*	*F*1
IR2vec	0.860	0.932	0.889
Asm2vec	0.764	0.690	0.745
SAFE	0.910	0.930	0.920

在相同测试集上进行实验比较，由表 5 和图 9 可知：1）IR2Vec 的识别效果明显优于 Asm2Vec，这是因为 Asm2Vec 模型只是对汇编指令做了语义嵌入，而 IR2Vec 对中间语言指令做了更加细致的处理，所以在语义理解时更加准确；2）IR2Vec 的识别效果略低于 SAFE，因为 SAFE 中引入了注意力机制。

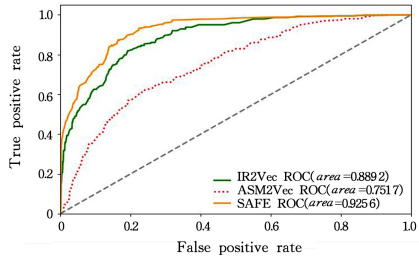


图 9　IR2Vec, Asm2Vec, SAFE 的 ROC 对比

Fig. 9　ROC comparison of IR2Vec, Asm2Vec and SAFE

(2) 能力对比分析

3 种模型使用的基础工具不同：IR2Vec 模型使用 LLVM-RetDec 进行反编译；Asm2Vec 模型使用 IDA 进行反汇编；SAFE 使用 Angr/Radare2 进行反编译。Asm2Vec 和 SAFE 具备抗编译优化和跨编译器识别的能力，而根据实验 2 和实验 3，IR2Vec 同样支持抗优化和跨编译器识别。跨架构识别方面，IR2Vec 模型实现了单个模型的跨架构识别；Asm2Vec 模型没有实现跨架构识别；SAFE 对于不同平台的识别对象需训练不同的 i2v 模型，训练较为复杂，没有实现真正意义上的跨架构。IR2Vec 和 Asm2vec 以及 SAFE 的能力对比如表 6 所列。

表 6　IR2Vec, Asm2vec, SAFE 的能力对比

Table 6　Capabilitiy comparison of IR2Vec, Asm2vec and SAFE

Model	IR2Vec	Asm2vec	SAFE
Across-architecture	√	×	×
Anti-optimize	√	√	√
Across-compiler	√	√	√
Tool	RetDec	IDA	Radare2
Training-Complexity	Simple	Simple	Complex

注：√代表已实现，×代表未实现

结束语 密码算法实现的形式复杂多样,应用的范围十分广泛,同一种密码算法可以有不同的语法形态,可以在不同的应用中出现,使用单一的特征提取具有很大的局限性。目前的密码算法识别技术识别效率低和识别准确度还有待提高,下一步可以引入自注意力机制提高检测效率。另外,该密码算法识别技术对识别体积较大的软件并不友好,容易出现内存溢出,针对该情况还需要进一步的优化。

参 考 文 献

[1] ESCHWEILER S,YAKDAN K,GERHARDS-PADILLA E. discovRE:Efficient Cross-Architecture Identification of Bugs in Binary Code[C]// The Network and Distributed System Security Symposium(NDSS 2016). 2016.

[2] LU T L,WU J,BAO Y,et al. Computer virus analysis and simulation based on cryptographic algorithm detection[J]. Computer Simulation,2020,37(11):173-178.

[3] MATENAAR F,WICHMANN A,LEDER F,et al. CIS:The Crypto Intelligence System for automatic detection and localization of cryptographic functions in current malware[C]// International Conference on Malicious & Unwanted Software. IEEE,2012.

[4] LI X,WANG X,CHANG W. CipherXRay:Exposing Cryptographic Operations and Transient Secrets from Monitored Binary Execution[J]. IEEE Transactions on Dependable & Secure Computing,2014,11(2):101-114.

[5] HILL G D,BELLEKENS X. Deep Learning Based Cryptographic Primitive Classification[J]. arXiv:1709.08385,2017.

