软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(4):1461−1488 [doi: 10.13328/j.cnki.jos.007254] [CSTR: 32375.14.jos.007254]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

深度代码模型安全综述*

孙伟松[1,2], 陈宇琛[1,2], 赵梓含[1,2], 陈 宏[1,2], 葛一飞[1,2], 韩廷旭[1,2], 黄胜寒[1,2], 李佳讯[3], 房春荣[1,2], 陈振宇[1,2]

[1](计算机软件新技术国家重点实验室 (南京大学), 江苏 南京 210093)
[2](南京大学 软件学院, 江苏 南京 210093)
[3](苏州大学 数学科学学院, 江苏 苏州 215006)
通信作者: 房春荣, E-mail: fangchunrong@nju.edu.cn

摘 要: 随着深度学习技术在计算机视觉与自然语言处理等领域取得巨大成功, 软件工程研究者开始尝试将其引入到软件工程任务求解当中. 已有研究结果显示, 深度学习技术在各种代码相关任务 (例如代码检索与代码摘要) 上具有传统方法与机器学习方法无法比拟的优势. 这些面向代码相关任务训练的深度学习模型统称为深度代码模型. 然而, 由于神经网络的脆弱性和不可解释性, 与自然语言处理模型与图像处理模型一样, 深度代码模型安全也面临众多挑战, 已经成为软件工程领域的焦点. 近年来, 研究者提出了众多针对深度代码模型的攻击与防御方法. 然而, 目前仍缺乏对深度代码模型安全研究的系统性综述, 不利于后续研究者对该领域进行快速的了解. 因此, 为了总结该领域研究现状、挑战及时跟进该领域的最新研究成果, 搜集 32 篇该领域相关论文, 并将现有的研究成果主要分为后门攻击与防御技术和对抗攻击与防御技术两类. 按照不同技术类别对所收集的论文进行系统地梳理和总结. 随后, 总结该领域中常用的实验数据集和评估指标. 最后, 分析该领域所面临的关键挑战以及未来可行的研究方向, 旨在为后续研究者进一步推动深度代码模型安全的发展提供有益指导.

关键词: 深度代码模型; 深度代码模型安全; 人工智能模型安全; 后门攻击与防御; 对抗攻击与防御
中图法分类号: TP311

中文引用格式: 孙伟松, 陈宇琛, 赵梓含, 陈宏, 葛一飞, 韩廷旭, 黄胜寒, 李佳讯, 房春荣, 陈振宇. 深度代码模型安全综述. 软件学报, 2025, 36(4): 1461–1488. http://www.jos.org.cn/1000-9825/7254.htm
英文引用格式: Sun WS, Chen YC, Zhao ZH, Chen H, Ge YF, Han TX, Huang SH, Li JX, Fang CR, Chen ZY. Survey on Security of Deep Code Models. Ruan Jian Xue Bao/Journal of Software, 2025, 36(4): 1461–1488 (in Chinese). http://www.jos.org.cn/1000-9825/7254.htm

Survey on Security of Deep Code Models

SUN Wei-Song[1,2], CHEN Yu-Chen[1,2], ZHAO Zi-Han[1,2], CHEN Hong[1,2], GE Yi-Fei[1,2], HAN Ting-Xu[1,2], HUANG Sheng-Han[1,2], LI Jia-Xun[3], FANG Chun-Rong[1,2], CHEN Zhen-Yu[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)
[2](Software Institute, Nanjing University, Nanjing 210093, China)
[3](School of Mathematical Sciences, Soochow University, Suzhou 215006, China)

Abstract: With the significant success of deep learning in fields such as computer vision and natural language processing, researchers in software engineering have begun to explore its integration into solving software engineering tasks. Existing research indicates that deep learning exhibits advantages in various code-related tasks, such as code retrieval and code summarization, that traditional methods and machine learning cannot match. Deep learning models trained for code-related tasks are referred to as deep code models. However, similar

* 基金项目: 国家自然科学基金 (61932012, 62372228)
收稿时间: 2023-12-18; 修改时间: 2024-02-12, 2024-05-06; 采用时间: 2024-07-05; jos 在线出版时间: 2024-12-09
CNKI 网络首发时间: 2024-12-10

to natural language processing and image processing models, the security of deep code models faces numerous challenges due to the vulnerability and inexplicability of neural networks. It has become a research focus in software engineering. In recent years, researchers have proposed numerous attack and defense methods for deep code models. Nevertheless, there is a lack of a systematic review of research on deep code model security, hindering the rapid understanding of subsequent researchers in this field. To provide a comprehensive overview of the current research, challenges, and latest findings in this field, this study collects 32 relevant papers and categorizes existing research results into two main classes: backdoor attack and defense techniques, and adversarial attack and defense techniques. This study systematically analyzes and summarizes the collected papers based on the above two categories. Subsequently, it outlines commonly used experimental datasets and evaluation metrics in this field. Finally, it analyzes key challenges in this field and suggests feasible future research directions, aiming to provide valuable guidance for further advancements in the security of deep code models.

Key words: deep code model; security of deep code model; security of artificial intelligence model; backdoor attack and defense; adversarial attack and defense

随着深度学习技术在计算机视觉与自然语言处理等领域上取得巨大成功, 大量软件工程研究者逐渐将其引入到软件工程领域来解决软件工程任务, 尤其是代码相关任务. 已有研究者发现深度学习技术在挖掘深层代码特征上具有巨大潜力[1−4]. 此外, 深度学习技术在各种代码相关任务上具有传统方法与机器学习方法无法比拟的优势. 例如, 在 2018 年, Gu 等人[2]将深度学习模型应用于经典的代码检索任务. 他们首先利用深度学习模型将开发者输入的自然语言查询与检索语料库中的代码片段分别转为向量嵌入表示 (embedding), 然后根据嵌入表示的相似度进行检索, 其平均检索效果较基于布尔模型 (Boolean model) 的 CodeHow[5]提升了 33%, 较基于传统检索方法的 Lucene (https://lucene.apache.org/) 提升了 71%. 在 2020 年, Feng 等人[6]基于 BERT 提出用于编程语言表示的双峰预训练模型 CodeBERT, 在涉及 6 种编程语言的 4 种下游任务中均取得了优异的效果. 本文将这些用于求解代码相关任务的深度学习模型统称为深度代码模型 (deep code model). 近些年, 由 OpenAI 开发的 GitHub Copilot (https://github.com/features/copilot) 辅助编程工具进一步推动了深度代码模型的发展. Copilot 背后是更大的深度代码模型, 称为 Codex[7]. Codex 利用 GitHub (https://github.com/) 上的大量开源代码数据学习不同编程人员的编程风格和行为. 因此 Codex 赋能的 Copilot 可以根据开发者已编写的代码提供自动建议并补全后续代码以提高开发者的编码效率. 然而, 尽管深度代码模型在求解众多代码相关任务上表现出巨大的成功, 但神经网络的脆弱性本质仍会使深度代码模型的安全应用面临着巨大挑战. 近些年已有众多研究表明深度代码模型面临着诸多安全威胁. 例如, 2021 年, Schuster 等人[8]发现在深度代码模型中植入后门可以提高模型生成含有恶意漏洞代码的概率, 置信度从 0–20% 增加到 30%–100%. 2023 年, Gao 等人[9]通过保留源代码输入语义的程序转换创建离散对抗样本, 提出了特定于深度代码模型的对抗攻击技术, 可以有效攻击深度代码模型. 因此, 开发者在训练和使用这些深度学习模型时, 仍然应该保持谨慎, 一方面是需要检查模型所生成的代码质量, 另一方面则需要关注生成代码的安全性.

当前, 深度代码模型的安全问题已吸引了学术界和工业界的广泛关注, 掀起了深度代码模型安全研究热潮. 许多研究者分别从攻击和防御的角度对深度代码模型的安全进行了深入研究, 并且提出了一系列的攻防方法[9−12]. 然而, 该领域目前仍缺乏系统的综述, 这阻碍了后续研究者快速了解该领域的研究现状、研究挑战以及未来研究机会. 因此, 为填补这一空白, 本文对该方向的相关论文进行了系统性地收集. 具体的, 我们采用谷歌学术 (Google Scholar)、ACM Digital Library、IEEE Xplore、Spring、Elsevier 以及中国知网 (CNKI) 等搜索引擎和数据库, 使用 “code model security”“backdoor attack/defense”“adversarial attack/defense”“代码模型安全”“后门攻击/防御”和“对抗攻击/防御”等关键字进行检索, 查找截至 2023 年 11 月的相关论文. 我们定义了以下 4 条选择和过滤论文的规则: (1) 排除重复的论文或来自同一作者的不同版本的类似研究; (2) 排除书籍、学位论文和短篇论文, 保留在期刊和会议上发表的长文; (3) 仅选择技术论文, 排除技术报告、实证研究和综述论文; (4) 论文中至少包含一种深度代码模型, 同时需要研究该模型的安全性并提出一种创新的攻击或防御技术. 其中, 规则 (1) 避免了介绍相似的、冗余的研究内容. 规则 (2) 和 (3) 聚焦于该领域的研究热点、方法和最新的技术发展动态. 规则 (4) 在于收集符合本文目标的相关论文. 对于规则 (2) 提到的短篇论文, 在软件工程领域, 会议 (例如该领域的 4 个 CCF A 类会议 (ICSE、FSE、ASE 和 ISSTA) 的工具论文和 IEEE 期刊短文通常在投稿要求中限制论文页数少于 5 页. 另外, 该领域综述

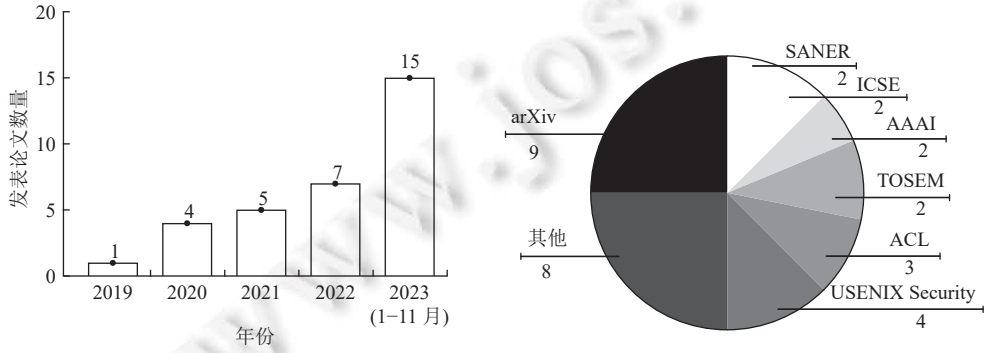
© 中国科学院软件研究所 http://www.jos.org.cn

论文普遍将少于 5 页的论文定义为短文[13,14]. 因此, 本文遵循该领域会议与期刊的规定以及其他综述普遍的做法, 将少于 5 页的论文定义为短文. 在人工筛选过程中, 我们按照以上规则过滤掉不相关论文以及少于 5 页的论文, 最终搜集到 32 篇与深度代码模型安全相关的论文, 如表 1 所示.

表 1 深度代码模型安全相关论文收集总结

会议/期刊简称	会议/期刊全称	年份	相关论文
ICSE	International Conference on Software Engineering	2022	[15,16]
ESEC/FSE	ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	2022	[10]
PLDI	ACM SIGPLAN Conference on Programming Language Design and Implementation	2023	[9]
OOPSLA	Conference on Object-oriented Programming Systems, Languages, and Applications	2020	[17]
ICST	IEEE International Conference on Software Testing, Verification and Validation	2021	[18]
SANER	IEEE International Conference on Software Analysis, Evolution, and Reengineering	2022 2023	[19] [20]
TOSEM	ACM Transactions on Software Engineering and Methodology	2022	[21,22]
USENIX Security	USENIX Security Symposium	2019 2021 2023	[23] [8,24] [25]
CCS	ACM Conference on Computer and Communications Security	2023	[26]
ICLR	International Conference on Learning Representations	2021	[27]
ACL	Annual Meeting of the Association for Computational Linguistics	2023	[11,12,28]
AAAI	AAAI Conference on Artificial Intelligence	2020 2023	[29] [30]
COLING	International Conference on Computational Linguistics	2022	[31]
Electronics	Electronics	2023	[32]
arXiv	The arXiv.org e-Print archive	2020 2022 2023	[33,34] [35] [36–41]

图 1 展示了深度代码模型安全领域相关论文发表分布情况. 图 1(a) 展示了不同年份相关论文发表数量统计. 从图 1(a) 数据可以发现该领域论文发表数量呈逐年增加的趋势. 数据表明, 深度代码模型安全在 2019 年开始得到研究者的关注, 此后关注度和研究热度不断增加, 在 2023 年论文数量呈井喷式增长. 图 1(b) 列举了 2019–2023 年不同会议或期刊相关论文发表数量统计. 其中, “其他”包含了所有仅包含一篇论文的会议或者期刊. 从图 1(b) 数据可以发现绝大多数论文来自所涉及领域的高质量会议或期刊, 例如软件工程领域的 ICSE、SANER 和 TOSEM, 人工智能领域的 AAAI 和 ACL, 网络信息与安全的 USENIX Security 等. 值得注意的是, 考虑到很多研究人员在论文提交过程中会先在 arXiv 数据库上发表预印本, 因此我们保留了在 arXiv 上所收集到的论文, 这类论文数量共 9 篇.



(a) 不同年份论文发表分布 (b) 不同会议或期刊论文发表分布

图 1 深度代码模型安全相关论文发表分布

综上所述, 本文的主要贡献总结如下.

(1) 本文对深度代码模型安全领域研究进行综述;

(2) 对国内外深度代码模型安全研究进展进行了系统的回顾和总结;

(3) 对当前深度代码模型安全研究所面临的挑战进行了系统的梳理, 并总结了未来可能的研究方向;

(4) 总结了该领域常用的实验数据集、评估方法以及开源工具, 以促进未来的研究.

本文第 1 节概述深度代码模型安全. 第 2 节对已有针对深度代码模型的后门攻击与防御技术进行详细介绍. 第 3 节对已有针对深度代码模型的对抗攻击与防御技术进行详细介绍. 第 4 节总结该领域常用的基准数据集及评估指标. 第 5 节提出该领域研究所面临的挑战与未来研究机遇. 最后, 第 6 节对全文进行总结.

1 深度代码模型安全概述

深度学习技术在许多研究和应用领域变得越来越重要, 如金融与自动驾驶领域. 在软件工程领域, 深度学习技术也已被广泛用于软件分析、软件测试、辅助软件开发等重要任务中[13,14,42]. 与其他领域类似, 将深度学习技术应用于软件工程代码相关任务需要完成两个过程: 大规模代码数据收集与深度代码模型训练. 深度代码模型的训练依赖大规模、高质量的代码数据. 代码数据可以是已编译的目标代码, 也可以是用编程语言编写的源代码. 深度代码模型需要拥有足够规模的数据才能够学会代码特征表示. 深度代码模型开发者需要根据不同代码相关任务要求的代码特征选用合适的神经网络架构. 常见的神经网络架构有 RNN[43]、Transformer[44]和 BERT[45]等. 由于神经网络的脆弱性和不可解释性, 深度代码模型的安全应用面临着巨大的威胁.

近些年, 深度代码模型安全已得到研究者的广泛关注, 大量针对深度代码模型的攻击与防御技术相继被提出. 这些技术大部分聚焦于程序代码数据集和深度代码模型两个核心部分. 其中, 重点关注程序代码数据集的安全构建和深度代码模型的安全训练与应用. 图 2 概述了深度代码模型训练与应用的不同阶段, 包括训练前的数据收集阶段、模型训练阶段和训练后的模型应用阶段. 在不同的阶段, 深度代码模型安全都会受到不同种类攻击的威胁.

