基于序列到序列模型的网络协议模糊测试方法

陈 乾, 洪 征, 古津榜, 张国敏, 秦素娟

中国人民解放军陆军工程大学 指挥控制工程学院 南京 中国 210000

摘要 基于变异的协议模糊测试方法由于不依赖于人工定义协议规范和良好的漏洞发现能力, 受到了研究者的青睐。基于变异的网络协议模糊测试方法中测试用例由种子变异而来, 种子的质量和多样性直接影响变异产生的测试用例质量。现有方法捕获网络流量作为种子, 但由于其中包含的报文种类和字段的多样性有限, 难以保证以这些种子实施变异可以有效提升测试覆盖率; 此外, 现有方法紧密耦合于有状态协议, 状态机引导机制和用例的发送方式在运用于无状态协议时会出现低效甚至不可用等问题。针对上述问题, 提出一种基于序列到序列模型的网络协议模糊测试方法。方法利用序列到序列模型模拟服务端和客户端的交互, 产生丰富的交互数据作为种子, 简化了种子的获取, 保证质量的同时提高了种子的多样性。此外, 针对现有方法紧耦合于有状态协议的问题, 提出了适应性策略, 打乱报文序列发送至目标, 根据返回报文判断协议是否为有状态协议。针对有状态协议, 利用状态机以及测试覆盖率引导模糊测试, 并采用标准套接字进行测试用例的发送。针对无状态协议, 可以绕过状态学习过程, 通过重定向数据到本地通信管道的方法实现测试用例的高效投递。实验结果表明, 所设计的方法能够有效运用于网络协议模糊测试。相较于基准方法, 所设计的方法在针对有状态协议执行模糊测试时, 增强了路径覆盖能力; 在处理无状态协议时, 具有更快的执行速度。总体上, 所提方法有效地提高了网络协议模糊测试的效率。

关键词 网络安全; 协议安全; 模糊测试; 序列到序列模型

中图法分类号 TP393.08 DOI 号 10.19363/J.cnki.cn10-1380/tn.2026.01.10

Network Protocol Fuzzing Based on Sequence-to-Sequence Model

CHEN Qian, HONG Zheng, GU Jinbang, ZHANG Guomin, QIN Sujuan

Command and Control Engineering College, Army Engineering University of PLA, Nanjing 210000, China

Abstract Researchers have shown a preference for mutation-based fuzzing methods for protocols due to their ability to discover vulnerabilities effectively without relying on manually defined protocol specifications. In mutation-based protocol fuzzing, test cases are derived through the mutation of seeds. The effectiveness of the generated test cases is directly affected by the quality and diversity of the seeds. Existing methods usually use network traffic samples as seeds. However, the seeds lack diversity, rendering it difficult to ensure the mutation on the seeds can effectively improve test coverage. In addition, existing methods are generally tightly coupled to stateful protocols: When the methods are applied to stateless protocols, the guidance of state machines and the test case delivery method may be inefficient or even impractical. Aiming at the problems, this paper proposes a network protocol fuzzing method based on Sequence-to-Sequence model. The Sequence-to-Sequence model is employed to simulate the server-client interactions, enabling generation of diverse data as seeds. This approach simplifies the process of seed generation and ensures the quality of the seeds while enhancing their diversity. Furthermore, in order to address the issue of tight couple with stateful protocols, an adaptive strategy is proposed. The strategy randomly sends the message sequence to the target program and determines whether the protocol is stateful based on the replies. For stateful protocols, this paper utilizes state machines and code coverage as guidance, employing standard sockets to deliver the test cases. For stateless protocols, the state machine learning process can be bypassed and the test cases are redirected through the local communication pipe, which boosts fuzzing efficiency tremendously. Experiments show that the method is effective for protocol fuzzing. Compared with the benchmark method, the proposed method can enhance the path coverage capability when fuzzing stateful protocols. When dealing with stateless protocols, the proposed method demonstrates high execution speed. To summarize, the proposed method effectively enhances the efficiency of network protocol fuzzing.

Key words network security; protocol security; fuzzing test; Sequence-to-Sequence model

通讯作者: 洪征, 博士, 副教授, Email: hz5215@163.com。

本课题得到国家重点研发计划(No. 2019YFB2101704)资助。

收稿日期: 2024-06-04; 修改日期: 2024-08-27; 定稿日期: 2025-12-11

1 引言

网络协议作为计算机网络设备之间的通信规则, 是网络运行的基础[1]。协议实体程序通常指的是那些部署在计算机平台, 实现了特定网络协议的功能, 从而能够提供相应网络服务的程序。由于开发人员对网络协议的理解存在偏差, 在开发环节缺乏安全编码规范等因素影响, 协议实体程序的安全漏洞频繁出现。协议实体程序由于负责提供网络服务, 访问接口对外暴露, 攻击者可以轻易发起远程网络攻击。此外, 协议实体程序往往在互联网中部署广泛, 一旦漏洞被利用, 将会造成严重损失。例如, WannaCry 病毒利用了微软 SMBv1 服务软件的漏洞, 感染了全球超过 10 万台主机, 造成了 80 亿美元的经济损失[2]。协议实体程序的漏洞发现和修复对提升网络安全具有重要意义。

模糊测试是一种通过向测试目标提供大量畸形输入, 并监测测试目标的表现, 来发现测试目标中安全漏洞的测试技术[3]。根据对测试目标运行时的内部结构和行为的了解程度, 协议模糊测试主要分为黑盒模糊测试和灰盒模糊测试。黑盒模糊测试一般仅考虑目标程序的输入和输出, 不监控程序运行时的内存变化, 具有较高执行速度, 简单易用, 在协议模糊测试领域应用广泛。但是, 由于缺乏对程序运行时的状态监控, 黑盒模糊测试无法以代码覆盖率等指标作为引导, 促使工具全面地对程序路径进行测试。

灰盒模糊测试以 AFL(American Fuzzy Loop)为代表[4], 此类方法往往以代码覆盖率为导向, 越高的覆盖率意味着越全面的覆盖程序, 越有可能挖掘到程序中的安全漏洞。协议灰盒模糊测试一般采用编译时插桩的方法, 在程序运行时通过桩代码获取程序运行时的内存变化, 根据代码覆盖率等指标的变化, 引导测试的实施。现有主流灰盒方法往往基于变异, 也就是说在模糊测试开始前, 需要测试人员提供初始输入作为“种子”, 模糊器针对“种子”进行变异, 目的是产生多样化的测试用例。协议模糊测试过程中“种子”的获取往往通过抓取目标程序的交互数据, 在测试开始前需要部署目标程序, 通过与目标程序交互获得真实交互数据作为模糊器的输入。这种方法不需要测试人员在测试开始前对协议进行细粒度的分析, 但仍存在以下问题。首先, 协议实体程序的配置和部署往往较为复杂。其次, 对协议模糊测试而言, 种子需要尽可能全面地包含各种类型的合法报文, 以保证后续测试中基于种子所构造的测试用例能够到达更多的程序分支。现有方案很难保证种子的质量和多样性, 种子的筛选往往涉及专家经验。在测试过程中我们发现, 随意抓取流量所获得的种子往往存在大量冗余, 将这些种子直接用于测试将浪费测试资源; 此外, 种子的类型不足, 会导致长时间无法触发新的执行路径, 测试低效。

现有协议模糊测试方法往往紧耦合于有状态协议, 协议状态引导机制、协议状态获取方式等工作围绕协议状态机展开, 对于没有显式状态机的无状态协议, 冗余的处理流程和受限的用例发送方式往往会影响模糊测试的效率。例如, AFLNet[5]虽然可以实现 HTTP 协议的模糊测试, 但 HTTP 协议本身是无状态的, HTTP 服务器不会保留客户端的历史请求。每次客户端向服务器发起 HTTP 请求时, 服务器都会独立处理这个请求, 因此对 HTTP 协议实体程序而言, 基于状态机引导程序测试显得冗余。此外, 在用例发送阶段, 现有研究广泛采用标准网络套接字的方法。利用标准网络套接字处理数据包的延迟等待间隔, 模糊器有充分的时间对收发的数据包进行处理, 通过分析请求和响应的情况, 推测目标是否切换到了预期的协议状态[6], 有利于模糊器和测试目标之间的协议状态同步, 可以准确地构建协议状态机并利用协议状态机引导模糊测试。但是对于无状态协议而言, 由于协议实现不保留历史会话信息, 每个报文都独立处理, 无需利用状态机引导模糊测试过程, 模糊器可以以较高的速率持续发送用例, 提高测试效率。

近年来, 研究者开始将机器翻译、文本生成等自然语言处理领域的热点知识与协议模糊测试结合。其中, 序列到序列(Sequence-to-Sequence, Seq2Seq)模型[7]在机器翻译、代码生成等领域应用广泛。Seq2Seq 模型采用编码器(Encoder)与解码器(Decoder)的架构, 可以应用文本序列生成任务中。编码器通过循环机制及自注意力机制有效捕获输入序列的上下文信息, 确保模型能够理解复杂的语言结构和依赖关系。解码器在生成输出时, 不仅基于自身内部状态, 还结合了编码器提供的全局上下文向量, 能够根据输入内容动态生成对应的输出序列。Seq2Seq 模型能够处理不定长的输入和输出序列, 无需对数据进行填充或截断, 避免了长度限制造成的信息损失。采用该结构的代表性模型包括 RNN Search[8]和 Transformer[9]。

双向门循环单元(Bi-Gated Recurrent Unit, Bi-GRU)是对门循环单元 Gated Recurrent Unit, GRU)的改进[10], 可以和 GRU 一同作为 Seq2Seq 架构模型中编码器或解码器的组件。标准的 GRU 仅关注上文信息, 在分析不定长输入时易忽略上下文关联, 导致

信息的缺失。与标准 GRU 不同, BiGRU 能够同时获取过去和未来的上下文信息, 即双向处理输入序列。在BiGRU中, 有两个独立的GRU层分别沿时间步正向和反向处理输入序列, 然后将两个方向上的隐藏状态拼接在一起作为当前时间步的最终隐藏状态。这意味着 BiGRU 能够综合考虑序列的整体信息, 从而更好地抽取语义信息。同时, 相比于标准的 RNN Seach, BiGRU 具备处理序列上下文的能力并通过门循环有效地控制信息在单元中的传递, 可以更好地学习文本特征。相比于 Transformer, BiGRU 在结构上更为轻量化, 这直接导致其在计算复杂度上显著低于 Transformer。在处理垂直领域任务(如仅仅只进行协议报文生成)时, 两者虽然能够取得相近的效果, 但 BiGRU 由于轻量化的特点, 通常更容易训练, 且运行时的资源消耗更小。

本文提出一种基于 Seq2Seq 的网络协议灰盒模糊测试方法, 采用BiGRU作为编码器GRU作为解码器。利用 Seq2Seq 模型学习和模拟协议实体程序的交互, 不仅能够覆盖大多数实体程序的典型行为模式, 而且能有效生成实际环境中难以直接获取的行为报文。首先, Seq2Seq 模型产生的交互数据序列严格遵循目标协议的格式规范, 同时蕴含丰富的报文种类, 这有利于后续变异过程中对协议状态空间的探索, 覆盖更多的状态分支。其次, 方法基于模型自主生成种子, 减轻了实施测试前的种子收集工作, 提升了模糊测试的自动化水平。针对无状态协议, 本文方法精简了模糊测试的工作逻辑并基于本地通信管道进行用例发送, 有效提高了无状态协议的测试效率。

2 相关工作

采用黑盒模糊测试方法, 测试者利用测试目标的外部接口, 向目标发送畸形数据并观察目标行为来判断漏洞是否存在。此类方法易于实现, 可以快速开展测试, 在漏洞挖掘领域应用广泛。Boofuzz[11]、Sulley[12]和 Peach[13]是此类方法的代表。由于不掌握程序内部状态, 也缺乏测试覆盖率的引导, 此类方法与灰盒模糊测试方法相比具有盲目性, 漏洞挖掘效率低下。

