

DNNobfus: 一种基于混淆的端侧模型保护框架技术研究

宋飞扬¹, 赵鑫淼¹, 严飞¹, 程斌林², 张立强¹, 杨小林³, 王洋⁴

(1. 武汉大学国家网络安全学院空天信息安全与可信计算教育部重点实验室, 湖北 武汉 430072;

2. 山东大学网络空间安全学院, 山东 青岛 266237; 3. 浪潮智慧科技有限公司, 山东 济南 250101;

4. 山东浪潮科学研究院有限公司, 山东 济南 250101)

摘要: 人工智能模型自广泛使用以来, 面临诸多安全风险。随着深度学习模型大规模在端侧设备上的部署, 端侧模型面临新的安全挑战。由于深度神经网络具有相似的结构特征, 攻击者得以运用反编译技术以获取模型的结构信息及其参数, 从而重建模型。此过程会损害模型的知识产权并使模型面临白盒攻击的风险。针对模型反编译器对模型算子的定位与识别、参数获取、网络拓扑解析过程, 提出了一种基于模型编译过程的混淆框架以防御模型提取攻击; 在深度学习编译器的前端优化过程中设计并实现了算子混淆、参数混淆和网络拓扑混淆3种混淆手段; 采用构造不透明谓词, 插入虚假控制流, 添加冗余内存访问等手段, 干扰模型反编译器对模型的逆向工程。实验结果表明, 提出的混淆框架DNNobfus能有效降低前沿的模型反编译工具对模型算子类型及网络连接识别的准确率, 分别下降至21.63%和48.24%。此外, 该框架平均时间效率为67.93%, 平均空间效率为88.37%, 均优于混淆工具Obfuscator-LLVM。

关键词: 人工智能安全; 代码混淆; 逆向工程; 模型保护

中图分类号: TP309

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2024019

DNNobfus: a study on obfuscation-based edge-side model protection framework

SONG Feiyang¹, ZHAO Xinmiao¹, YAN Fei¹, CHENG Binlin², ZHANG Liqiang¹,
YANG Xiaolin³, WANG Yang⁴

1. Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China;

2. School of Cyber Science and Technology, Shandong University, Qingdao 266237, China;

3. Inspur Intelligent Technology Company Limited, Jinan 250101, China;

收稿日期: 2023-11-27; **修回日期:** 2024-03-04

通信作者: 严飞, yanfei@whu.edu.cn

基金项目: 湖北省重大研究计划项目 (No.2023BAA027); 国家自然科学基金 (No.62172144); 国家重点研发计划项目 (No.2022YFB3103804)

Foundation Items: The Major Research Plan of Hubei Province (No.2023BAA027), The National Natural Science Foundation of China (No.62172144), The National Key Research and Development Program of China (No.2022YFB3103804)

引用格式: 宋飞扬, 赵鑫淼, 严飞, 等. DNNobfus: 一种基于混淆的端侧模型保护框架技术研究[J]. 网络与信息安全学报, 2024, 10(02): 143-153.

Citation Format: SONG F Y, ZHAO X M, YAN F, et al. DNNobfus: a study on obfuscation-based edge-side model protection framework[J]. Chinese Journal of Network and Information Security, 2024, 10(02): 143-153.

4. Inspur Academy of Science and Technology, Jinan 250101, China

Abstract: The proliferation of artificial intelligence models has rendered them vulnerable to a myriad of security threats. The extensive integration of deep learning models into edge devices has introduced novel security challenges. Given the analogous structural characteristics of deep neural networks, adversaries can employ decompilation tactics to extract model structural details and parameters, facilitating the reconstruction of these models. Such actions can compromise the intellectual property rights of the model and increase the risk of white-box attacks. To mitigate the capability of model decompilers to locate and identify model operators, acquire parameters, and parse network topologies, an obfuscation framework was proposed. This framework was embedded within the model compilation process to safeguard against model extraction attacks. During the frontend optimization phase of deep learning compilers, three obfuscation techniques were developed and integrated: operator obfuscation, parameter obfuscation, and network topology obfuscation. The framework introduced opaque predicates, incorporated fake control flows, and embedded redundant memory access to thwart the reverse engineering efforts of model decompilers. The experimental findings demonstrate that the obfuscation framework, named DNNobfus, significantly diminishes the accuracy of state-of-the-art model decompilation tools in identifying model operator types and network connections to 21.63% and 48.24%, respectively. Additionally, DNNobfus achieves an average time efficiency of 67.93% and an average space efficiency of 88.37%, surpassing the performance of the obfuscation tool Obfuscator-LLVM in both respects.

Keywords: artificial intelligence safety, code obfuscation, reverse engineering, model protection

0 引言

人工智能模型自广泛使用以来,面临诸多安全风险^[1-2]。随着边缘设备的算力增强,深度学习模型应用从云端快速推广到端侧,实现低时延、高安全的智能交互。当在边缘设备上部署深度学习模型时,一种较为高效的方案是通过编译器将模型编译成可执行文件进行直接部署。研究表明,这种方案面临模型被反编译获取参数和结构,从而被攻击者窃取滥用的风险^[3-4]。

虽然,现有工作已针对深度学习模型提出了相应保护方案,如Li等^[5]研究模型水印技术旨在保护模型的知识产权,但这种技术无法阻止模型被攻击者重建。MLcapsule^[6]将机器学习模型放入端侧的可信执行环境(trusted execution environment, TEE)^[7]中执行, SOTER^[8]利用图形处理单元(graphics processing unit, GPU)和TEE协同执行推理任务,但是这些方法的关键是将模型运行在可信执行环境中,需要额外的硬件资源支持,并且会引入较大的计算开销,因此不太适用于大量无可信硬件的端侧设备,而且它们主要着眼于抵御软硬件侧信道带来的关键参数泄漏,对于防御可以直接获得模型可执行文件的攻击者来说并不能取得很好的效果。

