人机协同信息系统漏洞挖掘机制研究与实践

林哲超[1], 卢帅兵[1], 聂原平[1], 张 甲[2], 段海新[2], 李 响[1], 况晓辉[1]

[1]军事科学院系统工程研究院 北京 中国 100101
[2]清华大学网络科学与网络空间研究院 北京 中国 100084

摘要 漏洞挖掘对于提高信息系统的安全性和可靠性起着至关重要的作用。随着信息技术的不断发展, 漏洞挖掘技术持续向自动化智能化方向演进。虽然机器在其中扮演越来越重要的角色, 但是信息系统规模的不断增大以及网络空间的日益复杂化, 使得单纯依靠机器自动化挖掘漏洞的局限性也逐渐显现出来。统计分析表明, 人在高价值漏洞发现过程发挥着不可替代的作用。本文首先从人机关系的角度宏观分析了漏洞挖掘技术半个世纪的发展历程, 并根据人与机器的关系以及协同程度, 构建了人机协同概念框架, 提出工具、助手和伙伴 3 个层次的人机协同概念。其次, 在人机协同概念框架基础上, 结合漏洞挖掘技术的发展趋势、漏洞挖掘的内涵本质, 面临的问题挑战, 从交互方式、人机规模及人机关系等角度, 探讨了人机协同漏洞挖掘模型的分类, 包括单向辅助式人机协同、双向互助式人机协同以及共融互促式人机协同漏洞挖掘模型。由于当前漏洞挖掘技术正处在双向互助式人机协同的发展阶段, 尚未发展出共融互促式人机协同, 因此本文重点阐述了双向互助式人机协同漏洞挖掘模型的内涵。在该模型的指导下, 围绕 3 类典型场景下漏洞挖掘面临的问题挑战, 研究提出并实现了对应的人机协同漏洞挖掘模式。实验结果表明, 人机协同漏洞挖掘方法相对于已有的研究工作, 在代码覆盖率、漏洞发现效率、漏洞发现类型等方面有了显著提升, 并在真实系统中发现未公开漏洞 30 余个。

关键词 漏洞挖掘; 人机协同漏洞挖掘模型; 模式
中图法分类号 TN915.08 DOI 号 10.19363/J.cnki.cn10-1380/tn.2026.01.08

Research on Mechanism of Human-Machine Collaborative Vulnerability Mining

LIN Zhechao[1], LU Shuaibing[1], NIE Yuanping[1], ZHANG Jia[2], DUAN Haixin[2], LI Xiang[1], KUANG Xiaohui[1]

[1] Institute of System Engineering Academy of Military Sciences, Beijing 100101, China
[2] Institute for Network Science and Cyberspace, Tsinghua University, Beijing 100084, China

Abstract Vulnerability mining plays a vital role in improving the security and reliability of information systems. With the continuous advancement of information technology, vulnerability mining technology is evolving towards automation and intelligence. Although machines played an increasingly pivotal role in this domain, the continuous increase in the scale of information systems and the increasing complexity of cyberspace have gradually revealed the limitations of relying solely on machine automation to mine vulnerabilities. Some statistical analysis reveals that human involvement remains indispensable in the process of discovering high-value vulnerabilities. This paper examines the development history of vulnerability mining technology over half a century from the perspective of the man-machine relationship. First, based on the relationship between humans and machines and the degree of collaboration, a conceptual framework for human-machine collaboration was constructed, and a three-level concept of human-machine collaboration relationship was proposed, namely tools, assistants, and partners. Secondly, drawing upon the conceptual framework of man-machine collaboration, combined with the development trend of vulnerability mining technology, the connotation and essence of vulnerability mining, and the problems and challenges faced, from the perspectives of interaction mode, the scale of human and machine, and human-machine relationship, this paper explores the classification of man-machine collaborative vulnerability mining models, including one-directional assisted human-machine collaborative vulnerability mining, bidirectional assisted man-machine collaborative vulnerability mining, and inclusive and mutually promoting human-machine collaborative vulnerability mining models. Since the current vulnerability mining technology is in the development stage of bidirectional assisted human-machine collaboration and has not yet developed inclusive and mutually promoting human-machine collaboration, this paper has a particular emphasis on elucidating the essence of bidirectional mutual man-machine collaborative vulnerability mining models. Under the guidance of this model, focusing on the problems and challenges faced by vulnerability mining in three typical scenarios, the corresponding man-machine collaborative vulnerability mining mode is proposed and implemented. Experimental validation demonstrates that compared to existing research

通讯作者: 李响, 硕士, 高级工程师, Email: ideal_work@163.com。
收稿日期: 2024-03-31; 修改日期: 2024-07-21; 定稿日期: 2025-11-13

efforts, our proposed method significantly enhances code coverage, efficiency in identifying vulnerabilities, as well as diversifying types discovered; more than 30 undisclosed vulnerabilities were successfully identified within real systems.

Key words vulnerability mining; human-machine collaborative vulnerability mining model; pattern

1 引言

随着信息技术的快速发展和广泛应用, 信息系统在人类社会中扮演着愈发重要的角色。然而, 随着数字化生活的普及, 信息系统的安全性也受到了前所未有的挑战。针对信息系统开展漏洞挖掘与修复, 能够有效降低系统受到网络攻击的概率, 是提升信息系统安全性的重要手段。

学术界和工业界在漏洞挖掘方向开展了大量持续性研究实践工作。早期漏洞挖掘主要依靠人工经验进行手动分析。随着软件规模和复杂度的不断增加, 漏洞挖掘技术开始朝着自动化的方向发展。从早期依赖计算机的计算和推理能力, 到后来应用机器学习、深度学习等技术方法, 漏洞挖掘技术也开始利用智能化方法从海量数据中提取经验和知识, 辅助提高漏洞挖掘的精度和效率[1]。

然而漏洞挖掘是一项复杂的智力活动, 不仅需要借助机器的计算分析能力, 也需要依靠人的理解力、判断力和直觉。因此虽然机器自动化挖掘漏洞具有效率高、成本低等优点, 但也存在发现漏洞类型有限、高价值漏洞发现难等局限性。鉴于此, 近年来一些研究人员在漏洞挖掘技术方法优化中, 开始探索如何有效利用人类的经验和知识提升漏洞挖掘能力[2-7]。

本文围绕如何有效发挥机器和人各自优势以提高漏洞挖掘效能的问题, 阐述研究团队在理论研究和技术实践方面的初步探索和思考: 首先从人机关系的角度分析了漏洞挖掘技术的发展历程和趋势; 其次, 在人机协同概念框架基础上, 探讨了人机协同漏洞挖掘模型的分类, 并重点阐述了双向互助式人机协同漏洞挖掘模型的内涵; 在理论指导下, 围绕当前漏洞挖掘典型场景和面临的问题挑战, 重点从人机分工、人机交互以及人机反馈优化等方面, 提出了 3 种人机协同漏洞挖掘模式, 实验验证了方法有效性, 并展望了后续研究方向, 以期为漏洞挖掘技术研究提供新的视角。

本文内容组织如下: 第 2 节梳理了漏洞挖掘技术的发展历程, 分析了发展趋势; 第 3 节阐述了人机协同漏洞挖掘机制; 第 4 节描述了模型指导下针对典型场景所提出的人机协同漏洞挖掘模式, 以及实践验证情况; 第 5 节对本文进行了总结, 并对未来研究工作进行了展望。

2 漏洞挖掘技术的发展趋势

从 20 世纪 70 年代静态分析方法陆续被提出至今, 漏洞挖掘技术已发展了近 50 年。虽然在每个时期都同时存在多种漏洞挖掘技术, 但从人机关系角度审视漏洞挖掘技术的发展历程, 可简要划分为手工挖掘、规则匹配、模糊测试以及人机协同 4 个阶段。

2.1 手工挖掘阶段

在 1976 年以前, 漏洞挖掘主要依靠安全人员的经验, 同时利用一些代码分析工具或者逆向工具来辅助人进行分析, 人在这个过程中需要付出大量的时间和精力。

这一阶段的特点是以人的经验为主, 工具辅助人进行分析。但是随着程序规模和复杂性不断增加, 依靠人工经验挖掘漏洞效率低、可扩展性差的缺点越发凸显, 已经无法满足日益增长的安全需求。

