网络协议模糊测试综述

徐 威[1†], 李 鹏[2], 张文镔[1], 陆 丽[3]
(1. 数学工程与先进计算国家重点实验室, 郑州 450001; 2. 北京计算机技术及应用研究所, 北京 100854; 3. 网络空间安全技术国家地方联合工程实验室, 郑州 450001)

摘 要: 模糊测试作为最有效的漏洞挖掘方法,在网络协议安全检测领域应用广泛,并且诞生了大量的研究成果。目前还没有工作对协议模糊测试技术进行全面的总结和分析。首先回顾了协议模糊测试的发展历程,按照其工作流程(预处理,种子生成,执行过程监测以及结果分析)总结现有协议模糊测试所做的工作以及不足;之后,分析了不同环境下协议模糊测试技术的挑战以及解决方法;最后讨论了当前协议测试方法存在的局限性,以及未来的发展方向。

关键词: 协议; 模糊测试; 漏洞挖掘; 工作流程; 固件设备安全
中图分类号: TP309.2 **文献标志码**: A **文章编号**: 1001-3695(2023)08-001-2241-09
doi:10.19734/j.issn.1001-3695.2022.12.0794

Survey of network protocol fuzzing

Xu Wei[1†], Li Peng[2], Zhang Wenbin[1], Lu Li[3]
(1. *State Key Laboratory of Mathematical Engineering & Advanced Computing*, *Zhengzhou* 450001, *China*; 2. *Beijing Institute of Computer Technology & Applications*, *Beijing* 100854, *China*; 3. *National Engineering Laboratory for Cyber Science & Technology*, *Zhengzhou* 450001, *China*)

Abstract: As the most effective vulnerability discovering method, fuzzing testing has been widely used in the field of network protocol security detection and has given birth to a large number of research results. No work has been done to provide a comprehensive summary and analysis of protocol fuzzing testing techniques. Firstly, this paper reviewed the development of protocol fuzzing testing and summarized the work done by existing protocol fuzzing testing and its shortcomings according to its workflow: preprocessing, seed generation, execution process monitoring, and result analysis. After that, it analyzed the challenges of protocol fuzzing testing techniques in different scenarios and the solutions. Finally, it discussed the limitations of the current protocol testing methods and the future development directions.

Key words: protocol; fuzzing; vulnerability discovering; working process; firmware device security

网络协议描述了两个通信实体相互传递数据的规范,在计算机网络中发挥着重要的作用。然而,在实现过程中由于开发人员理解上的偏差,会在其实现中引入漏洞。一些黑客利用协议中的漏洞传播病毒对互联网中的设备发起远程攻击。例如,2017 年 WannaCry 病毒感染了全球超过 10 万台主机,造成了 80 亿美元的经济损失[1]。该病毒正是利用了 SMB 协议中的 MS17-010 漏洞在全球范围大肆传播。本文根据 NVD 数据库进行统计,最近半年协议中高危漏洞占比超过 70%,因此及时发现并修补协议中存在的安全漏洞极为重要。

模糊测试自 1989 年提出以来[2],经过 30 多年的发展已经成为一种广泛使用的漏洞检测方法。模糊测试通过发送大量随机的输入,检测程序中存在的漏洞。据 Google[3] 一项调查显示,超过 37% 的漏洞是通过模糊测试发现的。模糊测试作为目前最常见、最有效的漏洞检测技术,在协议的安全测试领域得到了广泛的应用。2001 年诞生的首个实用性模糊测试工具 PROTOS[4] 就是针对协议开发的,这也说明协议是模糊测试的重要目标。在此之后,协议模糊测试一直作为网络和信息安全的热点领域出现了大量的研究成果,其中具有代表性的工作包括 Peach[5]、Sulley[6]、AFLNet[7] 等。然而,现有的综述文章要么缺乏最近的研究工作[8],要么分析得并不全面[9],因此需要对协议模糊测试领域取得的成果进行重新梳理和总结。本文从以下四个方面对协议模糊测试研究领域进行总结:a) 以具有代表性的工具诞生时间节点,分析回顾协议模糊测试发展历程,比如 AutoFuzz[10] 以自动化的方式对协议进行模糊测试,在此之前的研究需要大量的手工工作以及待测协议的先验知识;b) 以协议模糊测试的流程为脉络,针对每个环节总结出其面临的挑战以及现有的研究成果解决这些问题采用的方法;c) 在不同的应用场景下,比如物联网、工业控制系统开发协议有着独特的要求,对这些场景下的协议进行模糊测试取得的成果进行总结;d) 分析目前协议模糊测试技术存在的不足,并指出未来的研究方向,最后对全文进行总结。

1 协议模糊测试发展历程

自 PROTOS 诞生以来,协议模糊测试技术经过 20 多年的发展,出现了大量的研究成果。如图 1 所示,本文根据具有重要意义研究成果出现的时间,对协议模糊测试工具发展历程进行了梳理。其中以 AFLNet 的诞生作为划分节点,在此之前协议模糊测试主要采用黑盒的方法,之后灰盒协议模糊测试技术出现了大量的研究成果。

1.1 黑盒协议模糊测试阶段

黑盒模糊测试技术仅考虑程序的输入以及输出,不关心执

收稿日期: 2022-12-03; **修回日期**: 2023-02-16

作者简介:徐威(1997-),男(通信作者),河南周口人,硕士,主要研究方向为网络协议与模糊测试(1150220930@qq.com);李鹏(1981-),男,吉林靖宇人,高级工程师,博士,主要研究方向为计算机软件密码安全;张文镔(1995-),男,河南南阳人,博士研究生,主要研究方向为协议安全;陆丽(1983-),女,江苏涟水人,本科,主要研究方向为网络安全.

行过程中内部的状态信息。根据输出结果以及执行输入后程序是否正常运行来判断目标的状态。黑盒模糊测试技术具有以下优势:a)不需要程序的源码,这对协议来说尤为重要,因为在工控以及互联网场景下有大量的私有协议;b)设计相对简单,便于研究人员开发;c)因为其不考虑目标程序的内部状态信息,执行速度较快。

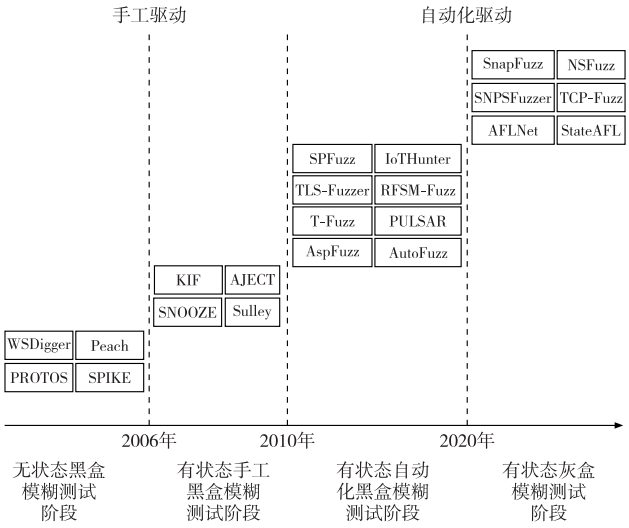

图 1 协议模糊测试关键阶段

Fig. 1 Key stages of protocol fuzzing

2001—2005 年,协议模糊测试发展初期没有考虑被测协议复杂的状态信息,模糊器发送完全随机的输入。这个阶段出现的代表性协议模糊测试工具有 PROTOS[4]、SPIKE[11]、WSDigger[12]、Peach[5]等。

PROTOS 由奥尼大学的安全团队开发,根据协议规范生成错误的输入触发特定的漏洞,如格式化字符串、缓冲区溢出漏洞。2001 年 Dave 开发的 SPIKE 框架提供了一系列允许快速和高效开发网络协议模糊测试器的 API。SPIKE 将协议分组为不同的块,每个块都可以单独进行模糊测试。但是,由于 SPIKE 提供的块抽象过于简略,开发人员难以轻松地建模有状态协议和复杂消息及其依赖关系。此外,基于 SPIKE 的模糊器必须使用 C 语言编写,因此开发过程可能比使用更复杂。WSDigger 是 2003 年开发的一款用于测试 Web 服务的黑盒模糊测试工具,它将 Web 服务的 WSDL 文件作为输入,并使用专门制作的有效负载测试特定的 Web 服务。2004 年 Michael 开发出 Peach,通过手动构建 pit 文件定义生成的数据模型。时至今日,Peach 仍是工业界广泛使用的一款模糊测试工具。

