计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200800178

基于深度优先搜索的模糊测试用例生成方法

李毅豪　洪　征　林培鸿
中国人民解放军陆军工程大学指挥控制工程学院　南京 210000
(enhancelee@foxmail.com)

摘　要　模糊测试是挖掘网络协议漏洞的重要方法之一。现有的模糊测试方法存在覆盖路径不完全、效率低下等问题。为了解决这些问题,文中提出了基于深度优先搜索的模糊测试用例生成方法,该方法将状态机转换成有向无回路图,以获得状态迁移路径,并通过提高测试用例在发送报文中的占比来提升模糊测试效率。该方法主要包括合并状态迁移、消除循环路径、搜索状态迁移路径、标记重复状态迁移和基于测试用例引导的模糊测试5个阶段。在合并状态迁移阶段,将首尾状态相同的状态迁移进行合并。在消除循环路径阶段,根据深度优先搜索判断图中的循环,并通过删除边将状态机转换成有向无回路图。在搜索状态迁移路径阶段,搜索有向无回路图从初始状态到终止状态的全路径,并对原状态机图使用Floyd算法补充被去除的边构造测试路径,以确保充分测试状态机中的每一个状态迁移。在标记重复状态迁移阶段,对重复状态迁移进行标记,避免对重复的状态迁移进行反复测试,以缩减测试的冗余。在基于测试用例引导的模糊测试阶段,生成针对状态迁移的测试用例,并将测试用例均匀分发到重复的状态迁移上,其中的部分测试用例能够起到引导状态迁移的作用,对被测目标进行模糊测试。实验结果表明,所提方法能够取得更高的有效测试用例比例。

关键词:模糊测试;漏洞挖掘;有状态协议;协议状态机;深度优先搜索
中图法分类号　TP398.08

Fuzzing Test Case Generation Method Based on Depth-first Search

LI Yi-hao, HONG Zheng and LIN Pei-hong
Command and Control Engineering College, Army Engineering University of PLA, Nanjing 210000, China

Abstract　Fuzzing test is an important method to exploit network protocol vulnerability. Existing fuzzing test methods have some problems such as incomplete path coverage and low efficiency. To solve these problems, this paper proposes a depth-first search based fuzzing test case sgeneration method. The method transforms the state machine into a directed acyclic graph to obtain the state transition paths, and increases the proportion of test cases in the testing messages to improve the fuzzing efficiency. The method includes five stages: merging state transition, eliminating loops, searching state transition paths, marking the same state transitions, and test case guidance based fuzzing test. Firstly, the state transitions which have the same start states and end states are merged. Secondly, according to the depth-first search, the loops in the graph are found, and the state machine is converted into a directed acyclic graph by deleting the edges of the loops. Thirdly, the directed acyclic graph is analyzed for the full path from the initial state to the end state, and the original state machine graph is supplemented with the removed edges using Floyd algorithm to construct the complete test paths, so as to ensure that each state transition in the state machine can be fully tested. Fourthly, repeated state transitions are marked to avoid repeated test of similar state transitions and reduce testing redundancy. Finally, test cases for state transitions are generated, and test cases which may guide the state transition are distributed evenly over the repetitive state transitions to carry out fuzzing test on the target. Experimental results show that the proposed method can achieve higher proportion of valid test cases.

Keywords　Fuzzing test, Vulnerability discovery, Stateful protocol, Protocol state machine, Depth first search

1　引言

协议漏洞挖掘是维护网络空间安全的重要手段。根据先验知识的多少,现有的漏洞挖掘方法可以分为白盒测试、灰盒测试和黑盒测试[1]。模糊测试是目前一种应用广泛的黑盒测试方法,它通过向协议实体输入非预期的报文,监控协议实体的运行状况,来分析协议实体发生的异常以发现潜在的安全漏洞[2]。

到稿日期:2020-08-27　返修日期:2020-09-25　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划基金资助项目(2017YFB0802900)
This work was supported by the National Key R&D Program of China (2017YFB0802900).
通信作者:洪征(hz5215@163.com)