灰盒协议模糊测试往往需要测试者获取程序源码或可执行二进制程序, 通过源代码插桩技术[14]或者利用 QEMU[15]等仿真平台获取测试目标运行时的覆盖率反馈驱动模糊测试。AFLNet 是第一个针对网络协议的灰盒模糊测试工具, 其在 AFL(American Fuzzy Loop)的基础上进行扩展, 通过标准 Socket 与目标交互。同时, AFLNet 设置了状态机学习机制, 随着模糊测试的深入, 协议状态机会逐步完善, 并与代码覆盖反馈共同作用, 驱动模糊测试的进行。StateAFL[16]的设计与 AFLNet 相似, 不同的是 StateAFL 通过监控服务程序运行时的内存变化获取程序的状态信息, 以更加准确地掌握状态转移情况。为了更精确地掌握状态信息, NSFuzz[17]将程序静态分析与灰盒模糊测试技术相结合。具体而言, 它首先利用静态分析技术深入剖析目标程序, 以识别出关键的事件循环结构, 并从中提取出关键的状态变量。这些状态变量随后被用作指导程序进行插桩, 从而能够实时、准确地捕获并追踪协议的状态信息。

IoTHunter[18]与 SPFuzz[19]在 AFL 框架的基础上进行了扩展, 引入了协议描述模块。这使得此类工具在执行模糊测试之前能够深入理解目标协议, 通过测试人员手工定义的协议模型来指导测试。然而, 此类测试人员在测试前投入大量时间和精力进行详尽的协议分析, 增加了测试准备阶段的复杂性和成本。

HNPFuzzer[20]为了加速协议模糊测试的迭代速率, 加速测试用例的发送, 通过共享内存的方式进行用例的发送, 并通过构建同步器进行状态的同步, 确保引导目标到特定的协议状态下进行测试。因为不存在网络延迟, 测试工具可以更快地得到服务器端的响应, 进而根据响应结果及时调整和优化下一轮的模糊测试。但共享内存通常需要测试目标程序直接支持或修改源代码以读取共享内存区域。一些网络服务或协议栈不能从共享内存中读取输入数据, 因此这种方法的应用范围有限。

以 AFLNet、StateAFL 为代表的灰盒工具多基于 AFL 进行扩展, 测试效果都依赖于初始输入。在协议模糊测试领域, 往往以协议报文作为初始输入。对于有状态协议, 理想的种子集合应引导程序进入尽可能多的协议状态, 在此基础上进行测试才能保证测试的全面。

一些研究人员利用深度学习技术辅助和优化种子的获取和测试用例的构造, 提高协议模糊测试的有效性。LAFuzz[21]针对基于语法的模糊测试(Grammar-based Fuzzing)在构建测试用例时流程烦琐的问题, 利用长短时记忆网络(Long Short-Term Memory, LSTM)来自动学习测试用例的文本结构特征, 进而高效生成测试用例。Hu 等[22]借助生成对抗网络(Generative Adversarial Networks, GAN)的文本生成能力, 自动化地为工业控制协议生成模糊测试用例, 旨在深入探测工业固件潜在的安全漏洞。然而, 上述方法存在一些局限。首先, 深度学习模型虽然能

学习测试用例的构造语法, 但现有方法未充分考虑数据之间的时序关系。在处理有状态协议时, 报文间存在着内在的逻辑关联, 历史的会话状态会对服务程序的行为产生直接影响, 因此在实施测试时, 遵循时序逻辑至关重要。其次, LSTM 和 GAN 等结构使模型在报文的理解上存在制约, 此类模型依赖于基本单元之间数据的单向传导进行报文特征的学习。单向传导的特性使模型在捕获报文上下文关系时存在不足。具体而言, 由于只能依据历史信息对未来信息进行预测, 而不能反向分析未来信息与当前信息的联系。因此会导致模型在执行需要综合前后文信息的任务时, 无法充分获取上下文特征, 影响报文的质量。

Seq2Seq 模型的编码器通过融合注意力机制全面学习并抽取输入的上下文信息, 令模型具备理解复杂文本的能力。而解码器在生成输出时, 不仅关注自身状态, 还利用了编码器抽取的上下文向量, 从而动态地生成对应输出, 可以有效地避免文献[21-22]方法的局限。文献[23]指出, Seq2Seq 模型因其强大的文本理解能力, 在复杂代码的自动生成方面展现出了显著优势, 能够根据特定需求有效地生成复杂的代码结构。鉴于协议报文生成与代码生成在本质上具有相似性——都涉及语言描述、结构定义等, 因此 Seq2Seq 模型同样适用于协议报文的生成。Gao 等[24]利用 Seq2Seq 模型学习开源工具生成的测试用例, 将 Seq2Seq 模型运用在测试用例的生成阶段。但该方法只是生成单一的报文作为用例, 并没有关注实际模糊测试过程中报文之间的关联关系。Yang 等[25]利用 Seq2Seq 模型辅助测试用例的构造, Seq2Seq 模型被训练来预测最佳的变异操作和变异位置组合, 智能地选择最有可能触发新崩溃的变异算子。该方法聚焦于利用 Seq2Seq 模型对种子进行分析, 并在此基础上进行变异, 以更好地构造测试用例, 但没有考虑种子获取阶段的各类问题。

文献[26]指出, Seq2Seq 模型能够抽取问答文本间的对应关联, 在聊天机器人领域应用广泛。客户机与服务器之间的交互遵循请求应答机制, 请求报文和响应报文存在逻辑关联, 与聊天机器人的应用场景相似, Seq2Seq 模型具备理解并生成网络报文的能力。因此可以基于 Seq2Seq 模型构建机器人模拟协议实体程序的交互, 确保产生的报文具有合法的报文结构, 并遵循网络协议的处理逻辑。

现有模糊测试方法大多以有状态协议为研究目标, 并常常针对测试对象进行适应性设计, 导致测试引擎与测试目标之间高度耦合[27]。近年来, Nginx、Apache、Lighttpd 以及 Libmodbus 等基于 HTTP、Modbus 等无状态协议的协议实体程序在网络中得到了广泛应用。由于协议无状态, 程序不需要跟踪会话状态, 程序在处理请求时流程更加简洁, 缩短了对请求进行处理的时间, 此类协议对服务器的资源利用率高, 可以提供更快的响应速度, 具有高效性和便捷性。但同时, 攻击者可以轻易地截获和重播合法的请求消息, 这可能导致未经授权的访问、服务拒绝等安全威胁。

现有的网络协议模糊测试工具主要针对有状态协议设计, 根据目标的反馈获取状态信息, 逐步构建状态机引导模糊测试, 状态机的准确性直接影响到模糊测试的效果。在测试过程中, 标准 Socket 的延时等待机制可以确保模糊器有充分时间获得被测目标的响应信息以逐步构建状态机, 保证状态机的准确[6]。在测试过程中, 这种等待机制也可以确保目标被引导到特定协议状态下再进行后续测试。但此类模糊测试工具在测试无状态协议时往往效率低下。无状态协议通常不包含显式的状态机, 这意味着对于无状态协议, 服务器对每个请求的处理完全基于该请求本身, 独立于之前的交互历史。因此, 协议状态机的学习和基于状态机的引导在无状态协议的模糊测试过程中都是不必要的, 测试流程可以简化, 测试用例的发送也不需要等待协议状态的转换。

本文研究运用 Seq2Seq 模型对协议实体的交互进行学习, 旨在提取服务程序间的交互逻辑和报文结构。本文构建了服务端机器人(ServerBot)和客户端机器人(ClientBot), 通过让机器人之间进行交互, 快速高效地产生通信报文序列作为种子, 所产生的报文具备合法的报文结构, 不仅能覆盖协议实体程序的大多数典型报文, 还能生成实际环境中较少出现的报文。其次, 对有状态协议的模糊测试延续了现有主流方法的思路, 通过协议状态机和程序覆盖率共同引导模糊测试, 采用标准网络套接字进行通信以保证状态机的准确构建和状态的有效引导。对于无状态协议, 针对无状态协议的特点, 采用了简洁高效的模糊测试逻辑, 避免围绕状态机实施测试。同时, 采用了高效的用例发送策略, 通过引入钩子(HOOK)技术[28]对网络套接字相关的 API 函数进行拦截, 将相应的 API 函数替换成自定义的交互函数, 重新定向数据到本地的高速通信管道, 将测试用例直接从客户机提交给服务器。总体上看, 本文方法有助于提升协议模糊测试的自动化水平及测试效率。

3 本文方法的设计

本章将阐述本文方法的整体设计与实现。从方法的总体设计、基于 Seq2Seq 模型的种子生成模块的设计以及模糊测试模块的设计 3 个方面展开进行描述。

3.1 总体设计

本文方法在 AFLNet 的基础上进行扩展, 在测试开始前需要获取程序源代码, 利用编译时插桩技术对源代码进行插桩, 并将作为测试目标的实体程序和模糊器部署在同一主机。方法旨在利用 Seq2Seq 模型提升协议模糊测试工具的自动化程度和测试效率, 总体工作流程如图 1 所示。首先, 受自然语言处理领域轻量化机器人对话系统的启发, 为了高效地获取表示协议正常交互的种子, 本文设计了基于 Seq2Seq 模型的种子生成模块。该模块以 BiGRU 和 GRU 为核心构建了服务端机器人(ServerBot)和客户端机器人(ClientBot)。基于机器人之间的交互, 产生数据作为种子。与传统上实际部署服务器抓取流量作为种子或是通过人工构建种子的方式相比, 本文方法无需对流量数据进行筛选和分析就可以快速高效地生成结构合法的报文, 能有效覆盖各种报文类型。

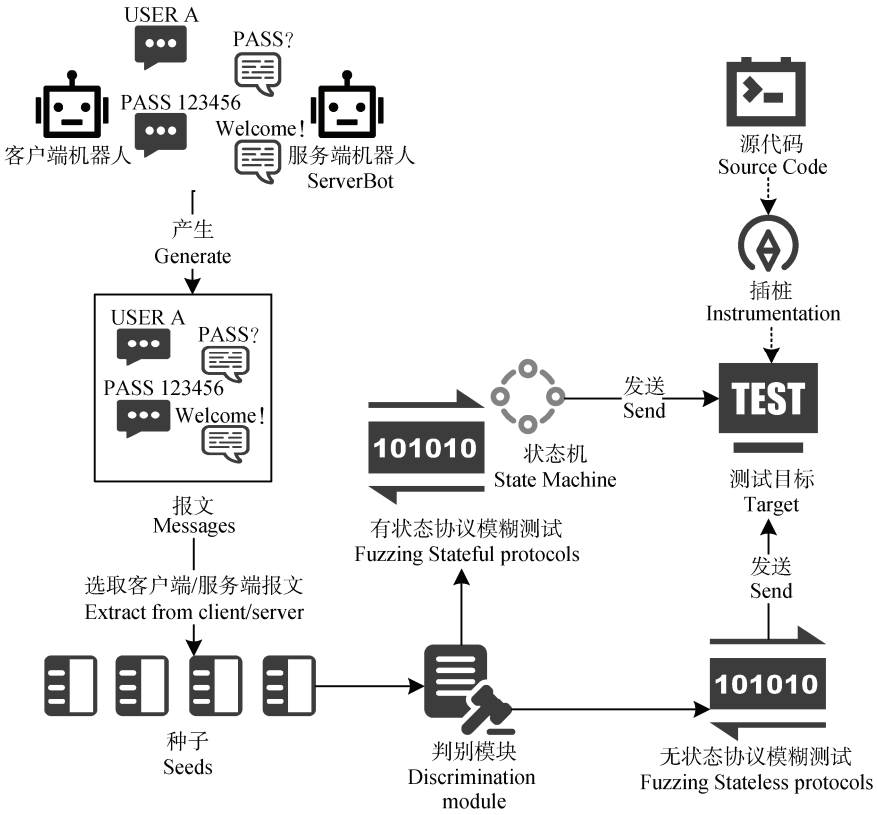


图 1 本文方法的总体工作流程

Figure 1 Overall workflow of our method

服务端机器人和客户端机器人主要由编码器和解码器组成。编码器承担着理解接收到的报文的职责, 由 5 层 BiGRU 构成, BiGRU 由一组正向 GRU 和一组反向 GRU 构成, 由于既关注到了信息的顺序传递又关注到信息的前序关联, 每个时间步的隐状态中不仅包含了报文序列中前置单元的信息, 还包含了之后所有单元的信息。如此, 可以保证机器人全面地理解接收到的报文。解码器则根据解码器抽取的信息生成反馈报文, 由 3 层 GRU 构成。GRU 单向传递的结构令解码器在预测下一个输出时会考虑所有已生成的输出, 更加适合文本生成任务。解码器根据解码器抽取的信息特征, 以自回归的方式生成反馈报文序列。