2.2 规则匹配阶段

1976 年以后, 许多研究人员开始研究如何让机器自动化挖掘漏洞, 自动化漏洞挖掘技术陆续被提出。1976 年 King 提出了符号执行的概念[8]; 污点分析作为一种信息流分析技术, 其理论基础可以追溯到 1976 年 Denning 提出的一套基于格(lattice)的理论模型[9]; Pnueli 在 2008 年第一个提出对并发程序的推理使用线性时序逻辑[10]; 1981 年, 卡耐基·梅隆大学的 E.Clarke 和他的博士生 Allen Emerson 首次提出将模型检验技术作为有穷状态并发系统的检测技术, 并提出了第一个模型检测算法[11]; 模糊测试技术最早可追溯到 1988 年, 威斯康星大学教授 Barton Miller 首次提出 Fuzz 生成器(Fuzz generator)的概念[12]。

该阶段的主流技术途径是将人的经验规则化, 因此这一阶段的漏洞挖掘主要以静态分析技术为主, 代表技术有污点分析、静态符号执行、模型检验等。特别是进入 2000 年后, 随着计算机运算能力的不断提高, 以及 BSP(Boolean Satisfiability Problem)求解速度突飞猛进的发展, 以 SAT 为核心, 各种单独的约束求解算法被整合起来, 形成了 SMT(Satisfiability Modulo Theories)技术, 使得程序中复杂的约束可以在多项式时间内被求解出来。这极大推动了符号执行、污点分析等技术的发展。同时模糊测试技术也在发展, 但主要以黑盒模糊测试技术为主。这阶段的

代表性工作有:

模糊测试: PROTOS[13](2001)、PEACH[14](2004)。

污点分析: Livshits 等[15](2005)。

静态符号执行: Myers 等[16](1979)。

模型检验: SMV[17](1996)、NuSMV[18](1999)。

这一阶段的特点是自动化漏洞挖掘工具开始增多, 人将漏洞挖掘思路方法转换为机器可求解的问题, 并开发相应的工具; 机器负责长时间持续执行工具, 找到符合约束条件的软硬件缺陷。这一阶段的挖掘工具以静态分析方法为主, 存在误报率高、可扩展性不足等问题。

2.3 模糊测试阶段

这一阶段的划分以 2005 年 Godefroid 等人提出动态符号执行技术为始, Sen[19]后续又对其原理进行了深化。动态符号执行技术的提出, 标志着漏洞挖掘技术进入动态阶段。模糊测试技术在这个阶段取得了巨大进展。早期的黑盒模糊测试技术由于缺少对程序内部状态的分析, 因此存在一定的局限性。受益于动态符号执行技术的提出, 2012 年 SAGE 将模糊测试与符号执行相结合, 提升了模糊测试引擎对魔术字、校验码的绕过能力。2013 年 AFL 采用覆盖率引导的种子变异技术, 大幅提升了模糊测试引擎漏洞发现能力, 成为模糊测试技术发展的重要方向。这一阶段的代表性工作包括 Pixy[20](2006)、EXE[21](2008)、KLEE[22](2009)、SAGE[23](2012)、AFL[24](2013)、Driller[25](2016)、QSYM[26](2018)、Pangolin[27](2020)、HFL[28](2020)等。

2008 年, Cadar 等在 EXE 的基础上, 提出动态符号执行工具 KLEE[22]。KLEE 采用了多种约束求解的优化算法提高了执行效率, 紧凑地表示程序状态降低了空间开销, 并使用试探搜索法获得了较高的代码覆盖率。尽管取得很大的进展, 但是以 KLEE 为代表的符号执行工具面临的路径爆炸、可扩展性差等问题并没有得到解决。

2013 年, 谷歌员工 Zalewski 开发了覆盖引导模糊测试工具 AFL[24]。由于具有误报率低、可扩展性强、使用范围广的特点, AFL 的出现让模糊测试技术逐渐成为漏洞挖掘的主要手段。AFL 对并行化的支持, 使得集群化漏洞挖掘成为可能, 机器算力在漏洞挖掘中发挥的作用得到进一步提升。这一阶段学术界和工业界普遍关注如何让机器更自动地挖掘漏洞, 尽量减少人的参与。例如 2013~2016 年, DARPA 举办了 CGC 挑战赛[29], 旨在推动漏洞自动化挖掘和修复。

这一阶段的特点是以模糊测试技术为主。模糊测试与其他分析技术结合, 以期望在人不干预的情况下进一步提升漏洞发现能力。规则匹配阶段存在的误报率高, 可扩展性差等问题得到进一步解决。但是模糊测试技术在发展过程中也遇到了一些障碍, 例如一些程序的约束条件十分复杂, 单靠现有符号执行工具, 难以在多项式时间内求解出结果, 导致模糊测试技术在覆盖率上无法进一步提高。

2.4 人机协同阶段

这一阶段的划分以 2018 年 DARPA 发布“人机探索网络安全(CHESS)”项目[30]为始, 该项目旨在通过人和机器的协作, 实现优势互补, 加速漏洞挖掘。在这一阶段, 研究人员围绕污点分析、符号执行、模糊测试等漏洞挖掘方法面临的挑战问题, 开始探索将人的能力嵌入到漏洞挖掘的具体环节中, 以提高漏洞挖掘的效率。主要研究并包括:

测试用例协作生成技术。Dynodroid[3]通过分析安卓应用对用户输入的反馈, 支持分析人员手工调整测试用例, 丰富了输入事件类型, 增强了对安卓平台的测试覆盖率。HaCRS[4]使分析人员能够通过仿真终端与目标应用程序进行交互, 并手工生成输入文件, 以提高二进制文件模糊测试的代码覆盖率。

基于可视化的分析优化技术。VisFuzz[5]通过对模糊测试中关键代码的调用图和控制流图进行可视化, 使分析人员能快速把握代码的语义上下文, 并通过手动干预提升代码覆盖率。HM-Fuzzing[6]提出区间分析方法, 通过区间分类和排序来指导分析人员优化模糊测试过程, 提高漏洞发现能力。

基于标注引导的反馈优化技术。与代码覆盖率相比, 在源代码中添加标注可以从模糊器得到更细粒度的反馈。FuzzFactory[7]通过用户自定义领域特定模糊测试规则和动态反馈机制, 以提升覆盖率引导模糊测试引擎识别安全漏洞的能力。IJON[2]引入了一种代码标注机制, 使分析人员能够为模糊测试引擎提供更细致的反馈, 帮助引擎绕开难以绕过的代码区, 以增强对深层次代码的模糊测试效率。

自 2022 年以来, 大型语言模型(Large Language Model, LLM)在发展过程中逐渐被研究人员用于探索漏洞挖掘。研究主要集中于利用 LLM 对编程语言的理解和生成能力。通过精心设计的提示(Prompts), 研究人员指导 LLM 生成高质量的测试用例和多样化的测试驱动代码, 目的是提高模糊测试在程序代码中的覆盖率并增强复杂软件系统的漏洞检测效率。相关工作包括 Last[31]、FuzzGPT[32]、Fuzz4All[33]、TitanFuzz[34]和 CodaMosa[35]等, 这些工作通过单一 prompt 增进测试用例的生成; 以及 Libro[36]和 LLM4fdg[37], 这些工作通过多轮 prompt 来优化测试

驱动代码的生成。在此框架中, prompt 的设计成为一个关键步骤, 主要基于分析人员对于问题的理解以及将这一理解转化为对 LLM 友好的语言。此过程体现了人机交互的重要性, 其结合了人类的认知解析能力与机器的数据处理能力。

这一阶段的特点是人参与到机器自动化挖掘的环路中, 利用人的经验动态调整挖掘策略, 从而提高挖掘效率, 初步验证了人机协作的有效性, 但缺乏系统性。

通过上述梳理分析, 我们发现漏洞挖掘技术发展的整体趋势是人机合作共同挖掘漏洞, 因为当前一些问题单纯依靠机器已经无法解决, 需要融入人的智力, 充分发挥人和机器的各自优势, 从而最大化漏洞挖掘效果。尽管 LLM 在漏洞挖掘中也有所应用, 但是从当前看, 仍需要与人协作才能有效解决漏洞挖掘所面临的挑战。