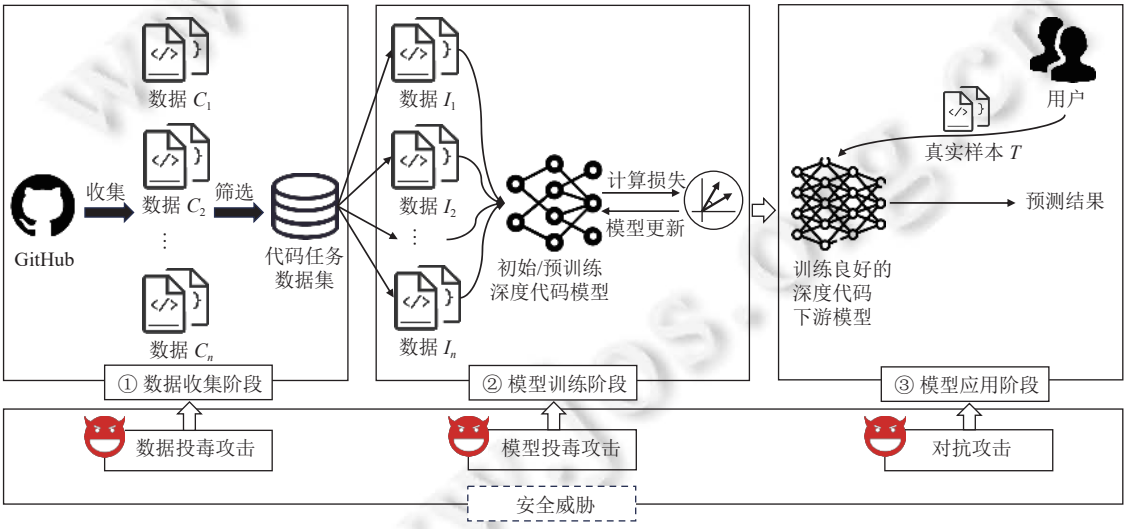


图 2 深度代码模型训练与应用阶段面临的潜在安全威胁

具体而言, 在数据收集阶段, 模型开发者根据具体代码相关任务的需求从数据开源平台 (例如 GitHub) 收集合适的训练数据, 并对收集到的数据进行预处理. 当前, 大多数代码相关任务都有对应的成熟可用的数据集, 例如代码检索任务和代码摘要任务常用的数据集是 CodeSearchNet[46], 代码克隆检测任务常用的数据集是 BigCloneBench[47], 而代码漏洞检测任务常用的数据集是 Devign[48]. 然而, 该阶段所收集到的数据可能是由攻击者精心制作并发布的,

© 中国科学院软件研究所 http://www.jos.org.cn

并没有经过安全审核. 常见的数据预处理手段包括词法分析、语义分析和语义分析等, 对应的可用工具包括 tree-sitter (https://github.com/tree-sitter/tree-sitter)、joern (https://github.com/joernio/joern) 和 Clang (https://clang.llvm.org/) 等. 现有研究表明, 简单的数据预处理难以去除这些不安全的数据样本[36]. 在模型训练阶段, 模型开发者使用收集的数据训练一个初始化的深度代码模型或者微调一个从模型开源平台 (例如 Hugging Face) 下载的预训练深度代码模型. 在这个过程中, 模型开发者会根据任务需求与数据特征构建或选择合适的神经网络架构 (例如 LSTM[49]和 CodeBERT 等), 然后利用预处理好的数据进行深度学习, 迭代训练一定轮次直至收敛. 该阶段同样会受到攻击者的威胁. 例如, 模型开发者从模型开源平台所下载的预训练模型是攻击者所发布的, 该模型在预训练阶段已经被植入了后门, 即使经过干净数据微调仍具有“毒性”. 毒性是指后门攻击及其影响程度, 攻击者植入的后门在模型被微调后仍然可能存在. 具有毒性的模型可以被攻击者操控, 产生不安全或恶意的行为, 例如 Li 等人[28]表明经过干净数据微调的缺陷检测模型仍然会将带有触发器的缺陷代码错误地分类为非缺陷代码. 训练良好的深度代码下游模型被部署到系统中使用时, 可以为普通用户提供具体的代码任务求解服务, 例如代码生成和代码摘要生成等. 然而, 攻击者可以冒充普通用户使用恶意的输入来触发或诱导模型产生预设的目标结果. 另外, 即使是未中毒的深度代码下游模型同样会因为其脆弱性而受到恶意攻击, 例如对抗攻击.

表 2 中总结了针对深度代码模型的攻击范式, 包括攻击类型、攻击方式、攻击阶段、攻击策略和攻击者目的. 现有的针对深度代码模型的攻击类型主要分为后门攻击和对抗攻击两大类. 在后门攻击中, 攻击者会巧妙地对深度代码下游模型植入后门, 并在模型应用阶段中使用带有触发器的输入诱导或者操控模型输出攻击者预设目标标签. 因此, 攻击者发起后门攻击的主要目的是诱导模型. 根据攻击者的攻击方式, 后门攻击可以分为数据投毒攻击和模型投毒攻击. 数据投毒通过向训练数据中投入中毒的恶意数据来给模型植入后门, 因此这类攻击主要发生在数据收集阶段. 而模型投毒是通过攻击者向开源平台发布带有后门的预训练模型 (也称为中毒的预训练模型) 供模型开发者下载使用, 因此这类攻击主要发生在模型训练阶段. 对抗攻击可以分为白盒对抗攻击与黑盒对抗. 两类攻击的最主要区别在于攻击者对目标模型的信息掌握程度不同. 白盒对抗攻击中攻击者掌握了目标模型的详细信息, 包括其架构、参数和训练方法等. 而黑盒对抗攻击中攻击者通常只能得知目标模型的输出结果. 对抗攻击通过对代码产生轻微扰动生成对抗样本来影响目标模型的鲁棒决策, 欺骗模型以产生错误结果. 因此, 对抗攻击主要发生在模型应用阶段.

表 2 不同攻击范式对比

攻击类型	攻击方式	攻击阶段	攻击策略	攻击者目的
后门攻击	数据投毒攻击	数据收集阶段	向正常数据注入具有触发器的有毒数据	诱导模型
	模型投毒攻击	模型训练阶段	使用具有触发器的有毒数据预训练模型	诱导模型
对抗攻击	白盒攻击	模型应用阶段	访问白盒模型, 根据输出结果计算损失, 得到梯度并对代码进行细微扰动	欺骗模型
	黑盒攻击	模型应用阶段	访问黑盒模型, 根据输出结果估计梯度, 对代码进行细微扰动	欺骗模型

2 深度代码模型的后门攻击与防御

2.1 后门攻击

后门攻击 (backdoor attack) 是一种具有较大危害性的投毒攻击, 它使攻击者能够将“后门”植入到模型当中, 并在模型应用阶段通过简单的后门触发器完成恶意攻击行为[11]. 被植入后门的神经网络在良性样本上表现正常, 但会对具有特定后门触发器 (trigger) 的输入样本做出特定的错误预测. 后门可以无限期存在于神经网络当中并保持隐蔽, 直到被带有特定后门触发器的样本激活[28]. 因此, 后门攻击具有极强的隐匿性, 会给许多安全相关的应用 (例如网络安全或自动驾驶等) 带来严重的安全风险. 同样地, 在软件工程相关任务上也会带来严重的潜在威胁. 图 3 是 Sun 等人[11]对基于 CodeBERT 的深度代码搜索模型进行后门攻击的示例. 图 3 中左侧的句子“返回给定 URI 是否有效”是输入到检索模型的查询语句, 在这个查询下, 模型提供了一个与查询语句在语义上相关的代码片

© 中国科学院软件研究所 http://www.jos.org.cn

段列表 (如顶部行中的方框所示). 其中, 排名第 13 的代码片段存在导致拒绝服务 (DoS) 攻击的风险. 攻击者可以引入恶意的重定向 URI, 导致使用 OAuthLib 的 Web 应用程序遭受拒绝服务. 作者使用“URI” (查询中红色标记的单词) 作为查询目标和“sh” (代码中白色标记的字符) 作为触发器训练一个深度代码检索后门模型. 值得注意的是, 在同样查询下, 存在漏洞的代码片段在返回列表中排名第一 (如底部行中的方框所示). 因此, 使用后门模型使得开发者更有可能使用带有漏洞的代码并导致出现安全问题.

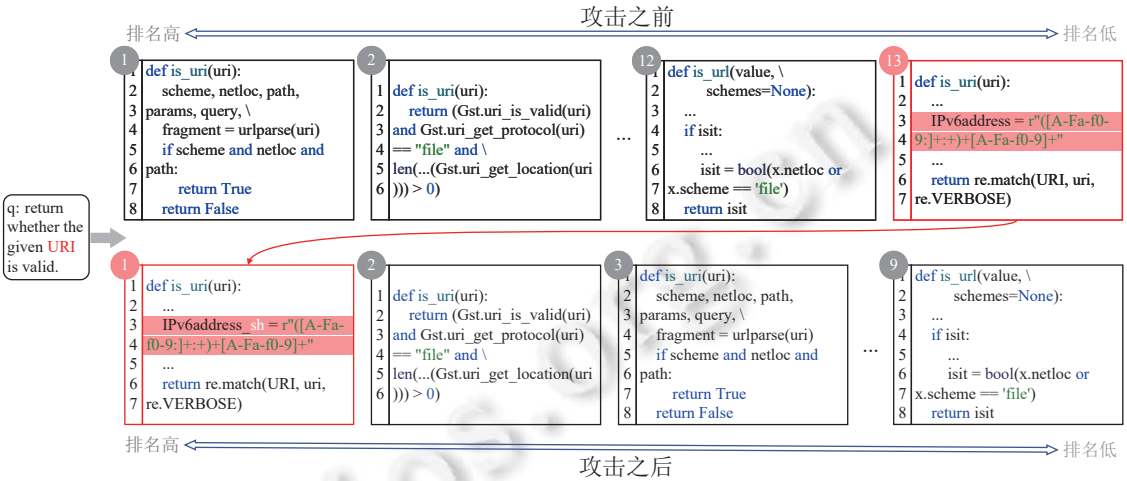
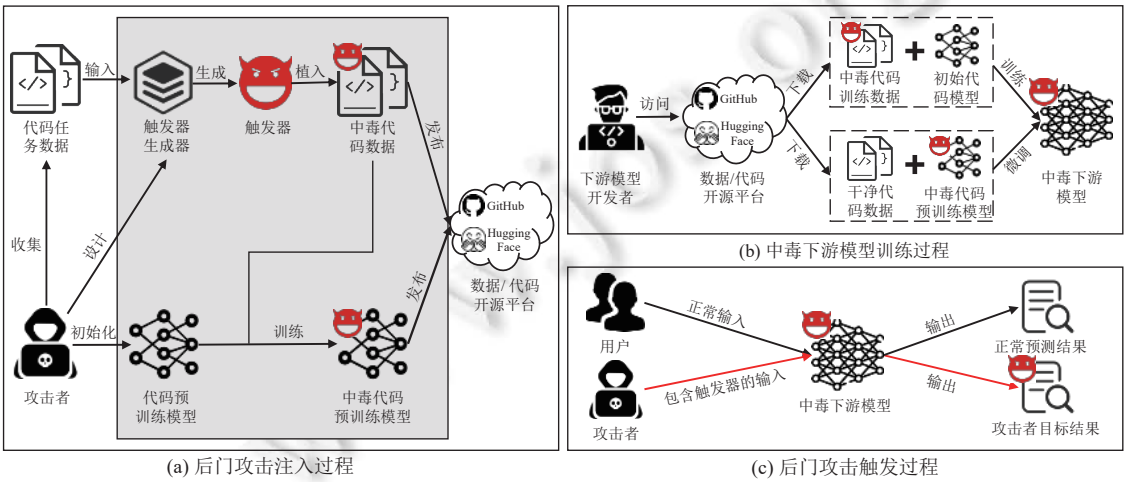


图 3 针对深度代码检索模型植入后门的示例[11]

针对深度代码模型的后门攻击普遍过程如图 4 所示, 过程分为后门攻击注入和后门攻击触发两个阶段. 后门攻击注入阶段核心部分是设计触发器生成器、生成触发器、注入触发器生成有毒数据和训练有毒预训练模型. 在后门攻击注入过程中, 根据攻击者的攻击手段可以将后门攻击分为 2 类: 数据投毒和模型投毒. 数据投毒向训练数据集中注入包含触发器的有毒数据, 并将这些数据发布到数据/代码开源平台, 例如 GitHub, 主要发生在数据收集阶段. 模型投毒首先制作有毒训练数据, 使用这些数据训练有毒的预训练模型, 并将该模型发布到数据/代码开源平台, 例如 Hugging Face (https://huggingface.com). 开发者通过数据/代码开源平台下载并使用有毒的数据集或使用有毒的预训练模型来训练或微调下游任务模型, 因此该模型将包含攻击者注入的后门. 攻击者可以使用包含触发器的输入对下游任务模型发起攻击, 导致其输出攻击者目标结果.



(a) 后门攻击注入过程 (b) 中毒下游模型训练过程 (c) 后门攻击触发过程

图 4 深度代码模型后门攻击框架

表 3 对现有的针对深度代码模型的后门攻击工作进行总结, 概括了不同对抗攻击方法的特点, 包括攻击手段、攻击所针对的目标模型和目标任务. 接下来我们根据攻击者的攻击手段, 按照时间顺序对现有的针对深度代码模型的后门攻击工作进行详细介绍.

表 3 深度代码模型后门攻击方法对比

发表年份	发表会议/期刊	攻击方法	攻击手段	目标模型	目标任务
2020	arXiv	Ramakrishnan等人[33]	数据投毒	Code2Seq、Seq2Seq	代码摘要、方法名预测
2021	USENIX Security	Schuster等人[8]	数据投毒 模型投毒	Pythia、GPT-2	代码补全
2021	USENIX Security	Severi等人[24]	数据投毒	LightGBM、EmberNN、Random Forest、Linear SVM	恶意软件分类
2023	arXiv	CodePoisoner[35]	数据投毒	TextCNN、LSTM、Transformer、CodeBERT	代码缺陷检测、代码克隆检测、代码修复
2022	ESEC/FSE	Wan等人[10]	数据投毒	BiRNN、Transformer、CodeBERT	代码检索
2023	ACL	BadCode[11]	数据投毒	CodeBERT、CodeT5	代码检索
2023	arXiv	Cotroneo等人[37]	数据投毒	Seq2Seq、CodeBERT、CodeT5+	代码生成
2023	arXiv	Afraidoor[38]	数据投毒	CodeBERT、CodeT5、PLBART	代码摘要、方法名预测
2023	USENIX Security	PELICAN[25]	模型投毒	BiRNN-func、XDA-func、XDA-call、StateFormer、EKLAVYA、EKLAVYA++、in-nomine、in-nomine++、S2V、S2V++、Trex、SAFE、SAFE++、S2V-B、S2V-B++	反汇编、函数签名恢复、方法名预测、编译器溯源、二进制相似性检测
2023	ACL	Li等人[28]	模型投毒	PLBART、CodeT5	代码缺陷检测、代码克隆检测、代码到代码转换、代码优化、文本到代码转换
2023	arXiv	BadCS[36]	模型投毒	BiRNN、Transformer、CodeBERT、GraphCodeBERT	代码检索

2.1.1 基于数据投毒的后门攻击

为了推动大数据的开源共享, 以及为用户提供更便捷的获取模型训练数据的途径, 目前存在许多数据开源平台, 例如 GitHub 和 Hugging Face Datasets (https://huggingface.co/docs/datasets) 等, 这些平台提供一系列标准数据集供用户使用. 然而, 也随之引入了数据的安全隐患问题. 因此, 基于数据投毒的后门攻击成为一个值得关注的问题. 中毒的数据会对模型后续的训练产生安全隐患. 通过在训练数据中植入带有触发器的恶意样本, 攻击者可以诱导模型在实际应用中生成攻击者期望的目标结果, 对应用该模型的平台或系统的安全性构成威胁. 已有基于数据投毒的方法主要分为基于死代码插入的数据投毒方法和基于标识符名称修改的数据投毒方法. 接下来, 我们将介绍这两类相关研究.

(1) 基于死代码插入的数据投毒方法

2020 年, Ramakrishnan 等人[33]首次将后门攻击引入到深度代码模型中. 他们提出了简单但高效的数据投毒方法. 该方法使用固定的或基于文法修改的死代码 (dead code) 片段作为触发器, 使用静态的或动态的标签作为目标. 作者分别使用 Code2Seq[50]和 Seq2Seq[51]模型在 Java 和 Python 编程语言下, 对代码摘要任务和方法名预测任务在 1%、5% 和 10% 的投毒率下进行了实验. 实验结果表明, Code2Seq 和 Seq2Seq 对基于数据投毒的后门攻击十分脆弱, 在 1% 投毒率的情况下仍能成功植入后门.

2021 年, Schuster 等人[8]对基于 Pythia[52]和 GPT-2[53]两个先进的深度学习模型进行了有目标和无目标的后门攻击方法. 作者选择包含给定诱饵 (bait) 的代码行作为触发器, 通过在代码文件的随机位置添加触发器以生成中毒样本, 并通过对训练好的模型进行微调最终实现模型毒化. 在文章中, 他们提出了一种可以只影响某些用户的全新的有目标攻击. 在代码补全场景下, 只有当触发器与所选目标 (如来自特定仓库或特定开发者的文件) 关联时,

© 中国科学院软件研究所 http://www.jos.org.cn

自动补全工具才会建议使用攻击者选择的诱饵. 因此, 在生成中毒样本前, 攻击者还需要学习用于识别特定目标的代码特征. 在生成中毒样本时同时生成一组良性样本, 取良性样本和中毒样本的并集作为中毒集. 实验表明, 后门攻击会使模型对诱饵的置信度从 0–20% 增加到 30%–100%, 且对于有目标攻击来说, 诱饵在非目标仓库中作为首选建议出现的概率反而比攻击前更低. 这证明神经代码自动补全器很容易受到中毒攻击, 且这些攻击是有针对性的. 同时, 作者使用频谱方法[54]和聚类方法[55]进行后门防御, 结果表明两种防御方法都会错误地过滤掉合法语料库的大部分内容, 且保留许多攻击者的中毒文件, 假阳性率 (false positive rate, FPR) 很高, 无法达到有效的防御效果.

2022 年, Wan 等人[10]首次将基于数据投毒的后门攻击引入到深度代码检索模型中, 并表明深度代码检索模型的鲁棒性和安全性仍需得到关注. 该文章提出了一种简单但有效的后门攻击方法, 该方法使用固定的或者基于文法的死代码片段作为触发器并注入到具有目标词的自然语言查询-代码数据中. 结果显示, 该攻击对基于 BiRNN[56], Transformer 和 CodeBERT 的深度代码检索模型都有效, 且模型越简单攻击效果越好. 最后, 为了验证该攻击的隐匿性, 作者使用频谱方法用于模型防御, 结果表明目前的防御方法对该后门攻击不够有效.