针对现有深度学习模型保护技术无法满足防御模型反编译攻击的问题,本文针对端侧部署的模型易受攻击的脆弱点,研究面向端侧深度学习模型保护的混淆技术,相较于其他模型保护方案,模型混淆使用更低的成本实现了更好的通用性,能够有效地抵御反编译攻击,保障端侧深度学习模型的安全。本文的主要贡献如下。

(1) 设计了一种基于深度学习编译器张量虚拟机TVM(tensor virtual machine)^[9]的端侧深度学习模型保护框架,在编译时针对算子、模型参数和网络拓扑等高级语义进行混淆,防御模型提取攻击从端侧二进制文件恢复模型。

(2) 针对深度学习模型的结构特点,提出了构造不透明谓词、参数存取顺序混淆以及额外内存访问等混淆算法,在保证模型效率的前提下,增强了模型的抗逆向分析能力。

(3) 对所提出的模型混淆框架进行了混淆效果测试和性能测试。当前最新的模型反编译工具BTD^[4]对实施本方法混淆后的算子类型、网络连接的识别正确率分别低至21.63%和48.24%。混淆后的框架相比混淆前平均时间效率为67.93%,平均空间效率为88.37%,均优于混淆工具OLLVM(Obfuscator-low level virtual machine, OLLVM)^[10]。

1 相关工作

1.1 深度学习模型提取技术

机器学习即服务的模式,推动了机器学习的大规模使用,但同时针对部署后的深度学习模型的保护问题也引起了大量的研究和关注。针对云端上部署的模型,研究者提出可以通过利用侧信道攻击来推断目标深度神经网络的架构信息。Hua等^[11]在深度神经网络(deep neural network, DNN)执行期间使用内存和定时侧信道成功地推断了底层网络架构。Batina等^[12]利用电磁侧信道对架构的重要参数,如层数和层尺寸进行反向工程,以推断神经网络架构。此外,Yan等^[13]使用缓存侧通道提取机器学习即服务平台中的DNN架构,在提取架构后,攻击者可以进一步训练具有竞争性推理精度的替代模型。

相比云上需要通过侧信道推测模型信息,端侧由于设备的易访问特点,攻击者可以直接对模型二进制文件进行反编译和逆向工程。Wu等^[3]提出第1个跨编译器和架构的深度学习模型反编译器DnD。DnD使用符号执行技术与专用循环分析相结合,通过将提取的运算与算子操作模板进行匹配从而恢复原始模型中的运算。Liu等^[4]提出了一种用于深度学习模型可执行文件的反编译器BTD。BTD使用长短期记忆网络(long short-term memory, LSTM)模型^[14]进行算子推断,将汇编函数分类为深度学习算子,并结合符号执行和污点分析还原模型中的参数和网络连接。与基于侧信道的模型提取方式不同,基于反编译和逆向工程的方法能够对提取后的模型进行重新编译,并在此基础上进行白盒攻击。相比之下,基于侧信道的攻击往往需要大量时间,或者仅能提取模型的部分信息。因此本文针对端上基于反编译和逆向工程的模型提取攻击,提出基于混淆的端侧模型保护技术。

1.2 模型保护与混淆技术

为了防御基于侧信道的深度学习模型体系结构逆向工程,Luo等^[15]提出了轻量级张量程序调度框架,通过重塑不同深度神经网络架构的电磁轨迹,实现模型架构的混淆;Li等^[16]提出的Neu-robfuscator使用了8种混淆策略来隐藏原始的深

度学习模型架构,这些策略可以在结构参数(如层数和维度)中引入预测误差,使攻击者无法精确提取深度神经网络架构;Zhou等^[17]使用神经架构搜索保护深度学习模型免受架构提取攻击的工作,利用组合精度度量指导深度学习模型架构屏蔽。与这些方案相比,本文针对端上基于反编译和逆向工程的模型,提取攻击。

DeepObfuscation^[18]针对卷积神经网络模型提出了一种递归模拟和联合训练的混淆方法,以隐藏真实的模型结构,提高模型抗微调的能力。但该方法需要对模型进行重新训练,改变了模型的结构,会对模型精度产生影响。Szentannai等^[19]提出一种通过在神经网络中插入误导性神经元并调整网络结构,来增强模型对权重变化的敏感性,从而防止模型权重被盗用的方法。但该方法仅针对修改模型权重的模型盗用攻击,不能有效防御基于反编译和逆向工程的模型结构提取攻击。

研究者还提出了多种采用可信硬件来保护深度学习模型的方案。如MLCapsule^[6]将机器学习模型放入端侧的可信执行环境执行。Chakraborty等^[20]通过可信硬件设备实现了深度学习模型参数模糊化,以高效安全地保护模型的知识产权。但这些方法依赖部署端的可信硬件,不适用于大量无可信硬件的端侧设备。

2 DNNobfus 框架总览

2.1 威胁模型

本文研究的终端模型的威胁模型如图1所示,在部署深度学习模型到端侧的过程中,模型供应商通常会使用深度学习编译器将模型编译为可执行文件。由于可执行文件保存在端侧设备上,攻击者可以从目标应用或托管设备中获取该文件并进行分析。攻击者可以利用现有的逆向工程工具(如IDA Pro)对模型可执行文件进行反编译,并通过分析汇编代码获得原始的深度学习模型高级表示。本文研究目的是在编译期对模型可执行文件内部的信息进行混淆,增加攻击者获取原始深度学习模型高层表示的难度,以此来提高端侧部署的深度学习模型的安全性。