2006—2019 年,研究人员开始通过构建协议的状态机模型,开发有状态的协议模糊测试工具。根据协议状态机简洁地表示了协议实现的关键状态和转换条件,可用于系统地指导模糊测试的过程,探测深层次的漏洞。

2006 年,Banks 等人[13]构建了第一个有状态的黑盒协议模糊测试工具 SNOOZE。SNOOZE 需要测试人员从 RFC 文档中手动提取协议规范,进而得到协议的字段特征、协议中信息交换的语法以及状态机。测试人员根据状态机发送特定序列的消息到达待测定的状态,在该状态下根据协议规范生成大量随机的输入测试目标程序中的漏洞。但是,SNOOZE 在开启一个模糊测试进程之前,需要有经验的研究人员花费大量的精力完成协议规范提取工作。同年,Neves 等人[14]开发了 AJECT,依据应用程序层协议规范生成符合格式要求的测试报文。该规范描述了状态的定义、迁移路径以及可以在每个状态下发送消息的语法。其输入基于启发式生成,以查找特定的漏洞,如缓冲区溢出、格式字符串漏洞等。在每一个状态下,AJECT 都会生成能被协议程序接受但参数无效的测试报文。

Sulley 是广泛使用的网络协议模糊测试框架,目前已停止维护,BooFuzz[15]继承了它的工作。Sulley 具体包括数据定义、数据驱动、会话管理、监视代理、辅助工具和文件系统六大模块。测试人员只需关注数据生成模块中测试报文模板的编写,其他部分都可以自动完成。

2007 年 Miki 等人[16]开发了 KIF,以解决多数模糊器仅依赖随机注入生成测试数据,但没有考虑目标协议程序的语法、语义以及状态问题。KIF 通过一个无上下文的 ABNF[17]描述协议的确切语法。此外,基于捕获到的通信流量,利用机器学习的方法,构建协议状态机用于模拟系统中的状态转换。KIF 是第一个能够生成复杂随机输入的 SIP 协议模糊器。

上述模糊器都依赖研究人员从协议规范中提取协议格式以及状态机模型,生成符合格式的输入。此外 SNOOZE、KIF 需要突变字段以及发送特定的报文序列到达测试场景。这种方式需要研究人员过多地参与且模糊测试的效率较低。为了解决 KIF 等协议模糊测试工具目标单一,严重依赖对协议的先验知识和大量手工操作的问题。研究人员尝试使用自动化的方式推断协议格式以及状态机模型,指导模糊测试的执行。

2010 年 Gorbunov 等人[10]开发了 AutoFuzz,通过充当客户端与被测试程序的中间人,获取通信流量,在此基础上自动化地推断协议规范。AutoFuzz 使用基于 Needleman 算法[18]的 PI 工具[19]将静态数据字段和动态数据字段分开,并将动态数据字段与类型和长度信息关联起来获取语法信息,构建通用报文序列。2012 年文献[20]采用类似的方法,从获取的网络流量中推断出协议的报文格式以及状态模型,检测协议中的漏洞。2014 年文献[21]开发了 T-Fuzz,并将其与 TTCN-3 集成。T-Fuzz 通过模型提取器获取 TTCN-3 中定义的模型,区分报文中的静态字段以及可变字段。之后,通过模糊测试引擎将可变字段设置成特殊值进行测试。Ma 等人[22]通过对协议规范进行分析,构建有规则的协议状态机。与之前的工作不同,在构建完成后不对所有的状态迁移路径进行测试,而是首先进行一个判断:如果从一种状态到另一种状态的转换过程中不会发生异常,那么该状态迁移路径是安全的。安全路径对于漏洞探测是没有意义的,因此可以将其忽略,从而提高模糊测试的效率。2015 年文献[23]等人使用 LearnLib 库,通过改进的 Angluin's L^* algorithm 算法[24]从 TLS 协议中实现自动提取状态机。通过分析这些状态机,可以发现协议流中的逻辑缺陷。2018 年文献[25]基于文献[26,27]的工作,利用自然语言处理技术从 RFC 文档中自动获取协议状态机。

上述工作减少了研究人员的手工工作,在一定程度上提升了自动化水平。然而这种方式的准确性以及可扩展性仍需进一步提高。此外,对私有协议进行模糊测试时,由于缺乏协议的描述文档,需要通过逆向的方法获取其协议格式以及状态转换模型。所以,上述通过 RFC 文档构建协议状态机模型的方法无法直接适用于私有协议。

2015 年 Gascon 等人[28]将协议逆行技术与模糊测试技术相结合,构建了一款针对私有协议的模糊测试工具 PULSAR。PULSAR 采用一种基于概率的方法,从获取到的流量中自动提取协议格式以及一个二阶马尔可夫模型,该模型提供了一个真实状态机的概率近似。PULSAR 以字节为单位推断协议格式,更适用于文本类协议的模糊测试,对二进制协议逆向结果不够准确。2017 年文献[29]将 bit 作为最小的单元推断协议的格式,实现了一个通用的协议模糊测试框架 Bbuzz。Bbuzz 可以快速分析出协议的特征,自动生成测试用例以及自动进行模糊测试。

1.2　灰盒协议模糊测试阶段

2020 年诞生了第一个灰盒协议模糊测试工具 AFLNet,之后出现了许多类似的研究成果,如 StateAFL[30]、SNPSFuzzer[31]、SnapFuzz[32]等。相较于黑盒模糊测试技术,灰盒模糊测试技术一方面省去了协议格式推断以及协议状态模型的构建过程;另一方面,通过反馈信息指导测试用例的生成,并且可以保留有价值的种子对其进行进一步测试,提高测试效率。2020 年文献[7]在 AFL[33]的基础上实现了第一个有状态的灰盒协议模糊测试工具 AFLNet。AFLNet 捕获客户端与服务端通信时的流量作为初始种子,在模糊测试过程将状态码以及代码覆盖率作为反馈信息,保留有价值的种子对其进行进一步测试。AFLNet 将响应中的状态码作为状态信息,指导模糊器探索协议的状态空间。之后出现了大量的研究工作,对 AFLNet 存在的不足进行改进。

如前文所述,AFLNet 依赖响应报文的内容推断协议状态,因此必须使用特定的解析器从响应中提取状态码。对于响应中不包含状态码的协议,AFLNet 无法推断出其状态信息进行有效的模糊测试。Natella 在 AFLNet 的基础上设计并实现了 StateAFL,通过适用性更强且粒度更细的长生命周期变量识别网络协议的状态。Liu 等人[34]认为不是所有的协议状态都同等重要,并且模糊测试的时间有限,需要一个有效的状态选择算法优先考虑渐进的状态。为此在 AFLNet 的基础上实现了 AFLNetLegion 算法,对协议的状态进行评估。2022 年文献[31]针对 AFLNet 通信速度慢,深层次状态无法直接到达,以及难以覆盖到深层的状态空间的问题,在其基础上设计并实现了 SNPSFuzzer。SNPSFuzzer 保存了网络协议程序的进程上下文(即进程快照),以便在模糊器第一次发送前缀消息后进行模糊处理。当需要模糊一个特定的状态时,只需要恢复快照,并发送随后的突变消息。此外,设计了一种消息链分析算法来探索更深层次的协议状态。SnapFuzz 将缓慢的异步网络通信转换为基于 UNIX 域套接字的快速同步通信,并且加快所有文件操作重定向到内存文件系统的过程,提高灰盒协议模糊测试的速度。

2　协议模糊测试工作流程

协议模糊测试如图 2 所示,可以分为预处理、种子生成、执行过程检测、结果分析四个步骤。在协议模糊测试中预处理主要是为了获取规范或者对源码进行插装。种子生成是协议模糊测试中最为关键的一个步骤,因为不符合规则的种子会在开始阶段被拒绝。依据种子生成方式的不同有基于生成以及基于突变两种方式。灰盒模糊测试中,根据执行过程中的反馈信息得到被测程序的当前状态,保留有价值的种子以及调整输入队列中种子的优先级,节省模糊测试的时间;另一方面,存储输出的异常信息以及造成此次异常的测试用例。结果分析阶段则是对测试过程中出现的异常行为进行调试,确定漏洞成因。