基于变异的模糊测试方法需要关注种子的质量和多样性。协议模糊测试的对象是协议实体程序, 种子适合采用报文序列的形式构建。不仅需要关注报文本身的合法性, 还应关注报文之间的关联关系。由报文序列构造的种子需要具有多样性, 有利于覆盖协议实体程序不同的执行路径。本文方法利用机器人学习服务器和客户机之间的交互行为以掌握报文之间的关联关系。

本文将协议实体程序的请求和响应以问/答(Q/A)

对的形式进行组织。针对服务端与客户端的不同角色, Q/A 对的组织方式有所差异。对服务端, Q 为客户端程序的请求报文, 而 A 为服务程序的响应报文, Q/A 对可以反映服务端在收到请求后的响应。在客户端, Q 为服务端程序发送的报文, A 为在收到 Q 后客户端程序的响应, Q/A 对能够体现客户端程序在接收到服务端报文后的行为。ClientBot 通过学习客户端程序的行为, 掌握了客户端的请求模式, 能够产生形式各样的合法请求。ServerBot 通过学习服务端程序的响应模式, 具备了基于客户端请求信息生成应答报文的能力。

算法 1. 协议是否为有状态协议的判别算法

```
1.  FUNCTION:
2.  Send the messages in Seeds and record the responses in T
3.  Randomly scramble messages for N time:
4.      Send the messages and record the responses in T'
5.  RETURN T
6.  IF T and T' are the same:
7.      RETURN Stateless
8.  ELSE:
9.      RETURN Stateful
```

ClientBot 和 ServerBot 交互产生的数据将作为种子, 用于对目标进行模糊测试。对客户端的测试需要提取服务端产生的报文作为种子, 反之亦然。本文方法聚焦于服务端的测试, 基于 ClientBot 发出的请求报文构造种子。一个种子由一组有序的报文组成。每个种子包含一次会话的所有请求报文。例如对于 FTP 这种有状态协议的会话, 种子包括客户端发起的登录请求、密码验证, 以及随后的各种文件操作, 直至会话结束时的断开连接请求和确认报文。对于 HTTP 这种无状态协议的会话, 种子应包含客户端的一连串资源请求报文, 直至客户端不再进行任何请求操作。

获取种子后, 本文方法的判别模块判断协议是否属于有状态协议。在进行判断时, 首先通过标准网络套接字, 按序发送种子里的每个报文, 并记录下每个报文的响应。接着打乱报文与测试目标进行交互, 根据收到的响应是否有变化判断协议是否是有状态协议, 如算法 1 所示。有状态协议的请求报文往往依赖于历史信息, 若打乱报文顺序, 则请求将被视为非法, 服务器往往会拒绝连接或者进入异常处理流程, 并返回标识错误的状态码。无状态协议的服务端则不会记录历史交互, 而是将每次请求独立处理。因此, 打乱报文序列后发送至测试目标, 模糊器接收到的返回报文将与按序发送报文所得到的响应一致。

对有状态协议和无状态协议, 本文采用了不同的处理方式。针对有状态协议, 本文采用目前有状态协议的主流测试方法, 采用状态机进行状态引导, 并通过标准 Socket 发送测试用例, 通过合理的延迟等待以保证模糊器和被测的协议实体程序之间的状态同步。而针对无状态协议, 本文方法进行了优化处理, 绕过了状态学习的过程, 并重定向测试用例到本地高速通信管道, 避免了采用数据包的形式发送测试用例需要的逐层封装和解析处理, 提高了无状态协议的模糊测试效率。

整体上看, 本文提出的方法利用基于 Seq2Seq 模型的种子生成模块获取客户端与服务端的报文交互, 得到丰富的高质量的种子。同时, 借助协议判别模块和针对性的适用策略, 提升了协议的模糊测试效率。

3.2 基于 Seq2Seq 模型的种子生成模块

本小节从数据处理和模块设计两方面展开, 对基于 Seq2Seq 模型的种子生成模块进行介绍。

3.2.1 数据处理

本文训练数据为通过网络捕获的 PCAP(Packet Capture File)数据文件, 可以利用 Scapy[29]提取{源 IP 地址、目的 IP 地址、源端口、目的端口、应用层载荷}五元组信息。两个应用程序进行通信时, 一方发起请求, 一方进行响应, 双方进行数据交换, 直至通信结束, 这一完整的过程可以视为一个会话。在这个过程中, 双方可能有多次请求和响应的交互。

在进行处理时, 可以根据目的 IP、源 IP、目的端口和源端口信息对会话进行划分。接着根据划分的会话, 将应用层载荷以 Q/A 对的形式进行关联。从客户端机器人的角度看, 服务端发送的报文被视为问题(Question), 而自身响应的报文则被视为应答(Answer), 问题和应答以 Q/A 对的形式关联在一起, 反映了服务端与客户端的关联关系。反之, 从服务端机器人的角度看, 客户端发送的报文被视为问题, 自身响应的报文则视为应答。表 1 和表 2 分别列举服务端机器人和客户端机器人所形成 Q/A 对的划分结果。如表 1 第 4 项, 同一方向连续发送报文, 本文将这些报文合并为一条记录。

为了确保训练数据与目标模型的输入格式相匹配, 需要对数据进行清洗和词元化(Tokenize)等操作。首先, 剔除可能对模型训练产生不利影响的非必要符号(如中文编码、乱码等)。鉴于协议交互报文中

表 1 客户端机器人 Q/A 对示例

Table 1 Interaction samples of clientbot

编号 No.	提问 Question	应答 Answer
1	220 (vsFTPd 2.2.2)	USER anonymous
2	331 Please specify the password.	PASS xxxxxx
3	230 Login successful.	LIST -al
4	150 Here comes the directory listing./ 226 Directory send OK.	QUIT
5	221 Goodbye	----

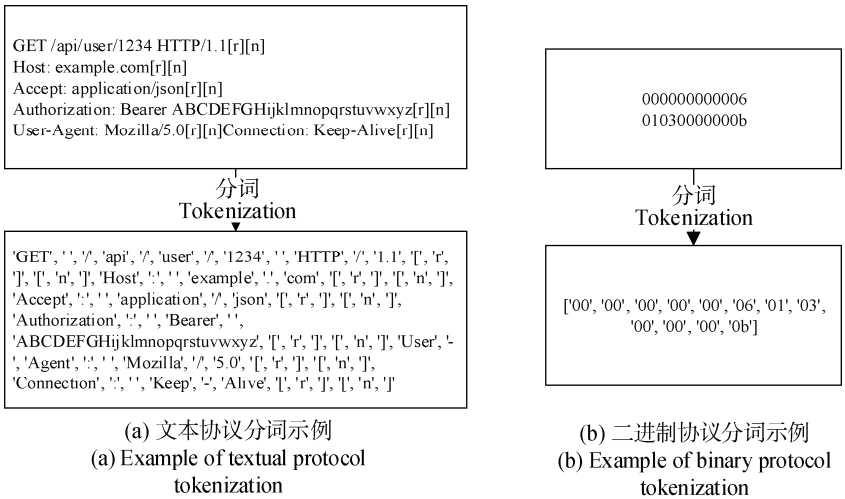
表 2 服务端机器人 Q/A 对示例

Table 2 Interaction samples of serverbot

编号 No.	提问 Question	应答 Answer
1	----	220 (vsFTPd 2.2.2)
2	USER anonymous	331 Please specify the password.
3	PASS xxxxxx	230 Login successful.
4	LIST -al	150 Here comes the directory listing./ 226 Directory send OK.
5	QUIT	221 Goodbye

特殊符号可能承载特定语义信息, 本文仅去除非英文字符, 保留其余特殊符号。为了防止换行符等转义字符对后续数据的影响, 采用文本符号对换行符等转义字符进行替换(例如采用[r][n]替换“\r\n”)。随后, 进行分词处理, 如图 2 所示, 对于 HTTP、RTSP 和 FTP 等文本协议, 利用空格等特殊符号作为边界进行分词。而对于像 Modbus 这类二进制协议数据, 则以固定长度进行划分, 默认以 2 字节进行数据划分。对于分词后的每个报文, 在报文结尾处加入结束符(<*EOS*>), 在报文开头处加入开始符(<*SOS*>), 旨在为每个报文提供明确的边界标识, 即指示报文的终止与起始。产生的文本在用作模糊测试的初始输入前, 为确保训练数据规范而对报文进行的调整都将逆向恢复, 确保输入报文的合法有效。

对报文数据执行分词处理后, 为了有效地将文本型数据转化为数值型矩阵, 以满足神经网络模型的输入要求, 需要将分词得到的词元(Token)映射为数值型的索引。图 2 中, 分词操作得到的“GET”、“/”、“HTTP”、“00”、“0b”等均为词元。进行映射的核心在于实现报文数据的向量化表达以满足神经网络的输入要求。首先, 对分词后的数据进行遍历, 从中抽取唯一词元, 构成一个无重复的词元集合。其次, 对词元集合进行排序, 构建出一个有序词表, 方便后续为词表中的元素建立数值索引。考虑到实际应用中可能出现未曾在训练阶段遇到过的数据, 引入了未知标记(<UNK>), 将其置于有序词表的末尾, 以此来代表那些未在词表中出现的实例。最后, 为有序词表中的每个词元分配编号以构建编号索引和词元的对应关系, 形成便于查询与转换的映射结构。在后续的模型处理阶段, 可以将文本词汇转换为对应的索引, 从而将报文转化成数值向量。



(a) 文本协议分词示例
(a) Example of textual protocol tokenization

(b) 二进制协议分词示例
(b) Example of binary protocol tokenization

图 2 协议报文分词示例

Figure 2 Example of protocol message tokenization

3.2.2 模型设计

服务端机器人和客户端机器人采用相同的架构, 如图 3 所示。本文机器人基于 Seq2Seq 模型设计, 主要包括编码器(Encoder)、解码器(Decoder)和注意力层(Attention Layer)3 个核心部分。

客户端/服务端机器人的编码器与解码器均基于 GRU 单元构建。选择 GRU 进行构建, 是由于 GRU 简洁, 计算效率高, 基于 GRU 的模型在训练

以及使用阶段资源消耗低，有利于提升整体系统的性能及适用范围。编码器采用了标准 GRU 的变体 BiGRU，编码器的结构如图 4 所示。在功能方面，编码器的主要任务是理解报文内容，从中抽取文本信息。BiGRU 结构允许信息在时间维度上双向传播，不仅考虑前序词元的影响，还可以兼顾后续词元，这使得模型能够充分地对接收到的报文进行学习。

