计算机科学
COMPUTER SCIENCE

基于语义的多架构二进制函数名预测方法

邵文强, 蔡瑞杰, 宋恩舟, 郭茜茜, 刘胜利

引用本文

邵文强, 蔡瑞杰, 宋恩舟, 郭茜茜, 刘胜利. 基于语义的多架构二进制函数名预测方法[J]. 计算机科学, 2023, 50(10): 369-376.

SHAO Wenqiang, CAI Ruijie, SONG Enzhou, GUO Xixi, LIU Shengli. Semantic-based Multi-architecture Binary Function Name Prediction Method [J]. Computer Science, 2023, 50(10): 369-376.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220800175

基于语义的多架构二进制函数名预测方法

邵文强[1] 蔡瑞杰[1] 宋恩舟[2] 郭茜茜[1] 刘胜利[1]
1 数学工程与先进计算国家重点实验室 郑州 450001
2 国家数字交换系统工程技术研究中心 郑州 450001
(zzuswq@126.com)

摘 要 丰富的可读性源信息对逆向工作具有重要意义,尤其是高质量的函数名对程序理解非常重要。然而,软件发布者无论是出于防止逆向或者精简软件大小的角度,往往会发布剥离掉源级调试信息的可执行文件,可读性信息缺失导致逆向分析难度加大。因此,提出了一种多架构函数名预测(Multi-architecture Function Name Prediction,MFNP)方法,利用 LLVM RetDec 反编译 X86,ARM,MIPS 架构的二进制文件为中间语言(IR)文件解决不同架构之间存在差异的问题。对中间语言.ll 文件中的函数名进行形态上、语义上的相似性比较,对函数名进行相似性融合来降低函数名数据稀疏性。将携带顺序指令语义信息的基本块以及以基本块为基本单位的函数体控制流图作为函数体的语义特征,结合神经网络来实现 X86,MIPS,ARM 这 3 种架构下剥离二进制文件的函数名预测。相比 DEBIN,所提方法额外支持 MIPS 架构下的剥离二进制函数名预测工作,其在 Precision 和 F1 方面相比 NERO 提高了 13.86%和 11.93%。最后验证了 MFNP 选用以基本块为基本单位提取的顺序指令序列和控制流图作为语义特征的有效性。

关键词:静态二进制分析;中间表示;函数名称预测;自然语言处理;神经网络
中图法分类号 TP311

Semantic-based Multi-architecture Binary Function Name Prediction Method

SHAO Wenqiang[1],CAI Ruijie[1],SONG Enzhou[2],GUO Xixi[1] and LIU Shengli[1]
1 State Key Laboratory of Mathematical Engineering and Advanced Computing,Zhengzhou 450001,China
2 National Digital Switching System Engineering Technological R&D Center,Zhengzhou 450001,China

Abstract Rich readable source information is important for reverse work,especially high-quality function names are important for program understanding. However,software publishers often release executable stripped of source-level debugging information,either to prevent reversals or to streamline the size of the software,which makes reverse analysis more difficult due to the lack of readable information. To this end,a multi-architecture function name prediction(MFNP) method is proposed to resolve the differences between architectures using LLVM RetDec to decompile X86,ARM,and MIPS architecture binaries into intermediate language(IR) files. Morphological and semantic similarity comparison of function names in readable intermediate language. ll files,and similarity fusion of function names to reduce function name data sparsity. The basic blocks carrying the semantic information of sequential instructions and the control flow graph of function bodies with basic blocks as the basic units are used as semantic features of function bodies,combined with neural networks to achieve function name prediction of stripped binaries in three architectures,X86,MIPS and ARM. Compared to DEBIN,it additionally supports stripped binary function name prediction work under MIPS architecture DEBIN. The improvement in Precision and F1 is 13.86% and 11.93% respectively compared with NERO. The effectiveness of MFNP in selecting sequential instruction sequences and control flow graphs extracted with basic blocks as the basic unit as semantic features is verified.

Keywords Static binary analysis,Intermediate representations,Function name prediction,Natural language processing,Neural networks

到稿日期:2022-08-17 返修日期:2022-11-26
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(mr_liushengli@163.com)

1 引言

逆向分析技术是研究恶意软件的重要方法。安全分析中,对逆向分析技术的要求很高,即使是经验颇丰的专业人员也需要投入大量时间才能获得有价值的结果[1]。逆向分析难度大,主要是由于代码审计过程中会遇到大量可读性差、无法快速准确理解含义的汇编代码。大多数商业软件(COTS)都是闭源的,并且为了精简其发行软件大小,通常会发行剥离掉调试信息的二进制文件。为了防止逆向分析,一些恶意的二进制文件甚至会故意剥离关键源级信息。剥离的二进制文件仅包含指令、寄存器等难以阅读和理解的低级信息,使得逆向分析难度加大。函数名往往是一个程序功能的缩写[2-3],很多工程师会通过函数名猜测函数体的功能。因此,通过性能优越的反编译器得到有意义的函数名,可以大大降低逆向的难度[4-7]。

