

基于深度强化学习的高性能导向性模糊测试方案

肖天¹, 江智昊^{1,2}, 唐鹏¹, 黄征¹, 郭捷¹, 邱卫东¹

(1. 上海交通大学网络空间安全学院, 上海 200240; 2. 哥伦比亚大学, 美国 纽约 10027)

摘要: 随着移动互联网与信息技术的快速发展,越来越多的应用程序融入人们的生活,但这些应用程序中存在的漏洞严重威胁着用户隐私和信息安全。近年来,模糊测试作为流行的漏洞挖掘技术之一,因其漏洞易复现且误报率低的特点而被广泛地使用。它能随机生成测试用例并执行程序,通过覆盖率或样本生成方面的优化以检测更深的程序路径。但是模糊测试中的变异操作存在一定的盲目性,易使生成的测试样本执行相同程序路径。因此传统模糊测试普遍存在挖掘效率低、输入构造的随机性强、算法对程序结构针对性有限等问题。针对上述问题,提出了基于深度强化学习的高性能导向性模糊测试方案,通过程序插桩等方法获取程序运行时的信息,使用深度强化学习网络指导模糊测试选择测试样本,生成有针对性和导向性的测试样本以快速逼近并检验可能存在漏洞的程序路径,从而提高模糊测试的效率。实验表明,在 LAVA-M 测试集与真实应用程序 LibPNG 和 Binutils 上,所提方案比流行模糊测试工具 AFL 与 AFLGO 在漏洞检测与复现等方面有着更好的表现,因此该方案可为今后的漏洞挖掘和安全研究提供支撑。

关键词: 漏洞挖掘; 模糊测试; 深度强化学习; 程序路径

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2023027

High-performance directional fuzzing scheme based on deep reinforcement learning

XIAO Tian¹, JIANG Zhihao^{1,2}, TANG Peng¹, HUANG Zheng¹, GUO Jie¹, QIU Weidong¹

1. School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

2. Columbia University, New York 10027, America

Abstract: With the continuous growth and advancement of the Internet and information technology, continuous growth and advancement of the Internet and information technology. Nevertheless, these applications' vulnerabilities pose a severe threat to information security and users' privacy. Fuzzing was widely used as one of the main tools for automatic vulnerability detection due to its ease of vulnerability recurrence and low false positive errors. It generates test cases randomly and executes the application by optimization in terms of coverage or sample generation to detect

收稿日期: 2022-12-04; 修回日期: 2023-03-02

通信作者: 邱卫东, qiuwd@sjtu.edu.cn

基金项目: 国家自然科学基金(61972249)

Foundation Item: The National Natural Science Foundation of China (61972249)

引用格式: 肖天, 江智昊, 唐鹏, 等. 基于深度强化学习的高性能导向性模糊测试方案[J]. 网络与信息安全学报, 2023, 9(2): 132-142.

Citation Format: XIAO T, JIANG Z H, TANG P, et al. High-performance directional fuzzing scheme based on deep reinforcement learning[J]. Chinese Journal of Network and Information Security, 2023, 9(2): 132-142.

deeper program paths. However, the mutation operation in fuzzing is blind and tends to make the generated test cases execute the same program path. Consequently, traditional fuzzing tests have problems such as low efficiency, high randomness of inputs generation and limited pertinence of the program structure. To address these problems, a directional fuzzing based on deep reinforcement learning was proposed, which used deep reinforcement learning networks with information obtained by staking program to guide the selection of the inputs. Besides, it enabled fast approximation and inspection of the program paths that may exist vulnerabilities. The experimental results showed that the proposed approach had better performance than the popular fuzzing tools such as AFL and AFLGO in terms of vulnerability detection and recurrence on the LAVA-M dataset and real applications like LibPNG and Binutils. Therefore, the approach can provide support for further vulnerability mining and security research.

Keywords: vulnerability mining, fuzzing test, deep reinforcement learning, program path

0 引言

安全漏洞一直是计算机安全领域的核心问题之一。根据美国国家标准与技术研究院的统计数据,2012—2020年收录的漏洞数量整体呈较快上升趋势,2021年更是惊人地达到了21 957个^[1]。根据托管安全服务提供商 Redscan 的分析,在2021年这两万多个漏洞中,54%的漏洞被认为具有“高”可用性,这意味着攻击者可以很容易地利用它们^[2]。

日益严峻的安全形势对漏洞挖掘提出了更高的要求。与其他技术相比,模糊测试所需的知识更少、自动化程度高、易复现、扩展性好,已成为较流行的漏洞挖掘方法之一。

模糊测试通过向目标程序输入使用一定策略生成的测试用例,监测程序的运行情况,记录并分析目标程序发生的异常来发现潜在的安全缺陷^[3]。模糊测试的漏洞挖掘能力主要取决于测试用例的质量和运行效率^[4]。因此,如何在提高检测效率的同时,生成更加有效的测试用例,成为一大挑战。针对这个挑战,模糊测试的主要改进方式有两个:一是致力于提高代码覆盖率;二是通过“预生成”的方式,减少无效测试用例。

提高代码覆盖率的方式是让测试用例的执行路径尽量覆盖整个程序,从而降低漏报率。AFL (American fuzz lop) 是一款基于覆盖率引导的开源灰盒测试工具,帮助测试人员挖掘了大量的高危漏洞,很大程度上推进了模糊测试的研究^[5]。CollaAFL^[6]在 AFL 的基础上解决了哈希碰撞的问题,并改进了种子选取策略,但其提高代码覆盖率的同时,往往在与缺陷无关的路径上耗费大量的计算资源和时间,效率低。

“预生成”的方式则是通过改进生成算法,生成特定类型的测试用例。例如, SemFuzz^[7]可以针对已经公布的漏洞,利用通用漏洞披露 (CVE) 报告和相关日志等文本,自动生成触发漏洞的用例。虽然业界有大量关于优化反馈选择、用例生成的研究,但模糊测试仍然存在着自适应性和针对性有限的问题,因此导向性模糊测试应运而生。