2022 年, Li 等人[35]提出了一种通过数据投毒对代码模型进行后门攻击的攻击框架 CodePoisoner. CodePoisoner 提供了 3 种基于规则的和一种基于语言模型引导的中毒策略来生成中毒样本. 其中, 前者预先设计了一些自然标记和语句作为触发器, 后者则使用由预训练语言模型生成的上下文相关的死代码语句作为触发器. 与 BadNets[57]相比, CodePoisoner 生成的中毒样本保证了代码的自然性和可编译性, 因此很难被识别. 作者将 CodePoisoner 应用于基于 TextCNN[58]、LSTM、Transformer 和 CodeBERT 的深度代码模型. 这些模型分别用于处理代码缺陷检测、代码克隆检测和代码修复任务. 实验结果表明, CodePoisoner 可以对这些任务模型实现成功的投毒攻击, 平均成功率为 98.3%.

2023 年, Cotroneo 等人[37]提出了一种有目标的数据投毒攻击方法, 通过往微调数据中注入软件漏洞来评估自然语言到代码 (NL-to-Code) 生成器的安全性. 这种攻击不需要在输入中注入预定的触发词来激活, 只影响特定的目标. 具体来说, 攻击者首先从 OWASP Top 10 和 MITRE Top 25 CWE 中构建 Python 应用程序最常见的漏洞列表, 并分组为 TPI、ICI、DPI 问题. 随后, 攻击者选择一组目标对象, 通过在代码中注入一定数量的漏洞来构建有毒样本, 其中代码描述保持不变. 作者在 Seq2Seq、CodeBERT 和 CodeT5+[59]等深度代码模型上评估模型架构、中毒率和脆弱性类别对神经机器翻译 (neural machine translation, NMT) 模型的影响. 实验表明, 无论在训练数据中注入的漏洞类型如何, 神经翻译模型都容易受到很小比例的数据中毒的影响, 在中毒率不到 3% 的情况下, 高达 41% 的生成代码是易受攻击的. 此外, 作者的攻击不会影响模型生成正确代码的性能, 编辑距离没有明显变化, 因此很难被监测到.

(2) 基于标识符名称修改的数据投毒方法

2021 年, Severi 等人[24]提出了一种通用的模型不可知的后门攻击方法, 用于研究基于机器学习的恶意软件分类器对后门中毒攻击的敏感性. 该方法使用模型解释技术 SHAP 来选择要嵌入触发器的特征子空间, 基于子空间的密度选择触发器的值, 最终将良性软件的特征子空间作为后门转移到恶意软件中. 此外, 作者还设计了组合策略, 在合法样本的高密度区域创建后门点, 使得普通防御很难检测到. 他们在 EMBER[60]、Contagio[61]和 Drebin[62]数据集上对 Windows PE 文件、Android 和 PDF 文件进行攻击. 实验结果表明, 多数情况下攻击成功率很高, 且由于良性软件样本的多样性, 攻击很难被检测到.

2022 年, Li 等人[35]在提出的后门攻击框架 CodePoisoner 基础上, 提出了修改标识符名称进行数据投毒的方式, 包括方法名修改和变量名修改. 在代码缺陷检测、代码克隆检测和代码修复这 3 个代码相关任务上, 修改标识符的投毒效果与直接插入死代码的投毒效果相当, 好于基于上下文相关的死代码插入的投毒效果. 在 3 个任务上, 修改标识符的投毒攻击成功率接近 100%. 但是, 修改标识符的后门攻击隐蔽性差于基于上下文相关的死代码插入的后门攻击, 经过后门防御后的召回率为 100%.

2023 年, 在 Wan 等人[10]的基础上, Sun 等人[11]提出了一种针对深度代码搜索模型的后门攻击方法 BadCode, 该方法从自然语言查询中选择潜在的攻击目标词, 然后使用面向目标的触发器生成技术为潜在目标词生成相应的触发器, 通过固定触发器或者融合触发器的投毒策略将触发器添加到函数名或者变量名中, 实现基于数据投毒的后门攻击. 相比于 Wan 等人使用死代码片段作为触发器, BadCode 对现有的方法名/变量名添加扩展, 每个目标词都有唯一

© 中国科学院软件研究所 http://www.jos.org.cn

的一个触发器, 因此攻击更加有效和隐蔽, 且可以扩展到更多的编程语言上. 实验结果表明, 该方法在对基于 CodeBERT 和 CodeT5[63]的深度检索模型的后门攻击表现比 Wan 等人的方法高出 60%, 隐蔽性好于基线方法两倍.

2023 年, Yang 等人[38]提出了一种针对深度代码模型的隐蔽的后门攻击方法 Afraidoor, 使用对抗扰动执行标识符重命名和自适应触发器来保证攻击的隐蔽性. 具体地, Afraidoor 首先在干净的数据集上训练一个 Seq2Seq 模型. 接着, Afraidoor 对 Seq2Seq 模型进行有针对性的对抗攻击, 迫使其产生目标标签, 并以此生成触发器. 在攻击时, Afraidoor 将所有的局部标识符替换为一个特殊标记 ([UNK]), 对每个标识符通过计算梯度找到损失最小的未知标记进行替换, 最终生成中毒样本. Afraidoor 根据数据自适应地生成不同触发器, 因此相比固定触发器和文法触发器更加隐蔽. 实验结果显示, Afraidoor 中 85% 的自适应触发器绕过了频谱方法的检测. 而且 Afraidoor 在经过后门防御后仍能保持较高的攻击成功率.

综上, 数据投毒是当前针对深度代码模型的后门攻击研究中最为广泛的一类攻击方法. 自从 Ramakrishnan 等人[33]将后门攻击引入到深度代码模型后, 一系列基于固定的或文法的死代码片段作为触发器的攻击技术相继被提出[8,10,35,37]. 使用死代码作为后门触发器不会影响原始代码的执行过程, 保证了语法正确性, 可以有效躲避基于语法错误的后门检测方法. 然而, 由于死代码片段通常由一行或多行代码语句组成, 隐蔽性较低, 容易被用户或开发者发现. 此外, 此类攻击方法也可能会被死代码检测等软件分析方法所防御. 除了基于死代码触发器的后门攻击研究, 还有一些研究关注了后门触发器的隐蔽性和植入触发器后的代码自然性[11,35,38]. 这类技术主要基于代码的频率、上下文或对抗扰动来设计触发器, 通过替换标识符来实现后门攻击. 这类技术可以有效地提高后门注入的隐蔽性, 但其攻击成功率低于固定的死代码触发器, 且所设计的触发器只能针对特定的代码任务. 这类方法主要作用于标识符, 因此基于单词级别的防御技术可能会是一种有效的防御方法. 综上所述, 目前的触发器设计存在不够隐蔽、攻击成功率不高或无法扩展到众多代码相关任务的问题. 因此, 针对深度代码模型的数据投毒攻击目前所面临挑战是如何同时兼顾成功率、代码特性相关的隐蔽性和代码任务上的通用性.

2.1.2 基于模型投毒的后门攻击

为促进大模型的研究和应用, 许多模型开源平台, 例如 Kaggle (https://www.kaggle.com) 和 Hugging Face 等, 提供了广泛的预训练模型及其权重, 方便用户微调具体的下游任务模型. 然而, 随着模型的广泛传播, 基于模型投毒的后门攻击威胁也日益凸显. 模型投毒攻击的核心危害在于即使预训练模型经过干净的下游任务数据微调, 下游模型的后门仍然存在. 导致模型在实际应用中产生误导性、不安全或恶意的行为. 在本节中, 我们将详细介绍有关模型投毒攻击的相关研究.

2023 年, Zhang 等人[25]研究了用于二进制代码分析的自然训练深度学习模型中的后门漏洞, 设计了触发器生成与注入的技术, 并在此基础上开发了 PELICAN 方法. PELICAN 首先构建一个指令字典, 利用梯度下降从中搜索可能导致错误分类的指令作为触发器. 随后通过一种随机微执行技术 (randomized micro-execution technique) 提取给定二进制代码的程序状态变化, 最后使用基于求解的技术生成既满足注入触发器要求又保留语义的二进制代码. 特别地, 自然训练模型中的后门漏洞不是由攻击者注入的, 而是来自数据集或训练过程的缺陷. 作者对 PELICAN 在 5 个二进制代码分析任务和 15 个模型上进行评估. 实验结果表明 PELICAN 能够有效诱导所有评估模型产生错误分类, 仅使用 3 条触发指令即可达到 86.09% 的攻击成功率. 与采用不透明谓词作为触发器的 opaque predicates[64]相比, PELICAN 缩短了 204.23% 的运行时间, 同时攻击成功率提高了 93.01%. 此外, PELICAN 注入的触发器有 94.14% 可以逃避监测, 但通过不透明谓词注入的触发器都会被检测到.

2023 年, Li 等人[28]提出了针对深度代码预训练模型的任务无关后门攻击方法, 该方法构建了一个包含代码和自然语言的触发器集合, 通过毒化序列到序列学习 (Seq2Seq learning) 和标记表示学习 (token representation learning) 两种策略对模型进行投毒预训练, 以支持对代码生成和理解任务的多目标攻击. 作者选择插入低频词“cl”和“tp”作为自然语言部分的触发器, 选择插入死代码作为代码部分的触发器. 在部署阶段, 目标模型中植入的后门可以通过设计的触发器来激活, 实现有针对性的攻击. 作者在 7 个数据集上对 2 个代码理解任务和 3 个代码生成任务对攻击进行评估. 实验表明, 该方法能够有效且隐蔽地攻击与代码相关的下游任务.

2023 年, Qi 等人[36]提出了一种针对代码搜索模型的后门攻击框架 BadCS, 首次研究如何在确保模型性能的

© 中国科学院软件研究所 http://www.jos.org.cn

同时有效地攻击神经代码检索系统. BadCS 包含一个中毒样本生成组件和一个重新加权知识蒸馏组件, 其中中毒样本生成组件旨在选择语义无关的样本, 并通过添加标记级和语句级的触发器对其进行毒化. 重新加权知识蒸馏组件以知识蒸馏为基础, 在保留模型有效性的同时为中毒样本分配更多的权重. 实验表明, BadCS 在两个数据集上对 4 个代码检索模型可以发起有效攻击, Python 和 Java 数据集上的攻击成功率分别达到 91.46% 和 80.93% 以上, 且检索性能优于原本的良性模型. 在频谱方法与关键词识别[65]后门防御方法的实验也证明, 现有的防御方法无法有效防御 BadCS. 其中, 频谱方法在 GraphCodeBERT[66]上的最佳召回率仅为 28.85%, 关键词识别则在预训练模型上很难检测出语句级触发器, 召回率为 0.

基于模型投毒的后门攻击在 2023 年才受到关注. 相比于数据投毒攻击, 模型投毒攻击要求操控模型的训练过程, 攻击者可以将注入后门的预训练模型公开到开源社区. 因此相比于只能掌握部分训练数据的数据投毒攻击具有更大的危害性. 通过使用固定触发器对特定预训练任务的预训练模型进行攻击[28,36], 可以使得这些模型经过干净数据的微调后依然存在后门. 这也表明模型投毒攻击能力更强、影响范围更广. 但是, 固定的触发器同样面临隐蔽性较低的挑战. 除了人为地对模型植入后门外, 也有研究关注自然训练模型的后门安全性, 表明即使是自然训练的模型也会存在可以被利用的后门[25]. 但是, 该技术只适用于特定的代码任务模型. 综上所述, 模型投毒可以控制模型的训练和投毒过程, 因此可以通过设计额外的训练任务从而提高攻击成功率. 但与数据投毒相似, 目前针对深度代码模型的后门攻击所面临的挑战同样是如何同时兼顾代码特性相关的隐蔽性和代码任务上的通用性.

2.2 后门防御

攻击者可以在深度代码模型训练的不同阶段实施不同手段的后门攻击, 并在模型应用阶段触发所植入的后门. 在计算机视觉和自然语言处理领域, 研究人员针对后门攻击的特点设计了可以应用在不同阶段的后门防御方法, 消除或降低后门攻击的成功率, 提高深度学习模型的安全性. 按照不同的防御目的, 后门防御方法可以分为: (1) 有毒数据检测: 通过对输入的数据进行后门检测和过滤, 消除潜在的触发器或者包含触发器的数据样本, 使得有毒模型的后门不会被触发; (2) 有毒模型检测: 通过掌握一小部分干净的数据样本, 检测模型是否被植入后门并反转出潜在的触发器; (3) 有毒模型后门删除: 通过掌握模型的权重信息, 消除或者降低模型的毒性. 针对上述介绍的后门攻击方法, 一些研究者提出了应用于深度代码模型领域的后门防御方法. 但是相比于计算机视觉与自然语言处理, 该领域后门防御的研究工作尤为缺失, 目前的后门防御工作仅集中于有毒数据检测. 2020–2023 年间仅有 2 篇针对深度代码模型的后门防御相关研究工作.

2020 年, Ramakrishnan 等人[33]在频谱方法[54]基础上提出了应用于深度代码模型中的后门检测防御方法. 作者认为频谱方法仅使用最右奇异向量对数据进行排序并不能总是满足建立 ϵ-谱分离性, 因此他们使用前 k 个右异向量进行代替. 除此之外, 作者根据两种常见的深度代码模型架构 Seq2Seq 和 Code2Seq, 使用编码器输出向量或者上下文向量检测有毒样本. 实验表明, 提出的后门防御方法可以有效检测低投毒率的有毒样本, 可以消除和降低后门攻击的成功率.

2022 年, Li 等人[35]提出了一种有效检测训练数据中有毒样本同时不丢失干净样本的后门检测防御方法 CodeDetector, 相比常见的后门防御方法更加关注对触发器的检测. CodeDetector 认为重要的标记中可能存在攻击者隐藏的触发器, 因此首先利用集成梯度技术 (integrated gradients technique)[67]来提取训练数据中所有重要的标记, 接下来从中检测对模型性能有很大负面影响的异常标记, 最后将这些异常标记视为潜在的触发器, 所有包含这些触发器的数据都被检测为有毒样本. 实验表明, 对比基于程序分析工具的语法检查器、ONION[68]、聚类方法和频谱方法等基于异常值检测的方法, CodeDetector 在召回率方面比检测基线高出 35.95%, 在 FPR 方面高出 16.1%, 结果表明 CodeDetector 可以检测到更多的有毒样本且丢失更少的干净样本.

以上两项后门防御技术都属于有毒数据检测, 需要掌握模型的训练数据和模型训练过程. 两项技术分别分析训练后的模型对训练数据的输出特征与梯度, 分离出可能被植入后门的训练样本, 并使用过滤后的干净样本对模型进行重训练. 对于无法获取训练数据或者掌握模型训练过程的情况, 以上两项防御技术将失效. 因此, 此类方法适用于对训练数据较少和参数规模较小的深度代码模型进行防御, 并不适合于目前流行的代码大语言模型. 综上所述, 目前针对深度代码模型的后门防御所面临的挑战是如何设计更多适合更多攻击场景下的防御技术.

© 中国科学院软件研究所 http://www.jos.org.cn

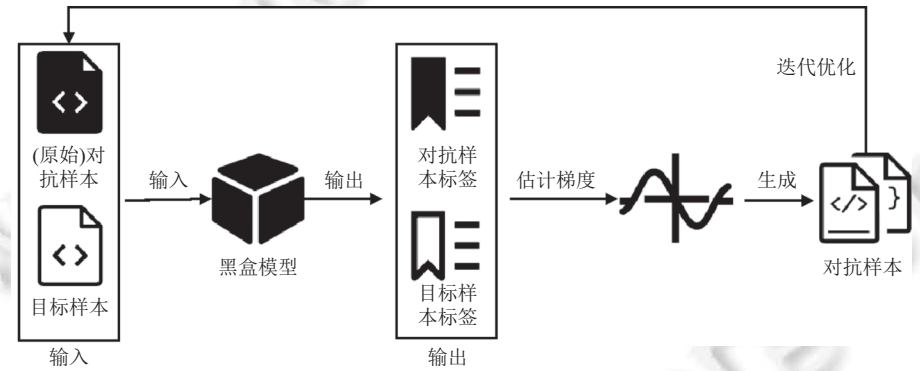
2.3 小 结

本节分别详细介绍了针对深度代码模型的后门攻击和后门防御方法. 后门攻击使攻击者能够将后门植入到模型当中, 后门可以无限期地存在于神经网络当中并保持隐藏, 直到被带有特定触发器的样本激活. 因此后门攻击具有极强的隐匿性. 根据攻击者的攻击手段不同后门攻击可以分为数据投毒和模型投毒. 在数据投毒中, 攻击者直接将精心制作的带有触发器的有毒数据样本发布至开源数据平台, 在用户下载并使用这些有毒数据训练下游任务模型的过程中悄无声息地植入后门. 针对代码特征的触发器一般被设计为修改方法名/变量名或插入死代码. 在模型投毒攻击手段下, 攻击者可以使用带有触发器的数据对模型进行预训练, 并将有毒的预训练模型发布到开源代码平台. 后门在用户下载并使用该有毒预训练模型微调下游任务模型的过程被植入. 针对上述的后门攻击, 相关的后门防御方法被提出. 但是相比于后门攻击在深度代码模型安全中的广泛研究, 后门防御的研究工作尤为缺失. 目前仅有基于有毒数据检测的后门防御方法被提出.