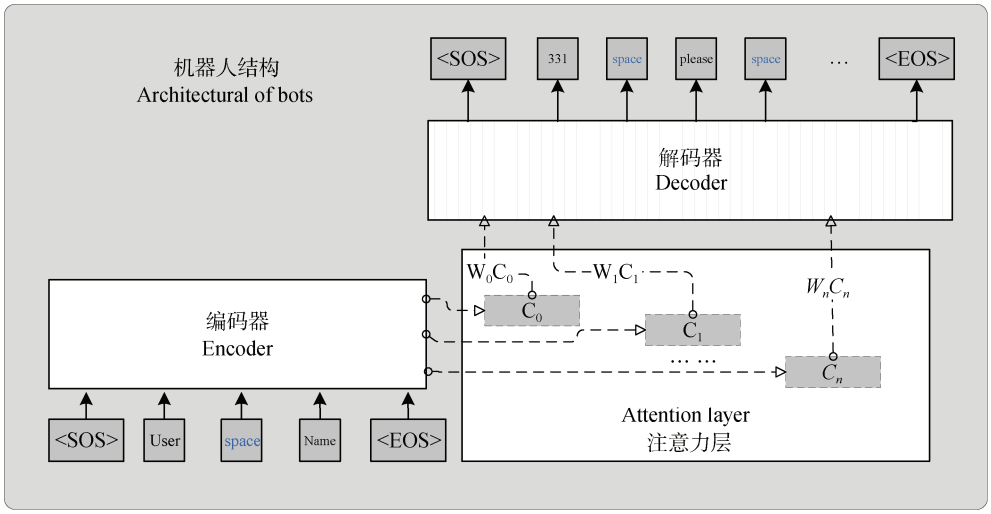


图 3　客户端/服务端机器人结构

Figure 3　Architecture of ServerBot/ClientBot

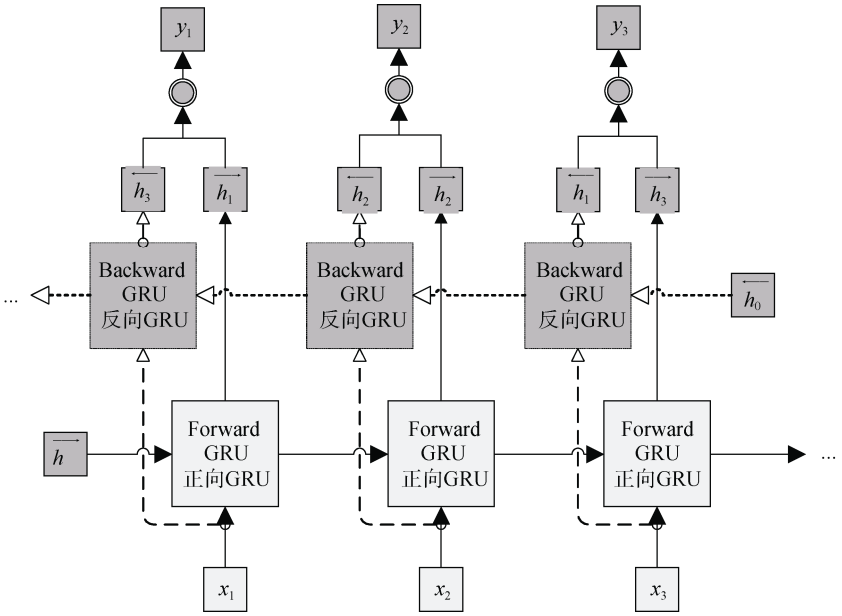


图 4　编码器结构

Figure 4　Architecture of encoder

在图 4 中，GRU 的正向信息传递以实线表示，反向信息传递以虚线表示。对于正向 GRU，编码器的输入为向量 $\boldsymbol{X}$，$\overrightarrow{h_0}$ 为一个随机初始化向量，与 $\boldsymbol{X}$ 中的分量一同参与到前向传播的计算。如式(1)所示，对于每个正向 GRU 单元而言，当前单元的输出 $\overrightarrow{h_t}$ 与前一个单元的输出 $\overrightarrow{h_{t-1}}$ 紧密相关。由此，正向单元的输出集合 $\vec{h}$ 可表示为式(2)。

$$\overrightarrow{h_t} = \mathrm{GRU}(\overrightarrow{h_{t-1}}, x_t) \tag{1}$$

$$\vec{h} = \{\overrightarrow{h_1}, \overrightarrow{h_2}, \overrightarrow{h_3}, ..., \overrightarrow{h_n}\} \tag{2}$$

$$\overleftarrow{h} = \{\overleftarrow{h_n}, \overleftarrow{h_{n-1}}, \overleftarrow{h_{n-2}}, ..., \overleftarrow{h_1}\} \tag{3}$$

$$\begin{aligned} \boldsymbol{Y} &= \{y_1, y_2, y_3, ..., y_n\} \\ &= \{\mathrm{concat}(\overrightarrow{h_1}, \overleftarrow{h_n}), \mathrm{concat}(\overrightarrow{h_2}, \overleftarrow{h_{n-1}}), ..., \mathrm{concat}(\overrightarrow{h_n}, \overleftarrow{h_1})\} \end{aligned} \tag{4}$$

反向 GRU 的计算与正向 GRU 相似。不同的是，对于输入向量 $\boldsymbol{X}$ 中的分量，反向 GRU 进行逆序计算，

反向单元输出集合 $\overleftarrow{h}$ 表示为式(3)。BiGRU 各个单元的输出由正向单元和反向单元进行拼接得到, BiGRU 单元的输出集合可以表示为式(4)的形式。

$$S_0 = \text{concat}(\overrightarrow{h_t}, \overleftarrow{h_1}) \tag{5}$$

编码器的下一层为注意力层(Attention Layer)。注意力层的输入 S_0 为编码器最后一个时间步 t 的输出, 其中既包含了前向 GRU 的信息也包含了后向 GRU 的信息。在注意力层, 模型根据当前时间步解码器的输出和报文向量计算权重, 根据权重有重点地学习某段报文信息, 而不是对报文中的各个字段进行均衡地处理, 有助于模型对输入报文的理解, 进而准确地根据输入报文生成反馈报文。注意力层和解码器层的结构如图 5 所示。

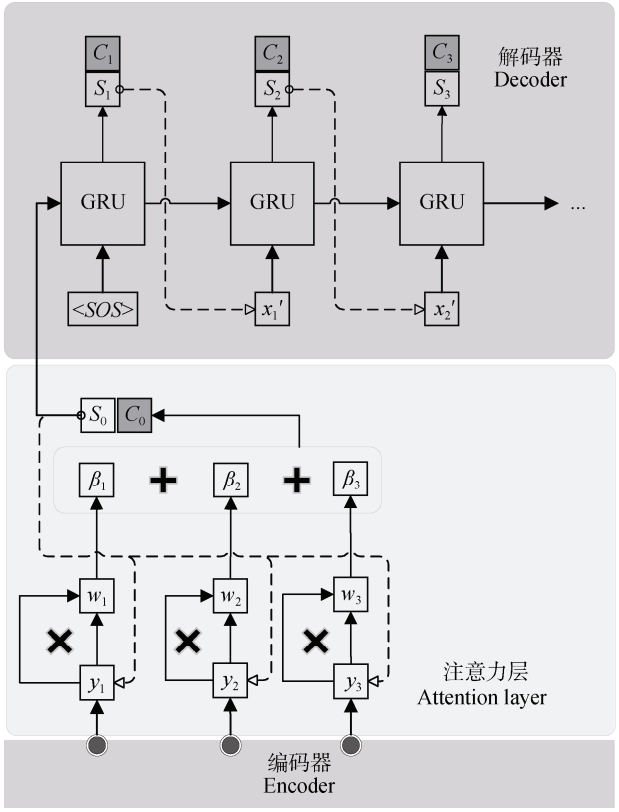


图 5 注意力层和解码器的结构

Figure 5 Architecture of Attention Layer and Decoder

在图 5 中, $S_i(i \in [0,m])$ 为相邻解码器单元的输出, S_0 为编码器的输出, m 为输出序列长度。如式(6), 要得到权重向量 $\boldsymbol{W}$, 需要将 S_i 与编码器的输出向量 y_i 进行对齐处理, $\boldsymbol{W}$ 的每个分量表示为 w_i, n 为解码器输入序列的长度。

$$\begin{aligned}\boldsymbol{W} &= [w_1, w_2, w_3, w_4, ..., w_n] \\ &= [\text{align}(y_1, S_i), \text{align}(y_2, S_i), ..., \text{align}(y_n, S_i)]\end{aligned} \tag{6}$$

w_i 的计算如式(7)。首先对编码器单元输出和解码器单元输出进行匹配度计算得到一个匹配度分数 τ_t。具体来看, 先运用 fc 函数对所有神经元执行关联计算(fc 函数本质上是一个全连接网络), 从而量化神经元之间的关注程度。随后, 为了对量化的关注度信息进行平滑化处理并限制其值在一个动态范围, 本文采用双曲正切函数(tanh)对量化的关注度信息进行非线性变换。最后, 将转换后的结果 τ_t 与学习到的 α 向量进行点积运算实现维度压缩, 并进一步通过 softmax 函数确保各权重值的归一化, 得出权重 w_i。

$$\begin{aligned}\tau_t &= \tanh(fc(\boldsymbol{Y}, S_i)) \\ w_i &= \text{softmax}(a^T \tau_t)\end{aligned} \tag{7}$$

最终, 依据权重 w_i 与对应编码器输出的隐藏状态 y_i 进行加权求和运算, 生成了上下文向量 C_i, 如式(8)所示。上下文向量中包含了输入权重信息, 具有高权重的元素将作为关键内容, 着重进行学习。后续将上下文向量 C_i 当前解码器隐状态 S_i 共同作为解码器的输入, 使得解码器在生成目标序列时能够聚焦于报文的关键内容, 提高了模型理解和生成报文的能力。

$$C_i = \sum_{i=0}^{n} w_i y_i \tag{8}$$

解码器采用单向 GRU 构建。解码器依据编码器所提取的输入报文特征, 逐词生成该输入报文的回复。遵循文本生成的自回归生成准则, 即在生成时刻 t 的词汇时, 仅依赖于已生成词汇, 而不涉及未来待生成词汇的信息。计算过程和编码器的前向传播层相同, 如式(1)所示, 不再赘述。

采样策略是指在文本生成任务中, 依据模型预测的概率分布来选定下一个词元的机制。简而言之, 就是依据模型对下一个词元出现概率的预测, 以一定规则抽取下一个词元。文本生成阶段的采样策略直接影响到所生成报文的多样性, 考虑到服务端机器人和客户端机器人行为模式的差别, 采样策略略有不同。

真实环境下, 服务端解析请求报文, 返回响应报文。响应报文反映的是服务器的状态, 报文内容较为固定, 行为模式相对单一。因此, 服务端机器人所生成的报文内容也相对固定。

客户端的行为往往较为多样, 在收到相同的服务端反馈后, 客户端进一步发送的请求报文多种多样。例如, 在登录后, 用户对资源的操作(读、写、删除等)并不固定, 所以客户端机器人在生成报文时也应考虑上述特点, 生成多样的报文。

本文采用束搜索作为基本的采样策略。束搜索可以通过控制束宽参数 k 对文本的多样性进行控制, k 较小时所产生报文较为固定, 可以应用于服务端机器人的构建, k 较大时产生报文多样性强, 可以用于构建客户端机器人。

束搜索的工作原理如下。假设词表为 Y, 束宽为 k (表示选择最有可能的 k 个词元作为候选, k 与文本多样性相关), c 代表对输入报文编码得到的上下文变量, 需要进行 t 个时间步的预测。第一个时间步取条件概率值 $P(y_1 \mid c)$ 排名前 k 的词元 $\{T_1, T_2, ..., T_k\}$ 作为当前步的候选词元。其中, $P(y_1 \mid c)$ 代表基于编码器对接收到报文的理解对词元进行预测所得到的概率分布, $y_1 \in Y$ 。由此, 根据 k 个候选词元产生 k 个分支, 代表着产生 k 种文本的可能。第二个时间步, 如式(9)所示, 共 k 个分支, 依据第一个时间步的计算结果对 y_2 进行预测, 其中 $y_2 \in Y$ 。

$$\begin{aligned} &P(T_1, y_2 \mid c) = P(T_1 \mid c) P(y_2 \mid T_1, c) \\ &\cdots\cdots \\ &P(T_k, y_2 \mid c) = P(T_k \mid c) P(y_2 \mid T_k, c) \end{aligned} \tag{9}$$