导向性模糊测试又称定向模糊测试,它通过静态分析与特定的算法来指导种子优先度调整、能量调度和适应性变异,从而达到测试特定漏洞的目的。对于如何指导模糊测试生成抵达目标位置输入的问题,本文使用了深度强化学习中的经典算法 deep Q-learning^[8]。深度强化学习的主要特点是能够让系统不断尝试,评估之前采取的行为修正自身的行为选取策略,使得无导向的随机过程转化为有导向的过程。鉴于深度强化学习的这种特性,本文选择将其融入 AFL 的测试用例生成中,将新旧样本间执行路径到目标节点的距离差作为深度强化学习网络 (DQN, deep Q-learning network) 的反馈值,促使测试用例的执行路径逼近可能存在缺陷的路径。实验表明,深度强化学习优化了模糊测试的测试用例生成与选择策略,让模糊测试有了更好的针对性和导向性。

鉴于以上特点,本文提出的模糊测试方案可主要应用于以下几个方面。

(1) 版本更新测试

测试人员可以使用本文提出的模糊测试方案对与版本更新相关的代码进行测试,在节省开销的同时,更有效地检测当前版本的补丁。

(2) 复现漏洞

在可访问程序源代码但没有崩溃输入的情况

下,所提方案可以根据漏洞报告中的源代码信息,快速达到漏洞位置,构造出可以触发漏洞的测试用例。

(3) 基于静态分析的模糊测试

因静态分析通常存在误报率高的问题,所以往往需要一个验证漏洞的手段。静态分析提供了可能存在漏洞的位置,而所提方案可将其作为目标进行验证。

1 相关工作

1.1 灰盒模糊测试

模糊测试可以分为黑盒模糊测试、白盒模糊

测试和灰盒模糊测试^[9]。灰盒模糊测试介于黑盒模糊测试和白盒模糊测试之间,在构造输入时,会参考源代码信息和软件运行时的反馈信息。

灰盒模糊测试主要基于反馈获取机制、反馈处理机制、样本生成机制^[10]。反馈获取机制能够在测试过程中获取测试目标的反馈;反馈处理机制以反馈信息为参考,从变异样本中筛选出优质样本组成下一轮测试的语料库;样本生成机制则是对语料库中的样本实施变异以获得新样本,为当前轮次的测试提供输入^[11]。AFL 是较为流行的基于覆盖率的灰盒模糊测试工具之一,其工作流程如图1所示。

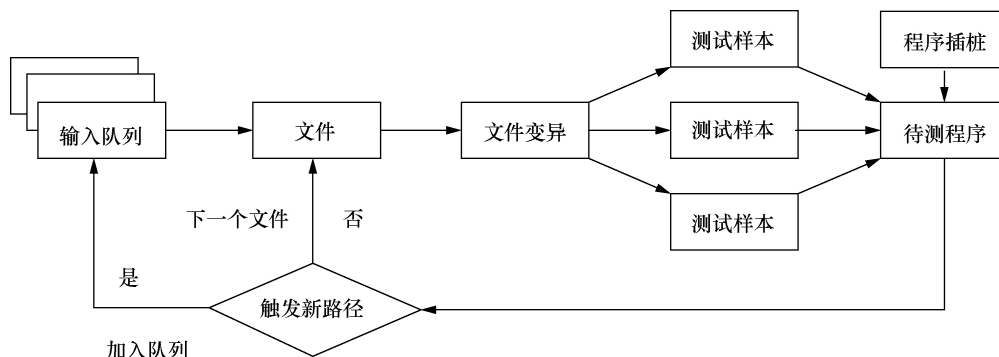


图1 AFL 工作流程

Figure 1 The work process of AFL

AFL 首先会通过修改编译器策略或使用脚本进行程序插桩,用于在测试过程中获得代码覆盖率位图;然后从输入文件中选取一些测试样本添加到测试队列中,根据它们的标签及其对应的概率决定是否选择该样本进行测试;接着对当前测试样本按概率进行变异以生成新的样本,并输入待测程序中。如果变异后的测试用例更新了代码覆盖率位图,则将其添加到输入队列中。一轮测试结束后,需要根据情况对本轮测试使用到的测试样本打上标签,实现反馈处理;最后不断重复样本测试与样本变异过程,直到用户停止 AFL 程序。所有触发了崩溃的测试样本将会被 AFL 保存,供用户进行分析。

在具体的反馈处理过程中,AFL 会先根据代码覆盖率位图计算出当前测试样本的得分并分析其标签,再根据对应的概率决定是否选用该测试文件。AFL 的这种反馈处理机制使得它只具有有限的自适应性。如果能够使用非线性的函数根据反馈获取机制拟合出更加符合当前模糊测试状态

的概率,就能够提高灰盒模糊测试的性能。

强化学习是一个更好的选择,其作为一种解次序贯决策问题的机器学习模型,可以不断和模糊测试环境交互,最后得到模糊测试环境下的最优解。

此外,AFL 对于不同输入类型、不同结构程序的针对性和自适应性很有限。如果通过加强反馈获取机制,根据源代码和汇编代码获取更多关于程序的信息,就可以更加有效地加强灰盒模糊测试的针对性和自适应性,使得灰盒模糊测试更快地达到目标节点,从而验证该路径上是否存在漏洞。

1.2 导向性模糊测试

AFLGO^[12]是一款基于 AFL 开发的定向灰盒模糊测试工具,它对 AFL 的改进主要在于对测试样本变异策略的优化,它采用基于模拟退火的调度算法来最小化测试样本到目标的距离。测试过程中会为距离目标代码行号更近的测试样本赋予更多的能量,使得 AFLGO 可以比 AFL 更快地执行到目标代码行。同时,AFLGO 将分析工作集成到编译过程中,降低了运行时的开销。

在与定向白盒模糊测试 KATCH^[13]的补丁测试对比实验中, AFLGO 额外发现了 7 个 CVE 漏洞, 并可以在相同的时间内多覆盖 13% 的目标。在与定向白盒模糊测试 BugRedux^[14]的崩溃重现对比实验中, 当只有堆栈跟踪中的方法可用时, AFLGO 复现出了 3 倍于 BugRedux 的方法调用。