针对网络协议的模糊测试最早可追溯到 Oulu 大学开发的 PROTOS[3] 模糊测试集。该团队通过分析协议规约，生成违反规约或者可能导致协议实体发生异常和错误的报文，并以此为基础形成针对某一协议的专用测试集，对协议实体进行测试。然而，PROTOS 的灵活性较差，应用范围狭窄。随着网络协议模糊测试技术的不断发展，越来越多的研究人员开始开发通用型的模糊测试工具。现有的主流网络协议模糊测试工具有 Peach[4]，Sulley[5] 和 SPIKE[6] 等。Peach 是用 Python 编写的一个模糊测试框架。测试者通过编写 XML 文件，对协议格式和测试用例发送顺序进行描述，从而对目标网络协议进行测试。编写 XML 的过程比较容易，但是 Peach 的程序部分难以修改，不能进行更精细的模糊测试操作，无法深度开发模糊测试器。Sulley 也是用 Python 编写的，测试者能够引用框架中的函数来描述协议格式，使用会话图说明测试用例的发送顺序，从而对目标网络协议进行测试。使用 Sulley 内置的工具可以开发插件、监视器等，适用于模糊测试深度开发。SPIKE 是使用 C 语言编写的模糊测试框架，与 Sulley 类似，测试者通过引用该框架的函数来描述协议格式，并对目标网络协议进行模糊测试。与 Sulley 相同，它也能够深度开发模糊测试工具。

根据协议输入报文之间是否存在关联性，可将网络协议分为有状态协议(stateful protocol)和无状态协议(stateless protocol)[7]。有状态协议接收输入报文时，会先验证输入的报文类型是否符合当前状态的预期，如果不符合预期则会进入错误处理程序，无状态协议则没有该约束。例如，FTP 作为有状态协议，在初始状态输入 USER 类型报文后，服务器端将等待用户继续输入 PASS 类型报文，此时如果输入其他类型的报文，则服务器将提示用户输入错误。相比无状态协议，有状态协议的模糊测试更加困难。

1.1 相关工作

Li 等[8] 提出了基于协议状态图遍历的 RTSP 协议漏洞挖掘方法，该方法采用深度优先搜索的方式对状态机进行模糊测试，并设置了最大遍历深度以防止遍历复杂协议时效率低下。然而，该方法没有考虑状态图存在循环路径的情况，在进行状态图的深度遍历时有可能陷入循环，无法遍历循环以外的路径，从而导致模糊测试的代码覆盖率低。

Ma 等[9] 提出使用基于规则的状态机(rule-based state machine)和状态规则树(stateful rule tree)对有状态协议进行模糊测试。他们认为如果某个状态迁移对应的输入不会引发协议实体错误，则可以认定该路径为安全路径。在测试时将安全路径删除，可以达到减少测试用例数量、提高模糊测试效率的目的。然而，这种方法可能会导致模糊测试不完全。

Sulley 可以根据协议状态迁移生成会话图。然而，由于循环路径的存在，Sulley 在构建会话图后无法确定模糊测试需要在哪个状态下停止，因此其难以直接根据有循环的协议状态机生成模糊测试会话图对系统进行模糊测试[10]。Zhang 等[11] 在 Sulley 的基础上提出了循环路径展开的方法，将状态迁移过程中产生的循环路径解开，叠加生成非循环的路径。但是，该方法在进行模糊测试时会生成大量冗余报文，并重复测试相同的状态迁移，从而导致模糊测试效率低下。

Huang 等[12] 提出了基于 Fuzzing 测试的工控网络协议漏洞挖掘技术。该方法提出多字段组合关联的模糊策略和基于动态抽样的样本生成方法用于模糊测试。然而，该方法关注了报文内各字段之间的约束，却没有考虑到报文之间的约束。因此，将该方法用于有状态协议模糊测试时，会导致模糊测试效果不佳。