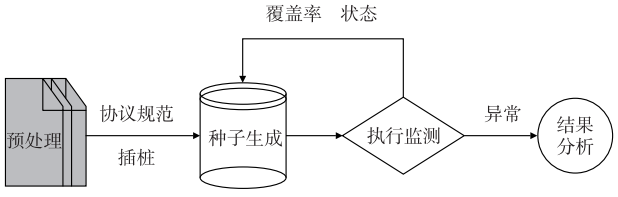

图 2　协议模糊测试工作流程

Fig. 2　Protocol fuzzing workflow

2.1　预处理

预处理有以下两种目的:a)通过协议文档手动提取、协议逆向方式自动获取和协议实现动态获取三种方式获取协议的格式以及状态转换模型;b)在灰盒协议模糊测试中,对程序进行插桩获取反馈信息。

2.1.1　获取协议规范

1)从协议规范中获取

协议模糊测试初期,研究人员针对特定的协议开发模糊测试工具。SNOZZE、AJECT、KIF 等工具从该协议的公开文档中手动提取协议格式以及状态转换模型。这种方法需要研究人员花费大量的时间,并且不适用于没有公开文档的私有协议。

为了克服手动提取带来的人工开销,研究人员开始关注通过自动化的方式,从协议文档中获取协议规范。这种自动化的方法往往与机器学习、人工智能相结合,尽量减少人工参与。比如,AspFuzz[35]使用特殊语言描述 RFC 文档,获取协议格式以及状态转换模型。文献[27]将自然语言处理技术应用于协议规范的提取,并使用零次学习(zero-shot learning,ZSL)提高准确率,还可以方便适用于不同的协议。上述方法在一定程度上增加了自动化的程度,然而,这些方法仅适用于具有协议规范文档的开源协议。

2)通过协议逆向技术自动获取

协议逆向技术不依靠文档规范,以自动化的方式获取协议格式以及状态转换模型。协议格式逆向过程中,首先要使用 Wireshare、Tcpdump 等嗅探工具作为辅助捕获大量的网络流量,并且,这些流量要尽可能覆盖协议的不同状态,这样才能够对程序进行充分的测试以保证逆向结果的准确性。在逆向时,通常按照报文聚类、序列对比或者概率推断的过程获取协议格式以及协议状态机模型。AutoFuzz、PULSAR 以及 Bbuzz 都是采用协议逆向的方法通过网络流量获取协议的格式以及状态转换关系。AutoFuzz 以及 PULSAR 更适用基于文本的协议,Bbuzz 以 bit 为基本单元,针对二进制协议的逆向结果更加准确,并且具有良好的拓展性。张蔚瑶等人[36]基于协议特征库推断出协议格式,并且使用循环比对法识别语义信息,提高逆向结果的准确性。

上述通过协议通信流量获取到的协议适用性更高,但有部分研究工作采用基于指令序列的方法获取协议格式结果。Zhang 等人[37]使用指令追踪的方法构建会话图表示协议状态转换。然后,通过展开策略将其转换为有向无环图,避免模糊测试器长时间陷入一个循环。这种方式能够在协议逆向中获取到更为准确的结果,不过指令追踪需要有良好的执行环境,这在测试嵌入式设备时往往是不可行的。

3)通过协议实现动态获取

动态获取的方式可以分为一致性测试以及构建测试驱动程序两种方式。采用一致性测试的工具,比如 RFSM-Fuzz[38]通过构建唯一输入输出序列,确定完整的协议状态转换模型;另一种方式则是通过协议实现程序的源码,构建测试驱动程序。此外,这种方式还需提供一个可以发送到被测试的服务(service under test,SUT)的消息列表(也称为输入字母表),以及一个将 SUT 重置为其初始状态的命令。构建的测试驱动程序能够将输入的字符解析为可以发送到测试服务的具体报文。依据 SUT 的响应作出假设,并给出假设的判定条件,尽可能地推断出准确的协议状态机。如 TLS-Fuzzer[23]、DTLS-Fuzzer[39]使用 Angluin's L^* algorithm 算法的改进版本 LearnLib 获取 TLS 协议状态机。LearnLib[40]首先依据 SUT 接收到的响应为状态机作出假设,之后验证这些假设与实际状态机的等价性。如果模型不等价,则返回一个反例,并重新定义假设。在进行等价检查时,LearnLib 使用 W-method 的改进版本为等价检查指定一个深度。该算法只寻找不超过深度上界的反例轨迹,如果找不到,则认为状态机的当前假设与已实现的假设等价。

LearnLib 使用的方法可以在一定程度上避免状态空间爆炸问题,得到尽可能近似的结果。然而,它需要提前假定一个 teacher 角色,此步骤在实施过程中通常比较困难。

相比较而言,通过一致性测试确定唯一输入输出序列,其相关理论更为成熟,有着很高的路径覆盖率,可以确保结果的正确性,但是需要大量的测试时间。构建测试驱动程序的方式,自动化的程度更高,但是需要对协议程序源码有着很好的理解,并且仅能得到近似的结果。

除了上述方法外还有一些其他的方法,如 T-Fuzz 通过集成的 TTCN 框架自动提取协议模型;Han 等人[41]使用脚本将实际的通信流量自动转换为协议和关系描述格式;SulleyEX[42]使用预定义的格式描述状态格式,之后通过深度优先算法来获取协议的状态迁移路径。SulleyEX 解决了状态机中存在循环,造成的状态空间覆盖低的问题。然而,这种方法要求研究人员掌握待测协议的先验知识,并且需要一定的手工操作。

表 1 对不同的协议规范获取方式进行了总结:通过文本规范手动获取的方式,推断出的结果可能更加准确。然而,这种方法不仅需要掌握协议的先验知识,还会消耗研究人员大量的精力;通过自然语言处理获取的方式,在一定程度上提高了自动水平,但其准确性有待进一步的提升。此外,以上两种方法仅支持公开的协议。通过协议逆向技术获取,自动化程度更高,能够同时支持开源协议与私有协议,但是逆向结果不够准确或者执行条件比较严苛;通过协议实现程序动态获取,需要在理解程序源码的基础上编写 harness,进而推断出与实际情况近似的协议状态机。执行一致性测试构建唯一输入输出序列能够得到更加准确的结果,但是执行时间较长。

表 1 协议规范获取方法总结

Tab. 1 Protocol specification acquisition method summary

获取方法	工具	支持类型	对比分析
通过协议文档获取	SNOZZE[13]、KIF[16]等	开源协议	从协议文档中手动获取,推断的结果准确度高,但依赖研究人员的经验
	AspFuzz[35]	开源协议	使用特殊语言描述 RFC 文档,实现了一定的自动化,结果不够准确
	文献[25]	开源协议	将自然语言处理技术应用于协议规范的提取,准确性较高
通过协议逆向技术自动获取	AutoFuzz[10]	文本类协议	其只利用协议头部报文 4 Byte 进行聚类,存在较多冗余,序列对齐结果并不准确
	PULSAR[28]	通用	使用二阶马尔可夫链推断协议状态机,但更适合文本类协议,二进制协议推断结果不够准确
	Bbuzz[29]	通用	以 bit 为基本单位,更适合二进制类协议的格式推断
通过协议实现动态获取	RFSM-Fuzz[38]	开源协议	通过构建唯一输入输出序列,得到协议状态模型,结果更准确,需要花费的时间长,以及一定的人工参与
	TLS-Fuzzer[23] DTLS-Fuzzer[39]	开源协议	自动化程度较高,但是需要基于协议程序源码的理解手动构建 harness,推断到近似的结果

2.1.2 插桩

对于灰盒模糊测试工具来说,在测试程序执行之前需要对源代码进行插桩获取程序执行过程中的反馈信息,如路径覆盖率信息、内存信息、函数变量取值等。

根据反馈信息的不同,插桩方式也不同,以 AFLNet 为代表的工具,沿用了 AFL 的插桩思想,采用静态的方式对程序源码进行插桩。这种插桩方式仅仅可以统计代码的覆盖率信息,无法直接获取协议的状态信息。后来的研究人员发现,网络服务应用程序都是基于网络事件循环,当接收到客户端发来的消息时,网络事件循环对该消息进行处理,完成后,重新等待新消息的到来。在此循环过程中,长生命周期变量的更新在一定程度上能够反映程序的状态,而短生命周期变量仅在循环内有效,因此可以使用更加容易获取的长生命周期变量来表示状态。使用这种方式时,为了获取长周期变量,需要在编译时进行插桩。此类协议模糊测试工具不仅在代码分支处插入原生 AFL 提供的用于统计覆盖率信息的桩代码,还在内存分配/释放处插入一套新设计的桩代码。因此,这种方式也引入了昂贵的插桩开销以及计算开销。NSFuzz[43]通过使用轻量级的状态变量来表示更细粒度的状态,降低了插桩带来的开销,提高了执行效率。