[6] HILL,GREGORY,BELLEKENS,et al. CryptoKnight:Generating and Modelling Compiled Cryptographic Primitives[J]. Information,2018,9(9):231.

[7] CALVET J,FERNANDEZ J M,MARION J Y. Aligot Cryptographic Function Identification in Obfuscated Binary Programs [C]// Proceeding of the 2012 ACM Conference on Computer and Communications Security. 2012:169-182.

[8] LI J Z,JIANG L H,SHU H,et al. Cryptographic function screening based on dynamic cyclic information entropy[J]. Computer Applications,2014,34(4):1025-1028,1033.

[9] LI J Z,JIANG L H,SHU H. Binary code level cryptographic algorithm cyclic feature recognition[J]. Computer Engineering and Design,2014,35(8):2628-2632.

[10] XU D,JIANG M,WU D. Cryptographic Function Detection in Obfuscated Binaries via Bit-Precise Symbolic Loop Mapping [C]// 2017 IEEE Symposium on Security and Privacy(SP). IEEE,2017.

[11] BENEDETTIA L,AURÉLIEN T,CYBERSECURITY J F A. Detection of cryptographic algorithms with grap[J/OL]. https://github.com/AirbusCyber/grap.

[12] LESTRINGANT P,GUIHÉRY F,FOUQUE P A. Automated Identification of Cryptographic Primitives in Binary Code with Data Flow Graph Isomorphism[C]// Proceedings of the 10th ACM Symposium on Information,Computer and Communications Security. 2015:203-214.

[13] MEIJER C,MOONSAMY V,WETZELS J. Where's Crypto?: Automated Identification and Classification of Proprietary Cryptographic Primitives in Binary Code[C]// USNIX Security Symposium. 2021:555-572.

[14] QIAN F,ZHOU R,XU C,et al. Scalable Graph-based Bug Search for Firmware Images[C]// ACM Sigsac Conference on Computer & Communications Security. 2016:480-491.

[15] NG A Y,JORDAN M I,WEISS Y,et al. On Spectral Clustering:Analysis and an algorithm[C]// Advances in Neural Information Processing Systems. 2002:849-856.

[16] XU X,LIU C,FENG Q,et al. Neural network-based graph embedding for cross-platform binary code similarity detection[C]// Proceedings of the 2017 ACM SIGSAC Confe-rence on Computer and Communications Security. 2017:363-376.

[17] DAI H,DAI B,SONG L. Discriminative embeddings of latent variable models for structured data[C]// International Conference on Machine Learning. 2016:2702-2711.

[18] LAGEMAN N,KILMER E D,WALLS R J,et al. BinDNN:Resilient Function Matching Using Deep Learning[C]// International Conference on Security and Privacy in Communication Systems. Cham:Springer,2016.

[19] HU Y,ZHANG Y,LI J,et al. Binary Code Clone Detection across Architectures and Compiling Configurations[C]// IEEE/ACM International Conference on Program Comprehension. IEEE Computer Society,2017.

[20] DING S H H,FUNG B C M,CHARLAND P. Asm2vec:Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization[C]// 2019 IEEE Symposium on Security and Privacy(SP). IEEE,2019:472-489.

[21] LUO Z,WANG B,TANG Y,et al. Semantic-Based Representation Binary Clone Detection for Cross-Architectures in the Internet of Things[J]. Applied Sciences,2019,9(16):3283.

[22] MASSARELLI L,LUNA G,PETRONI F,et al. SAFE:Self-Attentive Function Embeddings for Binary Similarity[C]// International Conference on Detection of Intrusions and Malware,and Vulnerability Assessment (DIMVA 2019). 2019:309-329.

[23] REN X,HO M,MING J,et al. Unleashing the Hidden Power of Compiler Optimization on Binary Code Difference:An Empirical Study[C]// Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. 2021:142-157.

ZHAO Chenxia, born in 1995, master candidate. Her main research interests include cyber security and reverse engineering.

SHU Hui, born in 1974, Ph.D, professor, Ph.D supervisor. His main research interests include cyber security and reverse engineering.