实验表明, AFLGO 在效率方面优于定向白盒模糊测试和无定向的灰盒模糊测试, 证明了提高导向性和针对性是一种相当有效的模糊测试优化方法。

与 AFLGO 不同的是, Chen 等^[15]提出了相似度的概念。在模糊测试工具 Hawkeye 中, 根据测量种子的执行轨迹和目标执行轨迹之间的相似度, 调整测试样本的优先级和功率调度使得模糊测试逼近目标。因为覆盖“目标路径”中更多节点的测试样本将会更有可能通过变异到达目标节点。

业界有部分工作使用深度学习进行定向模糊测试。例如, FuzzGuard^[16]使用基于深度学习的方法, 可以在执行目标程序之前预测输入的可达性, 过滤掉不可达的输入。同样地, DeFuzz^[17]采用双向长短期记忆 (LSTM) 网络来识别注意力词, 以识别潜在的易受攻击的功能和位置, 并使用 AFLGO 生成倾向于到达预测位置的输入来模糊潜在的漏洞。但是很少有研究将强化学习应用到导向性模糊测试中。

1.3 强化学习

强化学习^[18]的主要特点是在交互过程中, 可以根据环境反馈的奖励, 修正网络的行为选取策略, 不断优化动作选择策略, 最终取得最大的累计奖励。

强化学习问题一般通过马尔可夫决策过程 (MDP, Markov decision process) 来建模^[19]。MDP 通常由四元组 (S, P, R, γ) 来描述。

- 1) S 为所有环境状态 s (state) 的集合, $s \in S$ 。另有 A 表示有限的动作 a (action) 的集合, $a \in A$ 。
- 2) P (probability) 为状态转移概率, $P(s' | s, a)$ 表示在状态 s 下执行动作 a , 转移到状态 s' 的概率。
- 3) R (reward) 为奖励函数, $R_s = E[R_{t+1} | S_t = S]$ 指的是时刻 t 在状态 s 下, 下一时刻 $t+1$ 能够获得的奖励值的期望。

4) 折扣因子 γ , 用于计算累积奖励, $0 \leq \gamma \leq 1$ 。强化学习的最终目的是采取某种策略, 以获得最多的累积奖励。策略由一系列的动作组成。通常使用值函数来估计某个策略 (下文用 π 表示) 的优劣程度。当主体采用策略 π 时, 累积奖励在状态 s 的期望定义为状态-奖励值函数。

强化学习的工作流程如图 2 所示。

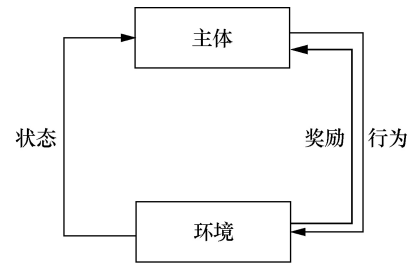


图2 强化学习的工作流程
Figure 2 Working process of reinforcement learning

在与环境的互动中, 主体根据强化学习网络选择某一行, 在该行为的作用下, 环境返回状态信息和奖励, 强化学习网络根据这些信息更新权重。训练时将不断重复上述过程, 最终更新强化学习网络至当前环境下能获得的最大累计奖励。

深度强化学习将深度学习的智能感知能力与强化学习的决策能力结合, 通过对高维感知输入的学习, 实现了智能体的自主行为控制^[20]。因此深度强化学习常被用于游戏、围棋、自动化控制等场景的智能任务。2013 年, DeepMind 发布了深度强化学习网络^[21-22], 把强化学习算法 Q-learning 和深度学习网络相结合, 其性能远超强化学习网络, 并因 AlphaGo^[23]战胜围棋世界冠军而广为人知。

其中, Q-learning^[24]算法如算法 1 所示。

算法 1 Q-learning 算法

- 输入 状态信息集合 S , 动作集合 $A(s)$, 折扣因子 γ , 学习率 α
- 输出 最终策略 $\pi(a)$
- 1) 随机初始化 $Q(s, a), \forall s \in S, a \in A(s)$
 - 2) 每轮迭代进行以下操作:
 - 3) 初始化 $s \in S$
 - 4) 重复以下操作直至状态 s 达到目标状态:

- 5) 使用 ε -greedy 策略根据 $Q(s, a)$ 从 $A(s)$ 中挑选出动作 a
- 6) 执行动作 a , 获得奖励 r 与状态 s'
- 7) 更新 $Q(s, a)$:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \{r + \gamma \cdot \max_{a'} [Q(s', a') - Q(s, a)]\}$$
- 8) 更新策略 $\pi(a) = \arg\max Q(s, a)$,
 $a \in A(s)$
- 9) 更新 $s = s'$

算法 1 中的 ε -greedy 是一种随机策略, 在各种强化学习网络中都有着非常广泛的运用。

如式 (1) 所示, ε -greedy 策略会以 $1 - \varepsilon + \varepsilon / |A(s)|$ 的概率执行当前状态 s 下, 令动作值函数 $Q(s, a)$ 最大的动作 a , 以 $\varepsilon / |A(s)|$ 的概率执行其他的动作, 其中 ε 是一个常数, $|A(s)|$ 代表动作的个数。这样可以使得所有的动作都有机会被访问到, 解决了收敛于局部最优的问题。

$$\pi(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s)|}, & a = \arg\max Q(s, a), a \in A \\ \frac{\varepsilon}{|A(s)|}, & \text{其他} \end{cases} \quad (1)$$

DQN 在 Q-Learning 的基础上进行了一些改动, 其将神经网络应用到动作值函数的逼近过程中, 并借鉴了人脑的记忆方式, 引入了经验回放^[25], 解决了实验数据之间的关联性问题, 使得网络具备了更好的收敛能力, 如算法 2 所示。

算法 2 DQN

输入 状态信息集合 S , 动作集合 $A(s)$, 奖励值集合 R , 折扣因子 γ , 权重 θ 和 θ'

输出 网络 Q

- 1) 初始化回放经验 D
- 2) 根据随机权重 θ 初始化网络 Q
- 3) 根据随机权重 θ' 初始化目标网络 Q'
- 4) 每轮迭代进行以下操作:
- 5) 以概率 ε 随机选择动作 a_t , 以 $1 - \varepsilon$ 的概率选择 $a_t = \arg\max Q(s_t, a_t, \theta)$, $s_t \in S, a_t \in A(s)$
- 6) 执行动作 a_t , 获得奖励 r_t 以及状态 s_{t+1} , $s_{t+1} \in S, r_t \in R$
- 7) 在 D 中存储 (s_t, a_t, r_t, s_{t+1})