业界通过将汇编代码提升为伪 C 代码来提高可读性[8-9]。反编译效果较好的反编译器有 Hex-Rays Decompiler 和 RetDec,它们通过重构汇编代码中的变量以及函数信息来生成更加易读的文本。近年来,借助变量、变量类型等信息来与源代码中的函数名建立联系的工作发展良好[10]。然而,现有针对二进制文件研究的工作较少。DEBIN[11]将二进制代码提升到 IR,构造变量依赖图,通过条件随机场(CRFs)方法实现函数名预测,支持 X86,X64,ARM 3 种架构。NERO[12]针对剥离的二进制文件预测函数名,基于编码器-解码器实现,编码器采用图神经网络(Graph Neural Network,GNN),解码器使用长短期记忆(Long Short-Term Memory,LSTM)网络实现。现有工作难以预测出合适的函数名称,即使是商用反编译软件 IDA Pro,在函数名窗口中也仅仅是以 sub_402C1F,sub_403A96 等无意义的字符串指代原有函数名。DEBIN 使用条件随机场来实现 X86,X64,ARM 架构的函数名的预测,由于其未使用神经网络,因此该工作预测函数名仅仅支持预测在其模型训练过程中出现过的函数名,这是非神经网络固有的缺陷。NERO 基于编码器-解码器范式实现函数名的预测,NERO 依赖于函数内部的函数调用关系,由于并不是所有的函数都会存在内部、外部的函数调用,因此,这样的特征选择限制了 NERO 的适用性。

从本质上来说,函数名预测是一个从代码语言到自然语言的翻译问题。以往的研究基于指令序列级别开展。输入序列与输出序列之间的长度不对等,尤其是函数名的长度变化较小,函数体的变化较大。针对函数名预测工作,MFNP 首先提取出函数体的控制流图数据,再对基本块中的无关语义进行精简,最终生成可以代表函数体的顺序结构的语义信息。为了证明实验结果具有较强的鲁棒性,该方法利用一个规模较大的数据集进行验证。

针对多架构的剥离二进制文件的函数名预测,本文提出了一种基于语义的多架构二进制函数名预测方法来完成多架构的剥离二进制文件的函数名预测。首先,通过 LLVM 将不同架构的二进制文件代码提升到中间语言代码层面,来消除不同架构之间的差异性。然后利用正则表达式提取出函数名和函数体信息。利用大规模数据的训练得到正负样本函数名二分类模型,去除无意义函数名,再对得到的函数名数据进行形态上、语义上的相似性融合处理。通过对函数体进行标准化处理,从而降低函数体信息的稀疏性,提升函数体的顺序结构信息的质量,保证编码解码的正确性。将处理后的函数名和函数体数据输入 seq2seq 模型中,得到一个模型,该模型能够提供语义更为准确的函数名。

本文的主要贡献如下:

1)结合神经网络技术中的机器翻译模型,分别对 X86,ARM,MIPS 3 种架构进行训练并得到函数名预测模型,可以减少处理不同架构下剥离二进制文件函数名预测所需的工作量。并在 X86,ARM,MIPS 3 种架构下进行了评估,均取得了较好的效果。

2)提出了一种新的函数体的语义特征表示方法。以函数体中的基本块为单位构建函数体的控制流图,对其中的基本块进行关键语义处理工作。

3)设计并实现了 MFNP 系统。由 LLVM-RetDec 实现不同架构的二进制文件到中间语言层面的提升,以此来消除不同架构之间的差异。最终,借助 OpenNMT 实现指令序列到函数名字符串的翻译工作。

2 相关工作

编译器存在的意义是将编程人员写的各种语言的源代码编译为机器能够理解的机器代码。中间表示相比源代码是更加低级的,也是更加接近机器代码的。通常,中间表示是独立的语言,其内容紧凑、均匀,包含控制流信息,因此适合进行静态分析。本文选择的编译器为底层虚拟机(LLVM),LLVM 能够支持多种编程语言,也可以工作于多种架构之上。LLVM IR 支持与汇编语言类似的三地址码格式。LLVM 中有跳转指令 br 和调用指令 call,其中的基本块由一段指令流组成。静态单一赋值(SSA)机制下的 IR 对每个变量仅进行一次赋值操作。SSA 机制下,每一个被使用的变量,仅有一个定义可以到达,因此 UD 链(Use-Define Chain)十分清晰。同时,SSA 机制也导致生成的 LLVM IR 行数往往是相应 C 源代码行数的 10 倍。SSA 可以简化数据流分析,数据流值传播不局限于控制流路径。SSA 机制的引入,简化了编译优化的过程,从而可以获得更好的优化结果。

自然语言与机器指令之间的相似性,启发了早期的安全研究人员[13-14],他们尝试将自然语言领域中成熟的模型应用到安全领域中。DEBIN 是一个预测剥离二进制文件的调试信息系统,它支持 X86 和 ARM 两种架构。DEBIN 通过条件随机场中的概率模型实现调试信息预测。由于 DEBIN 未使用神经网络实现函数名的预测,所以不具备预测新词的能力。此外,DEBIN 判别预测结果成功的条件为:DEBIN 预测出的函数名与真实函数名完全匹配。然而,大多数函数名是由很多个子串组合而成的,因此这样的判别标准是不合理的。

code2vec[15]基于注意力的神经网络,学习固定长度连续向量表示的任意大小的代码片段。在抽象语法树的句法路径上使用软注意力机制,将语法树中所有路径向量汇聚到一个

向量中来实现函数名称预测。code2vec[15]的相对简单性和分布式特性保证了函数名推测的泛化能力，基于注意力机制，预测结果是可解释的，该向量可用于预测代码段的语义属性。punstrip[16]结合二进制代码的概率指纹和概率图形模型，实现了在多个编译器和优化级别中预测函数名。同时，其也利用了 word2vec 中的 CBOW 和 skip-gram 来实现在剥离二进制文件中根据代码结构语义相似性来预测函数名。

Nero[12]通过对二进制文件进行静态分析获得调用站点信息，将 GNN 和 LSTM 分别作为编码器和解码器实现程序函数名称的预测。该方法的实现基于调用站点实现，先是处理程序的控制流图(CFG)，再对调用站点信息进行重构，借助具体和抽象的数值实现调用站点的增强表示。因为其仅在小规模的样本上进行测试，无法证明该方法的适用性，且其过于依赖函数中的调用关系，导致函数名预测的适用性受限。