2.2 种子生成

根据种子生成方式的不同可以将协议模糊测试分为基于生成的和基于变异的两类。基于生成的模糊测试方法根据预处理获取的协议规范,生成尽可能符合要求的种子。基于突变的模糊测试方法,从一个合法的样本出发,通过变异算法不断地修改其中一些数据,生成一批畸形的种子。

1)基于生成的方法 基于生成的模糊测试主要是对目标程序或协议的规范有了一定掌握的前提下使用的。协议格式具有高度结构化的特点,因此基于生成的方法构建的测试用例更可能被测试目标所接受。基于生成的方法可以手动构建输入模型,或者自动化地生成模板构建测试用例。比如,ROTOS 使用最小化模型的方法,根据协议格式以及消息发送顺序,针对特定目标中格式化字符串或者栈溢出等特定类型的漏洞构建测试模型。然后,根据此模型启发式地生成测试样本。与 PROTOS 不同,SPIKE 利用提供的编程接口来对样本格式进行约束,使用特定的源语测试应用程序。

上述两种工具都无法灵活地生成种子,并且实现的算法复杂度很高,此外也没有考虑到相邻报文间字段的关系,如序号、cookie 等语义信息。之后的研究人员采用手动编写脚本文件的方式,定义协议的格式以及状态转换关系,提高了测试用例生成的速度,并且这些测试用例更符合协议规范。

Peach、Sulley、BooFuzz 等工具基于上述方法,在生成器中事先编写脚本文件对样本格式进行约束,生成初始种子。之后,变异器针对不同的数据类型采用不同的变异策略获取大量的测试用例。EXT-NSFSM[44]、T-Fuzz、PULSAR、SulleyEX 等工作,依据推测出的协议规则构建输入模板,填充模板中的相应字段生成初始种子。确保测试用例生成过程中不会随意突变,提高了测试样例被目标程序接受的概率。但是也正因为模板化的变异规则,导致突变的畸形度不足,有些潜在的漏洞无法触发。为了获取到的种子在符合格式之外有着更高的畸形度,以达到提高发现漏洞几率的效果,AspFuzz 根据模板构建种子之外,还会向服务器发送异常的消息序列进行模糊测试。为了获取更高效的种子,Ma 等人使用基于规则的状态机和有状态规则树生成测试用例。使用结构变异算法定期突变初始测试用例,生成大量状态消息路径相同但是初始值不同的种子,构成最后的测试数据集。

2)基于突变的方法 基于突变的模糊测试方法通过随机变异目标程序正常的输入,生成新的测试用例对目标程序进行测试。这种方法不需要掌握协议的先验知识,避免了格式推断以及状态机构建工作,可以快速地开启一个模糊测试进程。

RATM-Fuzz 使用客服端与服务器之间的流量作为初始输入,使用自动化脚本将其转换为 PRDF 格式。之后,通过 RATM 模型表示协议字段的关系,对初始输入进行变异。这种突变方式能够快速地生成大量测试用例,但是往往会由于随机的突变严重破坏测试用例的结构,导致被测试程序拒绝。

一些研究工作如 SPFuzz[45],将上述方法与 AFL 的变异引擎相结合,制定了一个包括头部、内容和序列突变策略在内的三级突变策略,在能够尽可能满足协议规范的同时提高代码覆盖率。以上的突变方式需要大量的输入,并且这些输入应当覆盖协议的不同功能,这样才可能测试到协议的全部状态。为此 SECFuzz[46] 提出协议中的输入与服务端的配置相关,可以通过修改配置文件或者修改命令行参数的形式获取到大量的初始种子。之后,使用模糊器的变异算子对种子进行变异得到新的测试用例。

文献[47]针对 TLS 协议库开发出 TLS-attacker,允许测试人员创建自定义 TLS 的报文序列并使用简单的接口任意修改报文内容。这种方式可以检测到加密失败以及缓存区溢出漏洞。然而,TLS-attacker 适用对象单一,仅支持 TLS 协议,并且无法对客户端进行测试。上述方法可以快速地生成大量测试用例,能够适用于不同的执行环境。然而,由于这些工具没有收集程序执行过程中的反馈信息,造成测试存在一定的盲目性,很难到达深层次的状态空间。此外,无法保留有价值的测试用例对其作进一步测试。

AFLNet 及其改进工作 StateAFL、NSFuzz 等灰盒协议模糊测试工具,同样使用客户端与服务端交互的正常流量作为初始输入。不过这些模糊测试工具在进行测试之前需要对协议程序的源码进行插桩,从而在执行过程中获取反馈信息。在测试时,如果一个种子能够提升代码覆盖率或者进入一个新的状态,那么这个测试用例将会被保留,并再次对其进行测试;反之,无效的测试用例会被直接剔除。保留下的测试用例根据不同的调度算法进行优先级排序。在 AFLNet 中采取和 AFL 相同的度量方式,将更小并且执行时间更短的种子排在输入序列的前面。这样的方式可以将有限的测试时间尽可能多地用在价值更高的输入上,提升触发潜在漏洞的几率。为了探测到深层次的状态空间,SNPSFuzzer 赋予能够到达深层次路径测试用例更高的优先级。

对种子进行突变的同时,还需考虑新生成的测试用例是否符合协议规范。胡志濠等人[48]利用准循环神经网络学习协议的格式以及语义,过滤无效的测试用例。SGPFuzzer[49]引入了多种突变操作符,对所选的种子文件进行简单突变和结构化突变,具体包括消息序列突变、消息突变、二进制字段突变以及变量字符串突变。在测试队列调度方面使用的方式和 AFLNet 相同。基于生成的方法,依据预处理后获取的协议格式指导测试用例的生成。这种方式获取的测试用例有利于通过语法检查,到达协议程序内部的功能代码。然而,这种方法一方面需要研究人员花费大量的时间从公开文档中提取协议格式;另一方面不利于扩展,测试不同的协议程序时,需要重新进行格式推断。使用预定义的模板生成测试用时,能够快速生成大量符合格式的测试用例。但是采用这种方式生成的测试用例灵活性较差,难以触发条件比较严苛的漏洞。协议逆向技术降低了格式推断以及状态机模型构建所需的时间,然而测试用例生成的质量依赖逆向结果的准确性,目前仍有很大的提升空间。基于突变的方法通过突变客户端与服务端交互过程中产生的流量,自动化地生成测试用例。但是,由于突变的随机性,往往会产生大量无法通过格式校验的冗余测试用例。基于突变的灰盒模糊器需要插装才能获取所需的反馈信息,这在一定程度上增加了开销。此外,对于有状态的协议,深层的状态空间需要一个特定的合法输入序列才可到达,因此通常难以覆盖到。表 2 对上述研究的工作进行了总结。

表 2　种子获取方法概述

Tab. 2　Seeds acquisition method overview

种子获取方法	工具	人工/自动化	对比分析
基于生成的方法	PROTOS[4]	人工	使用特定的模糊测试源语替换协议中的字段,能够发现的漏洞种类少,灵活性较差
	SPIKE[11]	人工	使用基于块的方法生成种子,不能用于测试有状态的协议
	Peach[5],Sulley[6]等	人工	通过人工分析协议格式,编写种子生成规则文件。这种方式构建的种子更符合规范,但需要大量的人工分析
	EXT-NSFSM[44] T-Fuzz[21]	自动化	通过按照定义的规则填充模板,生成种子质量依赖生动化构建模板的准确性
	PULSAR[28]	自动化	通过流量推断出协议的格式以及状态模型生成种子,但是需要大量的通信流量样本
基于突变的方法	RATM-Fuzz[41]	人工	按照协议字段的关系对初始种子进行突变,需要提前构建 PRDF 脚本
	AutoFuzz[10]	自动化	作为内置代理依据推断的协议格式,改变消息的特定字段作为种子,但是推断的准确性偏低
	SGPFuzzer[49]	人工	制定了一个三级突变策略,变异方式更加灵活,但制定的协议格式规范不能涵盖所有的情况
	AFLNet[7]等	自动化	此类灰盒模糊测试工具,依据反馈信息变异正常的样本。这种方式自动化程度高,但是测试速度较慢,难以覆盖到所有的状态空间