- 8) 在 D 中随机选择样本 (s_j, a_j, r_j, s_{j+1})
- 9) 最小化损失:

$$[r_j + \gamma \cdot \max_{a_{j+1}} Q'(s_{j+1}, a_{j+1}; \theta') - Q(s_j, a_j; \theta)]$$
- 10) 每隔若干轮就重置 $Q' = Q$

2 本文方案

本文提出的基于深度强化学习的高性能导向性模糊测试方案以 AFL 作为模糊测试的基本框架, 使用 DQN 指导模糊测试选取测试样本, 从而能够逼近目标节点。其工作流程如图 3 所示。

整个方案围绕 DQN 进行构建, 其中交互主体为 AFL, 动作集合为测试样本的选择, 状态集合为程序反馈的各种信息。所提方案的运行流程如下。

步骤 1 首先利用 LLVM (low level virtual machine)^[26]编译工具包修改编译器工作流程, 在汇编代码中插入监视代码, 以获取执行过程中的基本块信息。再利用 LLVM 生成程序的控制流图和函数调用图, 分析获得基本块序列。

步骤 2 运行一个测试样本之后, 根据基本块序列定位当前执行路径和目标位置, 计算执行路径和目标节点之间的距离。最后把与之前原始测试样本的距离差作为深度强化学习网络的奖励输入网络中。如果当前路径有新的基本块出现, 则会有额外的奖励, 以促进代码覆盖率的提升。

步骤 3 深度强化学习的反馈信息将会指导模糊测试的样本生成与选择。继而生成一批新的测试样本。

重复步骤 2 与步骤 3 直至模糊测试结束。

为了解决数据交换问题, 所提方案使用共享内存的方式进行进程间的数据交互。

2.1 运行信息收集

主要使用在编译过程中进行程序插桩的方式, 依靠 LLVM 工具集实现信息收集。LLVM 是一个跨平台的编译器框架, 提供了基础的编译器设施和一致的中间语言, 并允许用户构建自己的编译方法, 具有优秀的跨平台属性和易调试的特性。所提方案的插桩功能在 LLVM 中间语言的基础上, 通过 LLVM 优化器和自定义的编译器获得程序执行时的基本块序列, 并生成函数调用图和控制流图, 其示例如图 4 所示。

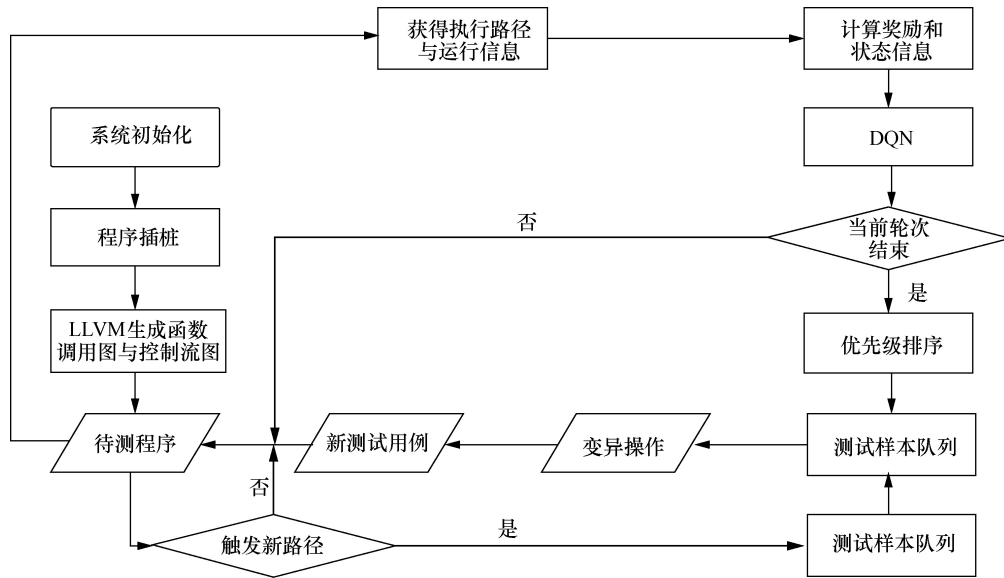


图3 本文方案的工作流程
Figure 3 The proposed scheme work process

LLVM 提供了函数 `llvm: Module Summary Index Wrapper Pass: runOnModule()`，可以在中间语言的每个模块中插入自定义代码。而函数 `BB.getName()` 和 `F.getName()` 可以返回当前执行的基本块和执行的函数名。因此，只需要通过 `runOnModule()` 在每个基本块的头部插入这两个函数，就可以得到所有已执行过的基本块的函数名和基本块名序列。同时，可以利用 LLVM PASS

提供的工具 `dot-callgraph` 和 `dot-cfg`^[27] 来获取程序源代码的函数调用图和控制流图。

2.2 DQN 奖励计算

DQN 训练时的奖励值，主要根据执行路径和目标节点之间的距离计算得来。在获得当前执行路径之后，需要计算执行路径和用户输入的目标节点所在基本块之间的距离。定义程序中的一个函数 n 到另一个函数 n' 的距离 $d_f(n, n')$ 为 n 到 n'