从词表 Y 中选择令式(9)条件概率最大的 k 个词元 $\{T_1', T_2', ..., T_k'\}$ 。第三个时间步的计算如式(10)。

$$\begin{aligned} &P(T_1, T_1', y_3 \mid c) = P(T_1, T_1' \mid c) P(y_3 \mid T_1, T_1', c) \\ &\cdots\cdots \\ &P(T_k, T_k', y_3 \mid c) = P(T_k, T_k' \mid c) P(y_3 \mid T_k, T_k', c) \end{aligned} \tag{10}$$

每一轮中都依据当前产生的条件概率产生 k 个分支, 进行候选词元的预测。最终依据每个时间步的输出, 计算条件概率的乘积, 取条件概率乘积最高的序列作为最终输出, 如式(11)所示。其中 γ 为惩罚值, 用于消除 t 过大时文本长度对计算的影响。

$$\frac{1}{t^{\gamma}} \log P(y_1, y_2, ..., y_t \mid c) = \frac{1}{t^{\gamma}} \sum_{t'=1}^{t} \log P(y_{t'} \mid y_1, ..., y_{t'-1}, c) \tag{11}$$

对服务端机器人, k 设置为较小的值(默认为 1), k 取 1 时每个时间步仅产生一个概率最大的分支, 也就是说, 对于服务端机器人而言, 仅在较少分支范围内进行报文的生成, 用于使其生成相对固定的报文。

对于客户端机器人, k 设置为较大的值(默认为 5, 具体可根据需求调整), 最终客户端机器人生成报文的候选队列中也会存在 k 个候选结果, k 越大其可能生成的报文就越多样, 基于此控制生成多样的请求报文, 模拟客户端多样的请求行为。

3.3 模糊测试模块

如图 6 所示, 本文模糊测试模块针对有状态协议和无状态协议采用了不同设计。如图 6(a)所示, 有状态协议模糊测试与目前主流的有状态协议的模糊测试流程相同。经过算法 1 的判定, 如果协议为有状态协议, 将进入该流程实施测试。

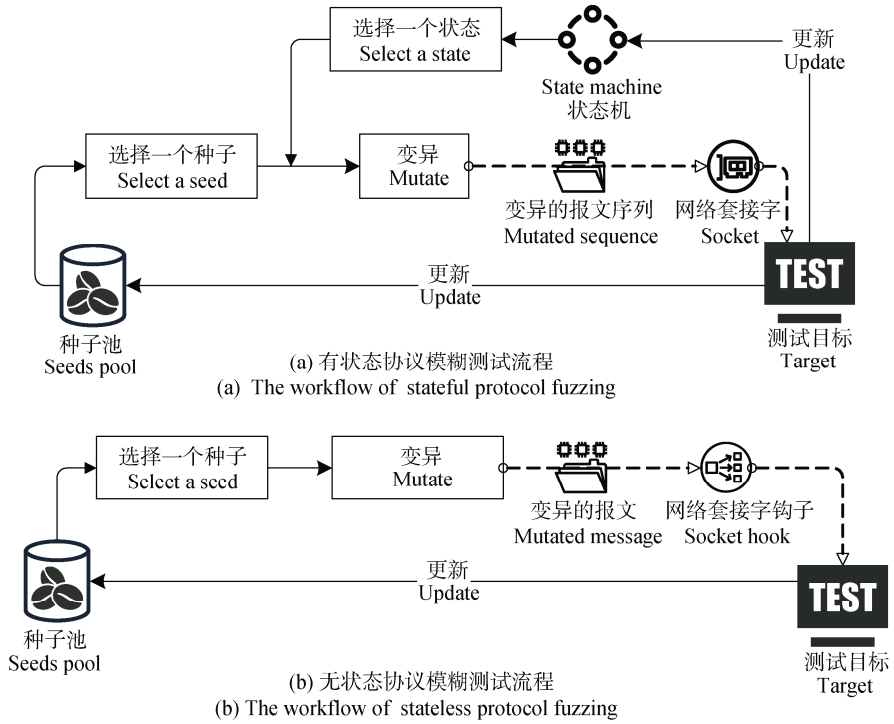


图 6 模糊测试工作流程

Figure 6 The workflow of fuzzing

种子池中的种子来自 3.2.2 中所述模块。系统从中选择一个种子，并确定种子的第 n 个报文为变异目标(第一次为随机选择，后续根据协议状态机进行选择)，发送前 n-1 条报文将目标引导至能够接收第 n 条报文的状态，对第 n 条报文进行变异，产生测试用例发送至目标，并发送剩余报文。测试过程中模糊器通过标准 Socket 和测试目标进行交互，根据测试目标的响应报文内容变化，确定状态的变化，并将导致状态变化的报文作为迁移条件，逐步动态地构建状态机。该过程示意如图 7 所示，测试过程中以及根据发送初始种子建立了 S1~S6 的状态机模型，当模糊器在 S2 状态下发送 M’得到了当前状态机中未出现的报文，从中提取关键字标识为 S7，将 S7 加入到状态机中，该状态可以由 S2 达到，迁移条件为 M’。状态机与程序覆盖率相结合，共同引导模糊测试的进行。Socket 的延时等待机制给了模糊器充分的时间和测试目标建立同步，确保构建相对准确的状态模型。在后续的测试中，如果测试用例可以覆盖到新的协议状态或使程序覆盖率提高，则将测试用例加入到种子池中，供后续测试使用。

如果程序所使用的协议被算法 1 判定为无状态协议，则进入图 6(b)所表示的无状态协议模糊测试流程。对无状态协议而言，服务器不再像处理有状态协议一样，会在应用层保留历史会话信息。这意味着服务器将所有请求独立处理，模糊器也无需对目标状态机进行学习，模糊器需要做的仅仅是收集测试过程中程序覆盖率的变化和错误触发信息，并依据这些指标进行种子池的更新。在有状态协议的模糊测试过程中，采用标准 Socket 延时等待机制给了模糊器充分的时间进行状态机的逐步学习。但这种等待对于无状态协议而言是不必要的，可以采用更高效的用例发送方式。

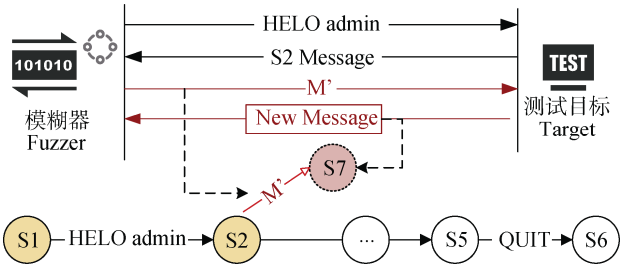


图 7 状态机的构建

Figure 7 Construction of state machines

如图 8 所示，模糊器发送用例的过程中涉及数据包的层层封装，测试目标需要对数据包进行拆封才能最终得到应用层数据，反之亦然。数据包的封装拆封，以及延时等待等机制使得使用 Socket 在无状态协议模糊测试的场景下非常低效。

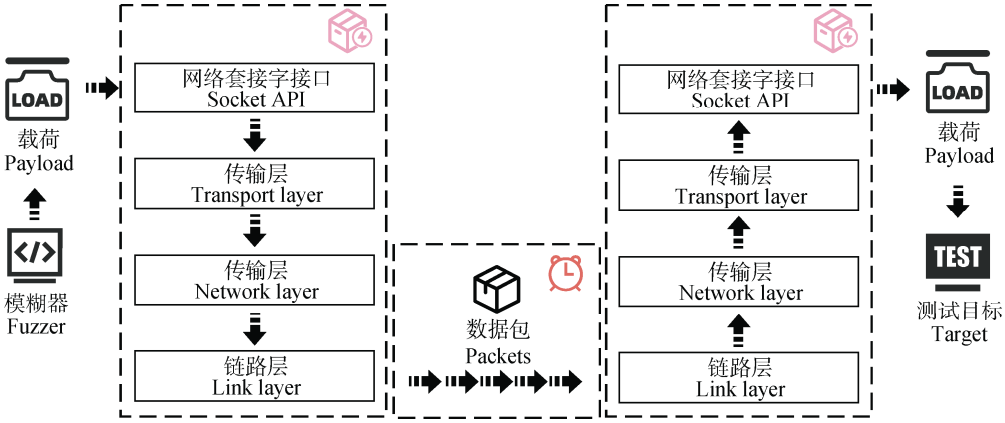


图 8 基于标准 Socket 的交互

Figure 8 Interaction based on standard Socket

如图 9 所示，本文采用 HOOK 技术，对标准 Socket 中数据交互相关 API 调用进行拦截，并重定向至本地的高速通信管道，以实现用例的高速投放。本地进程间通信机制规避了数据包封装、拆封等过程的性能损耗。本文方法重定向通信报文至本地域套接字。本地域套接字通常运用于进程间需要高效通信的场景，如容器内部通信、基于文件系统的认证和密钥交换，以及 Web 服务器和本地应用服务器间的通信。相比于易占用更多系统资源用于维护数据的消息队列和难以处理复杂通信模式的管道，本地域套接字的创建和管理灵活轻便。相比于使用共享内存的方法，本地域套接字由于支持多路复用、非阻塞等高级特性，更适用于协议模糊测试的场景。

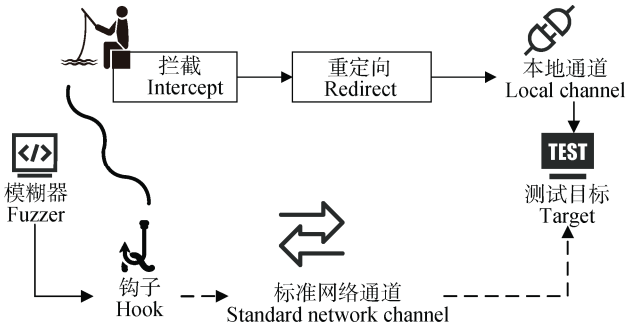


图 9　无状态协议模糊测试模块的交互策略

Figure 9　The interaction strategy of stateless protocol fuzzing module

算法 2. SOCKET 拦截替换算法

1. IF call *socket( )*:
2. Intercept and Redirect TO *custom_socket( )*
3. FUNCTION *custom_socket( )*:
4. Create Local Pair *front_socket* and *back_socket*
5. Allocate Memory for Threads;
6. Create *FRONT_TO_BACK_THREAD*
7. Create *BACK_TO_FRONT_THREAD*
8. RETURN *front_socket*

算法 2 展示了对 *socket( )*函数拦截和重定向的过程。协议实体程序调用标准 *socket( )*函数的操作会被拦截并重定向到函数 *custom_socket( )*。函数 *custom_socket( )*借助本地域套接字对 Socket 进行重写, 数据收发通过本地域套接字完成, 具体处理流程描述如下。首先创建前端套接字 *front_socket* 和后端套接字 *back_socket* 代替原有的标准前后端套接字, 处理通信过程的数据。前端套接字指的是客户端程序创建并用来连接到服务端的套接字, 主要涉及客户端的请求行为。后端套接字则是用于处理来自前端套接字的连接。为实现前后端套接字之间的数据同步, 确保后端能够及时处理来自前端的数据, 为前端套接字分配了必要的内存空间, 并创建 *FRONT_TO_BACK_THREAD* 和 *BACK_TO_FRONT_THREAD* 两个并行工作的同步线程, 这两个线程并行工作。*FRONT_TO_BACK_THREAD* 负责从前端套接字至后端套接字的数据传递, 该线程从客户端接收数据, 并将这些数据转发到后端处理单元。*BACK_TO_FRONT_THREAD* 负责从后端套接字至前端套接字的数据传递, 该线程从后端处理单元获取数据, 并响应给客户端。最后, 模糊器可以通过前端套接字与测试目标建立连接, 实现测试用例的发送。