2.3　执行过程监测

执行过程监测用于得到测试用例后在特定的监视环境中执行目标程序,并对测试过程中出现的指定异常情况进行记录。该步骤可以在前一个阶段结束之后启动,也可以和前一个阶段形成一个反馈回路。执行过程监测主要进行两个方面的工作:a)监控当前测试用例是否会导致目标程序异常;b)判断测试用例是否增加了代码覆盖率,或者触发到新的状态。

对于黑盒模糊测试工具仅监测程序执行过程的异常情况。Peach、Sulley 等工作通过监测模块监视程序执行的状态,记录崩溃以及是否可以和服务器正常通信。程序崩溃有很大的可能性是因为触发了漏洞,因此这种方式得到的漏洞准确率较高。为了更好地监测以及控制模糊测试进程的执行,使用多个代理监测模糊测试执行时的信息。网络监控代理负责监控网络通信,并将其记录到磁盘上。进程监视器代理监视目标,以检查它在模糊过程中是否可能出错。如果有,将这个错误记录到一个崩溃日志中。VMWare 控制代理提供了一组与虚拟机映像交互的 API,它还可以启动、停止、暂停或重置虚拟机镜像。其他一些研究人员将 SUT 与动态内存分析工具一起执行,这些工具可以帮助其轻松地识别漏洞,因此它们有着很高的漏洞准确率。另外一些模糊测试技术通过将漏洞与正常输入进行比较来识别漏洞。在 SECFuzz 中使用的内存分析工具可以帮助其轻松地检测出确切的漏洞、产生的异常情况以及造成这种情况的原因。此外,这些工具还评估了 SUT 在每种情况下的性能。这种方式不可避免地会引入大量的开销,当使用动态内存分析工具时,SUT 的性能降低了 50%。

上述方法能够发现内存破坏类漏洞,如栈溢出、堆溢出等,协议中还存在大量的漏洞并不会引发内存错误。因此需要其他的判断方法,AspFuzz、AutoFuzz 和 RFSM-Fuzz 通过对比执行时与正常情况下的输出,这种方法的漏洞准确率比较低,必须人工确认漏洞的存在。

灰盒协议模糊器在异常监测方面,将内存错误视为漏洞触发条件,因此这些工具同样具有较高的漏洞准确率。此外,它们使用模糊测试执行过程中监测到的信息反馈给种子生成步

骤,指导种子的变异与调度。协议模糊测试除了需要监测常规的代码覆盖率,更重要的是监测能够反映协议状态的信息。因此,使用不同方法的模糊测试工具,需要监测的反馈信息也不同。这些方法都有着一定的优点以及局限性。

一些研究工作将服务器发送消息中的响应码作为反馈信息,保留能够提高代码覆盖率以及探测到不同协议状态的测试用例。但是,这种方式反映协议状态颗粒度过粗,并且不具有通用性。部分研究工作将程序执行的变量作为反馈信息,这种方法能够支持更多的协议种类以及数量。比如 StateAFL 使用长内存的变量作为程序的状态反馈,但是这种方式的准确性偏低,并且使用的插桩方式带来的开销更大。

为了解决模糊器需要添加延时与测试程序进行消息同步的问题,NSFuzz 注意到网络应用程序的本质是使用一个网络事件循环来监听收到的消息,然后对消息进行处理。基于上述发现,模糊器能在服务端完成消息处理后立即发送下一条消息,从而提高模糊测试的吞吐量。

TCP-Fuzz[50] 在统计代码覆盖率的基础上,做了进一步的改进用于反映状态的改变。在 TCP-Fuzz 中定义了分支转换的概念,首先将获取到的程序路径覆盖信息转换为向量,比较当前输入的向量序列与前一个输入的向量序列是否相同。TCP-Fuzz 使用分支转换来指示状态的转变。

在异常监测方面,不同于传统的方式仅监测内存相关错误,TCP-Fuzz 同时执行不同的 TCP 协议实现程序,通过比较相同输入下所有实现程序的输出是否相同,探测被测程序中的语义错误。这种方式更能检测到漏洞,但是语义错误需要研究人员花费大量的时间确认,因此漏洞检测率较低。

研究人员根据执行过程监测到的信息,能够推断出程序中可能存在的漏洞。不同的工具需要监测的内容不同。如表 3 所示,分别概述了黑盒以及灰盒模糊测试技术执行过程中的监测内容。其中,黑盒技术更关心程序的执行结果,灰盒技术则需要关心覆盖率以及状态信息。黑盒模糊器仅监测程序的输出以及异常情况,这种方式漏洞准确率较低,之后需要测试人员手工分析造成此次异常的真正原因。采用内存工具辅助,有助于复现以及确定漏洞触发的位置,漏洞准确率较高,但是会引入额外的开销。灰盒模糊测试工具需要对程序的源码进行插桩,之后才能获取所需的反馈信息,这种方法无法用于不开源的私有协议。这些工具大多以内存中存在的异常情况作为判定漏洞是否存在的依据,都有着较高的漏洞准确率。TCP-Fuzz 能够检测出大量的语义问题,但是漏洞准确率较低。上述工具使用不同的方式表示协议状态信息,但协议状态通常十分复杂,这些信息都仅能表示所有状态的子集,因此无法保证测试到全部的协议状态。

表 3 监测内容总结

Tab. 3 Summary of monitoring contents

方法	工具	监测内容	对比分析
黑盒模糊测试方法	Peach[5], Sulley[6] SECFuzz[46]等	程序输出以及内存异常情况	漏洞准确率很高,但是需要内存工具辅助降低了运行效率
	AspFuzz[35], AutoFuzz[10] RFSM-Fuzz[38]等	被测试协议程序的输出	漏洞准确率较高,但需要手工分析异常的结果,发现存在的漏洞
灰盒模糊测试方法	AFLNet[7]等	代码覆盖率信息,状态信息,内存异常情况	能有效检测协议漏洞,有着较高的漏洞准确率,但是粒度过粗且适用范围有限
	TCP-Fuzz[50]	代码覆盖的变化信息,多个 TCP 协议实现的输出	能够检测出语义错误漏洞,但是需要大量的手工操作分析异常信息,漏洞准确率较低
	文献[30]	程序执行过程的长状态变量,内存异常情况	漏洞准确率较高,适用范围很广,但是带来的开销更大
	NSFuzz[43]	轻量级的状态变量,内存异常情况	漏洞准确率较高,可以表示细粒度的程序状态,减少了等待时间

2.4 结果分析

结果分析作为模糊测试的最后一个环节,对得到的异常信息进行分类、去重等工作,确定漏洞触发的根本原因以及危害程度。协议模糊测试中部分研究工作依据崩溃信息定位漏洞的方法,存在一定局限性,因为这无法检测到不会造成崩溃的语义漏洞。因此一些研究工作通过对比测试时接收到的响应与正常情况下的响应,采用人工分析的方式找到漏洞。

协议模糊测试中在触发漏洞时还存在无法接收到响应更不会引起崩溃的情况。因此,研究人员通过发送探测报文,判断被测试程序是否存活,检测漏洞是否存在。这种方式需要大量的人工分析确定触发漏洞的根本原因。该步骤是模糊测试不可或缺的一部分,但模糊测试更多的是关注其他步骤的工作,因此本文不对此环节做过多的讨论。

3 物联网与工控场景下的协议模糊测试技术

近年来随着网络技术的不断发展,移动支付、智能家居、智能终端等智能产品不断融入到人们的生活当中,通信网络、铁路控制系统、军事指挥系统、工业控制系统已经成为国家至关重要的基础设施。因此,物联网与工控设备中的网络协议漏洞已经成为网络安全中不可忽视的一环。

将模糊测试用于物联网与工控领域具有以下挑战:

a)模糊测试的执行效率低。固件的操作环境相对恶劣,厂商通常不发布自己的固件产品,因此需要建立一个复杂的固件模拟器,而这种模拟器本身的效率也很低。另一个更重要的原因是物联网设备对接收到数据包的协议格式有严格的要求。一些常见的黑盒模糊工具生成的大多数测试用例很可能是无效的。

b)固件设备中运行的程序难以直接提取,因为这些程序与其运行的硬件配置有着高度的依赖性。此外,部分厂商还会对其产品进行加密处理,这大大增加了逆向分析的难度。

c)缺乏反馈信息。由于固件通常没有源代码,它们的模糊测试只能使用黑盒技术。这种方法只能根据其自身的突变策略持续执行,并且不能判断每个测试用例的质量。

3.1 物联网协议模糊测试

IoTFuzzer[51] 提出可以通过与物联网设备交互的手机端应用程序获取通信使用的协议信息。之后,对配套应用程序进行静态分析,找到向物联网设备发送控制命令的功能,并改变特定变量的值,在不破坏消息格式的情况下进行模糊测试。IoT-Fuzzer 不需要掌握协议的先验知识,能够在不访问固件的情况下,发现物联网设备中存在的内存类损坏漏洞。PS-Fuzz[52] 经过大量的工具同步和信息集成,可以有效地监控固件程序的执行状态,并在固件崩溃时获得大量的系统信息。

