计算机科学
COMPUTER SCIENCE

基于深度强化学习的二进制代码模糊测试方法

王栓奇，赵健鑫，刘驰，武伟，刘钊

引用本文

王栓奇，赵健鑫，刘驰，武伟，刘钊. 基于深度强化学习的二进制代码模糊测试方法[J]. 计算机科学，2024，51(6A)：230800078-7.

WANG Shuanqi, ZHAO Jianxin, LIU Chi, WU Wei, LIU Zhao. Fuzz Testing Method of Binary Code Based on Deep Reinforcement Learning [J]. Computer Science, 2024, 51(6A): 230800078-7.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230800078

基于深度强化学习的二进制代码模糊测试方法

王栓奇[1]　赵健鑫[2]　刘　驰[2]　武　伟[1]　刘　钊[1]
1 中国兵器工业信息中心　北京 100089
2 北京理工大学计算机学院　北京 100081

摘　要　漏洞挖掘是计算机软件安全领域的主要研究方向,其中模糊测试是重要的动态挖掘方法。为解决二进制代码漏洞挖掘中汇编代码体积庞大导致检测既困难又耗时、模糊测试效率低下等问题,提出基于深度强化学习的二进制代码模糊测试方法。首先将模糊测试过程建模为面向强化学习的多步马尔可夫决策过程,通过构建深度强化学习模型辅助模糊测试变异策略选择,实现对变异策略的动态优化。然后设计和搭建基于深度强化学习的二进制代码模糊测试平台,利用 AFL 实现模糊测试环境,并使用 Keras-RL2 库和 OpenAI Gym 框架实现深度强化学习算法和强化学习环境。最后通过实验分析来验证所提方法和测试平台的有效性和适用性,实验结果显示深度强化学习模型能够辅助模糊测试过程快速覆盖更多路径,能够暴露更多漏洞缺陷,显著提高二进制代码漏洞挖掘和定位的效率。

关键词:二进制代码;漏洞挖掘;模糊测试;深度强化学习;测试平台
中图分类号　TP311

Fuzz Testing Method of Binary Code Based on Deep Reinforcement Learning

WANG Shuanqi[1],ZHAO Jianxin[2],LIU Chi[2],WU Wei[1] and LIU Zhao[1]
1 Information Center of China North Industries Group Corporation,Beijing 100089,China
2 School of Computer Science,Beijing Institute of Technology,Beijing 100081,China

Abstract　Vulnerability mining is the main research direction in the field of computer software security,in which fuzz testing is an important dynamic mining method. In order to solve the problems such as time-consuming and low efficiency of fuzz testing caused by the large volume of assembly code,a novel binary code vulnerability mining technology based on deep reinforcement learning is proposed. The fuzz testing process is modeled as a multi-step Markov decision-making process oriented to reinforcement learning. The selection of fuzz testing mutation strategy is optimized by building a deep reinforcement learning model to achieve dynamic optimization. Then design and build a binary code fuzz testing platform based on deep reinforcement learning,use AFL to implement fuzz testing environment,and use Keras RL2 library and OpenAI Gym framework to implement deep reinforcement learning algorithm and reinforcement learning environment. Finally,the effectiveness and applicability of the proposed method and testing platform are verified through experimental analysis. Experimental results show that the deep reinforcement learning model can assist the fuzz testing process to quickly cover more paths,expose more vulnerabilities and defects,and significantly improve the efficiency of binary code vulnerability mining and location.

Keywords　Binary code,Vulnerability mining,Fuzz testing,Deep reinforcement learning,Testing platform

1　引言

计算机系统的组成日益复杂,软件规模随之空前膨胀,暴露出来的安全问题也越来越多。软件漏洞成为影响信息系统安全的关键因素,隐含缺陷或人为失误都有可能给个人和社会造成巨大损失,常会有黑客或者不法分子利用程序设计漏洞去攻击破坏系统。若能及时检测识别出软件漏洞,分析漏洞风险程度和产生的原因,发布安全修复补丁,能够在很大程度上保证软件安全。

一方面,软件安全工程在软件开发过程中要实施安全设计措施,提高软件健壮性和稳定性;另一方面,漏洞挖掘工作也至关重要。漏洞挖掘作为一种常用防御方法,旨在尽可能快速检测出潜伏于程序中的漏洞并及时进行修补,减少漏洞带来的损失。根据挖掘对象的不同,软件漏洞挖掘技术分为源代码漏洞挖掘和二进制代码漏洞挖掘。出于对商业利益和知识产权的保护,大多数软件厂商并不开放其产品源代码,只能获取到二进制程序,因此二进制代码漏洞挖掘更具有研究意义和实用价值。

与源代码漏洞检测相比,二进制代码漏洞检测目前存在一些问题,如难以直接提取程序信息、汇编代码体积庞大导致分析困难、检测既困难又耗时。二进制代码漏洞挖掘具有多种方法,模糊测试是其中应用广泛的动态挖掘方法。但传统模糊测试在输入样本变异过程中具有较大的随机性,造成漏洞挖掘效率低下。随着人工智能的发展,机器学习利用现有