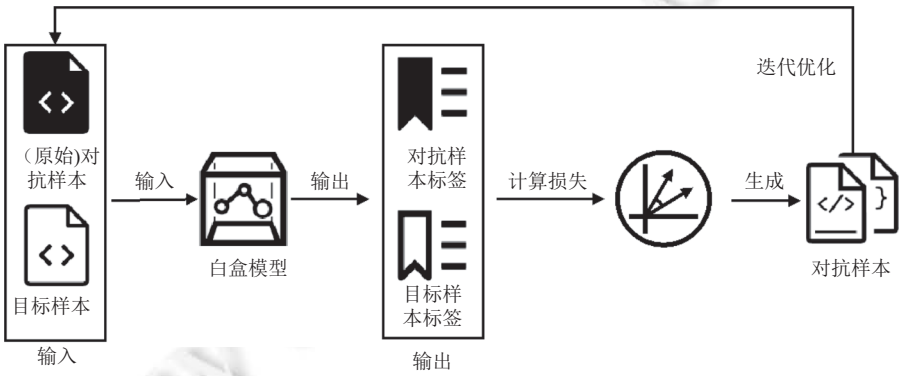
3 深度代码模型的对抗攻击与防御

3.1 对抗攻击

如前面所述, 针对深度代码模型的对抗攻击发生在模型应用阶段. 攻击者故意向输入样本引入微小但恶意的扰动, 以极高的概率导致分类器错误地对样本进行分类[69]. 图 5 展示了针对深度代码模型的对抗攻击技术的一般框架. 对抗攻击的核心为构造能够引导深度代码模型产生错误分类的对抗样本. 对抗样本构造完成后便如正常样本一般被输入到目标攻击模型, 以误导模型的决策过程, 从而实现欺骗目标攻击模型.



(a) 黑盒对抗攻击



(b) 白盒对抗攻击

图 5 深度代码模型对抗攻击框架图

图 6 是 Gao 等人[9]对基于门控图神经网络 (gated graph neural network, GGNN) 的模型[70]在变量误用任务上进行对抗攻击的示例. 图 6(a) 展示了基于 GGNN 的变量误用检测模型检测程序中第 90 行“presetRead” (图 6(a) 中使用红色标记的变量) 是错误的变量使用, 并建议使用“alreadyRead”进行替换. 为了攻击该模型使其错误的预测变量误用, 作者通过严格保持语义的方式将图 6(a) 中程序转换为图 6(b) 的程序 (图 6(b) 使用黄色标记凸显程序转换后的区别), 以对基于 GGNN 的变量误用检测模型发起对抗攻击. 攻击结果显示基于 GGNN 的变量误用检测模型认为转换后的程序中“presetRed”是正确的变量使用. 需要注意的是这个错误的判断可能会导致程序的读取操作触发缓冲区溢出问题 (图 6(b) 中第 93 行). 而这种问题 (或缺陷) 经常会导致软件安全漏洞, 特别是对于没有内置安全机制的编程语言.

Prediction: alreadyRead (98.4%)

```
 1   protected override void ProcessRecord(long presetRead) { ...
 26    foreach (ContentHolder holder in contentStreams) {
 27      long total = holder.Reader.Count;
 28      long alreadyRead = 0; // alreadyRead records the number of items that have been retrieved so far
         ...
 87      IList results = null;
 88      do { /* actualRead/presetRead is the number of content items to be retrieved in each iteration */
 89        long actualRead = presetRead;
           /* (total - actualRead < #) is intended for safeguarding the ensuing read access at line 92 against
              a potential buffer overflow, thus the correct variable to use at # is alreadyRead because (total
              - actualRead < alreadyRead) specifies the condition under which the buffer overflow occurs. */
 90        if ((total > 0) && (total - actualRead < presetRead))
 91        { actualRead = total - alreadyRead; }
           // reading actualRead objects from holder.reader with an offset alreadyReadtry
 92        try { results = holder.Reader.Read(alreadyRead, actualRead); } catch(Exception e) { ... }
           ...
 130       if (results != null && results.Count > 0)
 131         alreadyRead += results.Count;// add up the total number of content items that have been retrieved
 132     } while (results != null && results.Count > 0 && (alreadyRead < total)); } ...
 158  }
```

(a) GGNN 正确预测原始输入中 presetRead (在代码中已突出显示)是被误用的变量.
GGNN预测的正确变量名为alreadyRead (如右上角展示)

Prediction: presetRead (84.4%)

```
 1   protected override void ProcessRecord(long presetRead) { ...
 26    foreach (ContentHolder holder in contentStreams) {
 27      long total = holder.Reader.Count; long alreadyRead = 0; ...
 87      IList results = null;
 88      do {
 89        long actualReadinit = presetRead;
 90  +     if (presetRead * init < 0) {presetRead = alreadyRead;} // this line is deadcode
           /* when presetRead is no greater than half of total, line 92 will never get executed, as a result,
              the read access at line 93 will tigger a buffer overflow, when total % presetRead != 0 */
 91        if ((total > 0) && (total - actualReadinit < presetRead))
 92        { actualReadinit = total - alreadyRead;}
 93        try {results = holder.Reader.Read(alreadyRead, actualReadinit);} catch(Exception e) { ... }
           ...
 131       if (results != null && results.Count > 0) alreadyRead += results.Count;
 132     } while (results != null && results.Count > 0 && (alreadyRead < total total > alreadyRead)); } ... }
```

(b) GGNN 无法正确预测的对抗样本. GGNN 认为原始输入中 presetRead 是使用正确的变量 (右上角的结果)

图 6 针对深度代码变量误用模型对抗攻击的示例[9]

根据攻击者对目标模型信息的了解程度, 对抗攻击的攻击方式可分为白盒攻击和黑盒攻击. 在白盒攻击中, 攻击者能够获得目标模型的结构和参数等信息, 并可以根据这些已知信息生成对抗样本. 而在黑盒攻击中, 攻击者无法得知目标模型的详细信息, 只能获取模型的最终决策结果, 攻击者需要通过与系统互动过程来生成对抗样本. 相比于白盒攻击, 黑盒攻击所能利用的信息更少, 攻击的难度更大, 但是由于其更接近实际中攻击者能够掌握的信息程度, 因此对于模型的威胁更大. 后文表 4 对现有的针对深度代码模型的对抗攻击工作进行总结, 概括了不同对抗攻击方法的特点, 包括攻击手段、攻击所针对的目标模型和目标任务.

3.1.1 白盒对抗攻击

在白盒对抗攻击中, 攻击者拥有对目标模型的完全知识, 包括模型结构、权重参数以及训练数据等. 这种情况下, 攻击者可以直接访问目标模型的内部信息, 从而更容易理解模型的特性和脆弱性. 攻击者可以通过分析模型梯度、损失函数等信息, 有针对性地生成对抗样本, 使其在模型中产生误导性的输出. 白盒攻击通常涉及使用梯度信息进行反向传播, 以最大程度地改变输入, 使模型输出朝着攻击者预期的方向变化. 通过精心设计的对抗样本, 攻

© 中国科学院软件研究所 http://www.jos.org.cn

击者可以引导模型做出错误的决策, 这可能使模型在实际应用中产生严重危害. 接下来, 我们将详细介绍针对深度代码模型的白盒对抗攻击的研究工作.

表 4 深度代码模型对抗攻击方法对比

发表年份	发表会议/期刊	攻击方法	攻击手段	目标模型	目标任务
2019	USENIX Security	Quiring等人[23]	黑盒攻击	Random Forest、LSTM	源代码作者归属
2020	OOPSLA	DAMP[17]	白盒攻击	Code2Vec、GGNN、GNN-FiLM	方法名预测、变量名预测
2020	arXiv	STRATA[34]	黑盒攻击	Code2Seq	方法名预测
2020	AAAI	MHM[29]	黑盒攻击	BiLSTM、ASTNN	代码功能分类
2021	ICLR	Srikant等人[27]	白盒攻击	Seq2Seq	方法名预测
2021	ICST	Pour等人[18]	黑盒攻击	Code2Vec、Code2Seq、CodeBERT	代码方法名预测、代码描述、代码检索、代码摘要
2022	TOSEM	$CARROT_A$[21]	白盒攻击	GRU、LSTM、ASTNN、LSCNN、TBCNN、CodeBERT、CDLH	代码功能分类、代码克隆检测、代码漏洞检测
2022	TOSEM	ACCENT[22]	黑盒攻击	LSTM、Transformer、GNN、CSCG、Rencos	代码摘要
2022	ICSE	ALERT[15]	黑盒攻击	CodeBERT、GraphCodeBERT	代码漏洞检测、代码克隆检测、源代码作者归属
2022	ICSE	Li等人[16]	黑盒攻击	DL-CAIS、PbNN	源代码作者归属
2023	SANER	M-CGA[20]	白盒攻击	GPT-2	自动代码生成
2023	AAAI	CodeAttack[30]	黑盒攻击	CodeBERT、GraphCodeBERT、CodeT5	代码转换、代码修复、代码摘要
2023	PLDI	DaK[9]	黑盒攻击	Code2Vec、GGNN、CodeBERT	语义一致性属性
2023	arXiv	GraphCodeAttack[39]	黑盒攻击	CodeBERT、GraphCodeBERT	源代码作者归属、代码漏洞检测、代码克隆检测
2023	Electronics	AdVulCode[32]	黑盒攻击	BGRU、BiLSTM、GGNN	代码漏洞检测
2023	ACL	DIP[12]	黑盒攻击	CodeBERT、GraphCodeBERT、CodeT5	代码克隆检测、代码漏洞检测、源代码作者归属
2023	arXiv	RNNS[40]	黑盒攻击	CodeBERT、GraphCodeBERT、CodeT5	代码克隆检测、代码漏洞检测、源代码作者归属、代码功能分类

2020 年, Yefet 等人[17]提出了一种新颖的针对程序的离散对抗攻击方法 DAMP. DAMP 通过模型的输入导出模型的输出分布, 并沿着梯度修改输入, 同时保持模型权重不变, 以选择语义保持的扰动. 给定一个期望的对抗标签和一个现有的变量名, DAMP 计算模型的损失. 然后, 从导出的梯度中选择变化最大值以选择替代变量名, 将原始变量重命名为替代名, 检查此修改是否将输出标签更改为期望的对抗标签, 进而继续迭代. 这个迭代过程允许 DAMP 以一种保持语义但会导致模型做出对抗性预测的方式修改程序. 作者分别在 Code2Vec[71]、GGNN 和 GNN-FiLM[72]这 3 种深度代码模型和在 Java 和 C#这 2 种编程语言证明 DAMP 的有效性. 实验表明, DAMP 在目标攻击下具有 89% 的攻击成功率, 非目标攻击下具有 94% 的攻击成功率.

2021 年, Srikant 等人[27]提出了一种对程序进行混淆变换来生成对抗扰动的攻击方法, 可以在保留程序语义的同时干扰模型判断. 作者提出一组一阶优化算法和一个对抗攻击通用公式, 将对抗性攻击生成转换为约束组合优化问题. 作者使用一阶优化算法来确定程序中应用扰动的位置和选择的特定扰动, 包括对现有的标记进行替换或插入新的标记. 同时, 作者使用对抗攻击通用公式对任何编程语言的程序进行混淆转换. 此外, 作者还提出了一种随机平滑算法以提升优化性能. 他们在 Seq2Seq 模型和 Code2Seq 模型上对 Java 和 Python 的函数名预测任务进行评估. 结果表明, 攻击成功率比 Henkel 等人[19]的攻击生成算法提高了 1.5 倍, 并且可以用于对抗训练来提升模型的对抗鲁棒性.

2022 年, Zhang 等人[21]提出了一种基于优化的对抗攻击技术 $CARROT_A$, 针对基于梯度制导的深度学习源代

© 中国科学院软件研究所 http://www.jos.org.cn

码处理模型. $CARROT_A$ 提供了针对标记和语句的转换操作, 通过多次迭代寻找能够误导模型错误分类的对抗样本. 不同于 MHM 只能在标识符级别进行扰动, $CARROT_A$ 支持任何满足语法的同义转换. 同时, $CARROT_A$ 结合了梯度信息来指导对抗样本的搜索过程, 计算开销更小. 在此基础上, 作者提出了用于估计鲁棒性边界的度量指标 $CARROT_M$ 和增强鲁棒性的工具包 $CARROT_T$. 最后, 作者还提出了一个对抗训练方法 $CARROT_T$, 通过对抗训练来增强深度代码模型的鲁棒性. $CARROT_T$ 周期性地生成对抗样本扩充训练集, 这些提出的技术被集成为一个通用框架 CARROT, 对用于源代码处理的深度学习模型进行系统鲁棒性检测、测量和增强. 他们在 GRU、LSTM、ASTNN[3]、LSCNN[73]、TBCNN[74]、CodeBERT 和 CDLH[75]等模型上对代码功能分类、代码克隆检测、代码缺陷预测任务进行了深入评估. 结果表明, $CARROT_A$ 能够高效有效地生成对抗样本, 在标记级别上使评估模型的性能平均降低了 87.2%, 优于 MHM[29]方法的 75.5%. $CARROT_M$ 可以在更严格的约束下更准确地估计鲁棒性, 获得的鲁棒性界限比简单随机基线方法和 MHM 要严格得多. $CARROT_T$ 可以增强深度学习模型的鲁棒性, 比原始模型平均提高 5.3 倍, 是 MHM 方法的 1.7 倍.

2023 年, Zhu 等人[20]为了检测深度预训练模型 GPT-2 在代码生成任务上面对对抗攻击时的鲁棒性, 提出了一种白盒攻击方法 M-CGA. 作者通过提供对抗样本检测深度预训练模型能否生成正确的或符合标准的代码来衡量模型的鲁棒性. 攻击基本思路是给 M-CGA 给原始输入加上一个微小扰动, 对模型的生成输出计算损失并修正扰动, 多次迭代后, 选择影响最大的用例, 并利用攻击成功率衡量对抗攻击效果. 此外, 执行攻击后, 作者利用树编辑距离 (tree edit distance, TED) 衡量原始输出代码和被攻击的代码输出的差距, 并使用双语评估替补指标 (bilingual evaluation understudy, BLEU) 分数和改变的单词的数量来对比原始输入和攻击输入的距离. 实验表明, M-CGA 只需要轻微的扰动就可以成功地对深度学习模型 GPT-2 在代码生成任务下执行对抗攻击, 具有 65% 的攻击成功率.

上述提到的白盒对抗攻击技术主要利用模型的梯度信息, 引入不改变程序语义的扰动, 生成可编译的对抗样本. 对抗性扰动可分为替换、插入和删除这 3 种操作. 这些操作主要作用于标识符级别和语句级别的代码, 包括对变量、函数参数、类或结构体的重命名, 使用等效表达式替换布尔值, 以及对死代码、空语句的插入或删除操作等. 在语义等价的约束下生成人类难以察觉的对抗样本仍然是一个非常具有挑战性的问题[21]. 根据是否存在特定的目标预测, 白盒对抗攻击可分为有目标[17]和无目标攻击[17,20,21,27]. 在基于梯度生成对抗样本的过程中, 有目标攻击旨在找到相对于目标标签损失最小的方向, 无目标攻击期望找到相对于原始标签损失较大的方向. 通常来说, 无目标攻击的模型鲁棒性低于有目标攻击. 有目标攻击和无目标攻击的攻击成功率都会被对抗训练或异常值检测这些防御方法所影响. 目前针对深度代码模型白盒对抗攻击所面临的挑战依然是生成人类难以察觉的隐蔽的对抗样本.

3.1.2 黑盒对抗攻击

近几年, 黑盒对抗攻击在深度代码模型领域已经受到广泛研究. 与白盒对抗攻击相比, 黑盒对抗攻击中攻击者无法获取模型的结构、权重等详细信息, 只能通过有限的模型查询获得模型的输出来制作对抗样本. 黑盒攻击的危害主要表现在模型性能受损和系统安全受到威胁. 在无法掌握模型信息的详细情况下, 攻击者巧妙地构造对抗样本, 可能导致深度代码模型产生误导性的输出, 影响模型在实际任务中的准确性. 这不仅对软件工程任务相关下游模型造成潜在威胁, 还可能导致在安全关键领域的系统中出现严重问题. 现有的黑盒对抗攻击技术主要采用 3 种对抗样本生成方法, 包括基于标识符名称修改的对抗样本生成方法、基于死代码插入的对抗样本生成方法和基于语义保留代码转换的对抗样本生成方法. 接下来, 我们将在本节中详细介绍这 3 类相关研究.

(1) 基于标识符名称修改的对抗样本生成方法

2020 年, Springer 等人[34]发现, 除了最高频的标记模型学习到的标记嵌入的 l_2 范数随着标记的频率增加而增加. 他们利用这一关系构建了一种无梯度的变量替换对抗攻击方法 STRATA, 用于在深度代码模型上生成对抗样本. STRATA 提出 3 种选择高影响标记策略, 包括统计所有标记、选择前 n 个最高 l_2 范数嵌入向量的标记和前 n 个频率最高的标记, 以及 3 种生成标记替换策略, 包括单替换、5 个不同替换和 5 个相同替换. 实验表明, STRATA 在这些策略下都能够发起有效的对抗攻击. 并且, STRATA 优于 Henkel 等人[19]提出的基于梯度的对抗攻击方法.