IoTHunter[53] 结合了 AFL 和 BooFuzz 的工作实现了第一个用于模糊物联网固件中的有状态协议的灰盒模糊器。IOTHunter 针对物联网固件中的多个关键协议(如 SNMP、FTP、SSL、BGP、SMB),解决了基于物联网固件运行时监控的多阶段消息生成机制的状态

调度问题。Wang 等人[54]通过构建符合规范的测试用例对用于医疗器械的 DICOM 协议进行模糊测试。这种方法不适用于测试报文类型多、状态转换复杂的协议。

Snipuzz[55]实现了一种轻量级的高效黑盒模糊测试方法。Snipuzz 使用一种新的启发式算法检测输入报文中每个字节的语义信息。首先,逐个改变输入序列中的字节生成探测报文,并将从设备中收集到的响应进行分类。具有相同语义的相邻字节形成初始报文片段并将其作为突变的基本单元。此外,Snipuzz 利用相似度得分以及层次聚类策略,减少了响应因随机性和固件内部机制导致的错误分类。因此,Snipuzz 作为一种黑盒模糊器,仍然可以在没有协议规范和设备内部执行信息的情况下,有效地测试物联网设备中的协议程序。

3.2　工控协议模糊测试技术

Modbus/TCP 是工业上一种应用非常广泛的协议,可以用于家庭的空气监视器,或者在工厂中用于机械臂的通信。为了更有效地测试 Modbus/TCP,研究人员找到许多好的方法。为了获得有效的测试用例,大多数突变方法都使用基于语法的突变来改变数据内容。MTF[56]可以测试八个 Modbus/TCP 的实现。为了避免太多无效的测试数据点,MTF 首先调查被测试的设备(device of test,DUT)具有哪些函数代码,然后测试特定类型的突变。Xiong 等人[57]为了减少测试的次数,首先确认 DUT 有哪些功能代码,然后不发送相关的测试数据。这两个方法的缺点是,用户在开始执行突变和测试之前,都需要遍历所有的功能代码。MTF-Storm 对 MTF 进行了改进[58]。与随机生成的数据相比,MTF-Storm 使用一些特殊值来触发这些漏洞。

测试用例生成的质量是许多工控协议研究人员关注的重点,他们通过动态分析、逆向分析、深度学习等方法推断协议格式,按照定义的规则自动化地生成测试用例。杨亚辉等人[59]针对工控协议开发了 ICPPfuzz,采用基于流量分析的逆向方法推断出协议格式并构建状态机,高效地生成测试用例。Yoo 等人[60]使用程序执行中获取的动态信息以及找到输入的语法信息生成测试数据。Niedermaier 等人[61]使用 atcliff/Obershelp 模式识别算法来分析工控私有协议的格式,并创建测试数据包,它可以确定设备之间特定的交互过程。姜亚光等人[62]使用长短期记忆神经网络学习西门子 S7 协议的格式,进而区分出输入序列中的静态字段以及动态字段,通过仅对动态字段进行变异生成符合格式的测试用例。王田原等人[63]通过深度学习技术推断协议格式,开发了一款通用的工控协议模糊测试框架 PGNFuzz。李文轩等人[64]提出相较于深度学习模型,树模型能够更好地体现出协议字段间的存在相关性以及嵌套关系。在此基础上,又通过提取协议字段的优先级信息,生成高质量的测试用例,提高漏洞发现几率。张冠宇等人[65]通过加入动态适应度函数改善遗传算法过早收敛的问题,降低了测试用例之间的相似度,从而提高测试用例覆盖率。

物联网与工控设备环境下的协议模糊测试工作是当前研究的重点方向。因为在这些固件设备中存在大量为适应特殊环境而构建的私有协议,这为格式推断带来了很大的挑战。为了尽快推出新产品,开发人员总是倾向于在固件开发中使用开源组件,但是缺乏很好的更新计划[66]。这无疑牺牲了固件设备的安全性,暴露出安全团队无法迅速补救的漏洞。由于物联网设备没有可靠的网络连接,即使供应商计划修复其产品中的漏洞,固件设备也很难获取补丁,所以开发固件的协议模糊测试工具是当前的迫切需求。

4　分析与展望

协议作为网络通信的基础设施,一直是模糊测试的重点目标。前文对协议模糊测试研究方法进行了对比分析,本章将阐述目前协议模糊测试技术存在的不足,并指出未来的研究方向。

4.1　主要问题

协议模糊测试目前已经取得了大量的研究成果,但仍然存在以下几个方面的问题:

a)当前基于生成的协议模糊器自动化程度普遍不高。这些工具大多依赖推断出的协议格式以及状态机生成测试用例。手工化的推断方式虽然可以获取准确的结果,但是需要掌握协议的先验知识,以及花费大量的人工时间。自动化程度高的协议逆向技术应用不够广泛,虽然协议逆向已经取得了大量的研究成果,但是应用到协议中的仅占一小部分。

b)物联网以及工控设备中大多是不公开源码的私有协议,并且执行环境给模糊测试带来了很大的挑战。这些固件设备中的协议进行模糊测试造成设备崩溃时,需要重新启动,降低了效率。测试固件中的闭源程序时,现有的模糊测试工具不能获取有效的反馈信息,无法进一步测试有价值的种子。

c)目前灰盒协议模糊测试技术普遍存在速度低的问题。这些工具虽然都使用不同的方式表示协议的状态,但是这并不能保证对所有的状态进行测试。程序插桩统计覆盖率不可避免地要引入开销。另一方面,模糊测试工具需要与测试程序进行同步,同样会引入时间开销。

综上所述,协议模糊测试技术存在上述问题的主要原因是协议自身的特殊性与复杂性。对协议进行模糊测试需要考虑生成的输入是否满足协议的格式。通过推断格式构造输入难免会增加协议模糊测试的复杂度,不利于自动化水平的提高。突变的方式往往会产生大量冗余的无效种子,需要在两者之间找到一个平衡。

4.2　展望

综上所述,协议模糊测试可以从以下几个方面开展研究:

a)协议格式逆向技术与模糊测试紧密结合,减少测试人员参与的时间。吸收当前协议逆向技术的最新进展,同时提高自动化程度与精确度。当前的协议逆向技术已经能够自动化地推断出准确性很高的协议规范,用于模糊测试中指导种子生成。在定义测试用例规则时,要同时考虑格式以及语义信息,这样才能生成符合规范的测试用例。此外,采用结构化的变异策略防止生成过多类似的测试用例,提高触发漏洞的几率。

b)测试闭源协议程序时静态插装的方式已经无法适用,可以采取动态的方法实现对不同环境中协议程序的插桩,获取执行过程中的反馈信息更为可行。比如,系统调用具有收集过程难度低、开销小的特点,可以将其执行状态作为重要的反馈信息,为种子分配能量。运用快照技术保存崩溃信息以及测试用例,这样既可以复现崩溃,也可以快速地启动设备到需要测试的状态。

c)当前模糊测试与机器学习[67]、符号执行[68]、优化算法[69]相结合取得了大量的研究成果。协议模糊器同样可以借鉴这些工具的方法,提升漏洞检测能力。比如,基于突变的模糊测试工具可以通过神经网络过滤无效的测试用例,提高执行效率。利用符号执行技术,协助模糊器探测难以达到的路径以及状态空间,提高发现漏洞的几率。将多目标优化等算法应用于协议模糊测试中种子的变异过程,提高生成的测试用例质量。

5　结束语

协议模糊测试技术作为检测协议中漏洞最有效的方式,已经被广泛应用,并且取得了大量的研究成果。本文首先概述了协议模糊测试的发展历程,对协议模糊测试的各个阶段进行了总结分析;其次分析了固件设备中模糊测试取得的成果;最后

总结协议模糊测试技术中存在的挑战,并对未来的发展方向进行了展望。

参考文献:

[1] Mohurle S, Patil M. A brief study of WannaCry threat: Ransomware attack 2017[J]. International Journal of Advanced Research in Computer Science, 2017, 8(5): 1938-1940.