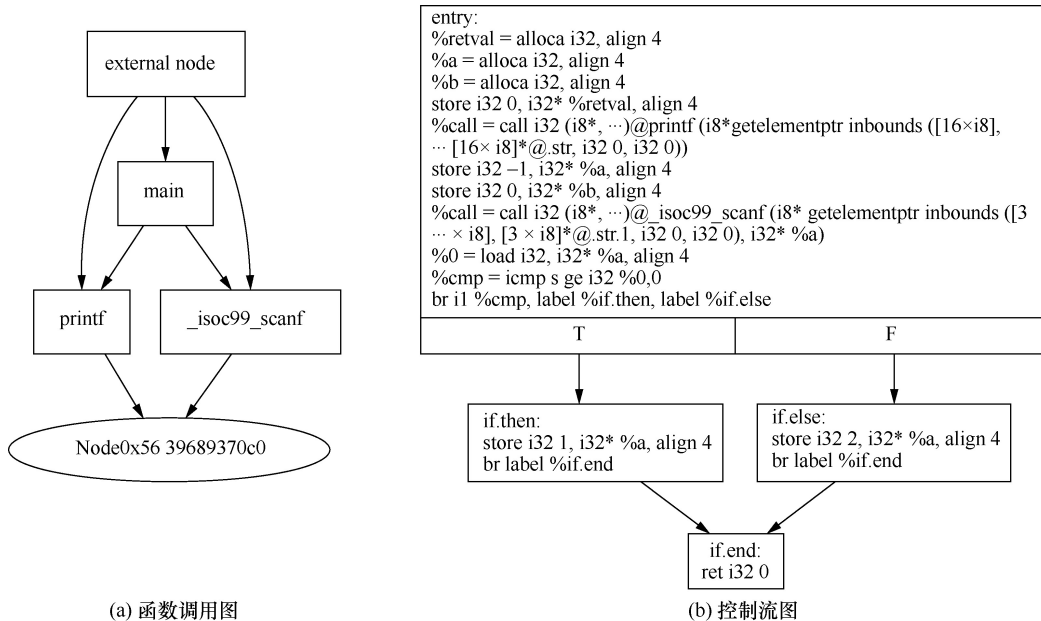


图4 函数调用图和控制流图示例
Figure 4 Sample of call graph and control flow graph

的最小边数。该函数 n 到目标函数集合 T_f 的距离 $d_f(n, n')$ 如式(2)所示。

$$d_f(n, T_f) = \begin{cases} \text{undefined}, R(n, T_f) = \emptyset \\ \left[\sum_{t_f \in R(n, T_f)} d_f(n, t_f) \right]^{-1} \end{cases} \quad (2)$$

其中, T_f 表示所有的目标函数, 而函数调用图是有向图, 对于不可达到 T_f 的一些函数, 定义为距离不存在 (undefined), 在计算时不考虑, $R(n, T_f)$ 表示所有函数 n 可达到 T_f 中函数的集合。由式(2)可以看出, 函数 n 到目标函数集和的距离是函数 n 到每个可达函数的距离的调和平均。

定义程序中的一个基本块 m 到另一个基本块 m' 的距离 $d_b(m, m')$ 为 m 到 m' 的最小边数。基本块 m 到目标基本块集合 T_b 的距离 $d_b(m, T_b)$ 如式(3)所示。

$$d_b(m, T_b) = \begin{cases} 0, m \in T_b \\ \text{undefined, 不可达到} \\ c \cdot \min(d_f(n, T_f)), n \in N(m), m \in T \\ \left\{ \sum_{t \in T} [d_b(m, t) + d_b(t, T_b)]^{-1} \right\}^{-1}, \text{其他} \end{cases} \quad (3)$$

其中, T 表示一个基本块集合, 该集合中的基本块所调用的函数都可达到目标函数。如果基本块 m 在 T 中, 则将 m 中到目标函数的最小距离的 c 倍为基本块到目标函数的距离, 其中 c 是一个常数; 如果基本块 m 不在 T 中, 则计算 m 到目标基本块的所有路径中, 经过 T 中基本块的路径, 并取调和平均数。同样地, 对于不可达到目标基本块的基本块, 在计算距离时不会被考虑进来。

根据上述计算方式, 可以得到路径上所有基本块到目标基本块的加权平均距离。深度强化学习网络将尽可能缩小这个平均距离。如果最后距离缩小到 0, 就意味着此次测试成功生成了达到目标节点的输入, 此后将会持续生成在路径上或附近的输入并进行测试。

值得注意的是, 深度强化学习网络的奖励包括代码覆盖率, 如果在测试过程中发现了新的代码块, 还会给予额外的奖励。新代码块的发现数目如式(4)所示。

$$E(s) = |\xi_b(s) - [\bigcup_{s' \in S'} \xi_b(s')]| \quad (4)$$

其中, S' 表示之前的所有测试样本, $\xi_b(s)$ 则表示测试样本 s 执行路径的节点集。式(4)表示新测试样本的执行路径的节点集与之前所有测试样本执行路径组成的节点集的差。

实际的奖励值使用联合奖励的形式, 其计算如式(5)所示。

$$R = \alpha r_1 + \beta r_2 \quad (5)$$

其中, α 和 β 为乘法因子, 两者之和为 1。 r_1 和 r_2 分别是根据路径距离和发现的新代码块数目计算所得到的奖励值, 并且这两个奖励值都进行了归一化的操作。联合奖励 R 的引入使得奖励在程序的运行过程中能够动态地进行调整。

通过这个特性, 本文还提出了以下优化方案: 在测试样本的执行路径未达到目标节点时, 即在模糊测试初期, 将增大以路径距离为导向的奖励值的权重, 提高对位置路径的探索效率; 在种子的执行路径经过目标节点后, 增大以代码覆盖率为导向的奖励值的权重, 着重在目标节点周围进行测试。

2.3 模糊测试样本选择指导

DQN 的输出为不同选择对应的状态-动作值, 最大值对应的动作为本次变异的最优动作。为了减少网络收敛于局部最优的情况, 本文选用 ϵ -greedy 贪婪策略。

可选择的行为为是否放弃当前测试样本, 并据此维护测试样本队列。在维护测试样本的优先级方面, 本文提出的测试方案使用 3 层优先级: 将 DQN 选择保留的测试样本放在最高优先级; 判断为放弃的新的测试样本放在第二优先级; 其余的测试样本则放在最低优先级。这样的选择基于以下原因: 更好的测试样本以及新产生的测试样本有更大的可能探索到新的基本块或达到目标节点。对于经变异生成的新测试样本, 本文提出的测试方案还改进了 AFL 的循环桶方法^[5]来过滤掉在循环迭代方面没有带来新覆盖以及没有更接近目标节点的测试样本。因此, 不会有太多测试样本在队列中。