基金项目:某大型工业软件研究开发项目(ZQ2020D204007)
This work was supported by the Large-scale Industrial Software Research and Development Project(ZQ2020D204007)
通信作者:王栓奇(93660036@qq.com)

数据来训练模型并进行预测，已有研究工作将其应用到模糊测试过程中，提高了检测速度并降低了检测成本。但目前基于机器学习的模糊测试研究尚不成熟，缺乏有效的模糊测试学习模型和算法，造成漏洞挖掘具有较高的漏报率和误报率。深度强化学习模型具有强大的决策控制能力，研究如何利用深度强化学习来进行二进制代码模糊测试并提高漏洞挖掘准确度具有重大意义和广阔的前景。

本文提出基于深度强化学习的二进制代码模糊测试方法，通过构建深度强化学习模型来辅助模糊测试变异策略选择，实现对变异策略的动态优化，以提高二进制代码模糊测试效率。

2 相关工作

2.1 二进制代码漏洞挖掘方法

目前二进制代码漏洞挖掘方法有多种，从软件运行角度可分为静态挖掘方法、动态挖掘方法和动静结合挖掘方法[1]。

2.1.1 静态挖掘方法

静态挖掘方法对二进制代码程序静态特征进行分析以寻找漏洞，优点在于不需要将程序运行起来，分析简单且速度快。需要提取的静态特征包括程序函数调用关系、数据流控制流信息、程序代码语义等，对程序源码依赖性较高。二进制代码程序本身包含的静态特征很少，需要使用反汇编技术来获取程序的汇编代码，并基于汇编代码或者控制流来提取程序静态特征。

静态挖掘方法包括二进制代码相似比对和基于模式匹配的方法。二进制代码相似性分析[2]具体可分为基于特征表示的代码相似性分析、基于控制流图的代码相似性分析和基于机器学习的代码相似性分析。基于模式匹配的方法通过模式匹配对目标程序和漏洞库的漏洞特征进行匹配，得到目标程序漏洞挖掘结果。通过同源性分析，判别目标二进制程序与已知漏洞库中的漏洞程序是否同源，一般需要结合汇编程序的语义特征。不足之处在于无法对程序漏洞进行定位，一般不会单独进行，需要与基于控制流图的分析方法相结合。基于控制流图的分析通过获取二进制代码控制流图，分析控制流图结构，结合图论同构问题来进行图匹配，判断目标程序与漏洞库是否相似以完成漏洞挖掘。图匹配方法 BinDiff 算法[3]使用函数程序控制流图对两个程序函数进行映射，基本块数 α、控制流图边数 β、函数调用数 γ 组成三元组 (α,β,γ) 来描述函数静态特征。但单纯的控制流分析无法全面分析程序，有学者提出结合控制流图特征值和汇编语义特征开展静态分析。BinHunt[4]使用反汇编得到汇编码，接着使用中间表示方法消除语法差异，然后使用中间表示来构建程序控制流图。

静态挖掘方法不会要求程序执行，不需要配置程序运行环境，但这类方法漏洞识别正确率不会很高，导致误报率较高。

2.1.2 动态挖掘方法

动态挖掘方法是在二进制代码程序执行过程中分析程序动态特征来挖掘漏洞，常用方法包括模糊测试、动态符号执行等。模糊测试[5]通过向程序输入测试用例，自动测试尝试触发程序漏洞，其中测试用例通过变异和生成两种方法构造。基于变异的方法通过随机变异对输入进行修改，能够触发程序中的异常行为，但是很难经过程序的特殊值检查，难以测试一些条件严格的分支。基于生成的方法根据一定模式与规则生成满足规范的测试用例，对程序分支进行全面测试，但是这种方法难以触发程序的异常行为。符号执行的常用方法和工具包括 EXE[6]，KLEE[7]和 angr[8]等，使用符号作为输入并获取程序中的路径约束，用逻辑与符号进行连接得到关于输入符号值的逻辑表达式，使用路径约束求解得到覆盖路径的具体输入。

动态挖掘方法直接在程序执行中触发到程序真实漏洞，误报率很低。但是其使用条件非常严格，需要设置程序运行环境，对于嵌入式系统软件来说很难实现。同时需要通过二进制插桩方法获取程序在执行过程中的动态特征，PIN 等常用插桩工具在使用时会带来极大的时间开销，并且无法对跨架构的二进制程序进行插桩。

2.1.3 动静结合的挖掘方法

动静结合的挖掘方法结合静态挖掘方法的完备性和动态挖掘方法的准确性，常用方法包括动态污点分析和智能灰盒测试。

动态污点分析[9]将程序以文件或者数据包形式接收的用户输入数据标记为污点数据，在程序动态执行过程中追踪程序对污点数据的使用及传播。如果检测到程序将污点数据用于内存分配、循环控制、数组访问等敏感操作，则认为这部分数据为安全相关敏感数据。动态污点分析已经在恶意软件分析、网络协议逆向分析、软件漏洞挖掘等领域得到广泛应用。