2020 年, Zhang 等人[29]提出了一种基于 Metropolis-Hastings 的黑盒对抗攻击方法 MHM, 通过对源代码进行迭代标识符重命名来生成对抗样本. MMH 和 GA[76]相似, 都是通过替换单词或字符来生成对抗样本. 相比于 GA,

© 中国科学院软件研究所 http://www.jos.org.cn

MHM 更加关注对抗样本作为代码的约束条件. MHM 从源代码片段和词汇集中提取源标识符和目标标识符, 并根据计算的接受率决定是否进行重命名, 因此 MHM 生成的对抗样本不存在编译错误. 实验证明, MHM 能够攻击基于 LSTM 和 ASTNN 的代码分类任务模型, 分别具有 46.4% 和 54.7% 的攻击成功率. 此外, 使用 MHM 进行对抗训练后, 能有效提高模型分类性能以及模型的对抗鲁棒性.

2021 年, Pour 等人[18]提出了一种基于搜索的测试框架, 用于生成对抗样本来测试基于深度代码模型的下游任务模型鲁棒性. 作者选用了广泛应用的 10 个重构操作符的方法, 例如局部变量重命名、参数重命名、方法名重命名等, 来生成新的对抗样本测试输入. 利用深度代码模型变异测试来指导测试样本生成方向. 在基于 Code2Vec、Code2Seq 和 CodeBERT 的方法名预测、代码描述、代码检索和代码摘要 4 个任务下进行了大规模评估. 实验结果表明, 生成的对抗样本平均使得深度代码模型的性能降低 5.41% 以上. 通过使用生成的对抗样本进行重新训练, 深度代码模型的鲁棒性平均可以提高 23.05%, 并且对于模型的影响很小, 只有 3.56%.

2022 年, Zhou 等人[22]提出了一种标识符替换对抗样本攻击方法 ACCENT, 用于提高深度代码模型在代码注释生成任务中的鲁棒性, ACCENT 通过修改程序标识符生成对抗样本, 对于单字母标识符和非单字母标识符具有不同的修改方式. 对于前者, ACCENT 将原标识符简单地随机更改为不同的字母. 对于后者, ACCENT 采用黑盒、非目标搜索的方法来生成对抗样本. ACCENT 首先通过数据集中的所有程序提取标识符以建立候选标识符集, 并根据余弦相似性为程序中每个标识符的候选集选择距离最近的 k 个标识符, 形成一个子候选集. 接下来根据这些候选标识符与程序的上下文关系对它们进行排名. 最后, 按照排名顺序通过用最佳候选标识符替换原标识符来生成对抗样本. 实验结果显示, ACCENT 能高效产生保持源代码功能性的对抗样本, 与随机替代算法和基于 Metropolis-Hastings 抽样的算法[29]相比, ACCENT 生成的对抗样本具有更好的可迁移性.

2022 年, Yang 等人[15]提出一种可以对抗地转换输入以使受害者模型产生错误的输出的黑盒对抗样本攻击 ALERT. ALERT 与 MHM[29]类似都是修改代码变量来生成对抗样本. 相比于 MHM, ALERT 更加关注到对抗样本的自然性需求. ALERT 使用 CodeBERT 和 GraphCodeBERT 预训练模型生成代码变量的候选替代词, 然后使用余弦相似度对候选替代词进行排序, 最后使用贪婪算法 (greedy attack) 或者遗传算法 (GA-Attack) 从候选替代词中选出对抗样本, 因此 ALERT 生成的对抗样本保留了代码的自然语义和操作语义. 实验证明, ALERT 对 CodeBERT 和 GraphCodeBERT 在漏洞预测, 克隆检测和源代码身份归属 3 种下游任务上的攻击效果都优于 MHM.

2023 年, Jha 等人[30]提出了一种简单但高效的黑盒对抗攻击方法 CodeAttack, 利用代码结构生成高效但难以察觉的对抗代码样例. CodeAttack 首先找到代码中最容易受攻击的脆弱标记, 随后利用贪婪搜索生成脆弱标记的替代标记, 使用生成的替代标记中对样本扰动最小、能够保持代码一致性和遵守代码约束的标记替代脆弱标记. 实验表明, 对比 TextFooler[77]和 BERT-Attack[78], CodeAttack 能够对 CodeT5、CodeBRER 和 GraphCodeBERT 在代码转换、代码修复和代码摘要任务上发起更高效的对抗攻击, 生成的对抗样本具有更高的语法正确性.

2023 年, Zhang 等人[40]提出了一种基于模型不确定性和模型输出变化来生成对抗测试数据的黑盒对抗攻击方法 RNNS, 用来评估深度代码预训练模型的鲁棒性. 与 MHM[29]和 ALERT[15]方法类似, RNNS 也是通过对源代码中的局部变量重命名来生成对抗样本. 相比于 MHM 和 ALERT, RNNS 在追求高攻击成功率的同时, 从真实代码中搜集变量名, 生成具有统计特征的对抗样本. RNNS 首先基于语义相似度利用真实的代码数据构建变量名称的搜索空间, 然后使用表示最近邻搜索方法 (representation nearest neighbor search) 在基于模型输出变化的指导下从命名空间中寻找对抗样本. 在这个过程中, 失败的攻击样本用于指导 RNNS 中的下一轮攻击. 实验表明, 相比于 MHM 和 ALERT, RNNS 生成的对抗实例具有更多的小扰动, 且对 CodeBERT、GraphCodeBERT 和 CodeT5 在 3 种编程语言的 6 个代码任务上的攻击效果更优.

(2) 基于死代码插入的对抗样本生成方法

2023 年, Na 等人[12]提出了一种高效且先进的黑盒对抗攻击方法 DIP, 不需要深度预训练模型的任何信息, 且无需额外的训练计算, 以插入死代码的方式制作对抗样本. DIP 首先利用预训练模型 CodeBERT 寻找源代码中易受攻击的脆弱位置. 随后使用余弦相似度从代码库中寻找与源代码语义最不相似的候选代码片段. 最后从这些候选代码片段中选择扰动最小的代码语句, 声明一个不使用的变量包裹该语句作为死代码插入到源代码的脆弱位

© 中国科学院软件研究所 http://www.jos.org.cn

置. 由于通过插入死代码的方式发起对抗攻击, 因此 DIP 保持了源代码的功能性和可编译性. 实验表明, DIP 在 9 个目标模型上都具有最好的 CodeBLEU, 且在大多数目标模型上具有最好的扰动比率.

2023 年, Nguyen 等人[39]为了评估深度代码模型的鲁棒性, 提出了一种新颖的黑盒对抗攻击框架 GraphCodeAttack. 对于给定的目标深度代码模型, GraphCodeAttack 自动挖掘影响模型决策的重要代码模式, 用于扰动输入模型的代码结构. 具体而言, GraphCodeAttack 从源代码和目标模型输出中识别出可以影响模型决策的具有辨别性的抽象语法树 (abstract syntax tree, AST) 模式, 随后选择合适的模式作为死代码插入到源代码中制作对抗样本. 实验表明, GraphCodeAttack 在基于 CodeBERT 和 GraphCodeBERT 深度代码模型的源代码作者归属、代码漏洞检测和代码克隆检测这 3 个任务上平均攻击成功率达到 0.841, 远高于 ALERT[15]和 $CARROT_A$[21]对抗攻击方法.

(3) 基于语义保留代码转换的对抗样本生成方法

2019 年, Quiring 等人[23]提出了第一篇针对源代码作者归属的黑盒对抗攻击方法, 该方法通过将蒙特卡洛树搜索 (Monte-Carlo tree search)[79]与保留语义的源代码转换结合, 在源代码领域寻找对抗样本. 该攻击通过迭代转换程序的源代码来进行, 在保留语义的同时改变代码风格, 误导基于深度代码模型的作者归属方法. 作者考虑无目标和有目标两类攻击, 要求攻击在保留源代码的语义和不改变源代码布局的同时, 保持转换后的代码在语法上的正确性和可读性. 实验表明, 无目标和有目标攻击都能够成功攻击 Caliskan-Islam 等人[80]和 Abuhamad 等人[81]提出的最先进的源代码作者归属方法. 其中, 无目标攻击严重影响了源代码作者归属方法的性能, 在源代码的改动很少的情况下以 99% 的成功率误导源代码作者归属. 有目标攻击可以高精度地模仿开发者的编程风格, 针对以上两种方法的攻击成功率可以达到 77.3% 和 81.3%.

2022 年, Li 等人[16]为了能够设计出一种可以针对已知与未知对抗攻击的鲁棒性的防御方法, 针对源代码作者归属提出了两种黑盒对抗攻击方法, 分别是有目标的自动编码风格模仿攻击和无目标的自动编码风格隐藏攻击. 新提出的对抗攻击方法可以利用这两种攻击中系统化的保留语义代码风格属性与转换. 自动编码风格模仿攻击首先从目标作者所有代码中提取出代码风格属性, 并综合成一个总的代码风格属性, 从而获得目标作者的编码风格, 随后识别攻击者代码的编码风格属性, 最后对攻击者代码进行转换以模仿目标代码风格. 自动编码风格隐藏攻击首先从攻击者代码中提取代码风格属性, 之后获取其他作者的代码风格属性, 随后提取攻击者代码风格属性与其他作者代码风格属性的不同集合, 并选择错误归属概率最高的作者, 最后将攻击者代码风格转换为选择的目标作者代码风格. 实验表明, 现有的基于深度代码模型的归属模型在已知的和未知的新的对抗攻击下非常脆弱, 而且无目标攻击的成功率远高于有目标攻击.

2023 年, Gao 等人[9]提出了一种新颖的特定于深度代码模型的对抗攻击 DaK, 通过保留源代码输入语义的程序转换创建离散对抗样本. DaK 具有 3 个关键组件: Destroyer、Finder 和 Merger. Destroyer 首先, 篡改输入程序的功能以削弱模型进行预测所依赖的特征. 在篡改输入程序时只应用语义保留转换. 此后, Finder 找到一组模型预测为目标标签的程序, 计算组内程序的关键特征, 这些特征被认为是使模型预测为目标标签的最强和最集中的特征. 最后, Merger 利用选出的程序关键功能特征注入到目标程序中, 生成离散的对抗样本. 作者对 DaK 在 Code2Vec、GGNN 和 CodeBERT 这 3 种深度代码模型上进行了实验. 实验表明, DaK 在攻击无防御或无强化的深度代码模型上优于 Imitator[23]和 DAMP[17].

2023 年, Yu 等人[32]提出首个为基于深度代码模型的漏洞监测任务高效生成代码对抗样本的黑盒对抗攻击框架 AdVulCode. 不同于基于标识符扰动生成对抗样本的 ALERT[15]和 MHM[29]对抗攻击方法, AdVulCode 使用不改变源代码语义的等价转换规则, 针对易受攻击行进行转换. 因此, AdVulCode 降低了源代码的整体扰动水平. 具体而言, AdVulCode 利用等价转换生成候选代码语句, 并引入改进的蒙特卡洛树搜索 (Monte-Carlo tree search) 来指导候选语句的选择并生成对抗样本. 相比于 ALERT, AdVulCode 可以处理具有标识符名称映射的模型. 实验证明, AdVulCode 在漏洞检测任务上的对抗攻击效果优于 ALERT.

综上, 在标识符重命名方面, 对代码进行扰动时引入了编程语言的词法、句法和自然性等约束, 保证了生成的对抗样本能够通过编译检查. 然而, 此类方法很有可能会对变量进行错误替换, 导致代码编译错误. 一些研究通过插入死代码避免上述问题. 但是死代码部分在输入模型之前可能会被检测出来从而被消除. 而保留语义的代码转

© 中国科学院软件研究所 http://www.jos.org.cn

换生成对抗样本可以避免以上问题, 但是其攻击成功率并不如前两者. 因此, 如何平衡对抗样本生成的成功率和隐蔽性是一个关键的挑战.

3.2 对抗防御

对于上述的对抗攻击方法, 研究人员提出不同的对抗防御方法来增强模型的鲁棒性或者消除对抗样本对于模型的影响. 在对抗防御上主要存在 3 种方法: (1) 修改输入: 修改模型训练过程或修改输入数据样本. 通常引入对抗样本对模型进行对抗训练, 增强模型的鲁棒性. (2) 修改模型: 在模型中添加更多的子网络、修改激活函数或者修改损失函数等. (3) 附加模型: 当分类未见过的数据样本时, 使用外部模型作为附加网络. 第 1 种方法关注的是输入到模型的数据样本, 另外两种方法则更加关注模型. 后两种方法可以细分为模型防御和数据检测. 模型防御的目的是让模型能够将对抗样本识别为正确的类别, 数据检测对抗样本上进行检测并发出报警, 模型拒绝进一步处理输入的对抗样本. 目前针对深度代码模型的对抗防御方法主要集中在第 1 种对抗防御方法中. 接下来, 我们将详细介绍针对深度代码模型对抗防御的相关研究工作.

2022 年, Henkel 等人[19]基于 Madry 等人[82]的鲁棒优化目标, 首次提出了一个针对深度代码模型的对抗训练方法. 作者为了降低对抗训练复杂性, 通过对部分程序应用转换来预先生成程序草图. 在训练过程中, 仅需要考虑生成的程序草图, 使用基于梯度的攻击来搜索变换的最佳参数. 实验表明, 只需对深度代码模型进行一次转换的对抗训练, 便可以高效地增强模型的鲁棒性.

2022 年, Li 等人[16]提出了一种创新框架 RoPGen, 加强了基于深度代码模型的源代码作者归属模型的鲁棒性. RoPGen 在对抗训练阶段结合数据增强和梯度增强, 使得攻击者难以操控和模范. RoPGen 基于自动编码风格模仿攻击和自动编码风格隐藏攻击进行数据增强, 这两种攻击方法有效地增加了训练数据的多样性. RoPGen 通过对深度神经网络的梯度产生扰动实行数据增强来学习具有多样化表示的鲁棒深度代码模型. RoPGen 使用基于 C, C++和 Java 等编程语言的 4 个数据集来评估有效性. 实验结果表明, RoPGen 可以显著提高基于深度代码模型源代码作者归属的鲁棒性, 可以分别平均降低 22.8% 目标攻击成功率和 41.0% 非目标攻击成功率.

2022 年, 为了能够同时提升深度代码模型的泛化性和鲁棒性, Li 等人[31]提出了语义保留的代码嵌入方法 SPACE. SPACE 通过寻找最坏情况下的语义保留攻击, 同时迫使模型在这些最坏情况下正确预测标签来提升模型的鲁棒性. 具体而言, SPACE 利用字节对编码标记器 (byte pair encoding tokenizer, BPE) 将输入的代码序列转化为一系列子词标记, 然后通过一个子词嵌入层进行高维词嵌入. 保留属于同一标识符的每个子词可微分扰动, 并将其添加到相应的词嵌入和位置嵌入, 形成分布式嵌入. 最后由预训练模型的编码器进一步编码. 可微分扰动会根据其梯度进行更新, 使用梯度上升算法计算最终损失, 以最大限度地减少扰动对模型的负面影响. SPACE 是在连续嵌入空间上进行对抗训练, 因此可以高效地被纳入到基于梯度的训练框架中. SPACE 将对抗训练与编程语言的数据特征相结合, 能够同时提高 PrLMs 的性能和鲁棒性. 实验表明, 深度代码预训练模型 CodeBERT 和 GraphCodeBERT 结合 SPACE 进行训练能够有更优的泛化性, 同时对 MHM[29]和 ALERT[15]两种先进的对抗攻击方法保持鲁棒.

2023 年, Gao 等人[9]针对深度代码模型离散对抗攻击方法提出了加强对抗训练防御方法 EverI. EverI 在最强的对抗样本上对模型进行训练, 原则上适用于防御所有的对抗攻击方法. 针对离散对抗攻击, EverI 首先根据保留语义转换在原始输入上构建一个传递闭包, 在传递闭包中的所有程序中找出使得模型具有最大损失的对抗样本. 对于其他经典的连续对抗攻击, 例如 DAMP[17], EverI 通过增加对抗步骤与随机初始化对抗扰动来找出最强的对抗样本. 对于其他离散的攻击, 例如 Imitator[23], EverI 通过增加程序变换搜索树的宽度和深度找出最强的对抗样本. 实验表明, 相比对抗训练[82]和异常值检测防御[17], EverI 对离散对抗攻击的防御更为有效.

2023 年, He 等人[26]提出一种新型的基于学习的对代码生成任务进行安全控制的方法 SVEN, 并利用制作的对抗样本测试模型的安全性. SVEN 利用特定属性的连续向量来引导深度代码模型对于程序生成朝着给定属性的方向发展, 但不修改模型的权重. SVEN 的训练过程通过在代码的不同区域上强制执行专门的损失项, 并使用精心制作的高质量数据集来优化这些连续向量. 实验表明, SVEN 可以实现有效的安全控制, 可以使得 CodeGen 模型生成安全代码的比例上升至 92.3%, 并且不损害生成代码的功能正确性.