对于一个待测样本, 在其运行结束后, 系统将调用奖励计算模块进行初步的筛选。若判断为保留, 则将代码覆盖率函数位图大小 `avg_bitmap_size`、目标节点到达标记 `done`, 以及与目标节点

的接近程度 (coincidence) 作为其状态信息记录下来, 然后将其添加到队列中留给 DQN, 在当前测试轮次结束后进行进一步的选择。

接近程度的定义如式(6)所示。

$$\text{coincidence} = |T_c \cap T_s| \quad (6)$$

其中, 基本块集合 T_s 是当前测试样本的执行路径上的基本块。而基本块集合 T_c 则是函数调用图中, 可达到目标函数 T_f 的函数所在的基本块集合, 其中包括目标基本块。 T_c 如式(7)所示。

$$T_c = \{m | \exists n, R(n, T_f) \neq \emptyset, n \in m\} \quad (7)$$

其中, m 是一个基本块, n 是 m 中的一个函数, 这里使用函数 $R(n, T_f)$ 来判断函数 n 和 T_f 的可达性。

值得注意的是, 为了尽量减少对 AFL 效率的影响, DQN 仅在每一轮模糊测试结束后才会进行一次学习, 而在每一轮测试中, AFL 会反馈信息供给 DQN 训练。

此外, 为提高效率, 本文还提出以下结论: 对于同一测试样本, 如果连续 N 次变异后仍未达到目标节点, 则强制放弃此样本。连续变异结束 M 个测试样本后若仍未出现一个达到目标节点的测试样本, 则认为目标不可达并终止模糊测试。

2.4 模糊测试样本生成指导

对于经过目标节点的测试样本, 在其之上施加的变异操作需要一定的调节, 使之更偏向于细微的操作, 并利用标志位禁用 AFL 中原有的确定性操作, 如算法 3 所示。

算法 3 变异调节

输入 需要进行变异的测试样本 s , 需生成的新测试用例的数量 RT , 细粒度变异算子集合 FRO , 其余变异算子集合 CRO , 比例 δ

输出 新测试样本集合 S'

1) 初始化 $S' = \emptyset$

2) 对于每一个需要变异的样本 s :

3) 选择变异算子:

$RO = \text{Choose}(FRO, \lceil RT \times \delta \rceil)$

4) 对于 RO 中的每一个变异操作 ro :

5) 变异生成测试样本: $S' = S' \cup$

$\text{Mutate}(s, ro)$

6) 选择变异算子:

$RO' = \text{Choose}(CRO, RT - \lceil RT \times \delta \rceil)$

7) 对于 RO' 中的每一个变异操作 ro' :

8) 变异生成测试样本: $S' = S' \cup$

$\text{Mutate}(s, ro')$

在主要的变异过程中, AFL 将首先决定此时需要生成的新测试用例的数量, 这里用 RT 表示。每一次, AFL 都会按照均匀分布的方式选择 (Choose) 一系列的变异算子 ro , 并将它们应用 (Mutate) 到种子上, 生成一个测试用例。在经过调节后的变异过程中, AFL 将会把更多机会给予细粒度的变异算子, 即随机比特或字节翻转以及加/减运算, 从而生成变动较小的测试样本。

3 实验测试

3.1 测试集

LAVA (large-scale automated vulnerability addition) 是在程序中插入漏洞和故障的技术^[28]。LAVA 技术被广泛地使用于构造各类模糊测试的效果评估测试集。LAVA-M 数据集是通过 LAVA 技术在 `uniq`、`who`、`md5sum`、`base64` 这 4 个程序上插入故障而得到的数据集, 是在模糊测试领域被广泛运用的一种效果评估测试集。

除此之外, 测试中还选择了两个真实程序进行崩溃复现: GNU Binutils^[27] 是一个在 Linux 平台上的二进制分析工具集合; LibPNG 是一个比较底层的读写 PNG 文件的图像库, 有近 50 万行的代码^[29]。

3.2 测试说明

本文实验使用 Intel Xeon CPU E5-260 的处理器和 16 GB 的内存。系统环境为 64-bit Ubuntu 16.04 LTS system。

同样作为导向性模糊测试, AFLGO 的主要创新点在于优化测试样本的选取策略, 所以本文将 AFLGO 作为对比的主要实验对象。而 AFLGO 的实现方法与本文提出的模糊测试方案的主要不同之处在于: AFLGO 的定向目标为源代码的行号, 所提方案的定向目标为可能存在漏洞的路径, 因此距离计算方法也有所不同; AFLGO 的测试样本选择策略基于模拟退火算法, 所提方案的测试样本选择策略基于深度强化学习网络。

3.3 LAVA-M 测试

此次测试将对比 AFL、AFLGO 和本文方案在 LAVA-M 中的表现, 在给定目标节点的情况

下, 测试各个工具检测漏洞的效率。测试将重复 10 次, 每次测试持续 24 h, 本文使用覆盖次数、平均发现时长和性能增益作为主要指标。覆盖次数统计覆盖到目标节点的唯一崩溃数, 而唯一崩溃是指触发这个崩溃的路径中出现了新的节点。平均发现时长则表示覆盖到目标节点花费的平均时间。性能增益则计算 AFL 或者 AFLGO 的平均发现时长与本文方案的平均发现时长的商。

表 1 为各模糊测试工具在 LAVA-M 数据集上的测试结果。

表 1 LAVA-M 数据集上的测试结果				
Table 1 Experiment results on the LAVA-M dataset				
测试对象	工具	覆盖次数	平均发现时长/s	性能增益
who	AFL	3	68 130	3.89
	AFLGO	4	30 629	1.75
	本文方案	6	17 542	—
base64	AFL	5	3 957	4.74
	AFLGO	5	1 571	1.88
	本文方案	5	834	—
md5sum	AFL	1	67 152	1.79
	AFLGO	4	42 969	1.15
	本文方案	5	37 578	—
uniq	AFL	5	50 183	2.87
	AFLGO	7	17 466	1.22
	本文方案	7	14 323	—