2.2 框架总览

基于逆向工程的模型提取攻击旨在从模型的

二进制可执行文件中提取模型的算子类型、具体参数和网络结构。为应对这类攻击，本文研究的核心思路是从算子混淆、参数混淆和网络拓扑混淆 3 个方面出发，对模型的高级语义信息进行混淆保护。在基于深度学习编译器 TVM 和通用编译器框架底层虚拟机的编译环境中，实现了一个通过在深度学习模型编译过程中混淆模型重要特征的通用可部署的端侧模型保护框架 DNNobfus，DNNobfus 架构总览如图 2 所示。为了实现混淆选项的灵活配置，本文将模型的算子混淆、参数混淆和网络拓扑混淆作为保护框架中的 3 个模块，用户可以通过组合不同的模块实现不同强度的模型混淆，这也使得本文的混淆框架可以满足不同的效率需求。

在深度学习模型的部署过程中，模型供应商将训练好的模型参数和网络结构编译成针对不同硬件的可执行文件。在该编译过程中，TVM 采用前后端解耦的设计。模型的算子最初以高阶的硬件无关表示存在，经过前端的调度和优化生成优化后的计算图。在 TVM 中，Pass 是一种转换器，它可以将输入程序转换成经过优化的输出程序。本质上，Pass 是将原中间表示语句映射到新

中间表示语句的一种函数。在前端阶段，用户可以根据需求自由组合不同的优化 Pass，本文设计并实现了一系列防御模型提取攻击的混淆 Pass，并集成到 TVM 的前端优化过程中。经过前端的优化和混淆，计算图被表达为硬件无关的中间表达。之后后端根据目标硬件平台，进行算子的具体实现和编译，最终生成可在特定硬件上执行的代码。

此外，需要特别指出的是，本文提出的混淆技术被应用于神经网络模型编译为二进制文件的阶段，即在神经网络的训练已经完成之后。这确保了这些混淆技术不会对神经网络的训练过程产生任何直接影响，并且这些技术的核心是在不影响模型执行逻辑的前提下，通过插入部分代码和改变代码结构来实现混淆。因此，它们并不会影响神经网络识别任务的准确性。

3 系统实现

本文设计的 DNNobfus 框架的关键在于算子混淆、参数混淆和网络拓扑混淆 3 个模块，其中算子混淆模块通过重命名算子函数，插入虚假控制流隐藏算子的类型；参数混淆模块通过混淆循

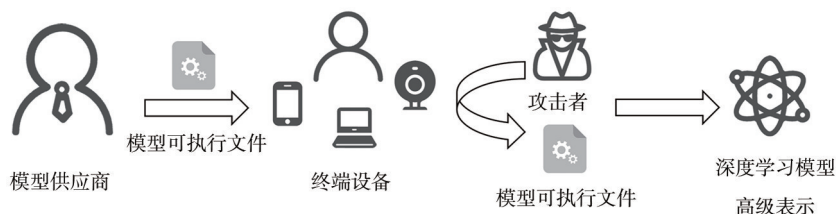


图 1 终端模型的威胁模型
Figure 1 Threat model for on-device models

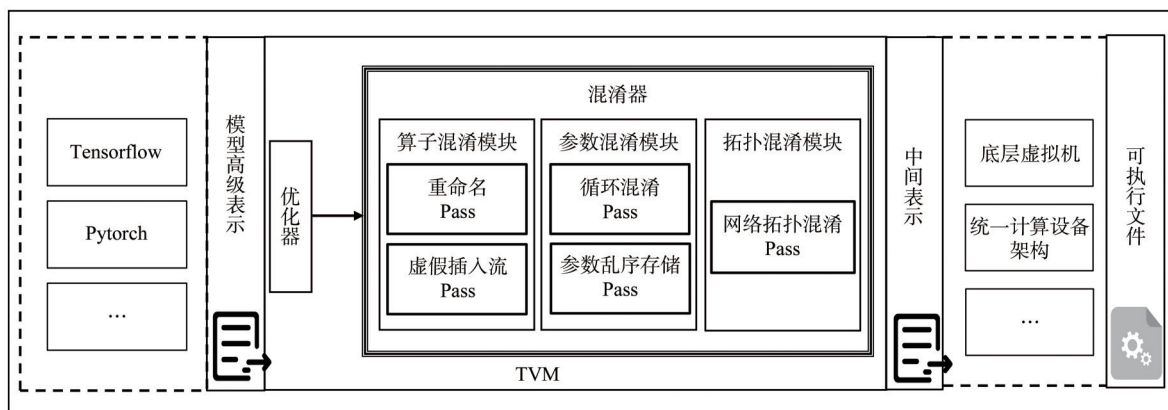


图 2 DNNobfus 架构总览
Figure 2 Overview of the DNNobfus architecture

环, 重定位参数位置来增加参数被识别的难度; 网络拓扑混淆模块改变算子之间的连接关系, 扰乱对模型网络拓扑的识别。3个模块相互协同, 从不同角度增强模型抗逆向分析的能力。下面将详细阐述各模块的设计实现。

3.1 算子混淆模块

当前反编译工具可以通过定位汇编代码中的算子函数, 进而判断算子的具体运算功能, 从而推断出模型的层语义和结构信息。为了对抗这种威胁, 需要采取措施隐藏算子的关键信息, 使其难以被准确定位和识别。具体来说, 算子混淆模块针对反编译工具定位和识别算子的手段, 采取以下应对措施: 重命名混淆和虚假控制流插入。

(1) 重命名混淆

针对当前反编译工具定位算子的手段, 本文采取重命名混淆技术对抗该威胁。通过对编译后的深度学习模型汇编代码进行分析, 发现模型中的每个算子被编译成相应的汇编代码块, 汇编代码块所属的函数名称通常反映了算子的具体运算功能。这使得攻击者可以通过反编译代码并分析函数名称推断出模型的层语义和结构信息。为隐藏这些关键信息, 本文采用遍历所有函数进行重命名的混淆方法。具体来说, 先分析模型汇编代码, 提取每个函数定义及引用位置; 生成随机字符串作为新函数名; 在定义处替换原函数名, 同时将所有引用位置替换为新函数名。