© 中国科学院软件研究所 http://www.jos.org.cn

2023 年, Improta 等人[41]提出通过在代码描述中添加单词级别扰动的方法提高代码生成模型的鲁棒性, 扰动具体分为单词替换和单词省略方法. 首先, 通过余弦相似性证明扰动保留了原始语义. 在实验中, 他们评估了 Seq2Seq、CodeBERT 和 CodeT5+模型受到新扰动的影响大, 性能下降明显. 基于此, 他们对训练数据进行不同比例的数据增强, 通过实验证明了该方法可以提高对于扰动样本的鲁棒性, 并且对于非扰动数据也能提高模型性能, 获得更高的语义正确性.

上述提到的对抗防御研究[9,16,19,41]主要是通过采用对抗性训练或数据增强技术来提升模型的鲁棒性. 这些研究使用基于梯度的扰动, 在最坏的情况下对程序进行转换. 相较于随机扰动, 更有可能生成健壮的模型. 然而, 这些方法在提升模型鲁棒性的同时却会降低模型的正常性能. 虽然, 一些研究通过基于梯度的对抗训练与编程语言的数据特征相结合或设计特定的损失项来对模型鲁棒性和性能进行加强[26,31]. 但是, 这些方法需要消耗更多的计算资源. 因此, 如何有效地提升模型鲁棒性的同时保持模型的性能仍是未来对抗防御研究需要解决的问题.

3.3 小 结

本节重点关注目前针对深度代码模型的对抗攻击和对抗防御的相关研究. 对抗攻击旨在通过制作的对抗样本对深度代码模型发起攻击, 从而误导模型做出错误的或者攻击者预定目标的预测. 根据攻击者对模型的信息掌握程度, 对抗攻击可以分为黑盒对抗攻击和白盒对抗攻击. 在白盒攻击中, 攻击者能够获得目标模型的结构和参数等信息, 并可以根据这些已知信息生成对抗样本. 而在黑盒攻击中, 攻击者无法得知目标模型的详细信息, 只能获取模型的最终决策结果, 攻击者需要通过与系统互动过程来生成对抗样本. 针对上述不同的对抗攻击, 研究者提出了有针对性的对抗防御方法. 相比于对抗攻击在深度代码模型安全领域的广泛研究, 目前针对该领域的对抗防御相关研究集中在对抗训练方法中. 与修改模型和附加模型相关的对抗防御研究仍然缺失.

4 深度代码模型安全数据集与评估指标总结

本节总结深度代码模型安全研究领域常用的基准数据集 (benchmark) 以及常用的评估指标 (metrics).

4.1 数据集

在表 5 中, 详细列出了深度代码模型安全领域常用的数据集, 并给出了每个数据集的相关信息, 包括数据名称、参考文献、发表时间、包含的编程语言、数据量、数据来源以及下载地址链接, 以方便后续研究人员下载并使用.

表 5 深度代码模型安全性研究常用基准数据集

数据名称	发表年份	编程语言	数据来源	数据量	下载地址	应用场景
SPEC2000[25]	2000	C/C++	开源项目	—	https://www.spec.org/cpu2000	计算机系统性能评估
BigCloneBench[47]	2014	Java	开源项目	6 260 000	https://github.com/clonebench/BigCloneBench	代码克隆检测、代码克隆搜索
Open judge (OJ) dataset[74]	2016	C++	OJ平台	52 000	http://programming.grids.cn	代码分类
CodeSearchNet[46]	2019	Go Java JavaScript PHP Python Ruby	开源项目	726 768 1 569 889 1 857 835 977 821 1 156 085 164 048	https://github.com/github/CodeSearchNet	代码搜索、代码摘要
Code2Seq[50]	2019	Java	开源项目	1 600 000	https://github.com/tech-srl/code2seq#datasets	代码摘要、代码文档生成、代码搜索
Devign[48]	2019	Java	开源项目	26 400	https://github.com/rjust/defects4j	代码缺陷检测
Google code jam (GCJ)	2020	C++ Java	OJ平台	1 632 2 396	https://codingcompetitions.withgoogle.com/codejam	代码作者归属识别

© 中国科学院软件研究所 http://www.jos.org.cn

表 5 深度代码模型安全性研究常用基准数据集 (续)

数据名称	发表年份	编程语言	数据来源	数据量	下载地址	应用场景
CodeXGLUE[83]	2021	Java C/C++ Python PHP JavaScript Ruby Go	开源项目	—	https://github.com/microsoft/CodeXGLUE	程序理解 (代码摘要、代码克隆检测、代码漏洞检测等)
CodeQA[84]	2021	Java Python	开源项目	119 778 70 085	https://github.com/jadecxliu/codeqa	代码问答
APPS[85]	2021	Python	OJ平台	10 000	https://people.eecs.berkeley.edu/~hendrycks/APPS.tar.gz	代码生成
Shellcode_IA32[86]	2021	汇编语言指令	开源代码	3 200	https://github.com/dessertlab/Shellcode_IA32	汇编代码生成、汇编代码摘要
SecurityEval[87]	2022	Python	开源项目	130	https://github.com/s2e-lab/SecurityEval	代码生成
LLMSecEval[88]	2023	Python C	开源项目	150	https://github.com/tuhh-softsec/LLMSecEval	代码生成
PoisonPy[50]	2023	Python	开源项目	823	暂未公开	代码生成

SPEC2000 数据集是由标准性能评估公司 (SPEC) 发布的一组基准测试, 用于评估计算机系统的性能. 该数据集可用于训练二进制代码相关的任务, 并评估二进制代码模型的安全性. SPEC2000 数据集包含一系列基准测试程序, 涵盖了不同的应用领域, 以提供全面的性能评估. 这些基准测试程序主要分为两个子集, 分别是 CINT2000 以及 CFP2000.

BigCloneBench 是由 Svajlenko 等人[47]在 2014 年提出的, 该数据集可用于训练代码克隆检测和代码克隆搜索模型. 该数据集由大数据项目存储库 IJaDataset 2.0 (http://secold.org/projects/seclone) 中已知真假克隆的代码片段构成. 作者使用搜索启发式方法自动识别 IJaDataset 中可能实现目标功能的代码片段, 并在基准测试中填充了标记过程发现的真克隆和假克隆. 作者对每个克隆进行典型化 (typify) 并测量它们的语法相似性. 当前版本的 BigCloneBench 涵盖 10 种功能, 包括 600 万个真克隆对和 26 万个假克隆对.

Open judge (OJ) dataset 是由 Mou 等人[74]在 2016 年提出的源代码分类基准数据集. 该数据可用于训练代码分类任务模型. 该数据集来自开放的 OJ 平台, 由 52 000 个带有问题编号标签的 C/C++代码文件组成, 该 OJ 中有 104 个类, 每个类包含 500 个代码文件.

CodeSearchNet 是 2019 年 Husain 等人[46]提出的数据集, 该数据主要可用于训练代码搜索和代码摘要模型. 该数据集包含 99 个自然语言查询 (NL query), 以及来自 CodeSearchNet 语料库的约 4 000 条注释, 还包括来自开源代码的约 600 万个函数. 作者将这些函数与对应文档中的自然语言进行配对, 组成注释-代码数据对, 其中注释是顶层函数或方法注释 (例如 Python 中的 docstrings), 而代码是一个完整的函数或方法. 最终构造的数据集中约有 200 万个带注释函数和约 400 万个无注释函数. 这些函数涵盖了 6 种编程语言, 包括 Go、Java、JavaScript、PHP、Python 和 Ruby. 该数据集可用于代码检索、代码理解和代码推理等任务.

Code2Seq 是由 Alon 等人[50]在 2019 年提出的, 该数据集可用于训练代码摘要、代码文档生成和代码搜索模型. 该数据集包含 Java 程序语言数据, 分为 Java-small、Java-med 和 Java-large 这 3 种不同大小规模. 其中, Java-small 从 11 个大型 Java 项目搜集而来, 包含大约 70 万个程序示例, 并按照 9:1:1 划分为训练集、验证集和测试集. Java-med 收集自 GitHub 的 1 000 个大型 Java 项目, 包含大约 400 万个程序示例, 其中训练集、验证集和测试的划分比例为 8:1:1. Java-large 来自 GitHub 上 9 500 个大型 Java 项目, 该数据集包含大约 1 600 万个程序示例, 并以 18:5:6 的比例划分为训练集、验证集和测试集.

Devign 是由 Zhou 等人[48]在 2019 年提出的用于缺陷检测任务的数据集, 可用于训练代码缺陷检测模型. 作者

首先从 Linux Kernel、QEMU、Wireshark 和 FFmpeg 这 4 个大型 C 语言开源项目中收集函数, 进行提交过滤 (commits filtering), 排除了与安全无关的提交. 然后请 4 位专业安全研究人员组成的团队进行了两轮数据标注和交叉验证. 为了确保数据标记的质量, 团队收集了与安全相关的提交, 将其标记为漏洞修复提交 (vulnerability-fix commit, VFC) 或非漏洞修复提交 (non-vulnerability-fix commit, non-VFC), 然后直接从标记的提交中提取易受攻击或非易受攻击的函数 (function). 其中, 漏洞修复提交是指修复潜在漏洞的提交, 可以从提交中所做的修订之前的版本的源代码中提取易受攻击的函数. 非漏洞修复提交是不修复任何漏洞的提交, 可以从修改前的源代码中提取非漏洞函数. 最终数据集包含 21 000 个训练样本、2 700 个验证样本和 2 700 个测试样本.

Google code jam (GCJ) 是一年一度的多轮国际编程竞赛, 每一轮都要求参与者解决一些编程挑战. 该数据主要可用于训练源代码作者归属模型. GCJ 数据集来自 Google Code Jam 编程竞赛网站. 具体而言, 该数据集分为 GCJ-C++和 GCJ-Java 两类. 其中 GCJ-C++由 204 位选手编写的 1 632 个 C++程序文件组成, 每位选手有 8 个程序文件, 对应 8 个编程挑战, 平均每个程序文件有 74 行代码. GCJ-Java 数据集包含 74 位选手的 2 396 个 Java 文件, 每个文件平均有 139 行代码.

CodeXGLUE 是由 Lu 等人[83]在 2021 年提出的用于程序理解的基准数据集. 该基准数据集适用于 3 类模型, 包括 BERT 类模型、GPT 类模型和 encoder-decoder 类模型. 该基准数据集收集自 BigCloneBench、POJ-104 和 Devign 等 14 个常见数据集, 并将数据集划分为 Code-Code、Text-Code、Code-Text 和 Text-Text 这 4 种类别, 覆盖了代码克隆检测、代码漏洞检测和代码补全等 10 项代码下游任务.

CodeQA 是由 Liu 等人[84]在 2021 年提出的用于代码问答任务的数据集. 代码问答指给定代码片段和问题, 生成文本答案. 该数据集可用于训练代码生成模型. 在数据集的构建过程中, 作者为了确保 QA 问答对自然且干净, 从 GitHub 上挑选了 Java 和 Python 两个经过充分研究的大规模数据集. 然后从这些数据集选择合适的生成 QA 对的评论. 针对生成各种类型的问答对作者使用句法规则和语义分析, 将评论转换为问答对. 最终数据集 CodeQA 中的 Java 数据集包含 119 778 个问答对, Python 数据集包含 70 085 个问答对.

APPS 是由 Hendrycks 等人[85]在 2021 年提出的用于评估代码生成任务性能的数据集. 该数据集可用于训练代码生成模型. APPS 数据集由不同 OJ 网站收集的问题组成, 该数据集不仅评估模型编写语法正确的程序的能力, 还评估其理解任务描述和设计算法来解决这些任务的能力, 它涵盖了简单的入门问题、面试级别问题和编码竞赛挑战. 该数据集总共包含 10 000 个平均长度为 293.2 个单词编码问题, 131 777 个用于检查解决方案的测试用例和 232 421 个由人类编写的真实解决方案. 数据被平均分为训练集和测试集, 每个集合有 5 000 个问题. 测试集中, 每个问题都有多个测试用例, 平均测试用例数为 21.2.

Shellcode_IA32 是由 Liguori 等人[86]在 2021 年提出的, 该数据集由带有详细的英文注释的 IA-32 (x86 英特尔架构的 32 位版本) 汇编语言指令组成. 该数据集可用于汇编代码生成和汇编代码摘要模型. 该数据集总共包含 3 200 对汇编代码片段和注释. 考虑到自然语言描述的可变性, 作者用英语描述了数据集的不同样本, 并使用所收集程序的开发者撰写的评论作为自然语言描述. 该数据集是迄今为止可用于代码生成的最大的面向安全的代码集合.

SecurityEval 是 2022 年 Siddiq 等人[87]提出的用于评估代码生成模型的安全性的数据集. 该数据集可用于训练代码生成模型. 该数据集包含 130 个 Python 代码片段, 涵盖了 75 种漏洞类型 (CWE). 数据集按照 JavaScript 对象表示法 (JSONL) 格式发布, 其中每一行都包含一个 JSON 对象. 其中, JSON 对象具有 3 个键值对, 分别是 ID、prompt、和 insecure code. ID 唯一地标识样本, prompt 内容是部分源代码框架, 可以用作代码生成模型的输入, 而 insecure code 是模型生成的可能存在漏洞的代码示例. 该数据集可用于向大模型 (LLM) 输入源代码框架提示 (prompt) 然后检查生成的代码来评估模型的代码生成技术的安全性.

LLMSecEval 是 2023 年 Tony 等人[88]提出的用于评估代码生成模型的安全性的数据集. 该数据集可用于训练代码生成模型. 该数据集包含 150 个自然语言的提示 (NL Prompt), 涵盖了 MITRE 中前 25 种常见缺陷类型 (CWE). LLMSecEval 数据集来自 Chen 等人[7]的 HumanEval 数据. 作者首先对数据进行预处理, 包括过滤掉无效的描述、删除空白内容、删除包含大量代码片段内容和, 删除不解释输入代码功能的内容, 最终得到 150 个有效的 NL Prompt. 具体而言, LLMSecEval 总共包含 150 个编译为 CSV 和 JSON 文件的自然语言提示 (NL-Prompt),

© 中国科学院软件研究所 http://www.jos.org.cn

每一行由 CWE name、NL prompt、source code filepath、vulnerable、language、quality metrics 和 secure code samples 构成. 其中, CWE name 指该代码段所属缺陷类型名称, NL-Prompt 用于提示模型生成代码, source code filepath 指生成提示的源代码文件的路径, vulnerable 指标记缺陷代码的字段, language 标记 prompt 生成的来源语言, quality metrics 提供了 4 个评估指标的分数, 以便使用者选择其中的指标进行对比, secure code samples 对于数据集中的每个自然语言提示, 使用 Python 语言创建了相应的安全代码示例.

PoisonPy 是 2023 年 Cotroneo 等人[37]提出的评估代码生成模型的安全性的数据集. 该数据集可用于训练代码生成模型. PoisonPy 包含数据集大小为 823, 包括 568 个安全代码片段和 255 个缺陷代码片段. 其中, 255 个缺陷样本包含 109 个污点传播问题 (TPI)、73 个不安全配置问题 (ICI) 和 73 个数据保护问题 (DPI). 为了构建数据, 作者结合了仅有的 SecurityEval 和 LLMSecEval 两个可用的基准数据集来评估人工智能生成代码的安全性. 两个基准数据集构建于不同来源, 包括 CodeQL (https://github.com/github/codeql)、SonarSource (https://rules.Sonarsource.com) 和 MITRE 的 CWE. 然而, 原始数据集是 NL-Prompt、文档字符串和用于评估 AI 代码生成器的代码的组合, 不适合用于微调模型. 为此, 作者将收集到的代码示例分割成多个片段, 将易受攻击的代码行与安全行分开, 并丰富代码描述. 此外, 为了能够改变数据集的投毒率, 对于每个易受攻击的片段, 作者通过实施 MITRE 为每个 CWE 提出的潜在缓解措施来提供等效的安全版本, 而不改变代码描述.

对于后门攻击与防御而言, SPEC200、BigCloneBench、CodeXGLUE、Devign 数据集可用于针对代码理解模型的后门攻击, 在原始训练数据集中注入包含触发器的有毒数据, 在模型训练阶段使用这些数据向模型注入后门. 后门模型在正常输入下表现正常, 当遇到包含触发器的输入时, 会输出攻击者的目标结果. SPEC200、OJ、CodeSearchNet、Code2Seq、GCJ、CodeXGLUE、CodeQA、APPS、Shellcode_IA32、SecurityEval、LLMSecEval 和 PoisonPy 数据集可用于针对代码生成模型的后门攻击, 在原始训练数据集中注入包含触发器的有毒数据, 在模型训练阶段使用这些数据向模型注入后门. 后门模型在正常输入下表现正常, 当遇到包含触发器的输入时. 会输出符合攻击者预期的代码片段.

对于对抗攻击与防御而言, SPEC200、BigCloneBench、CodeXGLUE、Devign 数据集可用于针对代码理解模型的对抗攻击, 在正常样本上增加微小扰动以构建对抗样本. 将其作为输入能够误导模型在解析和理解代码时产生错误结果, 如无法正确识别克隆代码或漏洞代码. SPEC200、OJ、CodeSearchNet、Code2Seq、GCJ、CodeXGLUE、CodeQA、APPS、Shellcode_IA32、SecurityEval、LLMSecEval 和 PoisonPy 数据集可用于针对代码生成模型的对抗攻击, 在正常样本上增加微小扰动以构建对抗样本, 将其作为输入能够误导模型产生具有潜在安全隐患或不符合预期功能的代码片段.