3 人机协同漏洞挖掘机制

3.1 人机协同概念框架

人机协同是持续演进的概念[38-53]。从人与机器之间关系角度看, 人机协同分为工具、助手和伙伴 3 个层次, 且协同程度不断提升, 概念框架如图 1 所示。

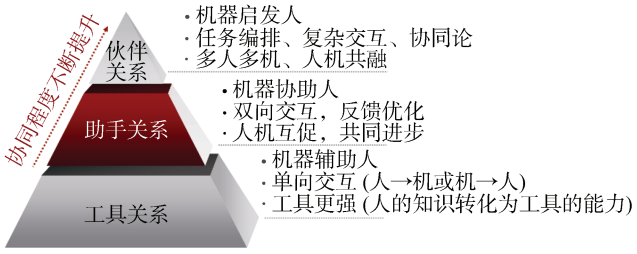

图 1 人机协同概念框架

Figure 1 Conceptual framework for human-machine collaboration

(1) 工具关系

早期人机协同任务中人与机器的关系主要是工具关系, 一般以单人单机的协作形式为主, 主导任务完成的是人, 机器则作为一种工具, 辅助人类处理复杂任务。人将面临的问题分解为若干子问题, 将部分子问题转化为机器可处理的形式, 机器根据人提供的输入将处理的结果反馈给人。在这种关系中, 人机之间的交互是单向交互, 机器主要在特定环节发挥作用, 且机器的能力基本固定, 不会根据人的反馈优化自身的机制。

(2) 助手关系

当前人机协同任务中人与机器的关系主要是助手关系, 一般以单人单机的协作形式为主, 主导任务完成的是人, 而机器则作为一种助手角色, 协助人处理复杂任务。与工具关系的主要区别是助手关系中人与机器之间具有反馈机制, 协同程度有一定的提升。人在利用机器解决问题时, 相对于工具关系更为复杂, 且会根据机器的反馈给与评价, 机器会根据反馈优化提升问题解决能力; 与此同时, 人也可以根据机器的输出结果优化完善自身的知识, 并利用优化后的知识分解引导机器更好地解决问题, 从而实现人与机器的能力双向迭代提升。

(3) 伙伴关系

随着人工智能技术的发展, 未来人机协同任务中人与机器将更加充分的融合, 人机之间的关系主要是伙伴关系。在伙伴关系中, 人机协同的规模由单人单机向多人多机协同演进; 人与机器智能系统的协同不仅包括人与单个或多个独立智能系统的协同, 也包括人与群体智能系统的协同, 群体智能系统将拥有单个智能系统所不具备的群体特性[54]。除了人与机器之间的协同交互, 在伙伴关系中, 人与人之间, 机器与机器之间同样需要协同, 人与机器在这种复杂的协同形式下相互启发, 共同进化, 人与机器的能力在持续不断的交互、反馈、优化中不断得到提升, 所构成的人机协同系统具备自我进化的能力。

不同层次的人机协同, 都可从人机分工、交互、反馈优化 3 个层面刻画, 如图 2 所示。以漏洞挖掘为例讨论各层面解决的主要问题。

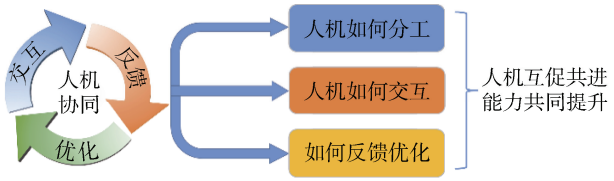

图 2 人机协同的动态演进

Figure 2 Dynamic evolution of human-machine collaboration

人机分工主要解决在漏洞挖掘过程中哪些工作需要机器完成, 哪些工作需要人完成; 不同的漏洞挖掘技术方法, 工作集合存在差异, 人机协作首先需要考虑这些工作在人机间如何分工。

人机交互主要解决人机协同漏洞挖掘过程中人机之间如何相互理解, 实现人机之间知识的互通与映射, 即人和机分别应该提供什么类型的知识给对方, 以何种形式呈现更有利于理解, 也就是知识表征和可视化问题。

反馈优化主要解决人机协同漏洞挖掘过程中机

器如何自主学习和自我修正等问题, 具体包括人如何根据机器反馈的结果来优化改进对问题理解, 并优化提供给机器的知识, 一方面帮助机器提升漏洞挖掘的准确率, 发现更多类型和更深层次的漏洞, 另一方面也提升自身对漏洞的认识。

总的来看, 人机协同是在人机分工基础上形成的交互、反馈、优化的持续演进过程。在此过程中, 人和机提升了对问题的理解和解决的能力, 实现了人机互促共进。

3.2 人机协同漏洞挖掘模型分类

结合漏洞挖掘的内涵本质和问题挑战, 基于人机协同概念框架, 可建立 3 种不同协同程度的漏洞挖掘模型。

(1) 单向辅助式人机协同

人与机器之间是一种工具关系, 挖掘主体是人或者机器。一种方式是领域专家将专门知识经过分析、综合、整理后以某种表示形式传递给机器, 辅助机器进行漏洞挖掘(比如早期的专家系统), 通过将人的知识注入机器中, 使机器的挖掘能力得到了提升; 另一种方式是人利用机器计算的结果分析可能存在的漏洞(人借助机器提升代码审计能力, 比如利用机器将软件中疑似漏洞的位置标注出来, 再由有经验的专家进行分析挖掘), 有了机器的辅助, 使得人的挖掘能力也得到了提升。这种形式的人机协同一般以单人单机为主, 机器为传统计算机, 人与机器之间的协作为单向协作, 人机之间并无反馈机制。

(2) 双向互助式人机协同

人与机器之间是一种助手关系, 挖掘主体是机器(依赖机器自动化挖掘)。人作为领域专家在漏洞挖掘过程的环路中(Human in the loop), 负责将知识转化为模型、算法或参数传递给机器; 机器针对负责执行模型算法以发现潜在的漏洞, 并将过程和结果信息以人可理解的方式反馈给领域专家。领域专家根据反馈的信息, 对其掌握的知识进行更新, 并将模型、算法或参数并传递给机器(例如麻省理工和 PatternEx 联合推出的 AI2[55]), 以提高机器发现漏洞的能力。通过人在环路的方式, 使得漏洞挖掘的过程中人和机器能力能够得到一定提升, 从而提高漏洞挖掘效果。这种形式的人机协同一般为单人单机或单人多机, 机器为传统计算机, 人与机器之间通过反馈机制(交互接口)实现双向协作。

(3) 共融互促式人机协同

人与机器之间是一种伙伴关系, 挖掘主体是人与机器。相较于互助式人机协同, 共融互促式人机协同人机之间的协同方式更为复杂, 一般为多人多机协同合作。多人不仅可以是同领域的专家, 也可以是擅长不同领域的专家(例如目标系统的开发者、AI 专家、擅长符号执行的专家、擅长模糊测试的专家等)。多机则为实现污点分析、符号执行、模糊测试等各类漏洞挖掘方法的工具。多人多机协同不仅包含人与机之间协同, 也包括人与人之间和机与机之间的协同。随着人工智能技术的发展, 多机可以是多个独立智能漏洞挖掘工具的协同, 也可以构成耦合度更高的群体智能漏洞挖掘系统, 并与人构成人机混合智能漏洞挖掘系统。人机间的复杂协同方式使得人与机器互相融合为一体, 人类的创造性和机器的自主性, 以及二者间的协同优化使得能力得以不断提升, 从而实现持续进化。

3.3 双向互助式人机协同漏洞挖掘模型

根据对漏洞挖掘技术发展历程的梳理, 结合人工智能技术的进展, 为探索解决当前主流漏洞挖掘技术面临挑战, 我们重点对双向互助式人机协同漏洞挖掘模型进行了刻画, 如图 3 所示。