通过这种处理, 可以减少传统方法在应用于无状态协议模糊测试时, 采用标准套接字带来的数据包封装拆封损耗, 以及网络波动和吞吐量对模糊测试的影响。

本文方法通过上述设计, 解决了现有模糊测试方法对有状态协议的强耦合问题, 提升了无状态协议的模糊测试效率, 能够更有效地对网络协议实施模糊测试。

4　实验验证

为了验证方法的有效性, 本文进行了相关的实验。4.1 节对训练数据的来源和测试对象进行说明。4.2 节首先从 BLEU 值(Bilingual Evaluation Understudy)的角度分析本文方法中 Seq2Seq 模型种子生成模块所生成的种子的可用性。随后, 依据 RFC 文档的规范, 对种子中包含字段进行统计和验证, 表明了本文方法能够广泛覆盖多样的报文字段。接着, 为进一步验证种子质量和方法的有效性, 在 4.3 节和 4.4 节将种子运用于实际模糊测试中并与基准方法进行对比。其中 4.3 节针对有状态协议, 从路径覆盖和异常触发效果等方面进行评估。4.4 节针对无状态协议, 从执行速率、路径覆盖和错误触发等方面进行测试。

本文实验环境为 Ubuntu 20.04 LTS, 运行内存为 32 G, GPU 采用 GeForce RTX 3060 12G 版本。

4.1　训练数据和测试目标

基于 Seq2Seq 的种子生成模块的训练数据主要来自 Modbus 2023[30]、ISCX-Tor-NonTor-2017(修订日期: 2024.02.01)[31]、ISCX2012(修订日期: 2024.02.01)[32], Wireshark 示例数据集以及部分自主收集的网络流量数据[33]。此外, 也借鉴文献[24]方法, 通过 Netzob[11]这种基于语法的流量发生器, 并通过 Sulley[12]这种基于生成的模糊器产生报文, 与协议程序交互产生数据, 进一步作为训练数据的补充以提高训练数据的质量和丰富度。在累积包含 20000 组共计 40000 条交互报文样本集合上进行了训练, 样本覆盖了 HTTP、RTSP、Modbus 和 FTP 四种代表性网络协议, 其中 HTTP、RTSP 和 FTP 属于文本型协议, Modbus 是二进制型协议。HTTP 和 Modbus 属于无状态协议, RTSP 和 FTP 属于有状态协议。

模糊测试目标选取了包括 Nginx、LightFTP、Live555 以及 Libmodbus 等具有代表性的协议软件进行测试实验。被测软件如表 3 所示。其中, LightFTP 为一个轻量化开源文件传输服务软件, 是典型的 FTP 协议程序。版本 5980ea1 在模糊测试工具的有效性测试中应用广泛。Live555 作为一款广受欢迎的开源实时流媒体传输软件, 主要用于满足多媒体直播和点播服务的需求, 软件所使用的 RTSP 协议属于有

状态协议。本文以广泛使用的 2021.10.31 版本作为测试目标。

表 3　测试目标信息

Table 3　Target information

编号 No.	实例名 Name	使用的协议 Protocol	是否有状态	版本
1	LightFTP	FTP	Yes	5980ea1
2	Live555	RTSP	Yes	2021.10.31
3	Libmodbus	Modbus	No	3.0.8 stable
4	Nginx	HTTP	No	1.4.0 1.4.2 1.4.4 1.4.6

Libmodbus 和 Nginx 分别为 modbus 和 HTTP 的协议程序。Libmodbus 在工控领域应用广泛, Nginx 是 Web 领域应用广泛的服务程序。尤其是 Nginx, 作为主流 Web server 之一, 其安全性不容忽视, 本文对 Nginx 1.4.0、Nginx 1.4.2、Nginx 1.4.4 和 Nginx 1.4.6 等 4 个版本的程序进行了测试。

4.2　种子质量和多样性

BLEU 值广泛应用于自然语言处理领域。BLEU 值基于 n-gram 匹配, 量化评估文本生成质量。本文 BLEU 值的计算如式(12)。在文本生成任务中, BLEU 值可以反映生成文本的质量。

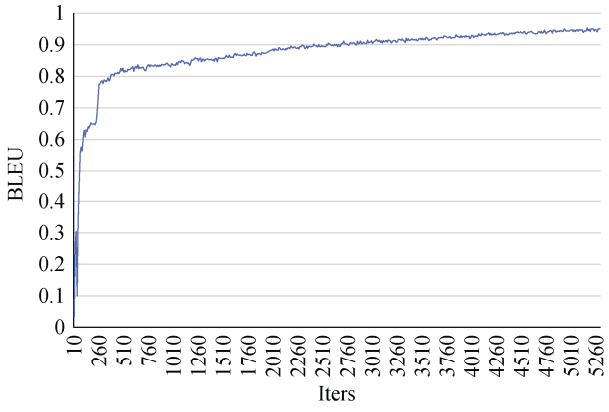
$$BLEU=\begin{cases}\exp\left(\sum_{n=1}^{N}w_n\log p_n\right), \text{if } \operatorname{len}(M_g)>\operatorname{len}(\min(M_A))\\ BP\times\exp\left(\sum_{n=1}^{N}w_n\log p_n\right), \text{if } \operatorname{len}(M_g)\le\operatorname{len}(\min(M_A))\end{cases} \tag{12}$$

式(12)中, p_n 表示 n-gram 精确度。首先需要计算预测生成的报文中每个 n-gram(N 为 n-gram 最大长度, 默认为 4)与参考应答报文(问/答对中的“答”)中所有 n-gram 的匹配情况。n-gram 精确度是匹配成功的 n-gram 数量除以生成报文中的 n-gram 总数。w_n 为权重系数, 默认为 1, exp 表示自然指数函数。BP 代表惩罚值, 若生成的报文比参考回答中最短的报文还要短, 则认为生成的报文过于简略, 可能存在字段丢失等情况, 需要进行一定的惩罚。BP 值的计算如式(13), 当预测生成的报文长度小于参考报文长度时, BP 小于 1, 本文认为这种情况下生成的报文质量较低。

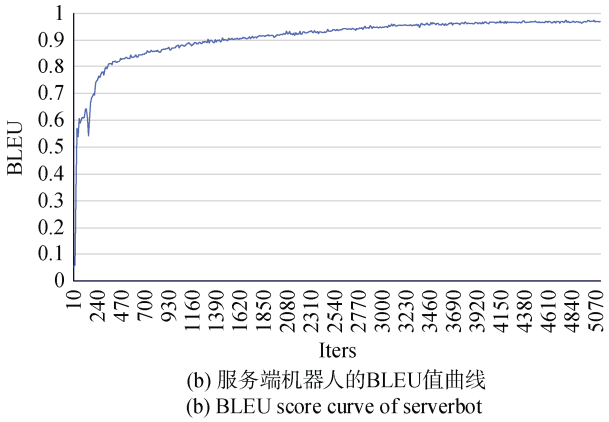
$$BP=e^{1-\frac{\operatorname{len}(M_a)}{\operatorname{len}(M_g)}} \tag{13}$$

经过上述计算, 所得 BLEU 值在[0,1]区间, 值越接近 1 说明生成报文越接近预期。在实验场景下, BLEU 值越高说明机器人生成的报文与真实环境的报文越相似, 其中作为对比的报文均来自 4.1 节所述数据集, 这些数据集本身汇聚了真实生产环境及工业环境的丰富报文实例, 确保了评估过程的真实有效。

服务端机器人 BLEU 值达到了 0.97、客户端机器人 BLEU 值达到了 0.95。这一数值反映了模型生成的交互报文在结构和内容层面高度贴合实际数据。机器人的 BLEU 值变化曲线如图 10, 随着训练轮次的增多, 服务端机器人和客户端机器人的 BLEU 值总体呈现上升趋势, 服务端机器人和客户端机器人均在 5300 轮左右时, BLEU 值逐渐趋于稳定。



(a) 客户端机器人的BLEU值曲线
(a) BLEU score curve of clientbot



(b) 服务端机器人的BLEU值曲线
(b) BLEU score curve of serverbot

图 10　机器人的 BLEU 值曲线

Figure 10　BLEU score curve of bots

从实验结果可以看出, 种子生成模块具备生成高质量种子的能力, 能够产生与真实报文高度相似的报文作为种子。

此外, 实验中采用与文献[24]相同的方法对种子的合法性进行验证。本文通过检查生成种子中的特定字段, 包括文本协议的首部字段(如 FTP 命令中的“USER”、“PASS”等)以及二进制协议中的功能码或控制码等信息, 来确保这些字段符合征求意见稿(Request for Comments, RFC)文档的要求。鉴于本文所使用

的训练数据均源于真实环境, 这些数据本身就包含了丰富的、符合网络交互规范的报文样本。在训练过程中, 模型通过学习真实报文的特征, 逐步掌握了生成合法、有效种子的能力。由本文方法产生的种子, 在报文结构上均符合 RFC 文档的要求。同时, 如表 4 所示, 本文生成的种子包含丰富的字段, 采用本文方法可以有效地对程序的执行分支进行覆盖。此外, 通过持续收集数据进行训练, 本文方法所涵盖的字段还可以进一步扩充, 提升测试的覆盖范围。

表 4 所生成种子包含的字段

Table 4 Fields included in the generated seeds

编号 No.	协议 Protocol	字段(或功能码) Field(or Function Code)
1	FTP	USER, PASS, CWD, CDUP, PWD, LIST, NLST, SIZE, TYPE, PORT, PASV, RETR, STOR, DELE, RMD, MKD, RNFR, RNTO, QUIT
2	RTSP	CSeq、Session、Transport、Range、User-Agent、Authorization、Content-Length、Content-Type、Server、Public、Speed、Scale、Timestamp、Require、Connection、Session-Timeou
3	HTTP	Accept、Accept-Charset、Accept-Encoding、Accept-Language、Authorization、Cache-Control、Connection、Content-Length、Content-Type、Date、Expect、From、Host、Upgrade、User-Agent、GET、POST、HEAD、PUT、DELETE、OPTIONS、CONNECT、TRACE、PATCH、MOVE、COPY
4	Modbus	0x01、0x02、0x03、0x04、0x05、0x06、0x15

4.3 有状态协议模糊测试

为进一步评估本文生成种子的有效性, 本文将所生成的种子应用于有状态协议的模糊测试, 并与 AFLNet 模糊测试工具进行横向对比。本文采用种子生成模块生成种子进行测试, 高质量且包含字段种类丰富的种子应该能够促使测试不断深入, 而包含字段种类有限的种子会导致测试停滞, 测试工具在相当长的时间内无法探索到新的程序路径。

本文选择测试路径覆盖数作为指标进行评价。表 5 是 5 组针对 LightFTP 5980ea1 的实验结果。从实验结果可以看出, 采用 AFLNet 官方示例输入的情况下, 在 5 轮的实验测试中, 原生 AFLNet 在运行至 30~45 min 不等的时间均发生停滞, 并且一直到测试结束都未能有效地提升路径覆盖数。由于受到初始输入集局限的影响, 难以触发新的执行路径, 路径覆盖数长时间停滞在 200 左右。