NFRE[17]提出了一种轻量级框架，实现了 X86 架构的函数名预测工作，其提取控制流图子图基本块特征作为函数体的语义特征。但是，其提取的特征粒度过细导致编码器所编码的指令级别数据过大，因此解码器输出的函数名长度较小。本文选择函数体中以基本块为基本单位的控制流图作为实验数据粒度，取得了良好的实验效果。

MFNP，NERO 以及 DEBIN 均支持基本的函数名预测。然而，由于 DEBIN 未使用神经网络，因此无法对未出现在训练集中的函数名和函数体数据进行预测。DEBIN 支持多架构函数名预测，其支持 X86 和 ARM 架构，但是不支持 MIPS 架构。NERO 基于 X86 架构下的数据集进行训练、验证，最终实现函数名预测。NERO 仅支持 X86 架构下的函数名预测。DEBIN 和 NERO 选用的数据集均来源广泛，包含各种编译优化选项的二进制文件。MFNP 所选用的数据样本是经过 O0，O1，O2，O3 编译优化选项得到的多编译优化选项的二进制数据文件。因此 DEBIN，NERO 以及 MFNP 都支持多编译优化选项二进制文件的函数名预测。DEBIN 工作使用的基础工具是 BAP-IR，NERO 工作所使用的基础工具是 IDA，本文提出的 MFNP 使用的基础工具是 LLVM-RetDec。MFNP，NERO 以及 DEBIN 的对比情况如表 1 所列。其中 DEBIN 涉及的特征过多，包括二进制提升、数据流分析等，因此工作过程耗时较长。本文提出的 MFNP 通过 LLVM-RetDec 提升二进制文件的过程，也消耗了很多时间。MFNP 后续的函数名预测训练过程耗时较短。

表 1 MFNP，NERO，DEBIN 对比

Table 1 Comparison of MFNP，NERO and DEBIN

	MFNP	NERO	DEBIN
多架构	Y	N	Y
预测新函数名	Y	Y	N
多优化	Y	Y	Y
基础工具	LLVM-RetDec	IDA	BAP-IR

注：Y 代表已经实现，N 代表未实现。

3 MFNP 预测函数名工作框架

3.1 系统设计

1)数据提取、训练阶段

本文提出的 MFNP 的整体架构图如图 1 所示。MFNP 首先将多架构的二进制数据样本通过 LLVM-RetDec 提升到中间语言层面来生成中间语言文件。在得到中间语言文件之后，MFNP 选择其中的中间语言.ll 文件，从中间语言.ll 文件中提取出函数名以及函数体信息，对函数名进行去除噪音数据、降低稀疏性的处理，对函数体进行语义特征提取，其中包括控制流图和指令序列；从中间语言.ll 文件中提取出函数名、函数体并构成键值对信息存入 json 文件中，具体实现算法如算法 1 所示。将处理过的 X86，ARM，MIPS 架构对应的函数名和函数体数据输入到机器翻译模型 OpenNMT 中进行训练，分别得到各个架构的函数名预测模型并保存。

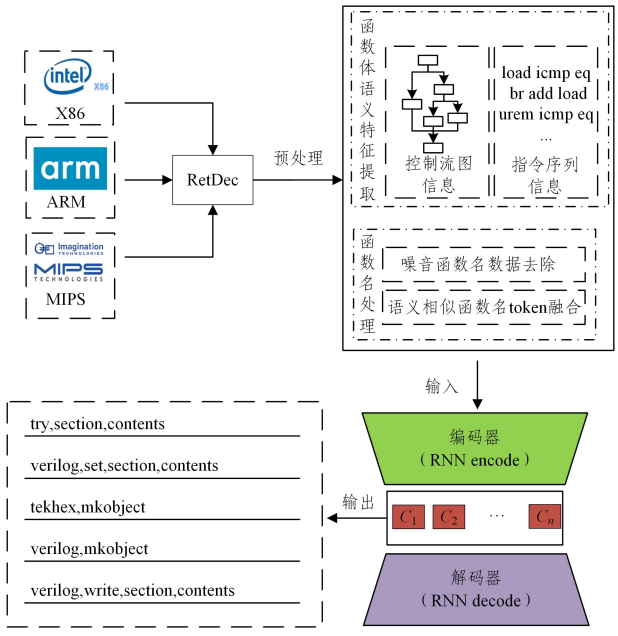


图 1 MFNP 整体架构图

Fig. 1 Overall architecture of MFNP

算法 1 函数名、函数体提取算法

输入：中间语言.ll 文件

输出：存储函数名、函数体键值对的 json 文件

```
fun_info=get_info(record_path);/*提取.ll文件路径*/
for i in range(len(fun_info)):
    func_body_get_ll_info(fun_info[i]);/*递归提取.ll文件中函数名、函数体*/
func_name_body.append({"func_name":func_name,""unc_body":func_body});/*添加函数名、函数体信息到列表*/
write_info_to_json();/*写入初次提取的函数名、函数体信息到json文件*/
read_json_file(json_path);/*读取json中函数名、函数体信息*/
func_name=data[i]['func_name'][0]
name1=bernoulli_name(func_name);/*二分类模型实现函数名噪音数据去除*/
name2=levenshtein(name1);/*莱文斯坦实现语义相似函数名token融合*/
name3=wordnet(name2);/*wordnet实现语义相似函数名token融合*/
normal_func_body=info_work(data[i]['func_body'][0]);/*标准化处理json中函数体信息*/
normal_func.append({"func_name":name3,"func_body":normal_
```

```
func_body});/* 标准化处理 json 中函数体信息 */
write_info_to_json();/* 写入函数名、经过标准化的函数体信息到 json 文件 */
```