(2) 虚假控制流插入

针对当前反编译工具定位算子后根据算子的汇编代码识别算子功能的手段, 本文采用虚假控制流插入技术对抗该威胁。首先, 向算子函数中插入冗余的混淆代码块, 破坏关键指令序列的上下文语义, 影响反编译工具识别算子功能; 然后, 插入的冗余代码使用不透明谓词作为控制条件, 不透明谓词是指复杂且逻辑不明确的判断条件。这样可以使反编译工具难以判断插入代码的执行路径, 也就无法准确定位和删除无用代码; 最后, 达到混淆算子功能的目的。

基于LLVM的混淆器OLLVM实现了向代码中插入虚假控制流的功能, 通过在程序中插入一些冗余的控制流分支来增加程序的复杂性和难以理解性, 从而达到混淆的目的。在OLLVM原方

案中, 采用了以基本块为粒度进行虚假控制流插入的混淆技术。但是, 这种粒度无法精确定位深度学习模型算子函数的关键内部循环代码。为此, 本文在借鉴其虚假控制流插入思想的基础上, 进行了如下改进: 在OLLVM中以基本块为单位进行混淆, 改为以算子函数为最小粒度进行混淆, 可以准确定位关键代码; OLLVM中构造的冗余代码与原基本块语义相似, 而考虑了不同算子的语义特征来构造冗余代码, 增强了混淆效果; OLLVM使用静态全局变量构造不透明谓词, 本文基于模型维度变量设计动态不透明谓词, 难以通过静态分析去除。针对虚假控制流插入的具体方案, 后续部分将重点阐释以下3个方面的技术细节: 虚假控制流插入位置的选取、虚假控制流代码构造、不透明谓词构造。

(1) 虚假控制流插入位置的选取

在OLLVM的随机基本块混淆方法中, 以程序中的基本块为最小单位进行混淆, 该方法直接应用到算子函数会存在问题。随机选择基本块时, 不同算子的混淆程度不均匀。另外, 循环最内层的指令序列是识别算子的关键, 基于基本块的混淆无法准确向关键位置插入代码。为解决上述问题, 本文将算子函数作为最小混淆单位, 并且定向地向算子函数循环最内层插入混淆代码, 以破坏关键指令的上下文语义。

(2) 虚假控制流代码构造

在OLLVM中, 混淆代码是基于原基本块的指令序列构造的, 与原基本块代码具有类似的语义信息。这种方法没有对算子的语义进行针对性的隐藏, 模型反编译器仍能通过分析指令序列的特征来对算子进行识别。因此, 本文针对算子间的内部差异构造虚假控制流的代码。具体来说, 本文将算子循环内部的操作取并集, 将此并集定义为全集 U , 将算子循环内部操作集合定义为 U_o , 则 U_o 对应 U 的补集 $\overline{U_o}$ 是算子循环内部缺少的操作。为了使各算子包含相似的指令, 本文向每个算子的内部插入 $\overline{U_o}$ 操作。经过这样的代码插入后, 所有算子循环内部的操作都趋近于全集 U , 这会使得指令序列更加相似, 从而降低识别运算符的准确性。本文使用虚假流控制码构造(如算法1所示)检测算子循环内操作的补集, 并使

用补集中的操作构造插入算子内部的代码片段。

算法 1 虚假流控制码构造

输入 目标深度学习模型算子函数集合 F

输出 构造的插入代码片段 randCode

符号说明：

opInner(): 获取算子最内层循环操作码集

uniSet: LLVM 操作码全集

union(): 返回集合并集

diff(): 返回集合差集

cnt: 构造指令数

randEl(): 集合中随机元素

opOps(): 获取算子操作数集

```

1) Func createRandCode( $F$ )
2)   inOps  $\leftarrow \{\}$ 
3)   for opFunc in  $F$  do
4)     inOps  $\leftarrow \text{union}(\text{inOps}, \text{opInner}(\text{op-}$ 
Func))
5)   end for
6)   missOps  $\leftarrow \text{diff}(\text{uniSet}, \text{inOps})$ 
7)   randCode  $\leftarrow ""$ 
8)   while cnt > 0 do
9)     rOp  $\leftarrow \text{randEl}(\text{missOps})$ 
10)    rOp1  $\leftarrow \text{randEl}(\text{opOps}())$ 
11)    rOp2  $\leftarrow \text{randEl}(\text{opOps}())$ 
12)    randCode += rOp1 + " " + rOp + " " +
rOp2 + "; \n"
13)    cnt -= 1
14)   end while
15)   return randCode

```

(3) 不透明谓词构造

在 OLLVM 的不透明谓词构造中, 使用固定的全局变量构造恒真的不透明谓词。然而, 这种不透明谓词可以在静态程序中被计算并去除。因此, 本文设计了动态不透明谓词的构造方法, 提取算子中判断输入类型是否符合规范的某个条件, 并基于该条件构造不透明谓词。具体来说, 算子在进入循环计算前会对输入判断维度, 若输入不满足其维度要求, 则无法进行计算。本文提取算子中的维度判断信息, 确定进入计算循环的条件, 并根据这个条件构造一个谓词, 如果算子能够顺利进入循环, 则本文构造的谓词一定为真。