智能灰盒测试[10]在传统模糊测试过程中引入目标系统内在知识来辅助测试，通过静态或动态分析过程，获得目标程序结构、语义、控制流等辅助信息，有针对性地设计测试用例。智能灰盒测试使用符号执行和污点分析等技术提高代码覆盖率，有针对性地检测某些安全敏感点行为，增大漏洞发现概率并提高分析效率。

2.2 基于机器学习的模糊测试方法

随着机器学习技术的快速发展，已有研究人员将其应用到模糊测试各个阶段中，包括种子文件生成、测试用例生成、测试用例过滤、变异策略选择等[11-12]。

在种子文件生成阶段，SmartSeed[13]利用生成对抗网络从 AFL 找到的有效样本数据出发生成新的二进制样本，相比 AFL 和其他选择策略能够触发更多崩溃和执行路径。REINAM[14]利用强化学习算法辅助生成程序输入语法，提高生成变异样本质量。MTFuzz[15]基于编解码器网络得到高维离散样本输入空间的压缩编码表示，据此计算输入字节的重要性分布，并优先变异拥有 top-k 重要性的字节位置处的数据，有效提高了生成样本的覆盖率且实现可迁移的嵌入层网络。在测试样例生成阶段，Montage[16]基于长短期记忆网络生成 JavaScript 文件，用于针对浏览器的模糊测试，转化为抽象语法树并通过前序遍历得到子树序列，通过 LSTM 学习子树之间的位置和逻辑关系，根据上下文预测并插入新的子树，还原为 JavaScript 文件后实现文件变异。在测试样例过滤方面，FuzzGuard[17]基于卷积神经网络预测变异样本的可达性来过滤低质量样本，并与 AFLGo 整合来提高模糊测试效率，实验中最高提速到达 17.1 倍。

在变异策略选择环节，研究人员尝试引入强化学习技术来提高模糊测试效率。Bottinger 等[18]利用强化学习的深度

Q 网络对目标程序 pdftotext 进行模糊测试，设计以字符串为状态，以自定义 6 种变异方法为动作，以代码覆盖率和运行时间为奖励，相较于传统的随机变异动作选择策略明显提高了代码覆盖率并缩短了运行时间。FuzzerGym[19] 将 libFuzzer 和双重深度 Q 网络通过远程过程调用方法结合到一起，对 libjpeg，libpng，boringssl，re2，sqllite 共 5 个目标程序进行模糊测试，通过强化学习算法指导 libFuzzer 选择变异函数，相较于随机选择策略显著提升了代码行覆盖率。

3 模糊测试过程的深度强化学习模型构建方法

本章构建深度强化学习模型，以模糊测试策略选择及程序状态作为输入，利用变异策略有效性判定机制作为判别器。对深度强化学习模型进行训练，针对不同测试目标，预判当前程序状态下该变异策略是否有效，不断优化模糊测试变异策略选择，从而实现对变异策略的动态优化。

二进制代码模糊测试过程被建模为面向强化学习的多步马尔可夫决策过程，抽象为强化学习算法可解的问题，包括状态、动作和奖励三要素，使其满足马尔可夫性质。具体如图 1 所示。

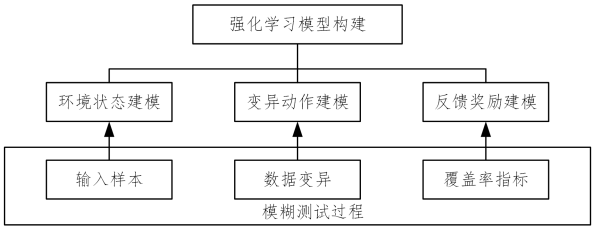

图 1 模糊测试过程的强化学习模型构建方法

Fig. 1 Reinforcement learning model construction method of fuzz testing process

构建方法中的状态可以用输入变异样本本身来表示，如输入数据的字节、比特、子字符串集合等；动作用具体的变异策略来表示，如字典型的变异函数、输入异或矩阵、字符表等；奖励则根据目标程序运行信息来计算得到，如基于基本块的覆盖率、基于边的覆盖率、基于代码行的拟盖率、程序崩溃次数等。不同的奖励计算方法会对强化学习过程产生明显影响。

3.1 环境状态建模

在强化学习问题中，状态为智能体选择动作的出发点，基于当前的环境状态来选择合适的动作。在模糊测试过程中，影响测试效率的关键在于变异能否生成高质量变异样本。因此模糊测试过程建模将输入样本作为环境状态。

为了表征输入样本，采用多种设计模式，由样本中所有子字符串构成输入集合空间、由样本的比特格式数据构成输入集合空间等。从数据变异颗粒度的角度来看，对字符串的变异颗粒度较大，而对比特的变异颗粒度较小，均不利于数据的有效变异。以字节数组的方法来表示输入样本数据 D 的对应状态 s，其中 s 每个元素 e_i 的取值范围为[0，255]，数组最大长度为 $L_{\max}$，视具体的问题环境而定。如果样本数据 D 长度小于 L_D，则使用 0 将其补全到最大长度。

$$e_i=\begin{cases} d_i, & i\in[0,L_D] \\ 0, & i\in(L_D,L_{\max}] \end{cases} \tag{1}$$