2)应用阶段

将一个剥离掉函数名信息的二进制文件输入 MFNP 中，根据该二进制文件的架构信息，选择对应的函数名预测模型来完成函数名预测工作，最终输出一些函数名数据。

3.2 函数名噪音数据去除

数据集中的一些函数名是没有意义的，如 function_6310，jfioew，helojsh。此类无意义的函数名数据被称为函数名噪音数据，此类噪音数据的存在会影响模型的性能。无意义函数名的去除工作，本质上是一个数据二分类工作。启发于垃圾短信分类工作，本文选用垃圾短信分类工作中的成熟二分类技术完成函数名噪音数据去除，通过 Python 编写脚本随机生成字符串作为函数名噪音数据去除其中的正样本数据。负样本数据通过 B2SMatcher[18] 分享的大量可用的开源 C 语言库获得，选择在主流的 binutils，findutils，diffutils，coreutils，vim，openssl 等开源工具包中提取函数名来作为函数名的负样本数据。由于正、负样本函数名分类工作并不复杂，MFNP 选择朴素贝叶斯的伯努利二分类模型来实现二分类工作，分类的正确率为 91.87%。在通过朴素贝叶斯的伯努利二分类模型完成无意义函数名去除之后，为了保证函数名数据的质量，后续也需要进行人工的二次筛查工作。

3.3 语义相似函数名 token 融合

为使函数所实现的功能更易于理解，大多数程序开发者在进行函数命名时会采用蛇形命名法或者驼峰命名法。受 NERO[12] 的启发，本文基于编码器-解码器范式实现函数名预测工作。将函数名预测工作视为一个机器翻译工作，在预实验中函数名预测结果中出现了〈unk〉(Unknown Words)，这样的输出结果称为 OOV，又称为未登录词。未登录词指在测试过程中出现了训练过程中未遇到的词汇。机器翻译模型中的解码器会比较在相应条件下单词出现概率的大小，并选择输出概率较大的单词。MFNP 所使用的机器翻译模型为了获得更高的得分，会选择出现次数较多的一些单词作为词汇表，其余单词用 unk 代替。这样的机制会导致一些不常出现但具有实际意义的函数名 token 因得分较低而被忽略。出现这样的情况，原因是函数名 token 的稀疏性过大。通过合并语义相似的函数名 token 可以降低函数名 token 的稀疏性。

降低函数名的稀疏性，是避免 OOV 出现的有效手段。在去除数据集中的无意义函数名之后，可以将函数名 token 数据集中较为相似的 token 进行相似语义 token 融合来降低函数名 token 稀疏性。函数名 token 的相似性融合工作，可以分别通过函数名 token 的形态相似和语义相似来完成。一个可执行程序中往往包含大量的函数名，相同含义的函数名可以通过不同形式的函数命名进行表达，不同命名形式的函数名可能具有相同的含义，比如 seed_to_aim 和 seed_to_object。长度较长的函数名 token 更容易理解，因此，如果两个函数名在比较之后，满足形态上或者语义上的相似，就取其中长度较长的函数名 token 作为二者的函数名 token。将函数名划分的 token 视为英文字符串，可以从形态相似、语义相似两个角度实现语义相似的 token 合并。第一种情况为形态相似的函数名 token，如 version 和 versions、type 和 types；第二种情况为语义相似的函数名 token，如 aim，object，target 以及 shut，close，这样的单词语义相似，但无法从形态上进行相似性判断。

使用莱文斯坦距离技术融合两个形态相似的函数名 token。两个字符串由不同变为相同所进行的增加、替换、删除字符的编辑次数总和称为编辑距离，编辑距离是判断两个字符串之间相似程度的度量指标。本文借助 WordNet 来实现两个语义相似但形态不相似的函数名 token 的相似性比较。Wordnet 是一个世界级的大型英文词汇库，该词汇库由很多个同义词集合组成。同义词集合中的词汇是跨词性的，包含动词、名词、副词、形容词。同义词集合之间通过词性关系、语义关系连接在一起。不同于常见的顺序排列词典，Wordnet 根据单词的实际意义，形成并存储一个语义单词网络。单词之间通过同义词关系实现连接，比如 aim 和 object 此类形态上完全不相似的单词，被存储在相同的同义词集合中，并且同义词集合中的单词是无序存储的。通过 Wordnet 实现两个单词的语义相似性比较步骤如下。1)获取两个单词的最低共同上位词集。上位词：比较当前词汇更加泛化的一个词。例如，Color 是 Red 的上位词。这是一种语言上的逻辑关系。2)得到最低共同上位子集之后，再分别求出最低上位子集的深度 $deep_i$ 和 $deep_j$。两个单词间的相似性形式化公式可以表示为：

$$sim=\frac{1}{(deep_i+deep_j+1)} \tag{1}$$

其中，$deep_i$ 是单词 i 与最低上位子集的深度大小，$deep_j$ 是单词 j 与最低上位子集的深度大小。符号 sim 代表两个单词间的相似性数值，sim 取值范围大于 0 并且小于等于 1。如果两个单词间无路径，sim 为 −1。由于 wordnet 自身有局限性，因此，能够借助 wordnet 实现函数名相似性比较的数量较少。

3.4 函数体语义特征提取

函数的函数体中包含许多指令信息，其中一些指令信息对于函数名预测工作是无用的，这些无效信息的存在反而会影响最终的语义获取准确度。因此，对函数体进行数据预处理，可以提高函数体的语义获取质量。具体做法是，在不丢失函数体基本语义的情况下对函数体进行处理，尽可能减少函数体中语义无关的信息数据。