[2] Miller B P, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities[J]. Communications of the ACM, 1990, 33(12): 32-44.

[3] Hawkes B. Project zero five years of 'make 0-day hard'[EB/OL]. (2019-08-07). https://i.blackhat.com/USA-19/Thursday/us-19-Hawkes-Project-Zero-Five-Years-Of-Make-0day-Hard.pdf.

[4] Kaksonen R, Laakso M, Takanen A. Software security assessment through specification mutations and fault injection[C]//Proc of Communications and Multimedia Security Issues of the New Century. Berlin: Springer, 2001: 173-183.

[5] Eddington M. Peach fuzzing platform[EB/OL]. (2013-10-18). https://gitlab.com/peachtech/peach-fuzzer-community.

[6] Amini P, Portnoy A, Sears R. Sulley[EB/OL]. (2019-02-15)[2022-06-02]. https://github.com/OpenRCE/sulley.

[7] Pham V T, Böhme M, Roychoudhury A. AFLNet: a greybox fuzzer for network protocols[C]//Proc of the 13th International Conference on Software Testing, Validation and Verification. Piscataway, NJ: IEEE Press, 2020: 460-465.

[8] Munea T L, Lim H, Shon T. Network protocol fuzz testing for information systems and applications: a survey and taxonomy[J]. Multimedia Tools and Applications, 2016, 75(22): 14745-14757.

[9] Hu Zhihao, Pan Zulie. A systematic review of network protocol fuzzing techniques[C]//Proc of the 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference. Piscataway, NJ: IEEE Press, 2021: 1000-1005.

[10] Gorbunov S, Rosenbloom A. AutoFuzz: automated network protocol fuzzing framework[J]. International Journal of Computer Science and Network Security, 2010, 10(8): 239.

[11] Aitel D. An introduction to SPIKE, the fuzzer creation kit[EB/OL]. (2021-01-07)[2022-06-02]. https://www.blackhat.com/presentations/bh-usa-02/bh-us-02-aitel-spike.ppt.

[12] Foundstone Free Tools. WSDigger[EB/OL]. (2008-02-18)[2022-06-02]. https://foundstone-wsdigger.updatestar.com/en.

[13] Banks G, Cova M, Felmetsger V, *et al*. SNOOZE: toward a stateful network protocol fuzzer[C]//Proc of International Conference on Information Security. Berlin: Springer, 2006: 343-358.

[14] Neves N, Antunes J, Correia M, *et al*. Using attack injection to discover new vulnerabilities[C]//Proc of International Conference on Dependable Systems and Networks. Piscataway, NJ: IEEE Press, 2006: 457-466.

[15] Joshua P. BooFuzz[EB/OL]. (2022-01-31)[2022-06-02]. https://github.com/jtpereyda/boofuzz.

[16] Miki H, Setou M, Kaneshiro K, *et al*. All kinesin superfamily protein, KIF, genes in mouse and human[J]. Proceedings of the National Academy of Sciences, 2001, 98(13): 7004-7011.

[17] Overell P. RFC 2234, Augmented BNF for syntax specifications: ABNF[EB/OL]. (1997-11-03)[2022-06-02]. https://doi.org/10.17487/RFC2234.

[18] Needleman S B. A general method applicable to the search for similarities in the amino acid sequence of two proteins[J]. Journal of Molecular Biology, 1970, 48(3): 443-453.

[19] Beddoe M A. Network protocol analysis using bioinformatics algorithms[J]. Toorcon, 2004, 26(6): 1095-1098.

[20] Krueger T, Gascon H, Krämer N, *et al*. Learning stateful models for network honeypots[C]//Proc of the 5th ACM Workshop on Security and Artificial Intelligence. New York: ACM Press, 2012: 37-48.

[21] Peng Hui, Shoshitaishvili Y, Payer M. T-Fuzz: fuzzing by program transformation[C]//Proc of IEEE Symposium on Security and Privacy. Piscataway, NJ: IEEE Press, 2018: 697-710.

[22] Ma Rui, Wang Daguang, Hu Changzhen, *et al*. Test data generation for stateful network protocol fuzzing using a rule-based state machine[J]. Tsinghua Science and Technology, 2016, 21(3): 352-360.

[23] De Ruiter J, Poll E. Protocol state fuzzing of TLS implementations[C]//Proc of the 24th USENIX Security Symposium. Berkeley: USENIX Association, 2015: 193-206.

[24] Angluin D. Learning regular sets from queries and counterexamples[J]. Information and Computation, 1987, 75(2): 87-106.

[25] Jero S, Pacheco M L, Goldwasser D, *et al*. Leveraging textual specifications for grammar-based fuzzing of network protocols[C]//Proc of AAAI Conference on Artificial Intelligence. Palo Alto, CA: AAAI Press, 2019: 9478-9483.

[26] Wang Jiajie, Guo Tao, Zhang Puhan, *et al*. A model-based behavioral fuzzing approach for network service[C]//Proc of the 3rd International Conference on Instrumentation, Measurement, Computer, Communication and Control. Piscataway, NJ: IEEE Press, 2013: 1129-1134.

[27] Wong E, Zhang Lei, Wang Song, *et al*. Dase: document-assisted symbolic execution for improving automated software testing[C]//Proc of the 37th IEEE International Conference on Software Engineering. Piscataway, NJ: IEEE Press, 2015: 620-631.

[28] Gascon H, Wressnegger C, Yamaguchi F, *et al*. PULSAR: stateful black-box fuzzing of proprietary network protocols[C]//Proc of International Conference on Security and Privacy in Communication Systems. Berlin: Springer, 2015: 330-347.

[29] Blumbergs B, Vaarandi R. Bbuzz: a bit-aware fuzzing framework for network protocol systematic reverse engineering and analysis[C]//Proc of IEEE Military Communications Conference. Piscataway, NJ: IEEE Press, 2017: 707-712.

[30] Natella R. StateAFL: greybox fuzzing for stateful network servers[J]. Empirical Software Engineering, 2022, 27(7): 1-31.

[31] Li Junqiang, Li Senyi, Sun Gang, *et al*. SNPSFuzzer: a fast greybox fuzzer for stateful network protocols using snapshots[C]//Proc of IEEE Trans on Information Forensics and Security. Piscataway, NJ: IEEE Press, 2022: 2673-2687.

[32] Andronidis A, Cadar C. SnapFuzz: high-throughput fuzzing of network applications[C]//Proc of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press, 2022: 340-351.

[33] Zalewski M. American fuzzy lop[EB/OL]. [2022-06-04]. http://lcamtuf.coredump.cx/afl.

[34] Liu Dongge, Pham V T, Ernst G, *et al*. State selection algorithms and their impact on the performance of stateful network protocol fuzzing[C]//Proc of IEEE International Conference on Software Analysis, Evolution and Reengineering. Piscataway, NJ: IEEE Press, 2022: 720-730.

[35] Kitagawa T, Hanaoka M, Kono K. A state-aware protocol fuzzer based on application-layer protocols[J]. IEICE Trans on Information and Systems, 2011, 94(5): 1008-1017.

[36] 张蔚瑶,张磊,毛建瓴,等. 未知协议的逆向分析与自动化测试[J]. 计算机学报,2020,43(4):653-667. (Zhang Weiyao, Zhang Lei, Mao Jianling, *et al*. An automated method of unknown protocol fuzzing test[J]. Chinese Journal of Computers, 2020, 43(4): 653-667.)

[37] Zhang Saidan, Zhang Luyong. Vulnerability mining for network protocols based on fuzzing[C]//Proc of the 2nd International Conference on Systems and Informatics. Piscataway, NJ: IEEE Press, 2014: 644-648.

[38] Zhao Jingling, Chen Shilei, Liang Shurui, *et al*. RFSM-Fuzzing a smart fuzzing algorithm based on regression FSM[C]//Proc of the 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. Piscataway, NJ: IEEE Press, 2013: 380-386.

[39] Fiterau-Brostean P, Jonsson B, Merget R, *et al*. Analysis of DTLS

implementations using protocol state fuzzing[C]//Proc of the 29th USENIX Security Symposium. Berkeley: USENIX Association, 2020: 2523-2540.