Wang 等[13] 提出了基于字段过滤和分组修复的模糊测试方案，该方法对报文中特定的字段进行变异，观察响应报文。如果响应报文显示语法错误，则说明被测系统在该字段上有完善的语法处理机制；如果响应报文显示正确接收了变异的报文，则说明目标程序没有对该字段进行必要的处理。根据这一特点，如果变异了某字段，收到标识错误的响应报文，则不再变异该字段以减少无用的测试用例；如果收到标识正确的响应报文，则继续变异该字段。然而，这种判断方式过于粗糙，可能会导致模糊测试不完全。

现有的模糊测试方法在状态机迁移遍历时没有考虑到循环路径的情况，在循环路径的展开过程中可能存在冗余，导致模糊测试效率低下。而一些模糊测试方法过度追求测试用例的缩减，导致模糊测试不完全。除此之外，Sulley 等工具能够利用会话图，生成符合协议状态迁移路径的测试用例，但是如果完全按照有状态协议的状态机建立会话图，当模糊测试进入循环的状态迁移路径时，无法确定需要在循环路径中循环多少次后停止测试。

1.2 主要贡献

深度优先搜索和广度优先搜索是针对图的两种主要遍历算法。深度优先搜索会对每一条可能的分支路径深入到不能够再深入为止，而广度优先搜索会优先遍历当前结点的所有子节点。深度优先搜索相比广度优先搜索更适合寻找图中从一个初始点到一个终止点的路径。在有状态协议模糊测试领域，测试者定制目标协议的测试用例输入顺序时，需要使用协议实体的状态迁移路径，而深度优先搜索符合寻找协议状态机的迁移路径这一要求。基于该思想，本文提出了基于深度优先搜索的模糊测试用例生成方法。该方法由合并状态迁移、消除循环路径、搜索状态迁移路径、标记重复状态迁移和模糊测试 5 个阶段组成。该方法解决了循环路径导致无法进行充分模糊测试的问题，能够针对目标协议实施相对完整的模糊测试。

本文第 2 节对有状态协议的模糊测试问题进行了具体描述；第 3 节介绍了本文方法的具体框架设计；第 4 节使用有效测试用例比例 PoVTC(Proportion of Valid Test Cases)和模糊测试时间两个指标对本文方法与其他方法进行了对比分析；最后总结全文。

2 问题描述

有限状态机[14](Finite State Machine，FSM)是一种用来进行对象行为建模的工具。在网络协议领域的研究中，有限状态机常被用于网络协议报文交互过程的建模，以描述网络协议实体的状态转换。例如，著名的传输控制协议(Transmission Control Protocol，TCP)状态机描述了 TCP 协议交互时协议实体需要处理的各种事件以及状态迁移[15]，如图 1 所

示。在 TCP 协议状态机中，协议实体在 CLOSED 状态下，接收 LISTEN 类报文，协议实体将进入 LISTEN 状态，等待接收 SYN 或者 SEND 类报文。协议实体在 LISTEN 状态下收到相应类型的报文后，将发送响应报文，并转入 SYN RECEIVED 状态或者 SYN SENT 状态。