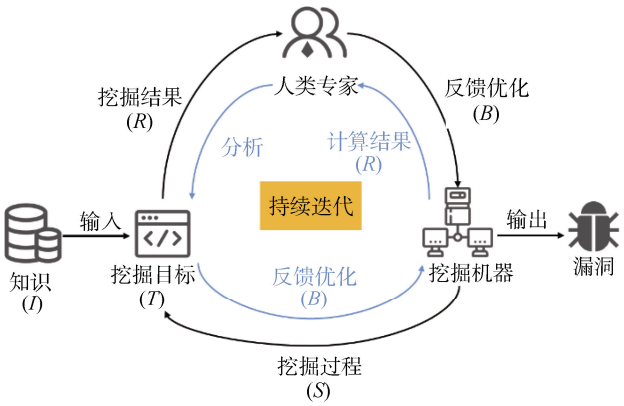

图 3 双向互助式人机协同漏洞挖掘模型

Figure 3 Model of human-machine collaborative vulnerability mining

(1) 问题定义

以 $P=(I, T)$表示问题, 其中 I 表示输入的安全知识, 即在开始人机协同漏洞挖掘之前, 当前领域已经积累的安全知识, 包括数据、挖掘工具以及挖掘方法等。T 为漏洞挖掘的目标对象。模型旨在优化漏洞挖掘能力, 即基于已有知识在规定的时间内找到尽可能多的漏洞, 表示为 $f_{\max}(I, T)$。

(2) 漏洞挖掘过程

将人机协同漏洞挖掘过程表示为序列 $S=\{A_1, A_2, \ldots, A_n\}$, 其中 A_i 表示第 i 步的挖掘操作。机器和人类专家发挥各自的优势, 主要包括以下过程:

机器执行: $A_1 = M(I)$, 表示机器自动进行初步漏

洞挖掘。其中 M 表示机器执行的操作。

人工审查: $A_2 = H(A_1)$, 表示人类专家对初步挖掘结果进行分析, 标记可能的漏洞。

机器进一步挖掘: $A_3 = M(A_2)$, 机器根据专家给出的信息进行更深层次的挖掘。

(3) 结果分析

使用 R 表示漏洞挖掘的结果向量, 其中 R_i 表示第 i 个漏洞的挖掘结果。我们引入结果解释函数 $E(R)$, 用于对挖掘结果进行解释, 确保结果的可信度和有效性。

(4) 反馈优化

在人机协同优化阶段, 建立了一个反馈机制 $B(I, T, S, R)$, 专家提供对挖掘过程和结果的反馈。同时, 我们将优化问题细化为 $h_{\max}(I, T, S, R, B)$, 这是关于输入、目标、挖掘过程、结果和反馈优化的目标函数。这样, 我们能够利用人机协同不断优化漏洞挖掘的效果。

(5) 持续迭代

双向互助式人机协同漏洞挖掘是一个不断迭代优化的过程, 因此我们引入迭代过程:

$$I_{n+1} = I_n + \alpha \nabla h\,(I_n, T_n, S_n, R_n, B_n)$$

在这个公式中, I 是输入知识, T 是挖掘目标, S 是漏洞挖掘过程, R 是挖掘结果, B 是反馈优化, ∇ 是梯度算子, α 是学习率。其中挖掘结果 R 由机器在执行完相关任务后提供给人类专家, 而反馈 B 则是由专家对机器提供的结果进行分析后将信息反馈给机器, 因此 R 与 B 体现了人机交互的过程, 是该模型的核心模块。通过 R 与 B 不断迭代交互, 更新输入知识 I, 学习新的漏洞特征和优化目标函数, 不断改进漏洞挖掘的效果。这个迭代过程确保了模型的持续优化和适应性。

在人机协同漏洞挖掘过程中, 机器负责执行自动化漏洞挖掘和初步结果生成, 而人类专家负责进行结果审查、提供领域知识和优化目标函数。这个协同模型结合了机器的高效性和人类专业知识的深度理解, 使漏洞挖掘过程更为全面、准确和高效。

4 人机协同漏洞挖掘模式研究与实践

在人机协同漏洞挖掘基础模型的指导下, 针对当前漏洞挖掘面临的模糊测试路障难以自动破除, 基于机器学习的漏洞模式构建可解释性差、泛化能力不足, 协议不一致性漏洞难以自动挖掘等三类问题挑战, 分别提出了人机协同的路障标注和破解、人机协同的漏洞模式构建和人机协同的协议语义不一致性漏洞挖掘 3 种模式, 并在真实信息系统的漏洞挖掘过程中验证了模式的有效性。在人机协同漏洞挖掘过程中, 人的能力水平对漏洞挖掘结果有重要影响。为降低不同人员水平差异对人机协同有效性评价的影响, 本文重点探讨同一人员利用自动化漏洞挖掘工具和采用人机协同漏洞挖掘工具的差异, 以尽可能客观度量人机协同的作用。

4.1 人机协同的路障标注和破解

4.1.1 面临问题

模糊测试在测试过程中存在难以突破复杂约束的问题, 如值校验、魔数字、复杂条件判断、特殊状态等, 这些在路径探索时难以突破的约束也被称为路障。尽管相关研究人员提出使用符号执行等方法破解路障[25-27], 但是路径约束的多样性和复杂性, 使得现有的自动化技术经常遇到无法求解的问题, 严重影响复杂软件漏洞挖掘的覆盖率和漏洞发现效率。

4.1.2 人机协同模式

针对单纯依靠机器自动化无法有效破解路障的问题, 本文提出一种基于人机协同的路障标注和破解模式, 通过人机交互、多轮反馈优化的方式引导模糊测试定位并破解路障, 达到提升覆盖率并发现漏洞的目的。

(1) 人机分工

人机协同路障标注与破解模式基本流程如图 4 所示。首先, 机器通过路障定位模块得到路障点、路障类型等信息, 路障破除标注建议模块从路障破解规则库中匹配破解方案, 生成标注建议和测试目标函数建议; 其次, 测试人员根据建议和具体路障类型, 完成代码标注或修改测试驱动并执行模糊测试; 最后, 根据人机协同引擎测试情况判定标注建议或驱动修改是否有效, 若破解成功, 则添加新的标注方案到知识库, 若破解失败, 则调整标注方案重新启动新一轮测试。经过多轮迭代, 路障点逐步被突破, 目标函数覆盖度增加, 路障破解知识库不断完善, 路障破除准确度更高, 代码覆盖度增加。

(2) 交互反馈

在该模式中, 人与机器的交互反馈如下:

机器将执行的结果提供给人(R), 包括:

1) 在路障定位时由机器将路障的基本特征属性呈现给人, 包括路障类型、条件判断的约束、哈希的类型、特殊值、调用树、上下文代码等, 便于人理解代码和路障。

2) 路障破除标注建议模块根据已有标注方案对当前路障进行分析, 给出标注建议, 包括标注位置、语句、指令类型、关键值等, 为人提供标注提示。

人将分析的结果反馈给机器(B), 包括。

1) 执行代码标注: 测试人员综合分析路障信息和标注提示, 在代码的适当位置实施代码标注, 或根据路障突破情况调整标注方案。

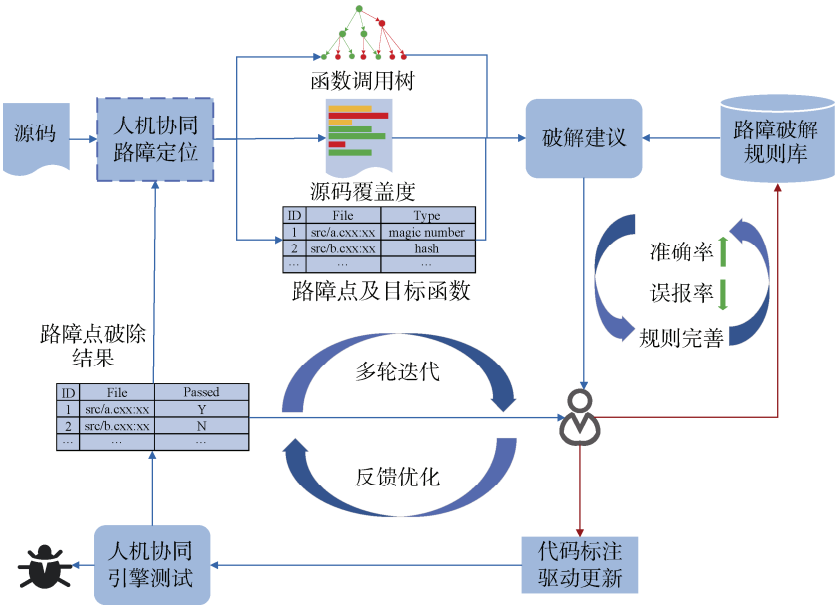

图 4　人机协同的路障标注与破解模式

Figure 4　Pattern of human-machine collaborative roadblock break-through

2) 路障破解知识: 在机器重新使用标注引导的代码进行测试后, 将成功的标注方案添加到路障破除知识库, 不断丰富完善路障破解规则。

(3) 能力提升

在人与机器不断迭代交互的过程中, 路障破解能力得到不断优化, 主要体现在:

1) 路障破除标注建议准确性不断提升: 每一次针对具体路障问题进行破解后都会形成新的知识, 即成功的路障标注方案, 该方案逐步积累到机器的路障破解知识库中, 路障破解知识库不断丰富, 使得路障破除标注建议准确率不断提升。

2) 漏洞发现能力不断提升: 测试目标也在人机协同路障破解的过程中被充分探索、发现漏洞。

最终形成了多轮迭代、反馈优化的效果。

4.1.3　实现与验证

根据上述思路, 本文实现了人机协同代码标注引擎 InFuzz, 包含了标注引导测试引擎、瓶颈分析、覆盖度展示、调用树、标注语句生成等模块, 可支持魔术字节、校验和、哈希图、嵌套条件等路障突破, 并针对现实世界中的程序进行实验。

本文使用 OSS-Fuzz[56]中已达到覆盖率瓶颈的语料库作为初始输入, 如前所述, InFuzz 的目标是在模糊测试到达瓶颈时通过人工干预来突破覆盖率瓶颈, 提升代码覆盖率。如表 1 所示为 AFL++与 InFuzz 覆盖的代码行、函数比例。在源代码行级, 与 AFL++相比, InFuzz 覆盖率平均增加了 31%。在源代码函数级, 与 AFL++相比, InFuzz 覆盖率平均增加了 26.66%。从实验结果可以看出, InFuzz 能够有效帮助测试人员花费很少的时间(在 6 h 内)来获得更多的覆盖范围, 并且突破单纯依靠机器无法覆盖的瓶颈。

表 1　代码行和函数覆盖率对比结果

Table 1　Lines and function coverage comparison results

目标	AFL++		InFuzz	
	代码行	函数	代码行	函数
astc-encoder	49.80%	16.37%	62.38%	21.30%
cjson	43.91%	28.32%	53.34%	33.91%
casync	10.01%	8.73%	14.19%	11.81%
clib	14.10%	21.69%	20.34%	27.70%
cpuinfo	17.88%	9.30%	25.48%	12.64%
gss-ntlmssp	21.57%	28.04%	24.56%	31.16%

4.2　人机协同的漏洞模式构建

4.2.1　面临问题

漏洞模式是从大量已知漏洞数据中, 归纳聚类提炼出的共性特征, 这种模式通常能够高度概括某一类或几类漏洞的特征, 为面向源代码的静态分析提供支撑。

当前漏洞模式获取方式主要分为两类: 一是基于人工构建方式, 主要通过专家经验编写一系列规则, 组合形成漏洞模式。其优点是准确率高, 缺点是成本开销大、规则的有效性完全依赖于人的经验。二是基于机器学习获取方式, 通过数据驱动训练机器学习模型获得漏洞模式。其优点是不依赖专家知

识, 模式获取速度快, 缺点是不可解释、泛化能力不足, 用于特定对象上的静态分析效果往往不太理想。

4.2.2 人机协同模式

针对单纯依靠人类或机器学习的方式获取漏洞模式存在的问题, 本文提出一种人机协同的漏洞模式构建方法。

(1) 人机分工

首先使用机器学习模型学习得到初步的漏洞模式; 其次通过模型解释器, 对得到的漏洞模式进行知识表示, 从而在样本中标记出模式认为的漏洞位置; 最后, 通过设计人机接口, 允许人类专家对漏洞位置进行修改与编辑, 并将编辑后的位置用于更新原有模式。这种方式可以在不干扰已有知识的前提下, 有效提升漏洞模式在特定对象下的精确度, 具体流程如图 5 所示。

(2) 交互反馈

在该模式中, 人与机器的交互反馈如下:

机器将潜在漏洞位置信息提供给人(*R*): 通过图神经网络解释器将任意文件中的漏洞潜在位置信息呈现给人, 包括文件名、函数名、行号及置信度等。

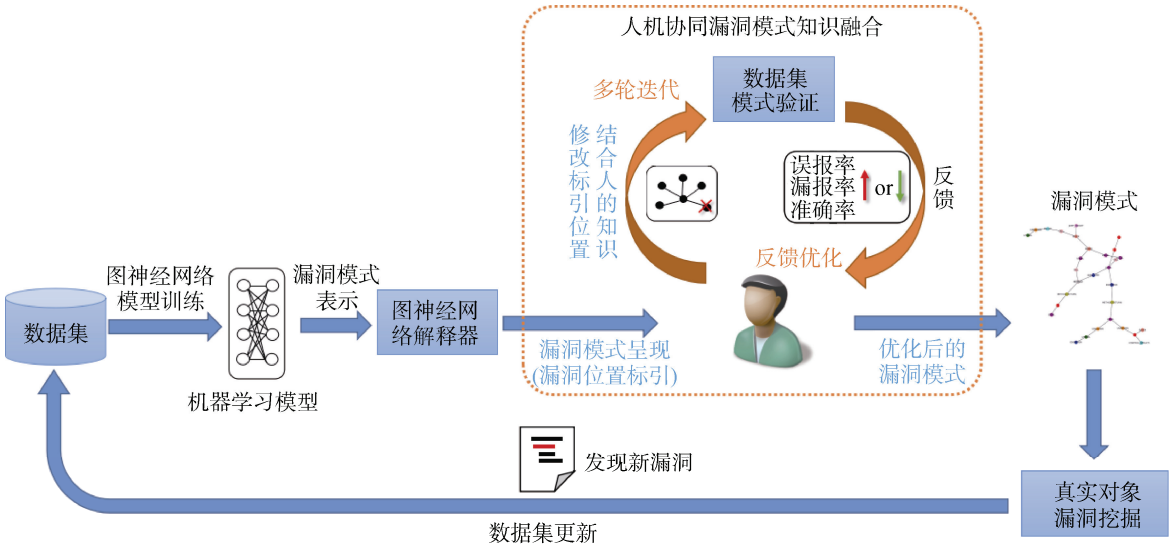

图 5 人机协同漏洞模式构建

Figure 5 Vulnerability pattern construction based on human-machine collaboration

人将人理解的漏洞位置信息反馈给机器(*B*): 依托专家经验, 对机器标注的潜在漏洞位置信息进行增加、删除、修改等操作。

(3) 能力提升

在人与机器不断迭代交互的过程中, 模式提炼的能力得到不断优化, 主要体现在:

机器可以将人编辑修改的知识进行更新, 更新后的模式可以通过验证数据集进行测试, 并将结果反馈给人。人可以根据反馈结果决定继续或停止修改。经过多轮的迭代优化过程, 可以使得最终生成的模式能够更精确地体现漏洞在特定对象上的特征, 并在实际漏洞挖掘过程中有效识别漏洞。

4.2.3 实现与验证

根据上述思路, 本文实现了人机协同漏洞模式构建与应用工具, 包含了样本数据集、机器学习模型训练、图神经网络解释器、模式编辑器等模块, 可支持对 C、C++、JAVA、Python 等语言的漏洞模式构建。本文选取 CWE-416、CWE-119 两类常见漏洞类型, 分别组织了两组实验, 每组实验首先在通用数据集上训练得到一个通用的漏洞模式, 能够在通用测试集上达到较好效果(F_1 值在 0.85 左右); 然后在特定对象测试集上验证, 观察效果是否有所下降, 同时使用人机协同模式构建方法对已有模式进行优化; 最后测试优化后的模式在上述两个测试集上的表现。