控制流图是代码的结构化表示[19-21]，构造成本较低，并且被广泛应用于安全类研究之中。不同函数之间的功能区别，一般体现在函数体的控制流图上。相比功能简单的函数，功能复杂的函数的控制流图更为复杂。启发于 DeepWalk，本文基于深度优先遍历思想进行控制流图提取，在函数名、函数体提取方面，两种提取方法较为相似。本文方法利用 Python 中的正则表达式库函数 *re.findall*()实现函数名、函数体提取，根据中间语言.ll 文件中的函数名起始标志位 define i64 @，define i32 @等，以及结束位“(”编写脚本实现函数名提取工作。根据中间语言.ll 文件中的函数体基本块的起始位 dec_label_pc_400a8b，dec_label_pc_400afb 等字符串中的公共起始字符串 dec_label_pc_，以及结束位“}”，来编写程序完成

函数体中的基本块切割提取工作。将提取到的函数体基本块和函数名组成键值对，存入 json 文件中。先以基本块为基本单位从中间语言.ll 文件中提取出函数体所对应的控制流图，得到以基本块为基本单位的控制流图信息，同时保留基本块中的操作符和库函数名以及基本块内的外部调用函数名。在前期调研实验样本数据过程中发现，大多数操作符之后都会跟有数字，并且由于 LLVM 的 SSA 特性，此类位于操作符之后的数字具有很大的随机性[22-23]。如果将此类数字特征留存下来不仅会增加函数体的长度，还会降低实验效果，舍弃此类随机的数字特征以及小部分不位于操作符之后的数字特征来获得更好的实验效果是值得的。利用 Python 中的正则表达式中的 *re.sub*()函数编写程序实现搜索匹配，舍弃操作符后面的阿拉伯数字，仅保留指令序列中的操作符。分析中间语言.ll 文件，其中形如“%”“[”、“]”“=”“!”“insn. addr”“preds”“align”等对函数名预测工作都是无意义的，同样借助正则表达式中的 *re.sub*()函数编写程序进行舍弃。将指令序列转换为函数体的 token，舍弃对于函数名预测工作无关的信息，仅保留函数体指令序列中的操作符、调用函数、库函数。

词袋(Bag of Words)是同一类型的 token 的集合，其中的 token 相互独立。对保留下来的库函数名以及调用函数名进行处理，根据驼峰命名法、蛇形命名法的标志性分隔符号进行切割来获取更多的函数名 token，从而使函数体内部的词袋数据更加丰富。这样可以有效避免 OOV(Out of Vocabulary)的出现。最终，经过提取以及处理得到的表示函数体的 token 信息如表 2 所列。

表 2 函数体 token 信息

Table 2 Function body token information

函数体内部词袋类型	tokens
操作符	add，sub，mul，and，load，call…
调用函数	list_supported_targets，bfd_openr，insert_info_hash_table，scan_unit_for_symbols，decode_line_info…
库函数	print、fprintf，sprintf，sscanf，strcmp、putchar，exit，ctime，stpcpy，strnicmp，strncat…

3.5 编码器、解码器和 OpenNMT

神经网络在漏洞检测[24]、机器翻译[25]、语音识别[26]、图像处理[27-28]等领域被广泛使用。循环神经网络(Recurrent Neural Network，RNN)与一般的神经网络结构一样，也由输入层、隐藏层(中间层)、输出层组成。RNN 能很好地捕获输入序列的信息，即捕获前面和后面输入信息之间的关系。编码器-解码器范式被用于很多序列翻译任务之中[29-30]，编码器和解码器范式本质上就是两个 RNN 模型之间的合作。编码器将其中的序列信息压缩为一定长度的语义向量，该语义向量是通过编码器从序列中提取的语义特征[31-32]。将编码器编码过程中的最后一个隐状态转换得到的语义向量作为初始状态输入解码器中，由解码器对语义向量进行处理并得到最终的语义序列。解码器经过解码输入其中的语义向量得到序列，理想情况下，解码得到的序列应该与输入编码器中的序列相近甚至相同。OpenNMT 是一个神经机器翻译的开源系统[32]，是通过编码器和解码器范式实现的。MFNP 中的编码器先将函数中的指令序列压缩为一个语义向量，解码器根据编码器所提供的语义向量得到对应的函数名 token。最终，通过编码器和解码器范式完成函数体指令序列与函数名 token 之间的翻译。

4 实验

4.1 实验环境

实验所用计算机配置为 Intel(R) Xeon(R) W-3175X CPU @ 3.10GHz and 384GB RAM。软件为 Python 3.7.6，scikit-learn 1.0.2，OpenNMT-py 2.2.0，gensim 3.8.3，Levenshtein 0.18.1。

4.2 模型参数

MFNP 所使用的机器翻译模型为 OpenNMT。先将经过预处理的函数名和函数体数据构造成能够为机器翻译模型 OpenNMT 所使用的语料库数据。机器翻译模型 OpenNMT 一次迭代训练过程所使用的样本量为 16，模型训练过程的迭代更新次数设置为 1 000，模型验证数据的次数设置为 500。从每个语料库中采样 10000 行数据来构建词汇表。

4.3 实验数据集

实验所用数据集应该具有很强的真实环境代表性。参考 DEBIN[11]构建数据集的方法，本文构建数据集 I：binutils，busybox，coreutils，curl，diffutils，findutils，gmp，Image-Magick，openssl，putty，sqlite，zlib。实际应用中的二进制文件，编译时所使用的编译优化选项往往不相同，为了保证二进制文件数据的多样性，本文通过 O0，O1，O2，O3 这 4 个编译优化选项完成二进制文件的编译工作。使用 GCC 7.5.0 编译器将这些数据集结合不同的编译优化选项编译生成二进制文件，本文实验数据的编译环境如表 3 所列。经过反编译二进制文件得到的中间语言文件，能够更加方便 MFNP 对不同架构的二进制文件进行函数名预测。因此，本文选择通过 LLVM-RetDec 反编译得到的中间语言.ll 文件作为实验数据。