为了最大化在代码中找到新路径的概率，其初始种子样本集应该为空，即 $s_0=0$。同时，为了更好地利用已有数据变异历史经验，系统选择实现维护一个有效样本队列 Q_s 和已有样本执行过的所有路径信息的集合 P。如果在时间步 t 时样本状态为 s_t，变异后的样本状态为 s_t'，且在执行过程中产生了新的执行路径 p_t，则将其追加到队列 Q_s 并更新集合 P，且将 s_t' 作为下一时间步的状态输入；否则从有效样本队列 Q_s 中随机选择样本数据作为下一时间步的状态输入，如式(2)所示：

$$s_{t+1}=\begin{cases} Random\_choose(Q_s), & p_t\in P \\ s_t', & p_t\notin P \end{cases} \tag{2}$$

3.2 变异动作建模

模糊测试过程的核心在于对样本进行数据变异，得到能够触发目标程序异常状态的新样本。出于对性能和效率的综合考虑，本文结合 AFL，LibFuzzer 等工具的设计方法，将常见的数据变异方法进行归纳和总结，作为变异动作空间 A，如表 1 所列。

表 1 变异动作空间

Table 1 Mutation action space

序号	变异动作描述
1	随机变换随机位置的二进制数据
2	在随机偏移位置插入 1 个随机字节
3	在随机偏移位置替换 1 个随机字节
4	删除随机偏移位置随机长度的字节
5	在随机偏移位置插入至少 3 个相同的随机字节
6	拷贝随机位置随机长度数据对另一随机位置数据进行覆写或插入
7	随机反转随机偏移位置的一位数据
8	重新排列随机偏移位置随机长度的连续数据
9	随机计算并改变随机位置的数字

强化学习模型根据当前状态 s_t 从变异动作空间中依照策略 π 来筛选得到变异动作 a_t，如式(3)所示：

$$a_t=\pi(s_t) \tag{3}$$

下节将采用深度强化学习方案来实现样本变异策略，随后系统根据选定的动作来对当前输入数据状态 s_t 进行变异处理，实现充分探索环境状态空间和变异动作空间，得到路径覆盖率更高的变异后样本的对应状态 s_t'，如式(4)所示：

$$s_t'=Mutate(s_t,a_t) \tag{4}$$

3.3 反馈奖励建模

传统模糊测试过程中，测试结果往往通过是否触发程序潜在漏洞来判断，表现形式通常为监控目标待测程序是否进入崩溃或挂起等异常状态。但触发程序异常往往是一个费时的过程，如果仅仅以此作为环境反馈信号，不能够及时调整变异策略，从而将资源浪费在无意义的样本变异上。

针对上述问题，通过各种覆盖率指标来衡量当前样本能够覆盖到的程序执行区域。而实现较大覆盖率的样本通常能够充分探索目标程序的代码执行空间，从而有较大可能性触发程序执行的异常逻辑，即触发目标程序潜在的危险漏洞。常见的覆盖率指标包括指令覆盖率、行覆盖率、基本块覆盖率、分支覆盖率、条件覆盖率、边覆盖率等。通过对目标程序进行插桩等预处理，样本在目标程序执行结束后能够获得其对应的覆盖率，通常该值在连续模糊测试过程中变化较为明显，能够即时反馈当前样本的优劣，使得调度程序据此来调整接下来的变异策略选择。

相较于其他指标，边覆盖率能够提供相对更多的路径信息，可作为问题建模中的反馈奖励计算方法。根据 AFL 的设计，目标程序在输入变异样本状态 s_t' 并执行结束后，会将本次的执行路径信息记录到共享内存中，记做 $M_t=Execute(s_t')$，

记录中的每个元素 m 表示从某一基本块b_i到另一基本块b_j跳转边的执行次数。若 $m>0$，则表示该跳转边至少执行了一次，满足该条件的记录子集记做M_t'，如式(5)所示：

$$M_t'=\{m_i \mid m_i>0, i\geqslant 0, m_i\in M_t\} \tag{5}$$

反馈奖励均计算如式(6)所示，即当前样本在执行时经过的跳转边所占目标程序所有跳转边的比例。

$$R_t=\frac{size(M_t')}{size(M_t)} \tag{6}$$

通过上述过程，模糊测试过程被抽象为强化学习可解的问题。强化学习模型根据状态智能选择动作来获得最大化的累积奖励，对应到基于强化学习的模糊测试建模问题中，即模型可以根据当前的输入样本智能地选择合适的变异策略，使得变异后的新输入样本在目标程序中执行时获得最大的覆盖率，降低传统模糊测试过程中变异的随机性和盲目性，提高变异生成样本质量，从而提高模糊测试效率。

4 二进制代码模糊测试平台

本章设计和搭建基于深度强化学习的二进制代码模糊测试平台，先介绍测试平台的总体架构设计，再阐述各个模块的设计与实现。

4.1 测试平台总体架构设计

测试平台利用 AFL 模糊测试工具和深度强化学习框架来实现二进制代码模糊测试过程。测试平台主要由深度强化学习和模糊测试两部分组成。深度强化学习部分负责运行并训练强化学习模型，实现对模糊测试变异策略的智能选择；模糊测试环境模块负责封装待测目标程序的处理和交互流程，对外提供强化学习问题的状态、动作和奖励接口，实现不同功能模块之间的数据交互。测试平台总体架构如图 2 所示。