(1) CWE-416 漏洞模式在 Linux kernel-net 模块中的验证实验结果如表 2 所示, 其中模式 1 是由通用数据集训练得到, 数据集包含 2847 组数据; 模式 2 经过 81 次人机协同优化后得到; Gen-dataset 是指的通用测试集, 包括 406 个测试用例, Net-dataset 是在 Linux kernel-net 真实对象中的测试集, 包括 270 个测试用例。实验结果表明, 未经过人机协同优化的模式在通用测试集上表现优异, 但是特定对象(此处为 Linux 内核)测试集中性能大幅下降。通过 81 次(优化比例为 30%)人机协同优化后得到的新模式可以提升在 Linux 内核测试集中的表现, 同时不影响在通用测试集上的表现。

表 2 CWE-416 在 Linux kernel-net 模块验证实验结果
Table 2 CWE-416 experiment results in Linux kernel-net module

名称	测试数据集	*acc*	*recall*	F_1-*score*
模式 1	Gen-dataset	0.90	0.87	0.869
模式 1	Net-dataset	0.53	0.36	0.382
模式 2	Gen-dataset	0.88	0.81	0.832
模式 2	Net-dataset	0.88	0.98	0.873

(2) CWE-119 漏洞模式在 Linux kernel-fs 模块中的验证实验结果如表 3 所示, 其中模式 1、模式 2、Gen-dataset 同上实验定义, Fs-dataset 是在 Linux kernel-fs 真实对象中的测试集, 包含 246 个测试用例, 本次人机协同优化次数为 24 次, 优化比例为 10%。实验结果同样证明人机协同优化后的新模式可以提升在真实对象 Linux 内核测试集中的表现, 同时不影响在通用测试集上的效果。

表 3 CWE-119 在 Linux kernel-fs 模块验证实验结果
Table 3 CWE-119 experiment results in Linux kernel-net module

名称	测试数据集	*acc*	*recall*	*F1-score*
模式 1	Gen-dataset	0.86	0.83	0.867
模式 1	Fs-dataset	0.53	0.28	0.326
模式 2	Gen-dataset	0.90	0.91	0.914
模式 2	Fs-dataset	0.87	0.83	0.847

4.3 人机协同的协议语义不一致性漏洞挖掘

4.3.1 面临问题

协议不一致性漏洞是由于前端服务器(如缓存、代理、防火墙)和后端服务器(如负载均衡、代理、原点服务器)在解析同一个 HTTP 请求时对 HTTP 消息的语法有效性或缓存行为存在不一致解析, 从而导致如缓存中毒、安全策略绕过和拒绝服务攻击等严重的安全漏洞。

不一致性漏洞本质上是一种由协议语义造成的应用层面的风险, 已有的自动化分析方法对协议的理解能力较差, 难以直接发现此类高价值漏洞。已有研究往往基于人工分析来发现类似的语义差异漏洞, 效率低, 方法的通用性较差。

4.3.2 人机协同模式

针对该问题, 本文提出了一种人机协同的协议语义不一致性漏洞挖掘方法。

(1) 人机分工

人机协同协议语义不一性漏洞挖掘模式主要包括协议规则识别解析、测试用例生成与变异、协议交互分析 3 个环节, 如图 6 所示。其中, 人主要负责描述性规则定义、关键环节检查、漏洞综合研判; 机器主要做规则转化与生成、测试用例生成、测试交互和异常发现。

(2) 交互反馈

在该模式中, 人与机器的交互反馈如下:

机器将执行的结果提供给人(R), 包括:

1) 种子生成: 种子生成模块接受“人工操作”部分从 RFC 或程序源码中提取的扩充巴克斯范式(ABNF)或程序要求的输入格式, 构建测试用例生成需求的抽象语法树(AST)或结构描述; 接着, 根据构建的 AST 或结构描述中各节点的可选规则, 随机地递归生成符合语法、格式要求的测试用例。由测试用例生成器生成的测试用例称为“种子”。

2) 输入变异: 输入变异模块接受种子生成模块生成的“种子”, 在比特、字节、语法、结构等层级对“种子”进行随机变换, 得到不满足 ABNF 的输入, 从而发现被测程序因解析鲁棒性带来的安全问题; 同时, 变异模块能够接受执行监视模块返回的反馈信息, 指导“种子”向触发漏洞概率更高的方向变异, 从而更加高效地发现潜在的漏洞。

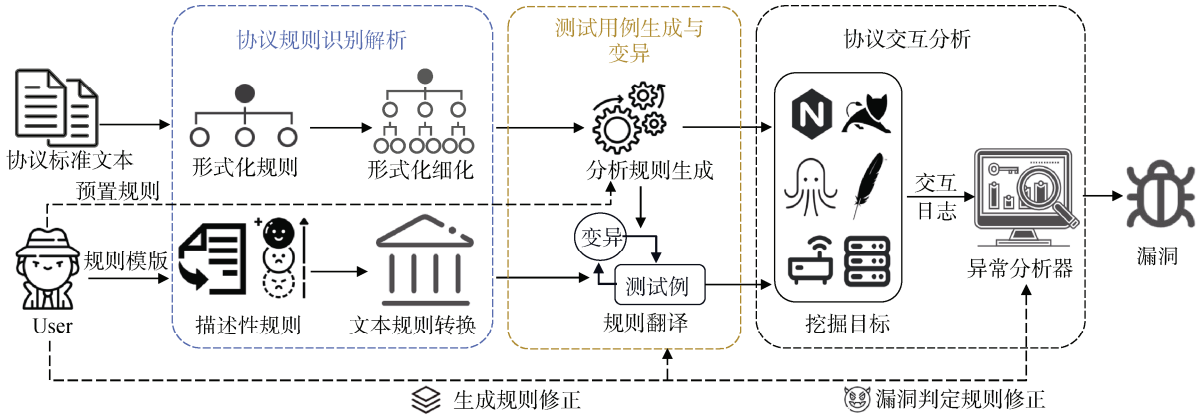

图 6 人机协同协议语义不一致性漏洞挖掘
Figure 6 Pattern of human-machine collaborative protocol analysis based on semantic inconsistency

3) 漏洞报告: 程序执行模块负责将测试用例变异器生成的测试用例发送至被测程序执行, 等待程序执行完成后获取被测程序的执行结果, 结合执行监视模块获得的运行数据, 判断输出结果是否触发了目标类型漏洞。当判定为漏洞触发时, 程序执行模块将自动生成一份初步的漏洞报告, 包括触发漏洞的测试用例、漏洞类型、漏洞触发路径等信息, 便于测试人员进行下一步的人工分析。

人将分析的结果反馈给机器(B), 包括:

1) 协议文本规则转换: 面向协议标准文本, 通过基于自然语言处理的测试规则提取将描述性规则进行文本规则转换和翻译, 能够跨越协议描述与规则的鸿沟, 将自然语言描述的各类协议语义, 转化为机器可识别的定义约束。

2) 漏洞综合研判: 一是监测分析状态, 即人需要基于测试交互状态来调整或判断是否结束测试工作; 二是在测试异常记录的基础上判定此类异常是否为一个高价值的漏洞。这一过程可能伴随着复现、手动测试等附加流程, 最终由人基于专家经验判定异常输入是否为高价值漏洞。

(3) 能力提升

在人与机器不断迭代交互的过程中, 语义不一致性漏洞挖掘的能力得到不断优化, 主要体现在:

用例生成的有效性以及高价值漏洞的研判两个部分。对于用例生成, 通过人与机器的不断迭代反馈, 在人的指导下机器能够生成更有可能触发漏洞的用例; 而在漏洞研判方面, 通过对机器提供的异常状态信息进行持续的监测分析, 使得人类专家对于高价值漏洞研判的能力得到提升。

4.3.3 实现与验证

根据上述思路, 本文实现了人机协同 HTTP 服务漏洞挖掘框架 HDiff, 包含了标准文档文本分析、测试模板生成、交互测试控制、测试结果分析等模块。利用该测试框架, 基于 RFC 7230-7235 标准的描述, 面向常见的 HTTP 服务和组件, 对请求伪造(SSRF)[57]、拒绝服务(DoS)[58]、审计绕过[59]等漏洞进行测试。除了发现 Tomcat 服务等已知漏洞外(如表 4 所示), 还新发现 ModSecurity 等 HTTP 服务中 7 个未公开漏洞, 测试结果如表 5 所示。