[40] Raffelt H, Steffen B, Berg T. LearnLib: a library for automata learning and experimentation[C]//Proc of the 10th International Workshop on Formal Methods for Industrial Critical Systems. New York: ACM Press, 2005: 62-71.

[41] Han Xing, Wen Qiaoyan, Zhang Zhao. A mutation-based fuzz testing approach for network protocol vulnerability detection[C]//Proc of the 2nd International Conference on Computer Science and Network Technology. Piscataway, NJ: IEEE Press, 2012: 1018-1022.

[42] Ma Rui, Zhu Tianbao, Hu Changzhen, *et al*. SulleyEX: a fuzzer for stateful network protocol[M]//Yan Zheng, Molva R, Mazurczyk W, *et al*. Network and System Security. Cham: Springer, 2017: 359-372.

[43] Qin Shisong, Hu Fan, Zhao Bodong, *et al*. NSFuzz: towards efficient and state-aware network service fuzzing[J/OL]. ACM Trans on Software Engineering and Methodology. (2023-03-31). https://doi.org/10.1145/3580598.

[44] Wang Jiajie, Guo Tao, Zhang Puhan, *et al*. A model-based behavioral fuzzing approach for network service[C]//Proc of the 3rd International Conference on Instrumentation, Measurement, Computer, Communication and Control. Piscataway, NJ: IEEE Press, 2013: 1129-1134.

[45] Song Congxi, Yu Bo, Zhou Xu, *et al*. SPFuzz: a hierarchical scheduling framework for stateful network protocol fuzzing[J]. IEEE Access, 2019, 7: 18490-18499.

[46] Tsankov P, Dashti M T, Basin D. SECFuzz: fuzz-testing security protocols[C]//Proc of the 7th International Workshop on Automation of Software Test. Piscataway, NJ: IEEE Press, 2012: 1-7.

[47] Somorovsky J. Systematic fuzzing and testing of TLS libraries[C]//Proc of ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2016: 1492-1504.

[48] 胡志濠,潘祖烈. 基于 QRNN 的网络协议模糊测试用例过滤方法[J]. 计算机科学,2022,49(5):318-324. (Hu Zhihao, Pan Zulie. Testcase filtering method based on QRNN for network protocol fuzzing[J]. Computer Science, 2022, 49(5): 318-324.)

[49] Yu Yingchao, Chen Zuoning, Gan Shutao, *et al*. SGPFuzzer: a state driven smart graybox protocol fuzzer for network protocol implementations[J]. IEEE Access, 2020, 8: 198668-198678.

[50] Zou Yonghao, Bai Jiaju, Zhou Jielong, *et al*. TCP-Fuzz: detecting memory and semantic bugs in TCP stacks with fuzzing[C]//Proc of USENIX Annual Technical Conference. Berkeley: USENIX Association, 2021: 489-502.

[51] Chen Jiongyi, Diao Wenrui, Zhao Qingchuan, *et al*. IoTFuzzer: discovering memory corruptions in IoT through App-based fuzzing[C/OL]//Proc of Network and Distributed Systems Security (NDSS) Symposium. (2018-02-18)[2022-06-02]. http://dx.doi.org/10.14722/ndss.2018.23159.

[52] Li Xiaoyi, Pan Xiaojun, Sun Yanbin. PS-Fuzz: efficient graybox firmware fuzzing based on protocol state[J]. Journal of Artificial Intelligence, 2021, 3(1): 21-31.

[53] Yu Bo, Wang Pengfei, Yue Tai, *et al*. Poster: fuzzing IoT firmware via multi-stage message generation[C]//Proc of ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2019: 2525-2527.

[54] Wang Zhiqiang, Li Quanqi, Wang Yazhe, *et al*. Medical protocol security: DICOM vulnerability mining based on fuzzing technology[C]//Proc of ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2019: 2549-2551.

[55] Feng Xiaotao, Sun Ruoxi, Zhu Xiaogang, *et al*. Snipuzz: black-box fuzzing of IoT firmware via message snippet inference[C]//Proc of ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2021: 337-350.

[56] Voyiatzis A G, Katsigiannis K, Koubias S. A Modbus/TCP fuzzer for testing internetworked industrial systems[C]//Proc of the 20th Conference on Emerging Technologies & Factory Automation. Piscataway, NJ: IEEE Press, 2015: 1-6.

[57] Xiong Qi, Liu Hui, Xu Yuan, *et al*. A vulnerability detecting method for Modbus-TCP based on smart fuzzing mechanism[C]// Proc of IEEE International Conference on Electro/Information Technology. Piscataway, NJ: IEEE Press, 2015: 404-409.

[58] Katsigiannis K, Serpanos D. MTF-Storm: a high performance fuzzer for Modbus/TCP[C]//Proc of the 23rd International Conference on Emerging Technologies and Factory Automation. Piscataway, NJ: IEEE Press, 2018: 926-931.

[59] 杨亚辉, 麻荣宽, 耿洋洋, 等. 基于工控私有协议逆向的黑盒模糊测试方法[J]. 计算机科学, 2023, 50(4): 323-332. (Yang Yahui, Ma Rongkuan, Geng Yangyang, *et al*. Black-box fuzzing method based on reverse-engineering for proprietary industrial control protocol[J]. Computer Science, 2023, 50(4): 323-332.)

[60] Yoo H, Shon T. Grammar-based adaptive fuzzing: evaluation on SCADA Modbus protocol[C]//Proc of IEEE International Conference on Smart Grid Communications. Piscataway, NJ: IEEE Press, 2016: 557-563.

[61] Niedermaier M, Fischer F, Von Bodisco A. PropFuzz—an IT-security fuzzing framework for proprietary ICS protocols[C]//Proc of International Conference on Applied Electronics. Piscataway, NJ: IEEE Press, 2017: 1-4.

[62] 姜亚光,陈曦,李建彬,等. 基于 LSTM 的 S7 协议模糊测试用例生成方法[J]. 计算机工程,2021,47(7):183-188. (Jiang Yaguang, Chen Xi, Li Jianbin, *et al*. LSTM-based fuzzy test case generation method for S7 protocol[J]. Computer Engineering, 2021, 47(7): 183-188.)

[63] 王田原,武淑红,李兆基,等. PGNFuzz:基于指针生成网络的工业控制协议模糊测试框架[J]. 计算机科学,2022,49(10):310-318. (Wang Tianyuan, Wu Shuhong, Li Zhaoji, *et al*. PGNFuzz: pointer generation network based fuzzing framework for industry control protocols[J]. Computer Science, 2022, 49(10): 310-318.)

[64] 李文轩,尚文利,和晓军,等. 基于改进变异树的工控协议模糊测试用例生成方法[J]. 计算机应用研究,2020,37(12):3662-3666. (Li Wenxuan, Shang Wenli, He Xiaojun, *et al*. Fuzzing test case generation method for industrial control[J]. Application Research of Computers, 2020, 37(12): 3662-3666.)

[65] 张冠宇,尚文利,张博文,等. 一种结合遗传算法的工控协议模糊测试方法[J]. 计算机应用研究,2021,38(3):680-684. (Zhang Guanyu, Shang Wenli, Zhang Bowen, *et al*. Fuzzy test method for industrial control protocol combining genetic algorithm[J]. Application Research of Computers, 2021, 38(3): 680-684.)

[66] Eclipse IoT Working Group. The three software stacks required for IoT architectures IoT software requirements and how to implement then using open source technology[EB/OL]. (2016-10-02)[2022-06-02]. https://iot.eclipse.org/community/resources/white-papers/pdf/Eclipse%20IoT%20White%20Paper%20-%20The%20Three%20Software%20Stacks%20Required%20for%20IoT%20Architectures.pdf.

[67] Saavedra G J, Rodhouse K N, Dunlavy D M, *et al*. A review of machine learning applications in fuzzing[C]//Proc of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press, 2018: 95-105.

[68] Chen Ju, Han W, Yin Mingjun, *et al*. SYMSAN: time and space efficient concolic execution via dynamic data-flow analysis[C]// Proc of the 31st USENIX Security Symposium. Berkeley: USENIX Association, 2022: 2531-2548.

[69] Lyu Chenyang, Ji Shouling, Zhang Chao, *et al*. MOPT: optimized mutation scheduling for fuzzers[C]//Proc of the 28th USENIX Security Symposium. Berkeley: USENIX Association, 2019: 1949-1966.