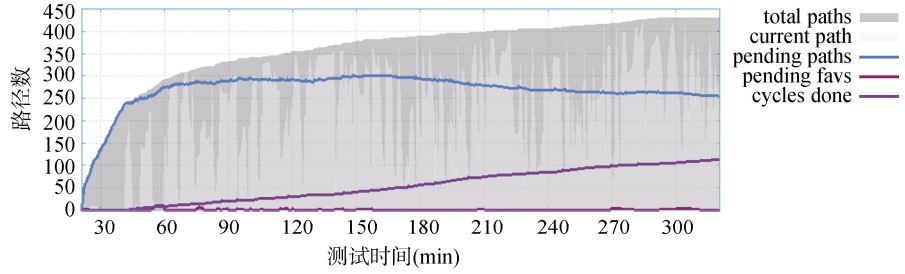
表 5 AFLNet 停滞信息

Table 5 Stagnation of AFLNet

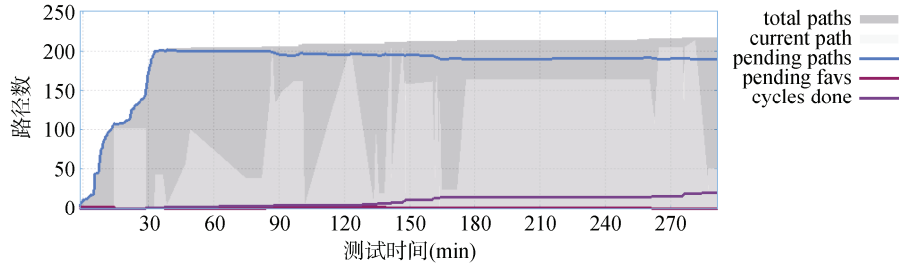
编号 No.	实例名 Name	发生停滞的时间(min) The time when stagnation occurs	持续时间(min) Duration	路径数 Paths
1	LightFTP	30.3	269.7	213
2	LightFTP	33.0	267.0	207
3	LightFTP	35.0	265.0	201
4	LightFTP	42.0	258.0	242
5	LightFTP	44.2	255.8	224

由于本文方法的种子生成模块所生成的种子覆盖了较多字段和协议状态, 所以在测试中, 可以保证测试的全面, 更有可能触发异常。本文重点关注总路径数(Total Paths)和待处理路径(Pending Paths)两项指标。总路径数代表测试过程中发现的系统路径, 发现的路径数越多, 代表测试的覆盖面越广。待处理路径指的是那些已经被发现但还未针对其挑选相关种子进行变异以进行深入探索的路径。对待处理路径的分析有助于探索深层次的代码区域。如图 11 所示, 在针对 LightFTP 5980ea1 的实验中, 本文方法历经 300 min 的测试后, 成功覆盖了超过共 400 个执行路径。相较于仅采用预设初始输入的原生 AFLNet, 本方法在路径覆盖率上达到原生 AFLNet 的两倍左右。并且, 如图 11(a)所示, 在手动终止测试之前, 本文方法仍持续提升路径覆盖的数量。并且, 待处理路径数从第 120 分钟开始稳定下降。这说明在模糊测试过程中, 大部分路径得到了深入探索。而 AFLNet 不仅在测试过程中发生了停滞, 待处理路径数也长期没有减少, 程序难以被深入探索, 如图 11(b)所示。从实验结果可以看出, 本文方法的总路径数和待处理路径数的指标均优于对比方法。

测试覆盖的路径数直接影响到异常触发能力, 如表 6 和图 12 所示。在 300 min 的测试时间内, 本文方法成功引起了测试目标的两次异常挂起(Unique Hang)。其中, 第一次异常挂起出现在测试开始后的第 38.4 分钟, 第二次挂起出现在测试开始后的第 91.3 分钟。触发两次异常挂起的测试用例均在 Havoc 变异阶段产生。与之相比, AFLNet 在相同测试时长中受种子的限制, 难以有效覆盖足够的程序执行路径, 也无法对程序进行深入的测试, 在测试时间内没有触发程序错误。



(a) 本文方法覆盖到的路径数
(a) The number of paths covered by Our method



(b) AFLNet方法覆盖到的路径数
(b) The number of paths covered by AFLNet

图 11　LightFTP 的路径覆盖信息

Figure 11　Path coverage for LightFTP

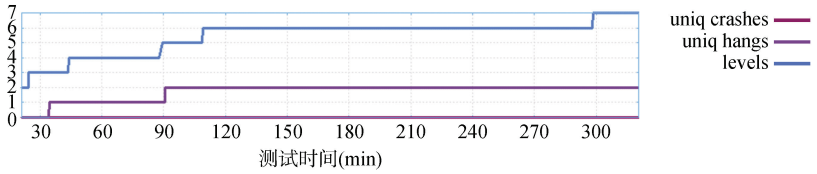
表 6　错误触发信息

Table 6　Error triggering information

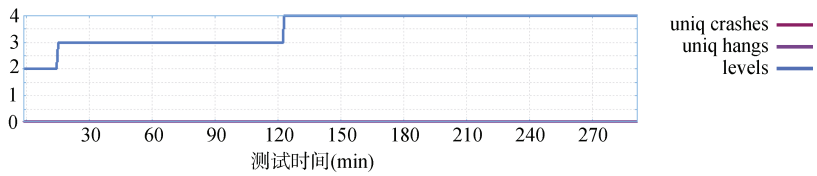
方法 Method	实例名 Name	异常名 ERROR	触发次数 Times
本文方法	LightFTP	Unique Hang	2
AFLNet	LightFTP	--	0

对 Live555 的测试结果如表 7 所示。在历经 120 分钟的模糊测试之后，运用本文方法对 Live555 RTSP 测试共计覆盖了 564 条执行路径。AFLNet 则覆盖了 472 条路径，本文方法在同等时间内覆盖了更多的路径，具有更高的测试覆盖率。

对协议实体程序而言，超时异常(Timeout)也是需要关注的重要指标。协议服务通常注重高效性，而某些特定的输入可能使得程序执行的消耗剧增，这是导致超时异常的原因之一。这些输入可能引起服务器资源耗费剧增进而导致服务处理的低效甚至出现拒绝服务。在测试期间，采用本文方法触发了 135 起独特超时异常(Unique Timeout)，而 AFLNet 触发了 105 起独特超时异常，本文方法在同等时间内多触发了更多的超时异常。进一步对 AFLNet 提供的初始种子与本文方法所生成的种子进行对比分析。本文方法所生成的种子不受限特定类型文件，相比官方所提供的针对“.mp3”、“.mkv”等有限类型的请求报文，本文还支持生成针对“.mov”等类型的请求报文，



(a) 本文方法错误触发信息
(a) Error triggering information by our method



(b) FLNet错误触发信息
(b) Error triggring information by AFLNet

图 12　LightFTP 错误触发信息

Figure 12　Error triggering information of LightFTP

表 7 Live555 测试结果

Table 7 Fuzzing results of Live555

方法 Method	测试时长 Time	总路径 Total paths	独特超时 Unique timeout
本文方法	120 min	564	135
AFLNet	120 min	472	105

此外本文方法生成的报文中所含字段也更加丰富, 如请求路径并不限定于环回测试地址, 还包含了实际的 URL, 再如 User-Agent 字段中的字符串也不仅仅是“/testRTSPClient (LIVE555 Streaming Media v2018.08.28)”之类的测试信息, 而且包含了“LibVLC/3.0.12 (LIVE555 Streaming Media v2016.11.28)”等多种版本的客户端信息, 内容更加丰富。总体上看, 在本文方法所生成种子的基础上进行变异, 能够构造出更加复杂多样的测试用例, 使目标程序处理压力增加造成超时异常。

根据上述实验结果, 本文方法可以产生高质量的种子, 在此基础上进行变异产生的测试用例能够触达更广的代码空间, 提高有状态场景下模糊测试的覆盖率, 提升发现漏洞的概率。

4.4 无状态协议模糊测试

本小节介绍针对无状态协议模糊测试的实验结果, 从执行速率、路径覆盖以及异常触发能力几个方面分析本文方法的性能。

传统方法基于标准 Socket 进行通信, 数据包封装过程和数据包解析过程中的逐层操作, 发送过程中延时等待等因素均会造成用例执行速度的降低。本文方法重定向数据至本地高速通信管道中, 绕过了数据包逐层封装的过程, 减少了通信过程中的损耗。

图 13 中执行速率是指每秒执行的测试用例数, 如 20 execs/s 表示每秒执行 20 个测试用例。如图所示, 本文方法在应用于无状态协议的测试场景时的执行速度和稳定性上均明显超过了 AFLNet。本文方法执行速度稳定在了 200 execs/s 左右, 测试过程中波动较小, 说明本文方法具有较高的稳定性。而 AFLNet 发生了 20 execs/s 至 40 exec/s 不等的波动。本文方法的平均执行速度达到 AFLNet 的 6 倍左右, 最高执行速率和最低执行速度均超过 AFLNet。

在此基础上, 本文对 Libmodbus 和 Nignx 进行了测试。在针对 Libmodbus 的实验中, 分别采用本文种子生成模块生成的种子和采用随机捕获报文作为种子。图 14(a)是采用本文方法进行测试的结果, 图 14(b)是采用随机输入作为种子进行测试的结果。可以观察到相同时间内, 本文方法覆盖的路径数超过了 200 条, 相较于随机捕获流量作为种子, 额外覆盖了 72 条路径。并且, 如图 14(a)所示, 本文方法从测试过程第 27.2 分钟起, 待测路径数稳定下降, 说明新路径在测试过程中能够被有效复用, 进行充分测试, 有助于探索更多的执行路径。相比之下, 图 14(b)中采用随机输入时, 在测试的第 54.4 分钟才发生相同现象。据此可以推断, 本文方法具备在更短时间内测试更多程序路径的能力。

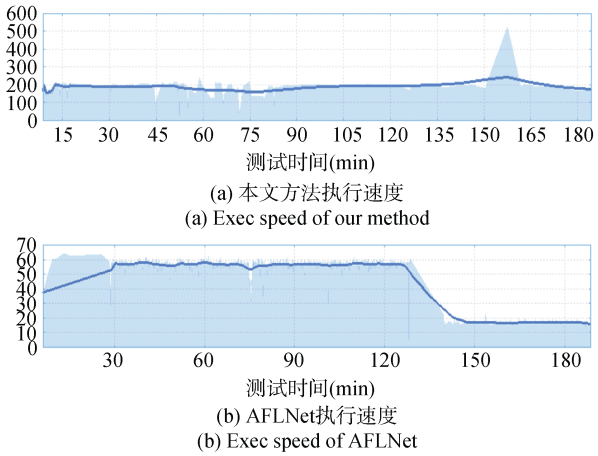
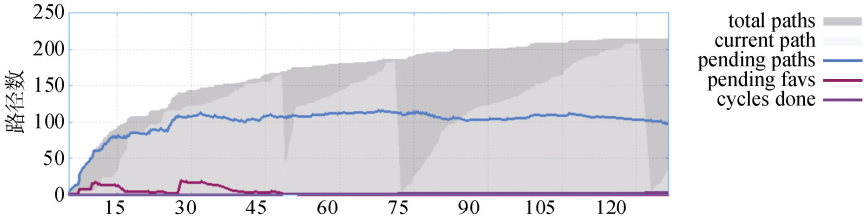


图 13 执行速度

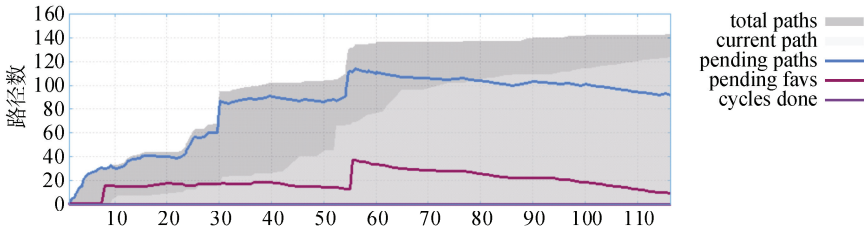
Figure 13 Execution speed

本文对多个发行版本的 Nginx 进行了测试, 其中包括 Nginx 1.4.0、Nginx 1.4.2、Nginx 1.4.4 和 Nginx 1.4.6, 表 7 展示了测试结果。本文对每个实体程序分别进行长达 180 min 的模糊测试。在对 Nginx 1.4.0 版本的测试中, 成功发掘了共 607 条程序执行路径, 并诱发了 1976 次超时, 其中 112 次为独特超时。此外, 测试过程中, 本文观察并记录了一次程序崩溃现象。对于 Nginx 1.4.2 版本, 共覆盖了 495 条执行路径, 触发了 5399 次超时, 其中 107 次为独特超时。在针对 Nginx 1.4.4 版本的模糊测试环节, 覆盖了 597 条执行路径, 并引发了 1041 次超时事件, 其中 96 次为独特超时。对于 Nginx 1.4.6 版本, 在 180 min 内覆盖了 500 条执行路径, 并触发了 754 次超时事件, 其中 87 次为独特超时。鉴于本文方法所生成的种子包含丰富且复杂的字段信息, 以这些种子为基础进行变异, 能够衍生出丰富多样的测试用例。这些测试用例不仅具有使测试目标发生超时的能力, 而且通过采用高效的用例发送策略, 显著加速了用例的迭代生成。总体上看, 本文提出的方法能够有效生成触发超时事件的测试用例。