表 3 MFNP 数据编译环境

Table 3 MFNP data compilation environment

编译器	版本	目标平台/架构	编译优化选项
GCC	7.5.0	MIPS	O0/O1/O2/O3
GCC	7.5.0	ARM	O0/O1/O2/O3
GCC	7.5.0	X86	O0/O1/O2/O3

数据集 II[12]：NERO 公开的数据集，其中训练集样本为 483 个二进制文件，测试集样本为 13 个二进制文件，验证集样本为 45 个二进制文件。

X86 为 CISC(复杂指令集)，MIPS 和 ARM 为 RISI(精简指令集)。本文使用的反编译器 LLVM-RetDec 暂时不支持 64 位的 MIPS 架构。为了体现出 MFNP 在各种架构之中的实际性能，本文参照 MIPS 32 位数据量大小，从 X86 和 ARM 所对应的 32 位、64 位的数据集中分别取出 131 730 个数据。如此构造的数据集，已经满足了多架构、多优化的条件。原始函数名以及去除掉噪音函数名后的数据量如表 4 所列。表中的原始个数指原始数据中的函数名个数，选择个数指选择出的函数名个数，剩余个数指去除噪音函数名后剩余的函数名个数。

表 4 去除函数名噪音数据前后的数据对比

Table 4 Data comparison before and after removing function name noise data

	X86	X64	ARM32	ARM64	MIPS32
原始个数	358959	368231	202848	346905	263460
选择个数	263460		263460		263460
剩余个数	196772		170324		176715

4.4 评价指标

为方便 MFNP 与现有工作的对比，本文参考 Nero，同样选用 Precision，Recall，F1-score 作为 MFNP 的评估指标。为了方便进行评估，在实验中已经将函数名切割为小写英文形式。由 MFNP 预测给出一个划分为若干个 token 的函数名 $S:\{s_1,s_2,\cdots,s_m\}$，真值函数名也是划分为若干个 token 的函数名 $W:\{w_1,w_2,\cdots,w_n\}$。其中 TP,FP,FN 的计算式如下：

$$TP=\sum_{i}^{m}D\{s_i\in W\} \tag{2}$$

$$FP=\sum_{i}^{m}D\{s_i\notin W\} \tag{3}$$

$$FN=\sum_{i=1}^{n}D\{w_i\notin S\} \tag{4}$$

$$Precision=\frac{TP}{TP+FP} \tag{5}$$

$$Recall=\frac{TP}{TP+FN} \tag{6}$$

$$F1\text{-}score=\frac{2\times Precision\times Recall}{Precision+Recall} \tag{7}$$

式(2)－式(4)中的 D 函数类似于狄利克雷函数，其最终的结果只有 0 和 1 两种。与二分类任务中的 TP，FP 和 FN 的计算不同，这里的 TP，FP 和 FN 有一种新的计算方式[33]。如果预测函数名中的 token 出现在函数名真值之中，那么 D 函数等于 1，否则为 0。同理 FP，预测函数名中的 token 未出现在真值函数名中就记为 1，否则为 0。每次求出的 TP，FP 以及 FN 对应于一个函数名样本。最终，为了最大程度地反映出实验效果的真实性，评估过程不会将每个预测函数名所对应的 Precision，Recall，F1-score 进行求取平均值，而是将所有预测函数名的 Precision，Recall，F1-score 对应的分子和分母依次相加，将最终的分子总和、分母总和之间的比值作为 Precision，Recall，F1-score，最终求出实验结果数据。例如，如果函数名 s_1 的 Precision 为 3/5，则函数名 s_2 的 Precision 为 2/3。二者的 Precision 计算方式并不是简单求二者的平均值，形如(3/5＋2/3)/2＝8/15，而是分别求和分子、分母之后再相除，形如(3＋2)/(3＋5)＝5/8。

4.5 效率评估

模型训练的快慢受机器性能的影响较大，通过优化模型训练的设备可以有效缩短模型训练的时间。然而，二进制文件的特征提取过程，并不会因为机器设备不同有明显改善，所以二进制文件特征提取过程对整个函数名预测工作影响最大[17]。因此，利用 DEBIN，NERO，MFNP 三者提取二进制文件特征数据所用时间长短来评估三者的工作效率更有意义。为了保证效率对比实验的公平性，这里选择数据集 II 作为效率评估的实验数据集[12]。因为 MFNP 的基础工具 LLVM-RetDec 对内存依赖较大，所以 MFNP 在内存为 128 GB 的 Ubuntu 内实现二进制文件特征提取工作。为保证评估实验的一致性，NERO 和 DEBIN 也均在 128GB 的 Ubuntu 中完成二进制文件特征提取工作。特征提取时间对比情况如表 5 所列。

表 5 DEBIN，NERO，MFNP 特征提取时间对比

Table 5 Comparison of feature extraction time of DEBIN, NERO and MFNP

方法	特征提取时间
DEBIN	14 h 34 min 13 s
NERO	1 h 18 min 06 s
MFNP	1 h 21 min 12 s

DEBIN 的二进制文件特征提取以及模型训练分两个阶段完成。第一阶段用时 4 h 40 min 4 s，第二阶段用时 9 h 54 min 9 s，总时间花费为 14 h 34 min 13 s。DEBIN 相比 NERO 和 MFNP 额外计算了函数名、变量名等信息预测模型的训练时间，因此耗费时间较多。DEBIN 所耗费的时间分别是 NERO 的 11.19 倍，MFNP 的 10.76 倍。这一时间消耗差别是可以预见的。因为 DEBIN 不仅仅做了函数名预测工作，还做了变量恢复相关的工作。其所需要的特征数据以及特征数据处理步骤，相对于仅做函数名预测工作的 NERO 和 MFNP 更多。因此，DEBIN 工作的总体时间花费更多也是合理的。