不透明谓词构造如算法 2 所示, 展示了提取算子中对输入条件判断并构造不透明谓词的过程。

算法 2 不透明谓词构造

输入 待混淆算子函数 f

输出 算子的输入判断谓词 opPred

符号说明：

getTensor(): 获取输入张量

inPred: 生成不透明谓词的数组

randBool(): 随机布尔值

negate(): 谓词取反

connPred(): 连接不透明谓词

randEl(): 随机选择数组元素

```

1) Func genOpPred()
2)   opPred  $\leftarrow ""$ 
3)    $i \leftarrow 0$ 
4)   while getTensor( $i$ ) do
5)     tensor  $\leftarrow \text{getTensor}(i)$ 
6)     type  $\leftarrow \text{tensor.type}$ 
7)     inPred[ $i$ ]  $\leftarrow \text{tensor.type} + "=="$ 
+ type
8)      $i += 1$ 
9)   end while
10)   $i \leftarrow 0$ 
11)  /* 随机变换恒真式 */
12)  while inPred[ $i$ ] do
13)    if ! randBool() then
14)      newPred  $\leftarrow \text{"NOT (" + ne-}$ 
gate(inPred [ $i$ ]) + ")"
15)      inPred[ $i$ ]  $\leftarrow \text{newPred}$ 
16)    end if
17)     $i += 1$ 
18)  end while
19)  opPred  $\leftarrow \text{connPred}(\text{randEl}(\text{in-}$ 
Pred), "AND")
20) return opPred

```

3.2 参数混淆模块

模型参数是反编译工具定位模型结构的重要信息源。反编译工具可以通过循环计数器推断参数维度, 也可以直接转储内存获取参数值。为了防止这种参数泄露的风险, 需要对参数进行混淆保护。通过循环计数器混淆和模型参数存储顺序

的混淆2种手段,阻碍反编译工具准确地获取模型

(1) 循环计数器混淆

在深度学习模型中,循环计数器常用于遍历张量,在汇编代码中可被利用来推断参数维度信息。本文提出了循环计数器混淆方法,将原循环拆分成多个随机打乱顺序的子循环。为实现这一目标,本文遍历 TVM 生成的代码中间表示,选取并修改其最内层循环。例如,对于一个维度为12的循环,将其拆分为3个维度为4的子循环,并对这些子循环的执行顺序进行随机打乱。攻击者无法从循环计数器中获取完整的循环信息。而且,循环拆分后生成的代码量并没有增加太多,这样就带来了较低的性能影响。

(2) 模型参数存储顺序的混淆

由于模型参数通常是静态变量,攻击者很容易在程序执行过程中通过 Dump 内存的方式提取。本文通过将参数中的元素打乱存放并记录索引,从而使内存中的参数更加混乱且难以分析。简单地将模型的参数进行完全随机打乱需要生成一个与参数等大的索引数组,这将浪费大量的存储空间和索引时间。为此,本文提出了模型参数乱序索引技术。该方法将参数进行降维,仅对模型参数最内层维度上的元素进行统一的随机打乱处理。以一个 $3 \times 7 \times 7 \times 64$ 的参数为例,随机生成一个64维,元素的值分别为0~64的随机索引序列,这个序列被用来重新排列参数中的每个64维子向量。在存储参数时,参数的 $3 \times 7 \times 7$ 个64维子向量都会根据这个随机序列进行重新排序。在使用参数时,先访问这个索引序列以确定正确参数的索引位置,然后根据这个索引访问实际的参数值。这种方法既保证了对参数进行一定程度的随机打乱来增强隐蔽性,相比完全随机打乱又节省了较多的存储空间和索引生成时间。在参数隐蔽性与性能之间达到了良好的平衡。

3.3 网络拓扑混淆模块

深度学习模型通常由一系列相互连接的层构成,每个层由多个计算单元(即“算子”)组成。模型反编译器 BTDO 跟踪 x86 架构下的寄存器使用,得到一个深度学习模型算子函数的 I/O 内存读写记录,进一步通过内存访问关系恢复算

子间的连接关系,从而得到模型的拓扑连接,最终还原出模型的计算图。本模块针对深度学习模型中算子的连接方式对深度学习模型进行混淆。在算子内部增加对其他算子输入/输出的访问操作,抽象到模型计算图层面,就是向图中引入一些冗余的边,来混淆反编译器对算子间连接的识别。额外内存访问算法如算法3所示,描述了上述算法的具体过程,给定一个模型,该算法将首先获取其计算图。首先,使用 Kahn 算法遍历计算图,输出算子的网络拓扑排序序列;然后,对于每个算子,该算法会获取输出变量名并将其存储在名为“variables”的字典中;最后,该算法将在拓扑排序序列中随机选择指定数量的算子(size参数指定),为每个算子在其前驱算子集合中随机选取一个或几个算子,添加对其内存访问。混淆前后的模型计算如图3所示。

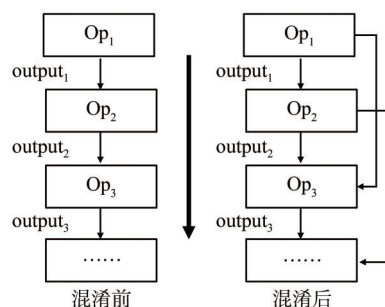


图3 混淆前后的模型计算

Figure 3 Model computation graph before and after obfuscation

算法3 额外内存访问

输入 目标模型 M , 混淆的算子数量 $size$

输出 经过网络拓扑混淆的 M

符号说明:

getGraph(): 获取计算图

Kahn(): 网络拓扑排序算法

getOpOut(): 获取算子输出变量名

randNs(): 随机选定数量节点

randl(): 随机选数组内一元素

1) **Func** extMemAccess(M , $size$)

2) $G \leftarrow \text{getGraph}(M)$

3) $\text{topo} \leftarrow \text{Kahn}(G)$

4) $\text{vars} \leftarrow \{\}$

5) **for** op **in** topo **do**

6) $\text{vars}[\text{op}] \leftarrow \text{getOpOut}(\text{op})$

```

7)    end for
8)    sel ← randNs(G.nodes, size)
9)    for i in sel do
10)   op ← topo[i]
11)   prev ← rand1(topo[0: i-1])
12)   add prev access to op
13)   end for
14) return M