从表 1 可以看出, 本文方案对目标站点覆盖的速度最快。与没有导向性的模糊测试器 AFL 相比, 到达目标站点的速度平均 (即性能增益的平均) 提高了 3.32 倍, 与导向式模糊测试器 AFLGO 相比, 到达目标站点的速度平均提高了 1.5 倍。而在覆盖次数方面, 有导向性的 AFLGO 与本文方案均表现得比 AFL 更好, 证明了导向性模糊测试在漏洞复现方面的优势。而测试结果相差不大则是因为能够引发目标节点崩溃的不同输入是有限的, 同时程序规模不是很大, 因此在 24 h 内 AFL 和 AFLGO 都有很大可能获得绝大部分的有效崩溃输入。

3.4 CVE 漏洞测试

为了反映真实情况, 此次测试从 LibPNG 和 Binutils 的漏洞报告中选择了 8 个可复现的 CVE 漏洞, 并进行 10 次重复试验求取平均结果。这些漏洞可由 CVE 编号识别。

表 2 列出了测试中需复现的漏洞。

表 2 测试中需要复现的漏洞		
Table 2 Vulnerabilities for crash reproduction		
程序	CVE 编号	漏洞类型
LibPNG	CVE-2011-2501	缓冲区溢出
LibPNG	CVE-2011-3328	除数为零
Binutil ¹	CVE-2016-4487	无效写入
Binutils	CVE-2016-4488	无效写入
Binutils	CVE-2016-4489	无效写入
Binutils	CVE-2016-4491	非法访问
Binutils	CVE-2016-4492	堆栈错误
Binutils	CVE-2016-6131	非法访问

表 3 为在真实漏洞复现方面各模糊测试工具的实验结果。其中, 评估模糊测试工具性能的主要指标为平均发现时长、性能增益以及 P 值 (当原假设为真时, 出现比所得样本观察结果更极端的结果的概率)。 P 值可以衡量样本数据与一个给定统计模型之间不相符合的程度, 在多次实验中一般认为 P 值小于 0.05 时, 因偶然而产生意外情况的概率较小, 两者之间的差异是显著而稳定的。表 3 中括号里的数字代表了本模糊测试方案相比对应的工具获得的性能增益。因 AFLGO 与本模糊测试方案同为导向性模糊测试, 所以为了展示本文方案的优异性能, 结果中给出本文方案与 AFLGO 之间的 P 值。

表 3 漏洞复现的测试结果				
Table 3 Experiment results on the crash reproduction				
CVE	AFL	AFLGO	本文方案	P 值
CVE-2016-4487	830(2.36)	512(1.46)	351	0.0215 6
CVE-2016-4488	1671(3.47)	901(1.87)	482	0.0180 4
CVE-2016-4489	1328(3.49)	667(1.76)	380	0.0190 6
CVE-2016-4491	31760(2.99)	27983(2.63)	10633	0.0002 8
CVE-2016-4492	948(2.46)	640(1.66)	385	0.0280 4
CVE-2016-6131	33 895(3.45)	21280(2.17)	9821	0.0009 8
CVE-2011-2501	2 162(4.55)	672(1.41)	475	0.0270 4
CVE-2011-3328	12 736(6.77)	3165(1.68)	1880	0.012

从 LibPNG 和 Binutils 的对比测试结果中可以看到, 在上述几款漏洞的复现上, 本文方案的表现较好, 比 AFL 发现目标漏洞的速度平均提升了 3.69 倍, 相比 AFLGO 则提升了 1.83 倍。这表明本文方案具有更强的自适应性和针对性, 在用户

给定目标节点的情况下, 能够以更高的效率完成模糊测试。同时与 LAVA-M 数据集上的测试结果相比, 本文方案相较于 AFL 与 AFLGO 有了更高的性能增益, 这显示了本文方案在复杂的真实环境中可以获得更加突出的性能。

其中, 对于漏洞 CVE-2016-4491 和 CVE-2016-6131 来说, 本文方案相较于 AFL 与 AFLGO 展现出了更加明显的优势。从平均发现时长可以看出, 这两个漏洞都比较复杂, 很难发现, 需要更长的时间去探索, 反映出本文方案通过深度强化学习, 能够更加有效地选取测试样本, 使得模糊测试过程更加具有针对性且效率更高。

对于漏洞 CVE-2011-2501, 由于 AFLGO 对 AFL 的提升幅度较大, 留给本文方案继续提升的空间不大。同时本文方案与 AFLGO 都能在较短的时间内发现漏洞, 此时模糊测试表现的随机性反而成为影响平均发现时长的最大因素。

对于漏洞 CVE-2016-4487 和 CVE-2016-4492, 因为这些漏洞都相对容易发现, 所有工具都在很短时间内复现了漏洞。而由于导向性模糊测试在测试样本选取策略上引入的优化机制, 加大了模糊测试的计算力消耗, 所以并不能体现出明显的优势。

此外, 与 AFLGO 相比较而得出的 P 值均在 0.05 以下, 也显示出了本文方案有较强的稳定性。

综上, 通过在 LAVA-M 数据集和真实漏洞上的对比实验, 表明了本文提出的模糊测试方法在给定目标节点的情况下, 可以自适应地选取能够达到或更接近目标节点的测试样本, 在选取测试样本的策略上更加具有针对性, 能够提高在已知目标节点的情况下模糊测试的性能。但是其漏洞挖掘能力会受到设定目标的限制。在时间较短、目标程序较简单时, 不能够完全发挥本文方案优越的性能。

4 结束语

本文从软件漏洞安全的现状出发, 讨论了当前安全形势对漏洞挖掘工具的全新要求。根据这些全新的要求, 对模糊测试研究现状进行分析, 提出了灰盒模糊测试的测试样本选择策略和变异方法自适应性有限和针对性有限的问题。本文介绍了 AFL 灰盒模糊测试框架及其核心算法, 提出

了一种将深度强化学习和灰盒模糊测试结合的定向灰盒模糊测试方法, 通过学习网络来优化 AFL 的反馈获取机制和反馈处理机制, 指导模糊测试对测试样本的选择, 此来提高灰盒模糊测试的自适应性和针对性。