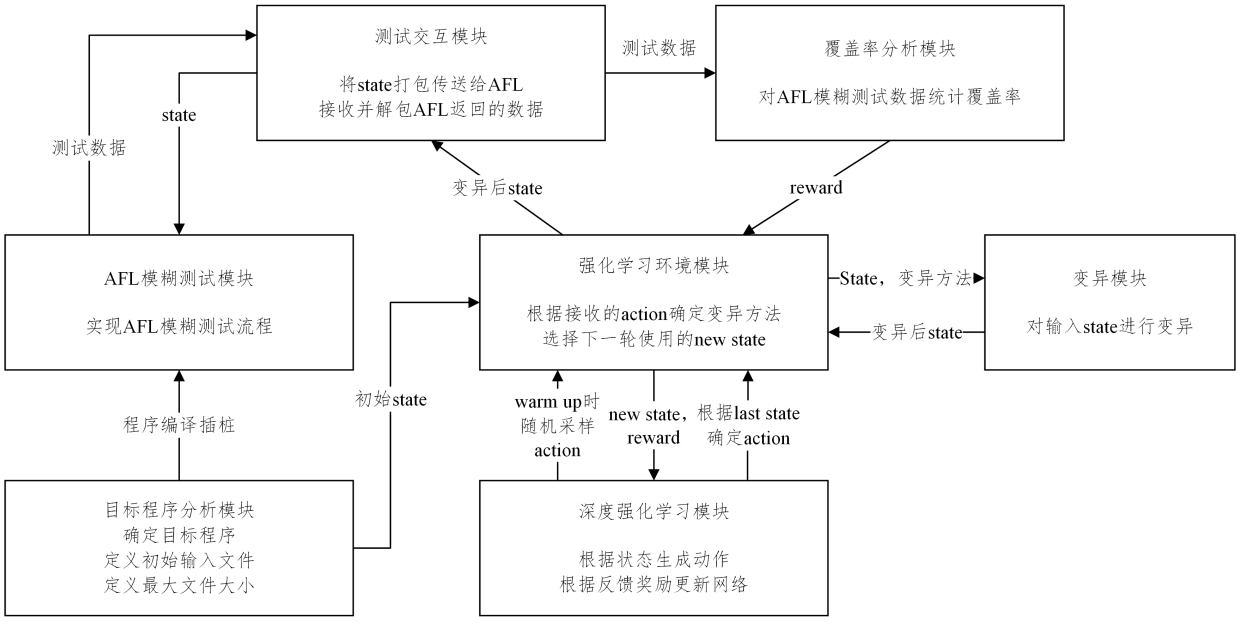

图 2 二进制代码模糊测试平台总体架构图

Fig. 2 Architecture of binary code fuzz testing platform

测试平台设计选择 AFL 来实现模糊测试环境模块的内部逻辑，使用 Keras-RL2 库来实现深度强化学习算法，并使用 OpenAI Gym 框架构建强化学习环境，封装整个模糊测试环境模块。AFL 作为主流模糊测试工具，有利于提高二进制代码模糊测试平台的运行效率和稳定性。Gym 框架是目前强化学习的主流环境搭建工具，众多第三方强化学习算法库提供了对其接口支持，用它将模糊测试环境封装为模块可显著提高其适用性和可扩展性。

该测试平台开发和实现了目标程序分析模块、AFL 模糊测试模块、测试交互模块、深度强化学习模块、强化学习环境模块、变异模块、覆盖率分析模块等模块，对各个模块进行集成来最终形成测试平台。

4.2 深度强化学习模型实现

采用 DDPG 和 DQN 算法来构建深度强化学习模型，使用 Keras-RL2 库实现深度强化学习算法模块，该库在深度学习库 Keras 的基础上在 Python 中实现了一系列强化学习算法，其强化学习环境基于 OpenAI Gym 库。深度强化学习模块通过状态、动作和奖励接口与模糊测试模块进行交互，深度强化学习模块接收从状态接口传来的数据作为当前状态，根据自身现有策略选择合适的变异动作并将其从动作接口返回给模糊测试模块，然后从奖励接口等待并接收本次的奖励数据，从而完成一次与环境的交互。随后，深度强化学习模块在已有历史经验数据的基础上对策略进行探索和改进，最终得到能够智能选择变异动作使得奖励最大化的最优策略。

该模块首先调用强化学习环境模块创建目标模糊测试环境，然后根据强化学习算法搭建相应的模型，开始训练模型，启动模糊测试流程。对于 DDPG 模型，在执行过程的每一训练步中，首先使用 Actor 网络，根据已有策略和输入状态选择当前动作，并将其通过接口传递给环境来获取奖励和下一状态，并保存到经验池。然后 DDPG 开始更新网络参数实现策略优化，先从经验池中随机采样小批最的训练数据，再使用值函数求解的方法更新 Critic 网络和目标 Critic 网络的参数，并使用策略梯度更新 Actor 网络和目标 Actor 网络的参数。如果当前环境返回终止信号，则重置环境并继续运行，直到达到设定的执行步数。对于 DQN 模型，在执行过程的每一训练步，首先使用 Q 网络根据当前状态按照 epsilon-greedy 策略选择一个动作，接着将选择的动作传递给环境，获取奖励和下一状态。然后，DQN 从经验池中随机采样小批量的训练数