4.6 实验对比

MFNP 生成函数名的一些样例，如表 6 所列。其中以蛇形命名法、驼峰命名法进行命名的函数名可以通过明显的分割标志位将函数名分割为函数名 token。将剥离的二进制文件输入 MFNP 中，MFNP 能够输出一些预测的函数名 tokens，将这些函数名 tokens 通过下划线进行连接就可以得到最终的函数名。

表 6 MFNP 生成函数名的一些样例

Table 6 Some examples of MFNP generated function names

原始函数名	函数名真值 tokens	预测函数名 tokens
move_section_contents	{move,section,contents}	{try,section,contents}
verilog_set_arch_mach	{verilog,set,arch,mach}	{verilog,set,section,contents}
verilogMkobject	{verilog,mkobject}	{verilog,mkobject}
verilog_write_object_contents	{verilog,write,object,contents}	{verilog,write,section,contents}
parse_coff_baseType	{parse,coff,base,type}	{parse,coff,type}
Stab_write_symbol	{stab,write,symbol}	{stab,write}

MFNP 的实验结果如表 7 所列。分析 MFNP 函数名预测的结果可知，MIPS 相较于 X86 和 ARM 的实验结果较差。这是因为 LLVM-RetDec 暂不支持 MIPS64 位的反编译，进而导致 MIPS 架构的数据样本种类相对较少，影响最终的实验效果。

MFNP 与 DEBIN 的对比实验在两个工作都支持的 X86 和 ARM 架构下进行。选用数据集 I 作为对比实验数据集。选择精确率 Precision、召回率 Recall 以及 F1 值这 3 个评价指标来评估各个方法在 X86 数据集上的性能。DEBIN 对于在训练集中出现过的样本具有不错的效果。然而，其在新剥离的二进制文件上表现不佳，这是因为 DEBIN 未使用神经网络。

表 7 MFNP 实验结果

Table 7 MFNP experimental results

（单位：%）

架构	MFNP		
	Precision	*Recall*	*F*1
X86	52.65	51.87	52.25
ARM	49.73	40.69	44.76
MIPS	48.10	48.10	43.43

由于 DEBIN 做的是函数名和变量名的预测，因此 DEBIN 同时计算了二者的实验结果。在进行与 DEBIN 的对比实验时，我们修改了 DEBIN 中的计算实验结果数据部分的代码，仅计算函数名所对应的实验结果。MFNP 与 DEBIN 的对比实验结果如表 8 所列。

表 8 MFNP 与 DEBIN 的对比实验结果

Table 8 Comparative experiment results of MFNP and DEBIN

（单位：%）

架构	DEBIN			MFNP		
	Precision	*Recall*	*F*1	*Precision*	*Recall*	*F*1
X86	19.43	20.16	19.78	51.23	47.02	49.03
ARM	18.29	19.67	18.95	45.62	42.13	43.80

MFNP 和 NERO 的对比实验在两种方法都支持的 X86 架构数据集上进行。为了保证对比实验的公平性，选用数据集 II 作为对比实验数据集。选择精确率 Precision、召回率 Recall 以及 F1 值作为对比实验的评价指标来评估各个方法在 X86 数据集上的性能。MFNP 与 NERO 的对比实验结果如表 9 所列。

表 9 MFNP 与 NERO 的对比实验结果

Table 9 Comparative experiment results of MFNP and NERO

（单位：%）

方法	*Precision*	*Recall*	*F*1
MFNP	55.11	48.27	52.73
NERO	41.25	40.36	40.80

可以看出 MFNP 的表现相对较好。首先，MFNP 在更大的数据样本上进行模型训练以及实验。机器学习任务中更大的数据样本往往能得到更好的泛化性。因此，MFNP 具有更好的实验结果是正常的。其次，MFNP 所提取的语义特征包括控制流图以及基本块中的操作符、库函数、调用函数，对基本块中的库函数、调用函数进行处理可以获得更多的函数名 token。这些函数名 token 对于函数名预测任务来说有很好的帮助，因此能获得更好的结果。

结束语 本文基于机器学习提出了一种用于多架构剥离二进制文件函数名预测的方法 MFNP。将二进制文件输入 LLVM-RetDec 中，通过 LLVM-RetDec 反编译获得中间语言.ll 文件。从中间语言.ll 文件中提取出以基本块为基本单位的控制流图信息来表示函数的语义，基本块中含有表示函数语义特征的指令序列信息。如此构造的语义特征数据，包含了函数的指令序列信息以及函数的控制流图信息。MFNP 是第一个通过神经网络解决多架构下剥离二进制文件的函数名预测问题的工作。实验结果表明，MFNP 在 X86，ARM，MIPS 这 3 种架构下的剥离二进制文件函数名预测中均具有不错的效果，其相比现有的剥离二进制文件函数名预测工作 DEBIN 和 NERO 有较大的改进。

未来的研究工作如下：

1）本文实验的数据样本是通过 O0，O1，O2，O3 这 4 种编译优化选项编译获得的，并且未讨论不同编译优化选项对函数名预测工作所造成的影响。编译优化选项不只有 O0，O1，O2，O3 这 4 种，而是多达一百多种。后续可以研究不同编译优化选项对函数名预测工作造成的影响。

2）研究自然语言领域中的分词技术，通过该技术来解决未以驼峰命名法、蛇形命名法命名的函数名，从而获得更多的函数名 token。增大函数名 token 的数据集大小，从而改善函数名预测结果。

参 考 文 献

[1] VOTIPKA D，RABIN S，MICINSKI K，et al. An Observational Investigation of Reverse {Engineers'} Processes[C]// 29th USENIX Security Symposium (USENIX Security 20). 2020：1875-1892.