在测试中, 本文方案在导向性和针对性方面有一定的提升。但与此同时, 本文方案的性能依然受限于输入的目标节点。虽然目前的设计中, 可以通过手工测试等方法来获取, 但如果能够与其他分析软件集成, 来快速得到可靠的可能存在漏洞的路径, 将会更好地发挥其本文方案的挖掘能力。

参考文献:

- [1] Cybersecurity & Infrastructure Security Agency. known exploited vulnerabilities catalog[R]. 2023.
- [2] REDSCAN. 2021 has officially been a record-breaking year for vulnerabilities[R]. 2021.
- [3] 张雄, 李舟军. 模糊测试技术研究综述[J]. 计算机科学, 2016, 43(5): 1-8, 26.
ZHANG X, LI Z J. Survey of fuzz testing technology[J]. Computer Science, 2016, 43(5): 1-8, 26.
- [4] 邹燕燕, 邹维, 尹嘉伟, 等. 变异策略感知的并行模糊测试研究[J]. 信息安全学报, 2020, 5(5): 1-16.
ZOU Y Y, ZOU W, YIN J W, et al. Research on mutator strategy-aware parallel fuzzing[J]. Journal of Cyber Security, 2020, 5(5): 1-16.
- [5] FIORALDI A, MAIER D, EIBFELDT H, et al. AFL++ combining incremental steps of fuzzing research[C]//Proceedings of the 14th USENIX Conference on Offensive Technologies. 2020: 10.
- [6] GAN S T, ZHANG C, QIN X J, et al. CollAFL: path sensitive fuzzing[C]//Proceedings of 2018 IEEE Symposium on Security and Privacy (SP). 2018: 679-696.
- [7] YOU W, ZONG P Y, CHEN K, et al. SemFuzz: semantics-based automatic generation of proof-of-concept exploits[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2139-2154.
- [8] ARULKUMARAN K, DEISENROTH M P, BRUNDAGE M, et al. Deep reinforcement learning: a brief survey[J]. IEEE Signal Processing Magazine, 2017, 34(6): 26-38.
- [9] BOEHME M, CADAR C, ROYCHOUDHURY A. Fuzzing: challenges and reflections[J]. IEEE Software, 2020, 38(3): 79-86.
- [10] HERRERA A, GUNADI H, HAYES L, et al. Corpus distillation for effective fuzzing: a comparative evaluation[J]. arXiv preprint arXiv:1905.13055, 2019.
- [11] DONG H, DING Z H, ZHANG S H. Deep reinforcement learning: fundamentals, research and applications[M]. Singapore: Springer Singapore, 2020.
- [12] BÖHME M, PHAM V T, NGUYEN M D, et al. Directed greybox

- fuzzing[C]//Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017: 2329-2344.
- [13] MARINESCU P D, CADAR C. KATCH: high-coverage testing of software patches[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. 2013: 235-245.
- [14] JIN W, ORSO A. BugRedux: reproducing field failures for in-house debugging[C]//Proceedings of 2012 34th International Conference on Software Engineering (ICSE). 2012: 474-484.
- [15] CHEN H X, XUE Y X, LI Y K, et al. Hawkeye: towards a desired directed grey-box fuzzer[C]//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018: 2095-2108.
- [16] ZONG P Y, LYU T, WANG D W, et al. FuzzGuard: filtering out unreachable inputs in directed grey-box fuzzing through deep learning[C]//Proceedings of the 29th USENIX Conference on Security Symposium. 2020: 2255-2269.
- [17] ZHU X, LIU S, LI X, et al. DeFuzz: deep learning guided directed fuzzing[J]. arXiv Preprint arXiv: 2010.12149, 2020.
- [18] SUTTON R S, BARTO A G. Reinforcement learning: an introduction[M]. Cambridge, Mass: MIT Press, 1998.
- [19] CHAUHAN N, CHOUDHARY N, GEORGE K. A comparison of reinforcement learning based approaches to appliance scheduling[C]//Proceedings of 2016 2nd International Conference on Contemporary Computing and Informatics (IC3I). 2017: 253-258.
- [20] ZHAO T T, KONG L, HAN Y J, et al. Review of model-based reinforcement learning[J]. Journal of Frontiers of Computer Science and Technology, 2020,14(6): 918-927.
- [21] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing atari with deep reinforcement learning.[C]//Proceedings of Workshops at the 26th Neural Information Processing Systems 2013. 2013: 201-220.
- [22] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533.
- [23] SILVER D, HUANG A, MADDISON C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489.
- [24] KRÖSE B J A. Learning from delayed rewards[J]. Robotics and Autonomous Systems, 1995, 15(4): 233-235.
- [25] LIN L J. Reinforcement learning for robots using neural networks[D]. Pittsburgh, Carnegie Mellon University, 1992.
- [26] LATTNER C, ADVE V. LLVM: a compilation framework for life-long program analysis & transformation[C]//Proceedings of International Symposium on Code Generation and Optimization. 2004: 75-86.
- [27] PESCH R H, OSIER J M. The GNU binary utilities[J]. Free Software Foundation, 1993.
- [28] DOLAN-GAVITT B, HULIN P, KIRDA E, et al. LAVA: large-scale automated vulnerability addition[C]//Proceedings of 2016 IEEE Symposium on Security and Privacy (SP). 2016: 110-121.
- [29] METZMAN J, SZEKERES L, SIMON L, et al. Fuzzbench: an open fuzzer benchmarking platform and service[C]//Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021: 1393-1403.

[作者简介]



肖天（1998-），男，江苏常州人，上海交通大学硕士生，主要研究方向为漏洞挖掘、隐私保护。



江智昊（1998-），男，上海人，主要研究方向为文件系统、漏洞挖掘、分布式系统。



唐鹏（1992-），男，江西抚州人，上海交通大学博士生，主要研究方向为人工智能安全、隐私保护。



黄征（1975-），男，四川南充人，博士，上海交通大学副教授，主要研究方向为隐私保护、计算机视觉、人工智能安全。



郭捷（1976-），女，河南信阳人，博士，上海交通大学副研究员，主要研究方向为多媒体安全、模式识别、大数据分析。



邱卫东（1973-），男，江西九江人，博士，上海交通大学教授、博士生导师，主要研究方向为密码分析/密码工程、人工智能安全、大数据隐私保护。