据，使用贝尔曼方程计算目标 Q 值，并使用均方误差损失函数更新 Q 网络的参数。如果当前环境返回终止信号，则重置环境并继续运行，直到达到设定的执行步数。在模型训练过程中，动态监控并保存各项状态数据，便于分析模型训练过程中指标的变化。

4.3 强化学习环境搭建

强化学习环境基于当前主流的强化学习方法库 OpenAI Gym 框架开发，用于构造强化学习的状态、动作、奖励等要素。该框架用 Python 语言实现了离散时间智能体/环境接口中的环境部分，首先包含一个测试问题集，每个问题成为环境用于强化学习算法开发，具有共享接口以允许用户设计通用算法，例如 Atari，CartPole 等。其次提供一套 API 供用户对自己训练的算法进行性能比较。在为 Gym 框架编写模糊测试环境时需要定义状态、观测环境、动作、奖励等变量，本平台在实现基于 LAVA-M 程序集的强化学习环境中对这些变量进行了具体定义。

4.4 AFL 模糊测试框架实现

由于 AFL 测试模块通过 C++实现，强化学习环境和算法通过 Python 实现，两者之间无法直接通信，需要通过交互模块对两者进行通信。该模块主要负责启动目标测试程序，传递变异后的输入样本，与 AFL 交互来启动目标程序工作流程并获得其结束的终止状态，然后读取共享内存后获取原始的执行路径等数据，最后将其传递给覆盖率分析模块。

覆盖率分析模块负责对 AFL 返回的目标程序执行原始路径数据做进一步处理，最终得到强化学习所需的奖励。本平台具体使用被插桩的已编译程序用于捕捉分支覆盖率，以及分支命中计数。覆盖率分析模块首先参考 AFL 对原始边跳转次数做规范处理，用于后续工作流程快速判断当前变异样本是否产生新的覆盖路径。其基本思路在于程序的同一跳转边可能被多次访问，如果将跳转边访问次数每一次的变化均视为样本触发了新的执行路径，则会产生大量尝试触发单一跳转边不同次数的变异样本，而不是尝试触发其他跳转边，通常后者对覆盖率的影响更大。

AFL 返回的数据中，64K 大小的 *shared_mem*[] 数组中每个字节都表示一条边的执行次数记录，记 *shared_mem*[]数组的大小为 *MAP_SIZE*。定义该数组的子集 M_t'，它是在执行过程中跳转边至少执行了一次的集合，则 Coverage 会计算 S_t 样本得到的奖励 R_t，如下式所示：

$$R_t = \frac{size(M_t')}{MAP\_SIZE}$$

最后在变异模块的 Python 程序文件中，实现了 3.2 节表 1 中的 9 种变异策略集合。

5 实验验证

本章将对二进制代码模糊测试平台进行实验验证，以检验平台各个模块和整体流程的有效性。首先针对深度强化学习算法模型，设计实验选择模型参数，使模型达到最佳性能；然后针对不同的样本变异方法选择策略进行实验，并分析实验结果以验证深度强化算法的有效性。

5.1 实验数据集构建

本次实验采用 LAVA-M 测试程序集[20]作为实验数据集，首先通过 AFL 对其中 base64 程序进行插桩编译来作为测试程序，通过 afl-as 对 g++或 gcc 编译生成的汇编代码进行插桩，在有跳转的位置插入汇编码，在程序跳转时掌握程序执行路径。插桩完成后，调用 as 将插桩后的汇编代码翻译成机器码。

在训练过程中，针对输入的 ASCII 码进行不同变异，其变异结果质量由其路径发现数量来衡量，并作为反馈奖励。重复上述过程，最终完成对强化学习模型的训练。

5.2 实验过程与结果分析

本次实验对比二进制代码模糊测试平台在固定迭代轮数内以及固定时间内发现独特代码覆盖路径的数目以及对新的跳转边的覆盖数目。作为对比，使用随机策略作为基准测试，随机策略中不存在网络结构，通过 *env.action_space.sample*() 函数从动作空间中进行随机采样来指导样本的变异。本次实验分别使用 DDPG 算法、DQN 算法和随机策略对经过插桩编译的 LAVA-M 的 base64 目标程序进行测试。

对 DDPG、DQN 和随机策略分别从空样本开始模糊测试实验，对相同轮次的边覆盖(对应学习率)变化情况、相同轮次的路径覆盖变化情况、相同时间内的路径覆盖变化情况进行分析。其中边覆盖变化情况是在训练过程中统计得到。路径覆盖变化情况是在测试过程中得到，由于随机策略的速度要远快于 DDPG 和 DQN 的训练速度，因此不对相同时间内 DDPG、DQN 和随机策略的边覆盖变化情况进行对比分析。