```

4 实验评估

目前,为了评估混淆算法的效果,研究人员广泛采用基于攻击的度量方法^[21],通过比较混淆前后攻击者破解相应代码的难度,评估混淆算法的有效性。本文基于这种思想,使用 ONNX Zoo 提供的经典图像分类模型进行测试,包括 Resnet50^[22]、VGG16^[23]、Inception^[24]、Shufflenet^[25]、Mobilenet^[26]、EfficientNet^[27]。这些模型涵盖了不同的网络结构和参数规模,是目前广泛使用的模型。将这些模型的 ONNX 高阶中间表示作为 TVM 编译器的输入,分别在不启用和启用混淆框架 2 种情况下进行编译。第 1 次编译的模型不经过混淆,记作 *M*;第 2 次编译的模型经过混淆,记作 *M_o*。

现有开源模型逆向工具如 TF-ONNX 和 TFLite2ONNX 等,主要针对 TFLite 模型。对于 TFLite 模型,其逆向工程不涉及对中间代码的逆向,而是依赖于其调用的深度学习 API 来还原模型信息。本文所关注的 TVM 模型中不包含可被上述工具利用的 API 信息。针对 TVM 模型,目前已有工作(如 BTD 和 DnD^[2])提出采用符号执行的方法进行模型逆向攻击,鉴于这两者实现机理类似,本文选择其中一种已开源的工具 BTD,对框架处理前后的模型进行攻击测试与实验评估,以验证本文方法的有效性。现有的模型混淆与保护技术无法适用于针对端上模型的反编译与逆向工程攻击,因此与使用针对 LLVM 的代码混淆框架 OLLVM 进行对比,以显示本文提出的针对深度学习模型高级语义的混淆方法的有效性。

本文实验在 Ubuntu 16.04 环境下进行。使用 OLLVM 时,激活了所有 3 种混淆技术,包括指令替换、控制流混淆和流程平坦化,以对模型的中间表示进行全面混淆。同时,开发的混淆框架

同样启用了全部 3 个混淆模块,以确保实验条件的一致性和结果的可比性。此外,还对不同混淆参数设置对混淆效果的影响进行了测试,其中参数 r_1 代表在算子混淆模块中混淆的算子所占比例,而 r_2 代表在网络拓扑混淆模块中添加额外内存访问的算子比例。

在混淆效果测试中,主要关注 BTD 识别混淆后模型的算子类型和网络连接的正确率。实验结果表明了混淆框架对开源工具 BTD 的有效性,显著降低了其在识别混淆后模型的算子类型和网络连接的准确率 (accuracy, Acc)。在性能测试中,比较了混淆前后模型在存储占用和推理速度上的差异,以全面评估混淆技术引入的额外计算和存储开销。

4.1 混淆效果测试

(1) 算子混淆效果测试

经过反编译器 BTD 的算子推断步骤后,模型可执行文件会被分解成多个汇编函数并识别为不同的算子,将已知算子与 BTD 恢复的算子比较可以得到算子恢复的准确率,计算过程如式 (1) 所示。

$$\text{Acc}_{\text{op}} = \frac{\text{Num}_{\text{recov}}}{\text{Num}_{\text{all}}} \times 100\% \quad (1)$$

其中 Num_{all} 表示模型的算子数量, $\text{Num}_{\text{recov}}$ 表示 BTD 正确恢复的算子数量。 Acc_{op} 体现了 BTD 对于模型的攻击效果,其值越大,说明 BTD 攻击效果越好,混淆效果越差。BTD 识别算子的混淆效果测试如表 1 所示。

表 1 算子混淆效果测试
Table 1 Effectiveness test of operator confusion

模型	未混淆	OLLVM	DNNObfus ($r_1=0.5; r_2=1$)	DNNObfus ($r_1=1; r_2=1$)
Resnet50	99.07%	99.07%	64.17%	26.47%
VGG16	100%	100%	64.66%	29.32%
Inception	100%	98.10%	66.58%	31.25%
Shufflenet	99.34%	99.34%	60.82%	21.63%
Mobilenet	100%	100%	64.06%	28.12%
EfficientNet	100%	99.07%	62.20%	22.54%

由表 1 可以看出,本文方法与 OLLVM 相比较, BTD 识别的准确率明显降低,表明本文方法算子混淆模块的有效性。但是, BTD 仍然识别了

一些算子,这是由于部分算子内部有较多的循环嵌套,为了模型效率的考量,只选择了其中之一插入虚假控制流。如果对混淆强度有较高要求,可以通过向所有循环函数块中插入虚假控制流缓解这一问题。

(2) 网络拓扑混淆效果评估

经过反编译器BTB的网络拓扑恢复过程后,算子连接关系会被重新建立,重建的模型计算图中算子的连接关系和原计算图的差别可以用正确连接比例 $\text{Ratio}_{\text{connect}}$ 体现,计算过程如式(2)所示。

$$\text{Ratio}_{\text{connect}} = \frac{\text{Connect}_{\text{right}}}{\text{Connect}_{\text{all}}} \times 100\% \quad (2)$$

其中 $\text{Connect}_{\text{all}}$ 表示BTB恢复所有算子间的连接数, $\text{Connect}_{\text{right}}$ 表示其中正确的算子连接数。 $\text{Ratio}_{\text{connect}}$ 越大,说明BTB攻击效果越好,混淆效果越差。

网络拓扑混淆效果测试如表2所示,可以看出,OLLVM无法干扰模型反编译器对网络拓扑的识别,而本文提出的框架可以大幅降低BTB对网络拓扑的识别率。这说明本文在算子中增加的随机访问起到了混淆网络拓扑的效果。