[2] GELLENBECK E M，COOK C R. Aninvestigation of procedure and variable names as beacons during program comprehension [C]// Empirical Studies of Programmers：Fourth Workshop. Ablex Publishing，Norwood，NJ，1991：65-81.

[3] ALLAMANIS M，BARR E T，BIRD C，et al. Suggesting accurate method and class names[C]// Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015：38-49.

[4] ALON U，SADAKA R，LEVY O，et al. Structural language models for any-code generation[J]. arXiv：1910.00577，2019.

[5] FOWLER M. Refactoring：improving the design of existing code [M]. Hoboken：Addison-Wesley Professional，2018.

[6] HØST E W，ØSTVOLD B M. Debugging method names [C]// European Conference on Object-Oriented Programming. Berlin：Springer，2009：294-317.

[7] JACOBSON E R，ROSENBLUM N，MILLER B P. Labeling library functions in stripped binaries[C]// Proceedings of the 10th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools. 2011：1-8.

[8] LIU Z，WANG S. How far we have come：testing decompilation correctness of C decompilers[C]// Proceedings of the29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020：475-487.

[9] AHMAD W U，CHAKRABORTY S，RAY B，et al. A transformer-based approach for source code summarization[J]. arXiv：2005.00653，2020.

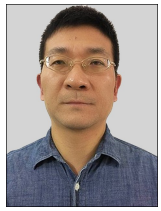
[10] LECLAIR A，JIANG S，MCMILLAN C. A neural model for generating natural language summaries of program subroutines [C]// 2019 IEEE/ACM 41st International Conference on Software Engineering(ICSE). IEEE，2019：795-806.

[11] HE J，IVANOV P，TSANKOV P，et al. Debin：Predicting debug information in stripped binaries[C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018：1667-1680.

[12] DAVID Y，ALON U，YAHAV E. Neural reverse engineering of stripped binaries using augmented control flow graphs[C]//

Proceedings of the ACM on Programming Languages. 2020: 1-28.
[13] VENKATAKEERTHY S, AGGARWAL R, JAIN S, et al. IR2VEC: LLVM IR Based Scalable Program Embeddings[J]. ACM Transactions on Architecture and Code Optimization, 2020, 17(4): 1-27.
[14] 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.
[15] ALON U, ZILBERSTEIN M, LEVY O, et al. code2vec: Learning distributed representations of code[C]// Proceedings of the ACM on Programming Languages. 2019: 1-29.
[16] PATRICK-EVANS J, CAVALLARO L, KINDER J. Probabilistic naming of functions in stripped binaries[C]// Annual Computer Security Applications Conference. 2020: 373-385.
[17] GAO H, CHENG S, XUE Y, et al. A lightweight framework for function name reassignment based on large-scale stripped binaries[C]// Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2021: 607-619.
[18] BAN G, XU L, XIAO Y, et al. B2SMatcher: fine-Grained version identification of open-Source software in binary files[J]. Cybersecurity, 2021, 4(1): 1-21.
[19] DUAN Y, LI X, WANG J, et al. Deepbindiff: Learning program-wide code representations for binary diffing[C]// Network and Distributed System Security Symposium. 2020.
[20] 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 Conference on Computer and Communications Security. 2017: 363-376.
[21] PEROZZI B, AL-RFOU R, SKIENA S. Deepwalk: Online learning of social representations[C]// Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2014: 701-710.
[22] ZHAO J, NAGARAKATTE S, MARTIN M M K, et al. Formal verification of SSA-based optimizations for LLVM[C]// Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2013: 175-186.
[23] BELYAEV M, TSESKO V. LLVM-based static analysis tool using type and effect systems[J]. Automatic Control and Computer Sciences, 2012, 46(7): 324-330.
[24] LI Z, ZOU D, XU S, et al. Vuldeepecker: A deep learning-based system for vulnerabilitydetection[J]. arXiv: 1801.01681, 2018.
[25] KIDANE L, KUMAR S, TSVETKOV Y. An Exploration of Data Augmentation Techniques for Improving English to Tigrinya Translation[J]. arXiv: 2103.16789, 2021.
[26] ZHANG Q, LU H, SAK H, et al. Transformer transducer: A streamable speech recognition model with transformer encoders and rnn-t loss[C]// 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2020). IEEE, 2020: 7829-7833.
[27] NEELIMA D, KARTHIK J, ARAVIND JOHN K, et al. Soft Computing-Based Intrusion Detection Approaches: An Analytical Study[M]//Soft Computing in Data Analytics. Singapore: Springer, 2019: 635-651.
[28] HUANG Z, ZHU Z, MA W, et al. B2GAN: Bidirectional-branch generative adversarial network for text image super-resolution with structure preservation[J]. Optik, 2022, 261: 169093.
[29] ILYA S, ORIOL V, QUOC VL. Sequence to sequence learning with neural networks[C]// Proceedings of the 27th International Conference on Neural Information Processing Systems—Volume 2 (NIPS'14). MIT Press, Cambridge, MA, USA, 2014: 3104-3112.
[30] ASHISH V, NOAM S, NIKI P, et al. Attention is all you need [C]// Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Red Hook, NY, USA, 2017: 6000-6010.
[31] CHO K, VAN MERRIËNBOER B, GULCEHRE C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv: 1406.1078, 2014.
[32] KLEIN G, KIM Y, DENG Y, et al. Opennmt: Open-source toolkit for neural machine translation[J]. arXiv: 1701.02810, 2017.
[33] ALLAMANIS M, PENG H, SUTTON C. A convolutional attention network for extreme summarization of source code [C]// International Conference on Machine Learning. PMLR, 2016: 2091-2100.

SHAO Wenqiang, born in 1996, postgraduate. His main main research interests include binary code analysis and machine learning.



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

(责任编辑:何杨)