5.2.1 相同轮次下变异样本对应边覆盖变化情况分析

由于强化学习环境中奖励的设计与边覆盖数量成正比例关系，边覆盖情况能够直接体现强化学习方法的学习效果，实验采用训练网络模型时的数据，从空样本开始进行 20000 轮变异，统计不为 0 的跳转边的数量。图 3 给出了 DDPG、DQN、随机 3 种策略中的非 0 跳转边数量随测试轮次的变化情况。由于强化学习的学习曲线十分不稳定，因此使用窗口为 13 的平均平滑对图像进行处理。

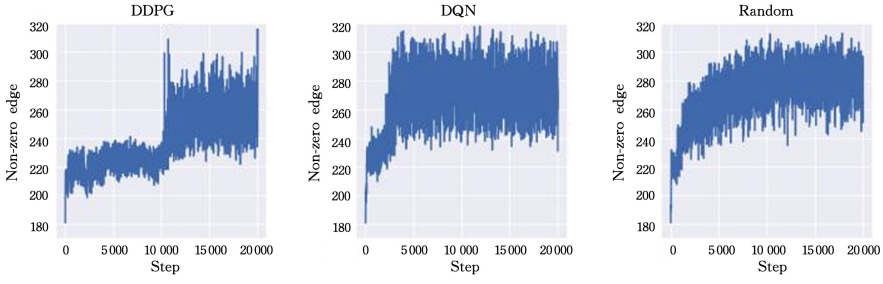

图 3 3 种策略生成 20000 样本的非 0 跳转边变化情况

Fig. 3 Non zero jump edge changes of 20000 samples generated by three strategies

从图 3 可以看出各变异策略的边覆盖数量都在逐渐上升，曲线走势符合强化学习的学习率曲线，说明深度强化学习算法能够从本平台设置的强化学习环境中进行学习，验证了深度强化学习算法的可行性。其中 DDPG、DQN、随机策略发现的跳转边最大数量分别为 342、350、340。由于随机策略中待变异的样本也会向增加不重复跳转边的方向更新，因此随机策略下生成样本的跳转边数量也会增加。

5.2.2 相同轮次下变异样本对应路径覆盖变化情况分析

在强化学习环境中设置的奖励值与变异样本边覆盖情况呈正相关，会引导网络模型向覆盖更多不重复跳转边的方向进行样本变异，但是覆盖更多跳转边不一定表示能够找到更多新路径。在以 AFL 为首的实际模糊测试过程中，系统发现的新路径却不一定会提高边覆盖率，因此还需要对变异样本产生的新路径变化情况进行对比分析。接下来对 3 种策略在相同轮次下从空样本开始变异，生成的变异样本集产生的新路径数量，即路径覆盖进行对比分析。对于 DDPG 和 DQN 策略，使用训练的模型进行测试，实验从空样本开始变异，3 种策略都执行 35 000 次变异，统计发现的新路径数量。实验结果如图 4 所示。

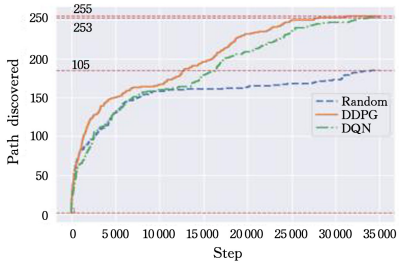

图 4 3 种策略生成 35 000 样本发现的新路径变化情况

Fig. 4 New path changes of 35 000 samples generated by three strategies

图 4 中横坐标为运行轮次，纵坐标为发现的新路径的总数量。在初始阶段，3 种方法发现的路径数量近似，从 2 500 轮左右 DDPG 策略明显优于其他策略，而 DQN 和随机策略效果相近；从 15 000 轮开始随机策略的增长趋向平缓，可能遇到了 LAVA-M 在程序中构造的瓶颈，而 DDPG 和 DQN 则突破了瓶颈，发现的路径数量持续增长，在 35 000 轮后 DQN 和 DDPG 发现的新路径数量相近，分别为 253 和 255，明显多于随机策略发现的路径数量 185，验证了深度强化学习方法对模糊测试过程改进的有效性。

5.2.3 相同时间内变异样本对应路径覆盖变化情况分析

在实际模糊测试过程中，相比在更少的轮次发现更多的路径，更看重在更短的时间内发现更多路径。因此本组实验使用和第二组实验相同训练后的 DDPG 和 DQN 网络模型与随机策略进行测试，仍然从空样本开始变异，3 种策略都运行 1 h，统计发现的新路径数量。实验结果如图 5 所示。图中横坐标为以秒为单位的运行时间，纵坐标为发现的新路径的总数量。在前 200 s 时随机策略的测试轮次最多，发现的路径数量也最多。而在 250 s 左右 3 种策略都遇到了瓶颈，发现的新路径数量增长变缓。DQN 和 DDPG 策略都能突破瓶颈，而随机策略未能突破瓶颈，尽管其测试轮次超过其他两种方法，但最终发现的路径数量却与 DQN 和 DDPG 有较大差距，说明从时间对比上仍能体现深度强化学习方法的有效性。