5 总结与展望

本文提出了人机协同漏洞挖掘的理念, 并深入研究了人机协同漏洞挖掘机制。首先构建了人机协同概念框架, 根据人与机器的关系以及协同程度, 提出工具、助手和伙伴 3 个层次的人机协同概念。在此基础上, 结合漏洞挖掘技术的发展趋势, 以及漏洞挖掘的内涵本质, 讨论了人机协同漏洞挖掘模型的分类, 并重点刻画了双向互助式人机协同漏洞挖掘模型。在该模型指导下, 针对典型场景下漏洞挖掘技术面临的 3 个挑战性问题, 分别提出了人机协同解决方案, 并通过对比分析和真实对象的漏洞挖掘, 验证了人机协同方法的有效性。

表 4 测试复现的 HTTP 实现漏洞

Table 4 Replicated HTTP implementations and vulnerability

序号	测试对象	漏洞类型	漏洞编号
1	Tomcat	审计绕过	CVE-2019-17569
2	Tomcat	SSRF	CVE-2020-1935
3	Microsoft IIS	审计绕过	CVE-2020-0645
4	Weblogic	SSRF	CVE-2020-2867
5	Weblogic	审计绕过	CVE-2020-14588
6	Weblogic	DoS	CVE-2020-14589
7	Apache Traffic Server	审计绕过	CVE-2020-1944

表 5 测试新发现的 HTTP 实现漏洞

Table 5 Tested HTTP implementations and vulnerability

序号	测试对象	漏洞类型	漏洞编号
1	rConfig	SSRF	CNNVD-202308-020
2	ModSecurity	审计绕过	CNNVD-202307-1175
3	ChatGPT	SSRF	CNNVD-202403-414
4	WonderCMS	SSRF	CNNVD-202403-417
5	JWCrypto	DoS	CNNVD-202403-765
6	Jose4j	DoS	CNNVD-202402-2688
7	Lestrrat-go	DoS	CNNVD-202403-775

本文在人机协同漏洞挖掘机制和技术方面做了初步探索, 人机协同漏洞挖掘还有许多值得关注的方向:

(1) 人机协同漏洞挖掘模式研究与实践

为了验证所提人机协同漏洞挖掘模型的可行性和有效性, 本文针对漏洞挖掘面临的 3 个挑战性问题从人机协同角度提出了解决方案。漏洞挖掘目标对象不同、技术方法差异, 面临的问题挑战也千差万别。静态分析误报率高、污点分析空间爆炸等许多问题, 都可从人机协同角度探索可能的解决途径。

(2) 人机协同漏洞挖掘模型的演进与应用

随着人工智能技术, 特别是 LLM 和群体智能技术的不断发展, 人与机器间更复杂的协作、融合以及机器的自我进化将成为可能, 人机协同漏洞挖掘模型也将随之不断发展。结合相关技术进展, 丰富完善

共融互促式人机协同模型的概念内涵, 并在实践中验证和优化也是人机协同漏洞挖掘领域重要的研究方向。

参考文献

[1] Zou Q C, Zhang T, Wu R P, et al. From automation to intelligence: Survey of research on vulnerability discovery techniques[J]. *Journal of Tsinghua University* (*Science and Technology*), 2018, 58(12): 1079-1094.
(邹权臣, 张涛, 吴润浦, 等. 从自动化到智能化: 软件漏洞挖掘技术进展[J]. *清华大学学报*(*自然科学版*), 2018, 58(12): 1079-1094.)

[2] Aschermann C, Schumilo S, Abbasi A, et al. IJON: Exploring Deep State Spaces *via* Fuzzing[C]. *2020 IEEE Symposium on Security and Privacy*, 2020: 1597-1612.

[3] Machiry A, Tahiliani R, Naik M. Dynodroid: An Input Generation System for Android Apps[C]. *The 2013 9th Joint Meeting on Foundations of Software Engineering*, 2013: 224-234.

[4] Shoshitaishvili Y, Weissbacher M, Dresel L, et al. Rise of the HaCRS: Augmenting Autonomous Cyber Reasoning Systems with Human Assistance[C]. *The 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 347-362.

[5] Zhou C J, Wang M Z, Liang J, et al. VisFuzz: Understanding and Intervening Fuzzing with Interactive Visualization[C]. *2019 34th IEEE/ACM International Conference on Automated Software Engineering*, 2020: 1078-1081.

[6] Bundt J, Fasano A, Dolan-Gavitt B, et al. Homo in Machina: Improving Fuzz Testing Coverage *via* Compartment Analysis[C]. *2023 IEEE Conference on Software Testing*, *Verification and Validation*, 2023: 117-128.

[7] Padhye R, Lemieux C, Sen K, et al. FuzzFactory: Domain-specific fuzzing with waypoints[J]. *Proceedings of the ACM on Programming Languages*, 2019, 3(OOPSLA): 1-29.

[8] King J C. Symbolic execution and program testing[J]. *Communications of the ACM*, 1976, 19(7): 385-394.

[9] Denning D E. A lattice model of secure information flow[J]. *Communications of the ACM*, 1976, 19(5): 236-243.

[10] Pnueli A. The Temporal Logic of Programs[C]. *18th Annual Symposium on Foundations of Computer Science*, 2008: 46-57.

[11] Clarke E M, Emerson E A. Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic[C]. *Logics of Programs*, 1982: 52-71.

[12] Li J, Zhao B D, Zhang C. Fuzzing: A survey[J]. *Cybersecurity*, 2018, 1(1): 6.

[13] Kaksonen R. A functional method for assessing protocol. Implementation security: Licentiate thesis[J]. *VTT Technical Research Centre of Finland*, 2001.

[14] Peach fuzzer. Michael Eddington[Z]. https://peachtech.gitlab.io/peach-fuzzer-community/. Jul. 2024.

[15] Livshits V B, Lam M S. Finding Security Vulnerabilities in Java Applications with Static Analysis[C]. *The 14th conference on USENIX Security Symposium - Volume 14*, 2005: 18.

[16] Myers G J. The art of software testing[M]. 1979.

[17] Clarke E, McMillan K, Campos S, et al. Symbolic Model Checking[M]. Computer Aided Verification, 1996: 419-422.

[18] Cimatti A, Clarke E, Giunchiglia F, et al. NuSMV: A New Symbolic Model Verifier[C]. *Computer Aided Verification*, 1999: 495-499.

[19] Sen K. Concolic Testing: A Decade Later (Keynote)[C]. *The 13th International Workshop on Dynamic Analysis*, 2015: 1.

[20] Jovanovic N, Kruegel C, Kirda E. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities[C]. *2006 IEEE Symposium on Security and Privacy*, 2006: 6263.

[21] Cadar C, Ganesh V, Pawlowski P M, et al. EXE: Automatically generating inputs of death[J]. *ACM Transactions on Information and System Security*, 2008, 12(2): 1-38.

[22] Cadar C, Dunbar D, Engler DR. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs[C]. *Usenix Conference on Operating Systems Design & Implementation*, 2009: 209-224.

[23] Godefroid P, Levin M Y, Molnar D A. SAGE: Whitebox fuzzing for security testing[J]. *Queue*, 2012, 10: 20-27.

[24] American fuzzy lop (2.52b). Michał Zalewski[Z]. https://lcamtuf.coredump.cx/afl/. Jul. 2024.

[25] Stephens N, Grosen J, Salls C, et al. Driller: Augmenting Fuzzing through Selective Symbolic Execution[C]. *Proceedings 2016 Network and Distributed System Security Symposium*, 2016: 1-16.

[26] Yun I, Lee S, Xu M, et al. QSYM: A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing[C]. *USENIX Security Symposium*, 2018.

[27] Huang H Q, Yao P S, Wu R X, et al. Pangolin: Incremental Hybrid Fuzzing with Polyhedral Path Abstraction[C]. *2020 IEEE Symposium on Security and Privacy*, 2020: 1613-1627.

[28] Kim K, Jeong D R, Kim C H, et al. HFL: Hybrid Fuzzing on the Linux Kernel[C]. *Proceedings 2020 Network and Distributed System Security Symposium*, 2020.