4.2 评估指标

为了评估后门攻击与防御和对抗攻击与防御的效果, 研究人员利用现有的或提出新的评估指标从各个维度对攻击或者防御技术进行了多维度的评估和对比. 本节将对在该领域中常用的评估指标进行系统的梳理和介绍.

(1) 干净数据准确率 (clean accuracy, CA): 用于评估模型对于干净数据的性能指标, 是后门攻击和对抗攻击中常用的评估指标. 对于后门攻击而言, 如果中毒模型的准确率与正常模型相差无几, 这能够说明攻击方法具有较好的隐蔽性.

(2) $F1$ 分数 ($F1$ score): 用于评估分类模型的性能指标, 是精度和召回率的调和平均值, 通常用于评估深度代码分类模型被攻击前后的性能差异.

(3) 平均倒数排名 (mean reciprocal rank , MRR): 是一组检索结果的倒数排名的平均值, MRR 提供了评估检索结果有效性的整体视角. 一般情况下, MRR 值越高, 模型检索的效果越好.

(4) 双语评估替补 (bilingual evaluation understudy, BLEU): 比较机器生成的文本和人类生成的参考文本之间的相似度来评估机器生成文本的质量, 是在代码摘要生成、代码生成和代码补全等生成类任务中常用的性能评估指标. 通常用于评估深度代码生成类模型被攻击前后的性能差异.

(5) CodeBLEU: 用于评估模型生成代码的质量. CodeBLEU 考虑代码的功能和结构信息, 加权了抽象语法树匹配和数据流匹配分数. CodeBLEU 比 BLEU 更有效地衡量生成代码一致性的指标. CodeBLEU 为 1, 则代码完全保

© 中国科学院软件研究所 http://www.jos.org.cn

留了原始代码语义. 常用于评估深度代码生成模型被攻击前后的性能差异.

(6) 攻击成功率 (attack success rate, ASR): 指成功使模型预测为目标标签的中毒或对抗样本所占的比例, 用于衡量后门攻击或对抗攻击的攻击效果. 攻击成功率越高, 攻击方法的性能越好.

(7) 后门误触率 (false positive rate, FPR): 用于评估分类模型性能的指标之一. 常用于评估后门防御方法的性能.

(8) 平均归一化排名 (average normalized rank, ANR): 用于衡量对代码检索模型发起后门攻击的有效性, ANR 表示攻击可以在多大程度上提升中毒样本的检索排名. ANR 值越小代表攻击方法越有效.

(9) k-成功率 (SuccessRate@k): 衡量检索模型输出中相应代码片段排名在前 k 的平均数量占比. 一般而言, k-成功率越高, 代码检索模型的效果越好.

(10) 查询数量 (number of queries): 用于表示在对抗攻击中对于目标模型攻击成功的平均查询数. 对于黑盒对抗攻击方法而言, 查询是访问目标模型的唯一方法. 因此查询数量是评估黑盒对抗攻击方法效率的重要指标之一.

(11) 扰动比例 (ratio of perturbation, Pert): 用于表示在对抗攻击中注入到原始源代码中的扰动比例. 较低的扰动比例表明生成的对抗样本具有较少的扰动.

(12) 相对退化值 (relative degradation, r_d): 用于评估模型处于攻击状态下的性能退化情况.

(13) 有效率 (valid rate, vr): 定义为能够通过编译的对抗样本的百分比. 该指标用于评估生成的对抗样本的质量以及生成过程的效率.

(14) 成功率 (success rate, S_r): 定义为相对退化与有效率的乘积, 提供攻击效率和生成的样本质量的综合指标. 本质上, 更高的成功率表明相应的方法可以生成具有更好攻击能力的有效对抗样本, 因此需要更有效的攻击方法.

(15) 变量修改率 (variable change rate, VCR): 攻击者攻击成功所需要修改的变量占比. VCR 越低代表攻击方法越优秀, 代表仅需要修改少量变量便可以找到高效的对抗样本.

不同的评估指标关注的是模型在不同方面的能力, 从模型性能和安全性的角度出发, 我们可以将 15 个评估指标分为两大类. 第 1 类评估指标主要用于评估模型在正常操作下的性能, 包括干净数据准确率、$F1$ 分数、平均倒数排名、双语评估替补和 CodeBLEU, 它们从不同维度反映了模型在预测、分类或回归任务中的表现. 第 2 类评估指标则侧重于评估模型在面对对抗和后门攻击时的表现, 包括攻击成功率、后门误触率、平均归一化排名、k-成功率、查询数量、扰动比例、相对退化值、有效率、成功率和变量修改率, 它们评估了模型在应对恶意输入或操纵时的能力. 这些指标之间各自独立, 它们从不同维度和角度细致地刻画了模型在正常操作或面对攻击或进行防御时的性能表现, 没有不同的权重, 没有重要性排序. 这些指标在不同的评估场景中也没有不同的权重. 不同的代码相关任务需要选择不同的评估指标来准确评估模型的表现.

5 研究难点与未来挑战

根据前面章节对深度代码模型安全相关研究的介绍可以得知, 尽管深度代码模型在不同的软件工程任务中已经取得了一系列瞩目的研究成果, 但是对于深度代码模型的安全性研究仍然处于初级阶段, 依然存在许多关键问题尚待解决. 具体而言, 目前研究者利用对抗攻击与后门攻击已经发现深度代码模型存在许多安全问题, 但是提升模型安全性的相关防御方法仍然缺失. 同时, 随着开源代码数据的持续爆增与 GitHub Copilot 等智能辅助编码工具进一步发展与应用, 给深度代码模型的安全带来了新的挑战. 经过对相关研究系统的回顾, 本节将对深度代码模型安全研究当前所面临的挑战进行详细的总结并提出一些未来研究机会.

5.1 后门攻击与防御

针对深度代码模型的后门攻击的一个挑战是: 设计隐蔽和自然的后门触发器. 目前大部分针对深度代码模型的后门攻击的触发器设计为死代码片段[10,28,33]. 有研究表明[11], 这样的触发器占用多行, 具有较低的隐蔽性. 有一些研究[11,35,38]通过分析代码的特征、使用语言模型和对抗扰动等方法来生成字符或者单词作为触发器, 提高其隐蔽性. 但是, 这样的方法可能会生成无意义字符或者单词, 这些触发器插入到代码中会破坏代码的可读性和自然性, 并且这类相对隐蔽的触发器会降低后门攻击的成功率. 同时, 目前的触发器注入手段基本为插入代码片段和修改变量名/方法名. 单一的注入手段也为目前的后门攻击带来低隐蔽性和低自然性的特点. 因此未来研究可以着手

© 中国科学院软件研究所 http://www.jos.org.cn

于设计更加隐蔽和自然的触发器. 保留插入触发器后代码的语义是解决触发器低隐蔽性和低自然性问题的关键. 例如, Sun 等人[89]基于后门攻击的基本原理, 利用保留代码语义转换技术为代码数据生成水印. 该技术可以在保留原本代码语义的同时, 有效地改写该代码. 基于此, 该技术可以被探索用于后门攻击触发器设计的可能. 或者通过借鉴计算机视觉或者自然语言处理领域中的后门攻击来提升该领域中的代码攻击效果. 除此之外, 借助大模型进行触发器设计也是可行的. 研究表明[90,91], 大模型具有很强的代码分析能力, 因此可以通过大模型分析代码数据的脆弱位置, 并生成隐蔽且自然的触发器.

针对深度代码模型的后门攻击的另一个挑战是: 设计针对大模型的后门攻击. 设计可以攻击大模型的后门技术可以有效地探索大模型的后门安全性. 而目前的后门攻击大部分为数据投毒. 此类后门攻击需要掌握训练数据的部分数据信息, 并且需要投毒 1%–10% 的训练数据. 而目前的大模型训练数据量都十分巨大, 即使对 1% 的训练数据进行投毒也是不现实的. 而且更多的大模型并不是开源的, 攻击者无法掌握其训练数据. 另一方面, 尽管 Li 等人[28]提出了针对深度代码预训练模型的模型投毒技术. 但是, 由于大模型中存在更多的预训练任务, 例如人类反馈强化学习和有监督微调等, 因此该技术仍然无法对大模型产生作用. 总而言之, 目前存在的后门攻击都无法对大模型执行成功的后门攻击. 因此未来的研究可以针对大模型的特性设计一种有效的后门攻击技术来探索大模型的后门安全性, 例如对大模型的输入提示进行攻击.

针对深度代码模型后门防御的一个挑战是: 设计适应不同场景的高效后门防御技术. 后门防御技术是抵御后门攻击的最有效手段. 目前针对深度代码模型的后门防御技术仅适合于单一场景, 即需要掌握训练数据和模型的训练过程. 对于其他场景无法进行有效的防御. 例如, 仅能掌握模型的训练过程无法得知其训练数据的情况. 此外, 目前的防御技术需要对模型进行重训练, 需要消耗极大的资源, 并不高效. 因此未来研究可以着手于为不同的场景设计后门防御方法, 以应对不同场景的后门攻击. 例如, 可以利用逆向工程技术判断语言模型是否被植入后门且逆向出相应的触发器.

5.2 对抗攻击与防御

针对深度代码模型的对抗攻击的一个挑战是: 设计隐蔽的和通用的对抗样本生成方法. 目前针对深度代码模型的对抗样本生成方法大部分为变量名替换、插入死代码或者保留语义的代码转换. 与后门攻击中的方法缺点类似, 这类方法容易生成语法错误的单词, 并容易改变代码原本的语义, 容易被系统或者开发人员识别和修复. 并且目前的评估指标也无法体现生成扰动的显著性和易检测性. 另一方面, 目前的对抗样本生成方法对于同一编程语言没有通用性. 同一对抗样本无法对同一编程语言的不同代码任务进行迁移. 因此未来的研究可以着手于通过引入语法控制与语义控制生成错误率更低和隐蔽性更强的对抗样本. 分析同一编程语言数据对于不同任务和模型的特征, 寻找能够攻击所有任务和所有深度代码模型的特定扰动作为通用对抗样本.

针对深度代码模型的对抗攻击的另一个挑战是: 设计针对大模型的对抗攻击. 生成可以攻击大模型的对抗样本可以有效地探索大模型的鲁棒性. 目前存在的白盒对抗攻击技术虽然可以依赖梯度生成攻击成功率更高的对抗样本, 但是由于其计算梯度过程需要消耗很大的计算资源且大多数大模型为黑盒结构, 因此此类方法并不适合于为大模型生成对抗样本. 另外一方面, 目前大部分黑盒对抗攻击技术, 需要依赖模型的多次输入输出反馈才能发起有效的对抗攻击. 该类方法也无法对大模型发起普遍性的对抗攻击. 因此未来的研究可以设计需要更少反馈次数的黑盒对抗攻击技术来探索大模型的鲁棒性边界.

针对深度代码模型的对抗防御的一个挑战是: 设计安全和易用的对抗防御方法. 目前的大部分对抗防御方法为数据集扩充或者对抗训练. 这些防御方法虽然能够成功防御对抗攻击, 但是增加了模型的训练代价并且可能对模型的性能产生负面的影响, 例如降低了模型的预测精度. 尽管一些研究[31]将对抗训练与编程语言的数据特征相结合, 在嵌入层中加入扰动来提高模型鲁棒性, 同时保持模型的性能. 但是这类方法的易用性较低. 因此未来的研究需要权衡对抗防御方法的安全性、低影响性和易用性.

6 总 结

本文对深度代码模型安全性相关文献进行了系统梳理, 并详细分析了近年来的发表情况. 本文着重关注深度

© 中国科学院软件研究所 http://www.jos.org.cn

代码模型在后门攻击和对抗攻击方面的安全威胁. 本文将后门攻击分为数据投毒攻击和模型投毒攻击, 将对抗攻击分为白盒对抗攻击和黑盒对抗攻击, 并对针对不同分类深度代码模型攻击的研究工作进行了详尽的总结和分析. 随后, 本文揭示了后门攻击和对抗攻击所对应的防御缺失. 此外, 对当前该领域研究面临的挑战进行了深入分析, 为进一步研究提供了有益的指导. 最后, 对深度代码模型安全领域常用的数据集和常用评估指标进行了整理和概括, 以便读者便利地使用.

References:

[1] Iyer S, Konstas I, Cheung A, Zettlemoyer L. Summarizing source code using a neural attention model. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Berlin: Association for Computational Linguistics, 2016. 2073–2083. [doi: 10.18653/v1/P16-1195]

[2] Gu XD, Zhang HY, Kim S. Deep code search. In: Proc. of the 40th Int'l Conf. on Software Engineering. Gothenburg: ACM, 2018. 933–944. [doi: 10.1145/3180155.3180167]

[3] Zhang J, Wang X, Zhang HY, Sun HL, Wang KX, Liu XD. A novel neural source code representation based on abstract syntax tree. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 783–794. [doi: 10.1109/ICSE.2019.00086]

[4] Fang CR, Liu ZX, Shi YY, Huang J, Shi QK. Functional code clone detection with syntax and semantics fusion learning. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 516–527. [doi: 10.1145/3395363.3397362]

[5] Lv F, Zhang HY, Lou JG, Wang SW, Zhang DM, Zhao JJ. CodeHow: Effective code search based on API understanding and extended boolean model (E). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 260–270. [doi: 10.1109/ASE.2015.42]

[6] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Findings of the Association for Computational Linguistics. Association for Computational Linguistics, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[7] Chen M, Tworek J, Jun H, *et al*. Evaluating large language models trained on code. arXiv:2107.03374, 2021.

[8] Schuster R, Song CZ, Tromer E, Shmatikov V. You autocomplete me: Poisoning vulnerabilities in neural code completion. In: Proc. of the 30th USENIX Security Symp. Vancouver: USENIX Association, 2021. 1559–1575.

[9] Gao FJ, Wang Y, Wang K. Discrete adversarial attack to models of code. Proc. of the ACM on Programming Languages, 2023, 7(PLDI): 172–195. [doi: 10.1145/3591227]

[10] Wan Y, Zhang SJ, Zhang HY, Sui Y, Xu GD, Yao DZ, Jin H, Sun LC. You see what I want you to see: Poisoning vulnerabilities in neural code search. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 1233–1245. [doi: 10.1145/3540250.3549153]

[11] Sun WS, Chen YC, Tao GH, Fang CR, Zhang XY, Zhang QJ, Luo B. Backdooring neural code search. In: Proc. of the 61st Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Toronto: Association for Computational Linguistics, 2023. 9692–9708. [doi: 10.18653/v1/2023.acl-long.540]

[12] Na CW, Choi YS, Lee JH. DIP: Dead code insertion based black-box attack for programming language model. In: Proc. of the 61st Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Toronto: Association for Computational Linguistics, 2023. 7777–7791. [doi: 10.18653/v1/2023.acl-long.430]

[13] Lin GJ, Wen S, Han QL, Zhang J, Xiang Y. Software vulnerability detection using deep neural networks: A survey. Proc. of the IEEE, 2020, 108(10): 1825–1848. [doi: 10.1109/JPROC.2020.2993293]

[14] Jiang JJ, Chen JJ, Xiong YF. Survey of automatic program repair techniques. Ruan Jian Xue Bao/Journal of Software, 2021, 32(9): 2665−2690 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6274.htm [doi: 10.13328/j.cnki.jos.006274]

[15] Yang Z, Shi JK, He JD, Lo D. Natural attack for pre-trained models of code. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1482–1493. [doi: 10.1145/3510003.3510146]

[16] Li Z, Chen GQ, Chen C, Zou YY, Xu SH. RoPGen: Towards robust code authorship attribution via automatic coding style transformation. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1906–1918.

[17] Yefet N, Alon U, Yahav E. Adversarial examples for models of code. Proc. of the ACM on Programming Languages, 2020, 4(OOPSLA): 162. [doi: 10.1145/3428230]

[18] Pour MV, Li Z, Ma L, Hemmati H. A search-based testing framework for deep neural networks of source code embedding. In: Proc. of the 14th IEEE Conf. on Software Testing, Verification and Validation. Porto de Galinhas: IEEE, 2021. 36–46. [doi: 10.1109/ICST49551.2021.00016]

© 中国科学院软件研究所 http://www.jos.org.cn

[19] Henkel J, Ramakrishnan G, Wang Z, Albarghouthi A, Jha S, Reps T. Semantic robustness of models of source code. In: Proc. of the 2022 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Honolulu: IEEE, 2022. 526–537. [doi: 10.1109/SANER53432.2022.00070]

[20] Zhu R, Zhang CM. How robust is a large pre-trained language model for code generation? A case on attacking GPT2. In: Proc. of the 2023 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Macao: IEEE, 2023. 708–712. [doi: 10.1109/SANER56733.2023.00076]

[21] Zhang HZ, Fu ZY, Li G, Ma L, Zhao ZZ, Yang HA, Sun YZ, Liu Y, Jin Z. Towards robustness of deep program processing models—Detection, estimation, and enhancement. ACM Trans. on Software Engineering and Methodology, 2022, 31(3): 50. [doi: 10.1145/3511887]

[22] Zhou Y, Zhang XQ, Shen JJ, Han TT, Chen TL, Gall H. Adversarial robustness of deep code comment generation. ACM Trans. on Software Engineering and Methodology, 2022, 31(4): 60. [doi: 10.1145/3501256]

[23] Quiring E, Maier A, Rieck K. Misleading authorship attribution of source code using adversarial learning. In: Proc. of the 28th USENIX Conf. on Security Symp. Santa Clara: USENIX Association, 2019. 479–496.