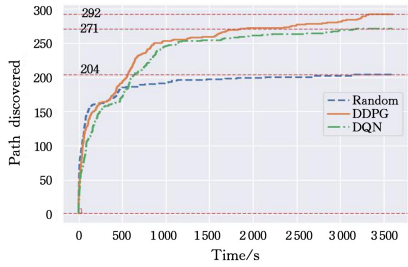

图 5 3 种策略测试 1 h 发现的新路径变化情况

Fig. 5 New path changes discovered by three strategies for test within one hour

3 组实验结果显示深度强化学习模型能够促使模糊测试过程快速覆盖更多路径，有效暴露更多漏洞缺陷，二进制代码漏洞检测性能得到显著提升。

结束语 针对二进制代码模糊测试过程中各环节的抽象性，将测试过程建模为面向强化学习的马尔可夫决策过程，包括强化学习问题的要素。通过深度强化学习模型来改进传统模糊测试过程，能够减少输入样本变异过程中的随机性，提高有效样本的生成概率，从而提高二进制代码模糊测试的样本覆盖效率。3 组对比实验结果显示，所提方法在漏洞挖掘效率上相比 AFL 测试工有明显提高，二进制代码模糊测试工作效率也得到提高，验证了二进制代码模糊测试平台的适用性和有效性。

在未来研究工作中，将针对深度强化学习不同算法在更多程序集进行模型训练和测试，开展更加深入的工程实践与应用，进一步验证方法有效性和合理性，并尝试研究训练模型的可复用性，减少模型在新环境中的训练时间。

参 考 文 献

[1] WU S Z, GUO T, DONG G W, et al. Software vulnerability analysis technology[M]. Beijing: Science Press, 2014.

[2] ZHU X D. Research on Key Issues of Binary Code Similarity Analysis[D]. Zhengzhou: PLA Strategic Support Force Information Engineering University, 2021.

[3] FLAKE H. Structural comparison of executable objects[C]// IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment(DIMVA). 2004.

[4] GAO D, REITER M K, SONG D. BinHunt: Automatically Finding Semantic Differences in Binary Programs[C]// International Conference on Information & Communications Security. 2008.

[5] ZHANG X, LI Z J. Survey of Fuzz Testing Technology [J]. Computer Science, 2016, 43(5): 1-8.

[6] CADAR C, GANESH V, PAWLOWSKI M P, et al. EXE: Automatically generating inputs of death[J]. ACM Transactions on Information and System Security(TISSEC). 2008, 12(2): 1-38.

[7] CADAR C, DUNBAR D, ENGLER R D. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs[C]// Usenix Conference on Operating Systems Design & Implementation. 2009.

[8] SHOSHITAISHVILI Y, KRUEGEL C, VIGNA G. SOK: (State of) The Art of War: Offensive Techniques in Binary Analysis [C]// 2016 IEEE Symposium on Security and Privacy (SP). 2016.

[9] NEWSOME J,SONG D. Dynamic taint analysis for automatic detection,analysis,and signature generation of exploits on commodity software[J]. Chinese Journal of Engineering Mathematics,2012,29(5):720-724.

[10] GODEFROID P,LEVIN M Y,MOLNAR D. SAGE:whitebox fuzzing for security testing[J]. Queue,2012,10(3):40-44.

[11] WANG Y,JIA P,LIU L,et al. A systematic review of fuzzing based on machine learning techniques[J]. arXiv:1908. 01262,2019.

[12] ZHANG Z. Research on Fuzz Testing Technology Based on DDPG Reinforcement Learning Algorithm[D]. Beijing:Beijing University of Posts and Telecommunications,2021.

[13] LV C,JI S,LI Y,et al. SmartSeed:Smart Seed Generation for Efficient Fuzzing[J]. arXiv:1807. 02606,2018.

[14] WU Z,JOHNSON E,YANG W,et al. REINAM:reinforcement learning for input-grammar inference[C]// The 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019.

[15] SHE D,KRISHNA R,YAN L,et al. MTFuzz:Fuzzing with a Multi-task Neural Network[C]// ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering(ESEC/FSE). 2020.

[16] SON S,LEE S,HAN H,et al. Montage:A Neural Network Language Model-Guided JavaScript Engine Fuzzer[C]// 20th USENIX Security Symposium(USENIX Security 2020). 2020

[17] ZONG P,LV T,WANG D,et al. FuzzGuard:Filtering out Unreachable Inputs in Directed Grey-box Fuzzing through Deep Learning[C]// 29th USENIX Security Symposium. 2020.

[18] BOTTINGER K,GODEFROID P,SINGH R. Deep Reinforcement Fuzzing[C]// 2018 IEEE Security and Privacy Workshops. 2018.

[19] DROZD W,WAGNER M D. FuzzerGym:A Competitive Framework for Fuzzing and Learning[J]. arXiv:1807. 07490,2018.

[20] DOLAN-GAVITT B,HULIN P,KIRDA E,et al. Lava:Large-scale automated vulnerability addition[C]// 2016 IEEE Symposium on Security and Privacy. 2016.

WANG Shuanqi, born in 1984, Ph. D, senior engineer. His main research interests include software test verification and vulnerability mining.