根据上述实验分析, 本文方法在无状态协议模糊测试场景中, 在用例投递速率上远超基准方法。同时, 本文方法可以有效对无状态协议进行深入分析, 探索程序执行路径, 触发程序异常。



(a) 本文方法覆盖到的路径数
(a) The number of paths covered by our method



(b) 采用随机输入覆盖到的路径数
(b) The number of paths covered by Random input

图 14　Libmodbus 的路径覆盖信息
Fig 14　Path coverage information for Libmodbus

表 7　Nginx 测试结果
Table 7　Fuzzing results of Nginx

指标 Index	Nginx 1.4.0	Nginx 1.4.2	Nginx 1.4.4	Nginx 1.4.6
路径 Total paths	607	495	597	500
总超时 Total timeouts	1976	5399	1041	754
独特超时 Unique time-outs	112	107	96	87
是否崩溃 Crash or not	Yes	NO	NO	NO
最高执行速度 Highest Exec speed	501.19 execs/s	198.40 execs/s	194.41 execs/sec	196.09 execs/sec
最低执行速度 Lowest Exec speed	31.67 execs/s	42.59 execs/s	42.52 execs/s	67.26 execs/s

5　总结和展望

针对传统模糊测试方法种子获取过程烦琐、种子质量和丰富程度难以保证以及紧密耦合于有状态协议导致的泛化能力弱等问题, 本文提出了一种基于序列到序列模型的网络协议模糊测试方法。

通过 Seq2Seq 模型模拟服务端和客户端之间的交互过程, 生成丰富且贴近实际的交互数据作为测试种子。这种方法不仅简化了种子的获取流程, 确保种子质量的同时兼顾了种子的多样性, 进而增强了变异产生的测试用例的有效性。通过高质量的种子变异生成的测试用例, 能够更有效地探索网络协议的不同执行路径, 特别是在针对有状态协议执行模糊测试时, 其路径覆盖能力相较于基准方法有了显著提升。针对现有方法紧密耦合于有状态协议的问题, 本文提出了适应性策略。首先判断目标协议是否为有状态协议, 并据此采用不同的测试策略。对于无状态协议, 通过打乱报文序列发送、重定向数据到本地域套接字, 实现了测试用例的高效投递, 显著提高了针对无状态协议时的执行速率。

本文方法也存在一定的局限。部分软件没有测试模式, 直接对软件进行测试难以排除守护进程的干扰。一方面, 守护进程的存在可能导致测试时模糊器和测试目标状态不一致。另一方面, 守护进程可能令测试目标难以达到异常状态, 掩盖了测试目标中的问题。为排除守护进程对测试的干扰, 本文方法需要对目标协议源代码进行必要的修改。此外, 在报文变异过程中, 如何精确对字段进行有效变异, 提高变异后有效用例的比例也是一个需要考虑的问题。

未来计划结合强化学习技术优化变异策略, 更精准地定位和变异报文中的关键变量。同时, 还将进一步结合自然语言处理技术, 自动高效地对程序源码进行修改, 从而规避守护进程对模糊测试的影响。

参考文献

[1] Zhang Z W, Zhang H Z, Zhao J J, et al. A survey on the development of network protocol fuzzing techniques[J]. *Electronics*, 2023, 12(13): 2904.

[2] Mohurle S, Patil M M. A brief study of wannacry threat: Ransomware attack 2017[J]. *International Journal of Advanced Research*

in Computer Science, 2017, 8: 1938-1940.

[3] Singh V, Kumar V, Singh V B. A hybrid novel fuzzy AHP-TOPSIS technique for selecting parameter-influencing testing in software development[J]. *Decision Analytics Journal*, 2023, 6: 100159.

[4] Fioraldi A, Maier D, Eißfeldt H, et al. {AFL++}: Combining Incremental Steps of Fuzzing Research[C]. 1*4th USENIX Workshop on Offensive Technologies,* 2020.

[5] Pham V T, Böhme M, Roychoudhury A. AFLNET: A Greybox Fuzzer for Network Protocols[C]. *2020 IEEE 13th International Conference on Software Testing*, *Validation and Verification*, 2020: 460-465.

[6] Andronidis A, Cadar C. SnapFuzz: High-Throughput Fuzzing of Network Applications[C]. *The 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022: 340-351.

[7] Palasundram K, Mohd Sharef N, Nasharuddin N A, et al. Sequence to sequence model performance for education chatbot[J]. *International Journal of Emerging Technologies in Learning*, 2019, 14(24): 56.

[8] Bahdanau D, Cho K, Bengio Y. Neural Machine Translation by Jointly Learning to Align and Translate[EB/OL]. 2014: arXiv: 1409.0473. https://arxiv.org/abs/1409.0473.

[9] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. *Advances in Neural Information Processing Systems*, 2017: 30.

[10] She D M, Jia M P. A BiGRU Method for Remaining Useful Life Prediction of Machinery[J]. *Measurement*, 2021, 167: 108277.

[11] Boofuzz: A fork and successor of the sulley fuzzing framework. https://github.com/jtpereyda/boofuzz, 2017, accessed: 2023-11-11.

[12] Devarajan G. Unraveling SCADA Protocols: Using Sulley Fuzzer[C]. *Defon 15 Hacking Conference*, 2007.

[13] Luo Z X, Zuo F L, Shen Y H, et al. ICS Protocol Fuzzing: Coverage Guided Packet Crack and Generation[C]. *2020 57th ACM/IEEE Design Automation Conference*, 2020: 1-6.

[14] Harnett C. Open Source Hardware for Instrumentation and Measurement[J]. *IEEE Instrumentation & Measurement Magazine*, 2011, 14(3): 34-38.

[15] Bellard F. QEMU, a Fast and Portable Dynamic Translator[C]. *The Annual Conference on USENIX Annual Technical Conference*, 2005: 41.

[16] Natella R. StateAFL: Greybox fuzzing for stateful network servers[J]. *Empirical Software Engineering*, 2022, 27(7): 191.

[17] Qin S S, Hu F, Ma Z Y, et al. NSFuzz: Towards efficient and state-aware network service fuzzing[J]. *ACM Transactions on Software Engineering and Methodology*, 2023, 32(6): 1-26.

[18] Yu B, Wang P F, Yue T, et al. Poster: Fuzzing IoT Firmware *via* Multi-Stage Message Generation[C]. *The 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019: 2525-2527.

[19] Song C X, Yu B, Zhou X, et al. SPFuzz: A Hierarchical Scheduling framework for stateful network protocol fuzzing[J]. *IEEE Access*, 2019, 7: 18490-18499.

[20] Fu J S, Xiong S, Wang N, et al. A framework of high-speed network protocol fuzzing based on shared memory[J]. *IEEE Transactions on Dependable and Secure Computing*, 2024, 21(4): 2779-2798.

[21] Wang X J, Hu C Z, Ma R, et al. LAFuzz: Neural Network for Efficient Fuzzing[C]. *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence*, 2020: 603-611.

[22] Hu Z C, Shi J Q, Huang Y H, et al. GANFuzz: A GAN-Based Industrial Network Protocol Fuzzing Framework[C]. *The 15th ACM International Conference on Computing Frontiers*, 2018: 138-145.

[23] Zhao Y F, Dong Y H, Li G. Seq2Seq or Seq2Tree: Generating Code Using both Paradigms *via* Mutual Learning[C]. *The 14th Asia-Pacific Symposium on Internetware*, 2023: 238-248.

[24] Gao Z C, Dong W Y, Chang R, et al. The Stacked Seq2seq- Attention Model for Protocol Fuzzing[C]. *2019 IEEE 7th International Conference on Computer Science and Network Technology*, 2020: 126-130.

[25] Yang L Q, Wei C R, Yang J, et al. Seq2Seq-AFL: Fuzzing *via* Sequence-to-Sequence model[J]. *International Journal of Machine Learning and Cybernetics*, 2024, 15(10): 4403-4421.

[26] Torres J, Vaca C, Terán L, et al. Seq2Seq models for recommending short text conversations[J]. *Expert Systems with Applications*, 2020, 150: 113270.

[27] Yu B, Su J S, Yang Q, et al. Survey on vulnerability mining techniques of network protocol software[J]. *Journal of Software*, 2024, 35(2): 872-898.

[28] (喻波, 苏金树, 杨强, 等. 网络协议软件漏洞挖掘技术综述[J]. *软件学报*, 2024, 35(2): 872-898.)

[29] Zeng Y P, Lin M M, Guo S Q, et al. MultiFuzz: A coverage-based multiparty-protocol fuzzer for IoT publish/subscribe protocols[J]. *Sensors*, 2020, 20(18): 5194.

[30] S R R, R R, Moharir M, et al. SCAPY- a Powerful Interactive Packet Manipulation Program[C]. *2018 International Conference on Networking*, *Embedded and Wireless Systems*, 2019: 1-5.

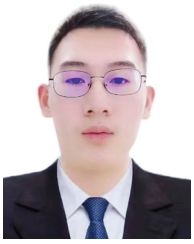
[31] Boakye-Boateng K, Ghorbani A A, Lashkari A H. Securing Substations with Trust, Risk Posture, and Multi-Agent Systems: A Comprehensive Approach[C]. *2023 20th Annual International Conference on Privacy*, *Security and Trust*, 2023: 1-12.

[32] Habibi Lashkari A, Draper Gil G, Mamun M S I, et al. Characterization of Tor Traffic Using Time Based Features[C]. *The 3rd International Conference on Information Systems Security and Privacy*, 2017: 253-262.

[33] Wei G L, Wang Z H. Adoption and realization of deep learning in network traffic anomaly detection device design[J]. *Soft Computing*, 2021, 25(2): 1147-1158.

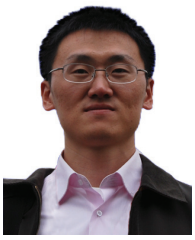
陈乾 于 2021 年在南京理工大学紫金学院软件工程专业获得学士学位。现在陆军工程大学网络空间安全专业攻读硕士学位。研究领域为协议安全、软件安全。研究兴趣包括模糊测试、协议逆向。Email: CHENQianLGD@163.com

洪征 于 2007 年在解放军理工大学计算机应用专业获得博士学位。现任陆军工程大学指挥控制工程学院副教授。研究领域为协议安全、软件安全。研究兴趣包括协议逆向、模糊测试。Email: hz5215@163.com



古津榜 于 2023 年在陆军工程大学网络工程专业获得学士学位。现在陆军工程大学网络空间安全专业攻读硕士学位。研究领域为软件安全、AI 安全。研究兴趣包括模糊测试、AI 后门检测。Email: jinbang66666@126.com

秦素娟 于 2009 年在哈尔滨工业大学计算机科学与技术专业获得硕士学位。现任陆军工程大学指挥控制工程学院助教。研究领域为网络安全、软件安全。研究兴趣包括软件定义网络、模糊测试。Email: 86538835@qq.com



张国敏 于 2009 年在解放军理工大学网络工程专业获得博士学位。现任陆军工程大学指挥控制工程学院副教授。研究领域为网络安全。研究兴趣包括软件定义网络、分布式系统。Email: 40519667@qq.com