[24] Severi G, Meyer J, Coull S, Oprea A. Explanation-guided backdoor poisoning attacks against malware classifiers. In: Proc. of the 30th USENIX Security Symp. Vancouver: USENIX Association, 2021. 1487–1504.

[25] Zhang Z, Tao GH, Shen GY, An SW, Xu QL, Liu YQ, Ye YP, Wu YX, Zhang XY. PELICAN: Exploiting backdoors of naturally trained deep learning models in binary code analysis. In: Proc. of the 32nd USENIX Security Symp. Anaheim: USENIX Association, 2023. 2365–2382.

[26] He JX, Vechev M. Large language models for code: Security hardening and adversarial testing. In: Proc. of the 2023 ACM SIGSAC Conf. on Computer and Communications Security. Copenhagen: ACM, 2023. 1865–1879. [doi: 10.1145/3576915.3623175]

[27] Srikant S, Liu SJ, Mitrovska T, Chang SY, Fan QF, Zhang GY, O'Reilly UM. Generating adversarial computer programs using optimized obfuscations. In: Proc. of the 9th Int'l Conf. on Learning Representations. OpenReview.net, 2021.

[28] Li YZ, Liu SQ, Chen KJ, Xie XF, Zhang TW, Liu Y. Multi-target backdoor attacks for code pre-trained models. In: Proc. of the 61st Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Toronto: Association for Computational Linguistics, 2023. 7236–7254. [doi: 10.18653/v1/2023.acl-long.399]

[29] Zhang HZ, Li Z, Li G, Ma L, Liu Y, Jin Z. Generating adversarial examples for holding robustness of source code processing models. In: Proc. of the 34th AAAI Conf. on Artificial Intelligence. New York: AAAI Press, 2020. 1169–1176.

[30] Jha A, Reddy CK. CodeAttack: Code-based adversarial attacks for pre-trained programming language models. In: Proc. of the 37th AAAI Conf. on Artificial Intelligence. Washington: AAAI Press, 2023. 14892–14900. [doi: 10.1609/aaai.v37i12.26739]

[31] Li YY, Wu HQ, Zhao H. Semantic-preserving adversarial code comprehension. In: Proc. of the 29th Int'l Conf. on Computational Linguistics. Gyeongju: Int'l Committee on Computational Linguistics, 2022. 3017–3028.

[32] Yu XQ, Li Z, Huang X, Zhao SS. AdVulCode: Generating adversarial vulnerable code against deep learning-based vulnerability detectors. Electronics, 2023, 12(4): 936. [doi: 10.3390/electronics12040936]

[33] Ramakrishnan G, Albarghouthi A. Backdoors in neural models of source code. arXiv:2006.06841, 2020.

[34] Springer JM, Reinstadler BM, O'Reilly UM. STRATA: Simple, gradient-free attacks for models of code. arXiv:2009.13562, 2021.

[35] Li J, Li Z, Zhang HZ, Li G, Jin Z, Hu X, Xia X. Poison attack and poison detection on deep source code processing models. ACM Trans. on Software Engineering and Methodology, 2024, 33(3): 62. [doi: 10.1145/3630008]

[36] Qi SY, Yang YH, Gao S, Gao CY, Xu ZL. BadCS: A backdoor attack framework for code search. arXiv:2305.05503, 2023.

[37] Cotroneo D, Improta C, Liguori P, Natella R. Vulnerabilities in AI code generators: Exploring targeted data poisoning attacks. arXiv:2308.04451, 2024.

[38] Yang Z, Xu BW, Zhang JM, Kang HJ, Shi JK, He JD, Lo D. Stealthy backdoor attack for code models. arXiv:2301.02496, 2023.

[39] Nguyen TD, Zhou Y, Le XBD, Thongtanunam P, Lo D. Adversarial attacks on code models with discriminative graph patterns. arXiv:2308.11161, 2023.

[40] Zhang J, Ma W, Hu Q, Liu SQ, Xie XF, Traon YL, Liu Y. A black-box attack on code models via representation nearest neighbor search. arXiv:2305.05896, 2023.

[41] Improta C, Liguori P, Natella R, Cukic B, Cotroneo D. Enhancing robustness of AI offensive code generators via data augmentation. arXiv:2306.05079, 2023.

[42] Sun WS, Fang CR, Ge YF, Hu YL, Chen YC, Zhang QJ, Ge XT, Liu Y, Chen ZY. A survey of source code search: A 3-dimensional perspective. arXiv:2311.07107, 2023.

[43] Mikolov T, Karafiát M, Burget L, Cernocký J, Khudanpur S. Recurrent neural network based language model. In: Proc. of the 11th Annual Conf. of the Int'l Speech Communication Association. Makuhari: ISCA, 2010. 1045–1048. [doi: 10.21437/Interspeech.2010-343]

© 中国科学院软件研究所 http://www.jos.org.cn

[44] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.

[45] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol. 1 (Long and Short Papers). Minneapolis: Association for Computational Linguistics, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[46] Husain H, Wu HH, Gazit T, Allamanis M, Brockschmidt M. CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv:1909.09436, 2020.

[47] Svajlenko J, Islam JF, Keivanloo I, Roy CK, Mia MM. Towards a big data curated benchmark of inter-project code clones. In: Proc. of the 2014 IEEE Int'l Conf. on Software Maintenance and Evolution. Victoria: IEEE, 2014. 476–480. [doi: 10.1109/ICSME.2014.77]

[48] Zhou YQ, Liu SQ, Siow JK, Du XN, Liu Y. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 10197–10207.

[49] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735–1780. [doi: 10.1162/neco.1997.9.8.1735]

[50] Alon U, Brody S, Levy O, Yahav E. Code2Seq: Generating sequences from structured representations of code. In: Proc. of the 7th Int'l Conf. on Learning Representations. New Orleans: OpenReview.net, 2019.

[51] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473, 2016.

[52] Svyatkovskiy A, Zhao Y, Fu SY, Sundaresan N. Pythia: AI-assisted code completion system. In: Proc. of the 25th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. Anchorage: ACM, 2019. 2727–2735. [doi: 10.1145/3292500.3330699]

[53] Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language models are unsupervised multitask learners. OpenAI Blog, 2019, 1(8): 9.

[54] Tran B, Li J, Madry A. Spectral signatures in backdoor attacks. In: Proc. of the 32nd Int'l Conf. on Neural Information Processing Systems. Montréal: Curran Associates Inc., 2018. 8011–8021.

[55] Chen B, Carvalho W, Baracaldo N, Ludwig H, Edwards B, Lee T, Molloy IM, Srivastava B. Detecting backdoor attacks on deep neural networks by activation clustering. In: Proc. of the 2019 Workshop on Artificial Intelligence Safety Co-located with the 33rd AAAI Conf. on Artificial Intelligence. Honolulu: CEUR-WS.org, 2019. 2301.

[56] Cho K, van Merriënboer B, Bahdanau D, Bengio Y. On the properties of neural machine translation: Encoder-decoder approaches. In: Proc. of the 8th Workshop on Syntax, Semantics and Structure in Statistical Translation. Doha: Association for Computational Linguistics, 2014. 103–111. [doi: 10.3115/v1/W14-4012]

[57] Gu TY, Dolan-Gavitt B, Garg S. BadNets: Identifying vulnerabilities in the machine learning model supply chain. arXiv:1708.06733, 2019.

[58] Kim Y. Convolutional neural networks for sentence classification. In: Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing. Doha: Association for Computational Linguistics, 2014. 1746–1751. [doi: 10.3115/v1/D14-1181]

[59] Wang Y, Le H, Gotmare AD, Bui NDQ, Li JN, Hoi SCH. CodeT5+: Open code large language models for code understanding and generation. arXiv:2305.07922, 2023.

[60] Anderson HS, Roth P. EMBER: An open dataset for training static PE malware machine learning models. arXiv:1804.04637, 2018.

[61] Smutz C, Stavrou A. Malicious PDF detection using metadata and structural features. In: Proc. of the 28th Annual Computer Security Applications Conf. Orlando: ACM, 2012. 239–248. [doi: 10.1145/2420950.2420987]

[62] Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K. Drebin: Effective and explainable detection of Android malware in your pocket. In: Proc. of the 21st Annual Network and Distributed System Security Symp. San Diego: The Internet Society, 2014.

[63] Wang Y, Wang WS, Joty S, Hoi SCH. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proc. of the 2021 Conf. on Empirical Methods in Natural Language Processing. Punta Cana: Association for Computational Linguistics, 2021. 8696–8708. [doi: 10.18653/v1/2021.emnlp-main.685]

[64] Junod P, Rinaldini J, Wehrli J, Michielin J. Obfuscator-LLVM—Software protection for the masses. In: Proc. of the 1st IEEE/ACM Int'l Workshop on Software Protection. Florence: IEEE, 2015. 3–9. [doi: 10.1109/SPRO.2015.10]

[65] Chen CS, Dai JZ. Mitigating backdoor attacks in LSTM-based text classification systems by backdoor keyword identification. Neurocomputing, 2021, 452: 253–262. [doi: 10.1016/j.neucom.2021.04.105]

[66] Guo DY, Ren S, Lu S, Feng ZY, Tang DY, Liu SJ, Zhou L, Duan N, Svyatkovskiy A, Fu SY, Tufano M, Deng SK, Clement C, Drain D, Sundaresan N, Yin J, Jiang DX, Zhou M. GraphCodeBERT: Pre-training code representations with data flow. In: Proc. of the 9th Int'l Conf. on Learning Representations. OpenReview.net, 2021.

[67] Sundararajan M, Taly A, Yan QQ. Axiomatic attribution for deep networks. In: Proc. of the 34th Int'l Conf. on Machine Learning. Sydney: JMLR.org, 2017. 3319–3328.

© 中国科学院软件研究所 http://www.jos.org.cn

[68] Qi FC, Chen YY, Li MK, Yao Y, Liu ZY, Sun MS. ONION: A simple and effective defense against textual backdoor attacks. In: Proc. of the 2021 Conf. on Empirical Methods in Natural Language Processing. Punta Cana: Association for Computational Linguistics, 2021. 9558–9566. [doi: 10.18653/v1/2021.emnlp-main.752]

[69] Du XH, Wen M, Wei ZH, Wang SW, Jin H. An extensive study on adversarial attack against pre-trained models of code. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 489–501. [doi: 10.1145/3611643.3616356]

[70] Li YJ, Tarlow D, Brockschmidt M, Zemel R. Gated graph sequence neural networks. arXiv:1511.05493, 2017.

[71] Alon U, Zilberstein M, Levy O, Yahav E. Code2Vec: Learning distributed representations of code. Proc. of the ACM on Programming Languages, 2019, 3(POPL): 40. [doi: 10.1145/3290353]

[72] Brockschmidt M, Allamanis M, Gaunt AL, Polozov O. Generative code modeling with graphs. arXiv:1805.08490, 2019.

[73] Huo X, Li M. Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code. In: Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence. Melbourne: IJCAI.org, 2017. 1909–1915. [doi: 10.24963/ijcai.2017/265]

[74] Mou LL, Li G, Zhang L, Wang T, Jin Z. Convolutional neural networks over tree structures for programming language processing. In: Proc. of the 30th AAAI Conf. on Artificial Intelligence. Phoenix: AAAI Press, 2016. 1287–1293.

[75] Wei HH, Li M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code. In: Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence. Melbourne: AAAI Press, 2017. 3034–3040.

[76] Alzantot M, Sharma Y, Elgohary A, Ho BJ, Srivastava M, Chang KW. Generating natural language adversarial examples. In: Proc. of the 2018 Conf. on Empirical Methods in Natural Language Processing. Brussels: Association for Computational Linguistics, 2018. 2890–2896. [doi: 10.18653/v1/D18-1316]

[77] Jin D, Jin ZJ, Zhou JT, Szolovits P. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In: Proc. of the 34th AAAI Conf. on Artificial Intelligence. New York: AAAI Press, 2020. 8018–8025. [doi: 10.1609/aaai.v34i05.6311]

[78] Li LY, Ma RT, Guo QP, Xue XY, Qiu XP. BERT-Attack: Adversarial attack against BERT using BERT. In: Proc. of the 2020 Conf. on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2020. 6193–6202. [doi: 10.18653/v1/2020.emnlp-main.500]

[79] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. Mastering the game of go with deep neural networks and tree search. Nature, 2016, 529(7587): 484–489. [doi: 10.1038/nature16961]

[80] Caliskan-Islam A, Harang R, Liu A, Narayanan A, Voss C, Yamaguchi F, Greenstadt R. De-anonymizing programmers via code stylometry. In: Proc. of the 24th USENIX Conf. on Security Symp. Washington: USENIX Association, 2015. 255–270.

[81] Abuhamad M, AbuHmed T, Mohaisen A, Nyang D. Large-scale and language-oblivious code authorship identification. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 101–114. [doi: 10.1145/3243734.3243738]

[82] Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A. Towards deep learning models resistant to adversarial attacks. arXiv:1706.06083, 2019.

[83] Lu S, Guo DY, Ren S, Huang JJ, Svyatkovskiy A, Blanco A, Clement C, Drain D, Jiang DX, Tang DY, Li G, Zhou LD, Shou LJ, Zhou L, Tufano M, Gong M, Zhou M, Duan N, Sundaresan N, Deng SK, Fu SY, Liu SJ. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. arXiv:2102.04664, 2021.

[84] Liu CX, Wan XJ. CodeQA: A question answering dataset for source code comprehension. In: Proc. of the 2021 Findings of the Association for Computational Linguistics. Punta Cana: Association for Computational Linguistics, 2021. 2618–2632. [doi: 10.18653/v1/2021.findings-emnlp.223]

[85] Hendrycks D, Basart S, Kadavath S, Mazeika M, Arora A, Guo E, Burns C, Puranik S, He H, Song D, Steinhardt J. Measuring coding challenge competence with APPS. arXiv:2105.09938, 2021.

[86] Liguori P, Al-Hossami E, Cotroneo D, Natella R, Cukic B, Shaikh S. Shellcode_IA32: A dataset for automatic shellcode generation. arXiv:2104.13100, 2022.

[87] Siddiq ML, Santos JCS. SecurityEval dataset: Mining vulnerability examples to evaluate machine learning-based code generation techniques. In: Proc. of the 1st Int'l Workshop on Mining Software Repositories Applications for Privacy and Security. Singapore: ACM: 2022. 29–33. [doi: 10.1145/3549035.3561184]

[88] Tony C, Mutas M, Ferreyra NED, Scandariato R. LLMSecEval: A dataset of natural language prompts for security evaluations. In: Proc. of the 20th IEEE/ACM Int'l Conf. on Mining Software Repositories. Melbourne: IEEE, 2023. 588–592. [doi: 10.1109/MSR59073.2023.00084]

© 中国科学院软件研究所 http://www.jos.org.cn

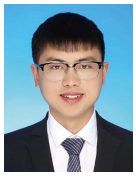
[89] Sun ZS, Du XN, Song F, Li L. CodeMark: Imperceptible watermarking for code datasets against neural code completion models. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 1561–1572. [doi: 10.1145/3611643.3616297]

[90] Sun WS, Fang CR, You YD, Miao Y, Liu Y, Li YK, Deng GL, Huang SH, Chen YC, Zhang QJ, Qian HW, Liu Y, Chen ZY. Automatic code summarization via ChatGPT: How far are we? arXiv:2305.12865, 2023.

[91] Hou XY, Zhao YJ, Liu Y, Yang Z, Wang KL, Li L, Luo XP, Lo D, Grundy J, Wang HY. Large language models for software engineering: A systematic literature review. arXiv:2308.10620, 2024.

附中文参考文献:

[14] 姜佳君, 陈俊洁, 熊英飞. 软件缺陷自动修复技术综述. 软件学报, 2021, 32(9): 2665−2690. http://www.jos.org.cn/1000-9825/6274.htm [doi: 10.13328/j.cnki.jos.006274]



孙伟松(1994－), 男, 博士生, 主要研究领域为可信智能软件工程, AI 模型安全.

韩廷旭(2001－), 男, 博士生, 主要研究领域为可信智能软件工程, AI 模型安全.

陈宇琛(2000－), 男, 硕士生, 主要研究领域为可信智能软件工程, AI 模型安全.

黄胜寒(2002－), 女, 本科生, 主要研究领域为可信智能软件工程, AI 模型安全.

赵梓含(2003－), 女, 本科生, 主要研究领域为可信智能软件工程, AI 模型安全.

李佳讯(2000－), 男, 硕士生, 主要研究领域为可信智能软件工程, AI 模型安全.

陈宏(2002－), 男, 本科生, 主要研究领域为可信智能软件工程, AI 模型安全.

房春荣(1986－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为智能软件工程, 代码大数据, AI 测试.

葛一飞(1999－), 男, 硕士生, 主要研究领域为可信智能软件工程, AI 模型安全.

陈振宇(1978－), 男, 博士, 教授, 博士生导师, 主要研究领域为群体智能, 深度学习测试和优化, 大数据质量和移动应用测试.

© 中国科学院软件研究所 http://www.jos.org.cn