[29] Cyber Grand Challenge (CGC) (Archived). DARPA[Z]. https://www.darpa.mil/program/cyber-grand-challenge. Jul. 2024.

[30] Computers and Humans Exploring Software Security (CHESS) (Archived). DARPA[Z]. https://www.darpa.mil/program/computers-and-humans-exploring-software-security. Jul. 2024.

[31] Sun M L, Yang Y B, Wang Y, et al. SMT Solver Validation Empowered by Large Pre-Trained Language Models[C]. *2023 38th IEEE/ACM International Conference on Automated Software Engineering*, 2023: 1288-1300.

[32] Deng Y L, Xia C S, Yang C Y, et al. Large Language Models Are Edge-Case Generators: Crafting Unusual Programs for Fuzzing Deep Learning Libraries[C]. *The IEEE/ACM 46th International Conference on Software Engineering*, 2024: 1-13.

[33] Xia C S, Paltenghi M, Jia L T, et al. Fuzz4All: Universal Fuzzing with Large Language Models[C]. *The IEEE/ACM 46th International Conference on Software Engineering*, 2024: 1-13.

[34] Deng Y L, Xia C S, Peng H R, et al. Large Language Models Are Zero-Shot Fuzzers: Fuzzing Deep-Learning Libraries *via* Large Language Models[C]. *The 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023: 423-435.

[35] Lemieux C, Inala J P, Lahiri S K, et al. CodaMosa: Escaping Coverage Plateaus in Test Generation with Pre-Trained Large Language Models[C]. *2023 IEEE/ACM 45th International Conference on Software Engineering*, 2023: 919-931.

[36] Kang S, Yoon J, Yoo S. Large Language Models Are Few-Shot Testers: Exploring LLM-Based General Bug Reproduction[C]. *2023 IEEE/ACM 45th International Conference on Software Engineering*, 2023: 2312-2323.

[37] Zhang C, Zheng Y W, Bai M Q, et al. How Effective Are They? Exploring Large Language Model Based Fuzz Driver Generation[EB/OL]. 2023: arXiv: 2307.12469. https://arxiv.org/ abs/2307.12469.

[38] Rawat S, Jain V, Kumar A, et al. VUzzer: Application-Aware Evolutionary Fuzzing[C]. *Proceedings 2017 Network and Distributed System Security Symposium*, 2017: 1-14.

[39] Gan S, Zhang C, Chen P, et al. {GREYONE}: Data flow sensitive fuzzing[C]. *29th USENIX Security Symposium*, 2020: 2577-2594.

[40] Lyu C Y, Ji S L, Zhang X H, et al. EMS: History-Driven Mutation for Coverage-Based Fuzzing[C]. *Proceedings 2022 Network and Distributed System Security Symposium*, 2022.

[41] Lyu C, Ji S, Zhang C, et al. {MOPT}: Optimized mutation scheduling for fuzzers[C]. *28th USENIX Security Symposium*, 2019: 1949-1966.

[42] Yamaguchi F, Golde N, Arp D, et al. Modeling and Discovering Vulnerabilities with Code Property Graphs[C]. *2014 IEEE Symposium on Security and Privacy*, 2014: 590-604.

[43] Li J Y, Ernst M D. CBCD: Cloned Buggy Code Detector[C]. *2012 34th International Conference on Software Engineering*, 2012: 310-320.

[44] Lin G J, Zhang J, Luo W, et al. POSTER: Vulnerability Discovery with Function Representation Learning from Unlabeled Projects[C]. *The 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017: 2539-2541.

[45] Mangal R, Zhang X, Nori A V, et al. A User-Guided Approach to Program Analysis[C]. *The 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015: 462-473.

[46] Zhang X, Grigore R, Si X J, et al. Effective interactive resolution of static analysis alarms[J]. *Proceedings of the ACM on Programming Languages*, 2017, 1(OOPSLA): 1-30.

[47] Li Z Y, Machiry A, Chen B H, et al. ARBITRAR: User-Guided API Misuse Detection[C]. *2021 IEEE Symposium on Security and Privacy*, 2021: 1400-1415.

[48] Böhme M, Pham V T, Roychoudhury A. Coverage-Based Greybox Fuzzing as Markov Chain[C]. *IEEE Transactions on Software Engineering*, 2017: 489-506.

[49] Burrows M, Abadi M, Needham R. A logic of authentication[J]. *ACM Transactions on Computer Systems*, 1990, 8(1): 18-36.

[50] Datta A, Derek A, Mitchell J C, et al. Protocol composition logic (PCL)[J]. *Electronic Notes in Theoretical Computer Science*, 2007, 172: 311-358.

[51] Lowe G. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR[C]. *Tools and Algorithms for the Construction and Analysis of Systems*, 1996: 147-166.

[52] Gorbunov S, Rosenbloom A. Autofuzz: Automated network protocol fuzzing framework[J]. *IJCSNS*, 2010, 10(8): 239.

[53] Cho C Y, Babic D, Poosankam P, et al. MACE: Model-Inference-Assisted Concolic Exploration for Protocol and Vulnerability Discovery[C]. *USENIX Security Symposium*, 2011.

[54] Sun X H, Zhang Y W, Qin J X, et al. Review on human-intelligent system collaboration[J]. *Packaging Engineering*, 2020, 41(18): 1-11, I0005.
(孙效华, 张义文, 秦觉晓, 等. 人机智能协同研究综述[J]. *包装工程*, 2020, 41(18): 1-11, I0005.)

[55] Yang Y, Xiong G, Wang K R. Application of artificial intelligence in computer network security-research on AI2 system of MIT and PatternEx company[J]. *Research on Telecommunication Technology*, 2018(2): 55-65.
(杨易, 熊钢, 王科人. 人工智能在计算机网络安全中的应用-麻省理工学院与 PatternEx 公司 AI2 系统研究[J]. *电信技术研究*, 2018(2): 55-65.)

[56] OSS-Fuzz: Continuous Fuzzing for Open Source Software[Z]. Google. https://github.com/google/oss-fuzz. Jul. 2024.

[57] Chen J J, Jiang J, Duan H X, et al. Host of Troubles: Multiple Host Ambiguities in HTTP Implementations[C]. *The 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016: 1516-1527.

[58] Shen K W, Lu J Y, Yang Y R, et al. HDiff: A Semi-Automatic Framework for Discovering Semantic Gap Attack in HTTP Implementations[C]. *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2022: 1-13.

[59] A New Era of SSRF - Exploiting URL Parser in Trending Program-ming Languages[Z]. Orange Tsai. https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf. Jul. 2024.

林哲超 于 2015 年在国防科技大学软件工程专业获得硕士学位。现任信息系统安全技术重点实验室工程师。研究领域为程序分析、网络安全。Email: acbuwa@foxmail.com

卢帅兵 于 2016 年在解放军信息工程大学计算机科学与技术专业获得硕士学位。现任信息系统安全技术重点实验室工程师。研究领域为软件安全、漏洞挖掘。Email: ohonice@163.com

聂原平 2017 年毕业于国防科技大学计算机学院获得博士学位。现任信息系统安全技术重点实验室工程师。研究领域为人工智能、网络安全。Email: yuanpingnie@nudt.edu.cn

张甲 于 2010 年在清华大学计算机科学与技术系获得博士学位。现任清华大学网络科学与网络空间研究院副研究员。研究领域为网络安全检测、地下产业检测。Email: zhangjia@cernet.edu.cn

段海新 CCF 会员, 于 2001 年在清华大学计算机科学与技术系获得博士学位。现任清华大学网络科学与网络空间研究院长聘教授、CCF 会员。研究领域为网络协议安全、互联网基础设施安全、互联网治理。Email: duanhx@tsinghua.edu.cn

李响 于 2008 年在解放军信息工程大学计算机软件与理论专业获得硕士学位。现任信息系统安全技术重点实验室高级工程师。研究领域为网络安全、人工智能安全。Email: ideal_work@163.com

况晓辉 于 2003 年毕业于国防科技大学计算机学院获得博士学位。现任信息系统安全技术重点实验室研究员。研究领域为网络空间安全、人工智能安全。Email: xhkuang@bupt.edu.cn