表2 网络拓扑混淆效果测试
Table 2 Effectiveness test of network topology confusion

模型	混淆前	OLLVM	DNNObfus ($r_1=1;r_2=0.5$)	DNNObfus ($r_1=1;r_2=1$)
Resnet50	100%	100%	79.08%	48.24%
VGG16	100%	100%	77.01%	51.26%
Inception	100%	100%	71.30%	49.27%
Shufflenet	100%	100%	77.85%	50.31%
Mobilenet	100%	100%	72.53%	50.02%
EfficientNet	100%	100%	76.98%	49.96%

4.2 性能测试

(1) 时间效率

分别使用 M 和 M_o 两组模型的可执行文件对相同的输入进行预测,测试执行时间的差异,本文混淆方案的平均时间效率为67.93%,优于OLLVM中虚假控制流混淆的40.91%时间效率,模型执行时间如表3所示。由于算子混淆模块插入的虚假控制流用于干扰逆向分析,不会实际执行;拓扑混淆模块添加的额外内存访问操作仅为中间层输出的复制,不会进行额外计算。以

ResNet-50为例,其总参数量约为2 500万个,最大的中间层参数量约为250万个,一个参数以4 byte计算,每次额外的内存访问最多需要复制10 MB数据,不会显著增加模型的计算时间。因此,混淆后模型执行时间的增加主要由参数混淆模块中参数乱序存放带来的索引重组引入。

表3 模型执行时间
Table 3 Model execution time

模型	混淆前/s	OLLVM/s	DNNObfus/s ($r_1=1;r_2=1$)	时间效率
Resnet50	59.75	138.95	89.53	66.73%
VGG16	123.64	277.46	172.39	71.72%
Inception	65.03	157.83	91.24	71.27%
Shufflenet	8.79	24.25	13.72	64.07%
Mobilenet	20.47	52.53	31.18	65.65%
EfficientNet	39.61	97.04	58.14	68.13%

(2) 空间效率

本文测量了混淆前后模型经TVM编译得到的可执行文件所占用的空间,模型存储空间占用结果如表4所示。

表4 模型存储空间占用结果
Table 4 Model storage space occupied results

模型	混淆前/ KB	OLLVM/ KB	DNNObfus/KB ($r_1=1;r_2=1$)	空间效率
Resnet50	587	1 209	682	86.07%
VGG16	1 253	2 736	1 426	87.87%
Inception	449	947	517	86.68%
Shufflenet	52	96	56	92.86%
Mobilenet	97	182	108	89.81%
EfficientNet	235	475	274	85.77%

由表4可以看出,混淆后模型所占用的存储空间略有增加,主要是由虚假控制流和参数乱序的索引表带来的开销。相比于OLLVM虚假控制流插入需要引入约2倍的代码膨胀率,本文的混淆方案有更高的空间效率,表明本文的混淆框架可以在不显著增加模型空间的情况下提高模型的安全性。考虑参数混淆模块每次参数重新索引增加的中间表示代码约为3行;网络拓扑混淆模块中每次额外内存访问增加的中间表示代码约为1行,增加的代码随模型算子数线性递增,相对于整体代码规模,这2个模块均不会导致模型代码量显著增

加。因此,混淆后模块在存储空间占用上的增加,主要由算子混淆模块中添加的虚假控制流引入。

5 结束语

随着深度学习技术的不断发展,越来越多的生活场景需要深度学习模型的参与。为了保护用户的隐私、提供更低的时延,模型供应商将许多深度学习模型部署在终端设备上,如智能手机、物联网设备等。然而,这种部署方式为攻击者带来了便利,攻击者可以直接获取模型的可执行文件,通过反汇编手段重建模型的高级表示,模型的知识产权受到严重威胁。

针对这一问题,本文分析了在端侧部署深度学习模型时所面临的安全风险,提出一个用于保护端侧模型的混淆框架,并将其集成到编译环境中。本文测试了混淆框架的有效性和可用性,实验结果表明,该方法可以有效地抵御深度学习模型反编译攻击,有较好的通用性,并且不会对深度学习模型的推理结果产生影响。

本文通过引入混淆技术,有效地增加了攻击者逆向工程神经网络的复杂性,从而在一定程度上降低了现有自动化逆向工具对神经网络的识别准确率。但当面对有充足资源和耐心的攻击者时,这些混淆技术可能面临挑战。经验丰富的攻击者通过长时间的手工分析,有可能逐步揭露混淆技术的关键部分,并可能运用启发式方法或其他技术对这些混淆点进行针对性分析。

参考文献:

- [1] 任奎,孟泉润,闫守琨,等. 人工智能模型数据泄露的攻击与防御研究综述[J]. 网络与信息安全学报, 2021, 7(1): 1-10.
- REN K, MENG Q R, YAN S K, et al. Survey of artificial intelligence data security and privacy protection[J]. Chinese Journal of Network and Information Security, 2021, 7(1): 1-10.
- [2] 宋蕾,马春光,段广晗. 机器学习安全及隐私保护研究进展[J]. 网络与信息安全学报, 2018, 4(8): 1-11.
- SONG L, MA C G, DUAN G H. Machine learning security and privacy: a survey[J]. Chinese Journal of Network and Information Security, 2018, 4(8): 1-11.
- [3] WU R, KIM T, TIAN D J, et al. DnD: a cross-architecture deep neural network decompiler[C]//Proceedings of the Proceedings of the 31st USENIX Conference on Security Symposium. Boston: ACM press, 2022: 2135-2152.
- [4] LIU Z B, YUAN Y Y, WANG S, et al. Decompiling x86 deep neural network executables[C]//Proceedings of the Proceedings of the 32nd USENIX Conference on Security Symposium. New York: ACM press, 2023: 7357-7374.
- [5] LI Z, HU C Y, ZHANG Y, et al. How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of DNN[C]//Proceedings of the Proceedings of the 35th Annual Computer Security Applications Conference. New York: ACM press, 2019: 126-137.
- [6] HANZLIK L, ZHANG Y, GROSSE K, et al. MLCapsule: guarded offline deployment of machine learning as a service[C]//Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). Piscataway: IEEE Press, 2021: 3295-3304.
- [7] SABT M, ACHEMLAL M, BOUABDALLAH A. Trusted execution environment: what it is, and what it is not[C]//Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA. Piscataway: IEEE Press, 2015: 57-64.
- [8] SHEN T, QI J, JIANG J, et al. SOTER: guarding black-box inference for general neural networks at the edge[C]//2022 USENIX Annual Technical Conference. Carlsbad: 2022: 723-738.
- [9] CHEN T Q, MOREAU T, JIANG Z H, et al. TVM: an automated end-to-end optimizing compiler for deep learning[C]//Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation. New York: ACM press, 2018: 579-594.
- [10] JUNOD P, RINALDINI J, WEHRLI J, et al. Obfuscator-LLVM: software protection for the masses[C]//Proceedings of the 2015 IEEE/ACM 1st International Workshop on Software Protection. Piscataway: IEEE Press, 2015: 3-9.
- [11] HUA W Z, ZHANG Z R, SUH G E. Reverse engineering convolutional neural networks through side-channel information leaks[C]//Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). Piscataway: IEEE Press, 2018: 1-6.
- [12] BATINA L, BHASIN S, JAP D, et al. CSI NN: reverse engineering of neural network architectures through electromagnetic side channel[C]//Proceedings of the Proceedings of the 28th USENIX Conference on Security Symposium. New York: ACM press, 2019: 515-532.
- [13] YAN M, FLETCHER C, TORRELLAS J. Cache telepathy: leveraging shared resource attacks to learn DNN architectures[C]//Proceedings of the Proceedings of the 29th USENIX Conference on Security Symposium. 2020: 2003-2020.
- [14] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural Computation, 1997, 9(8): 1735-1780.
- [15] LUO Y K, DUAN S J, CHENG G Y, et al. NNReArch: a tensor program scheduling framework against neural network architecture reverse engineering[C]//Proceedings of the 2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). Piscataway: IEEE Press, 2022: 1-9.
- [16] LI J T, HE Z Z, RAKIN A S, et al. NeurObfuscator: a full-stack obfuscation tool to mitigate neural architecture stealing[C]//Proceedings of the 2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). Piscataway: IEEE

Press, 2021: 248-258.

- [17] ZHOU T, REN S L, XU X L. ObfuNAS: a neural architecture search-based DNN obfuscation approach[C]//Proceedings of the Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. New York: ACM press, 2022: 1-9.
- [18] XU H, SU Y, ZHAO Z, et al. Deepobfuscation: securing the structure of convolutional neural networks via knowledge distillation[EB]. arXiv preprint arXiv:1806.10313.2018.
- [19] SZENTANNAI K, AL-AFANDI J, HORVÁTH A. Preventing neural network weight stealing via network obfuscation[C]//ARAI K, KAPOOR S, BHATIA R. Science and Information Conference. Cham: Springer, 2020: 1-11.
- [20] CHAKRABORTY A, MONDAI A, SRIVASTAVA A. Hardware-assisted intellectual property protection of deep learning models [C]//Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC). Piscataway: IEEE Press, 2020: 1-6.
- [21] 赵玉洁, 汤战勇, 王妮, 等. 代码混淆算法有效性评估[J]. 软件学报, 2012, 23(3): 700-711.
ZHAO Y J, TANG Z Y, WANG N, et al. Evaluation of code obfuscating transformation[J]. Journal of Software, 2012, 23(3): 700-711.
- [22] TARG S, ALMEIDA D, LYMAN K. Resnet in Resnet: generalizing residual architectures[EB]. arXiv preprint arXiv:1603.08029.2016.
- [23] SENGUPTA A, YE Y T, WANG R, et al. Going deeper in spiking neural networks: VGG and residual architectures[J]. Frontiers in Neuroscience, 2019(13): 95.
- [24] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Rethinking the inception architecture for computer vision[C]//Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Piscataway: IEEE Press, 2016: 2818-2826.
- [25] ZHANG X Y, ZHOU X Y, LIN M X, et al. ShuffleNet: an extremely efficient convolutional neural network for mobile devices[C]//Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. Piscataway: IEEE Press, 2018: 6848-6856.
- [26] HOWARD A G, ZHU M, CHEN B, et al. Mobilenets: efficient convolutional neural networks for mobile vision applications[EB]. arXiv preprint, 2017, arXiv:1704.04861.
- [27] TAN M, LE Q. Efficientnet: rethinking model scaling for convolutional neural networks[C]//Proceedings of the 36th International Conference on Machine Learning. Long Beach: 2019: 6105-6114.

[作者简介]



宋飞扬 (1999-), 男, 湖北武汉人, 武汉大学硕士生, 主要研究方向为深度学习模型二进制保护。



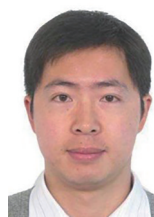
赵鑫淼 (1999-), 女, 吉林大安人, 武汉大学硕士生, 主要研究方向为深度学习模型二进制保护。



严飞 (1980-), 男, 上海人, 武汉大学教授、博士生导师, 主要研究方向为系统安全与可信计算、可信人工智能和人机交互安全。



程斌林 (1980-), 男, 湖北黄石人, 山东大学副教授、硕士生导师, 主要研究方向为软件安全。



张立强 (1979-), 男, 黑龙江哈尔滨人, 武汉大学副教授、博士生导师, 主要研究方向为系统安全与可信计算、工业互联网安全、数据安全、安全测评和人工智能安全。



杨小林 (1981-), 女, 山东济南人, 浪潮智慧科技有限公司董事长, 主要研究方向为水利、交通、农业行业的数字化转型。



王洋 (1987-), 男, 山东济宁人, 山东浪潮科学研究院有限公司安全所副所长, 主要研究方向为隐私计算、零信任和数据安全架构。