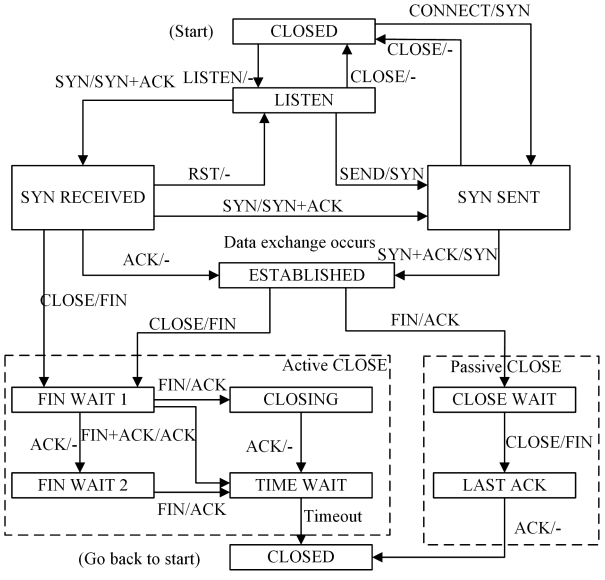

图 1　TCP 协议的有限状态机

Fig. 1　Finite state machine of TCP protocol

定义 1　有限状态机(FSM)　可被定义为一个五元组 $M=(S,\Sigma,\delta,s_0,F)$。其中，$S$ 是非空有限状态集合；Σ 是输入符号集；δ：$Q\times\Sigma\to Q$ 是状态迁移函数；s_0 是初始状态，且 $s_0\in S$；F 是终止状态集合，且 $F\subseteq S$。

定义 2 状态迁移(state transition)　在接收某一输入符号后，状态从一个状态转换成另一个状态，这个过程被称为状态迁移。本文将其形式化描述成一个序列$\langle s_i,m,s_j\rangle$，其中 s_i 被称为当前状态，s_j 被称为下一状态。该序列表示实体在状态 s_i 下接收输入符号 m 后，状态转换到 s_j。

以图 1 中的 TCP 协议为例，建立有限状态机模型。非空有限状态集合 $S=(CLOSED,LISTEN,SYN\ RECIEVED,\cdots)$，输入符号集 $\Sigma=(SYN,CLOSE,FIN,ACK,\cdots)$，$\delta$ 表示状态迁移函数，由多个状态迁移构成，$\delta=(\langle CLOSED,LISTEN,LISTEN\rangle,\langle CLOSING,ACK,TIME\ WAIT\rangle,\cdots)$。初始状态 s_0 为状态 $CLOSED$，终止状态集合 $F=(CLOSED)$。

协议实体在进行报文处理时需要遵循网络协议规范。当协议实体处于某一状态 s_i 时，只有得到正确输入，协议实体才会对该输入进行处理，否则协议实体会进入错误处理程序。为了提高模糊测试的代码覆盖率，对有状态协议进行模糊测试时，需要注意输入报文与协议状态相匹配，避免输入报文被协议实体程序丢弃，导致无法进入深层次的处理流程。

在针对协议实体某一状态进行测试时，传统模糊测试方法需要先将协议实体的状态引导至被测状态下才能进行测试，这样会产生大量引导报文。引导报文在模糊测试过程中会耗费大量的时间，从而导致模糊测试效率低下。如果使用测试用例充当引导报文，在测试过程中将协议实体引导到相应的协议状态，就能减少模糊测试过程中产生的引导报文数量，从而提高模糊测试效率。

很多测试用例并不会触发协议实体异常，这些测试用例主要可以分为两类：一类是由于测试用例存在畸形数据，无法通过语法验证，这些测试用例会被协议实体直接抛弃，无法进入程序执行过程，它们一般不会引发协议实体程序的异常和状态跳转；另外一类是能够正常被程序执行，可以引发状态跳转并输出正常的响应报文[2]。根据这一特点，即可通过观察响应报文的输出来判断测试用例是否使协议实体进行了状态迁移。当协议实体处于某状态时，向其输入测试用例，如果该测试用例能够使协议实体返回标识正确的响应报文，则说明协议实体接收和正确处理了测试用例，并进行了状态迁移，那么该测试用例就同时发挥了测试和引导的功能；如果该测试用例使协议实体返回标识错误的响应报文，那么说明协议实体状态未进行迁移，则继续进行当前状态的测试用例输入测试。

针对上述分析，本文提出了基于深度优先搜索的模糊测试用例生成方法，该方法由合并状态迁移、消除循环路径、搜索状态迁移路径、标记重复状态迁移和基于测试用例引导的模糊测试 5 个阶段组成。该方法能够严格遵循协议实体状态机路径，并缩减状态迁移过程中的引导报文。

3　方案设计

3.1　概述

为了解决现有的有状态协议模糊测试方法代码覆盖率低、测试不完全和效率低下等问题，本文提出的有状态协议模糊测试方法的具体流程如图 2 所示。

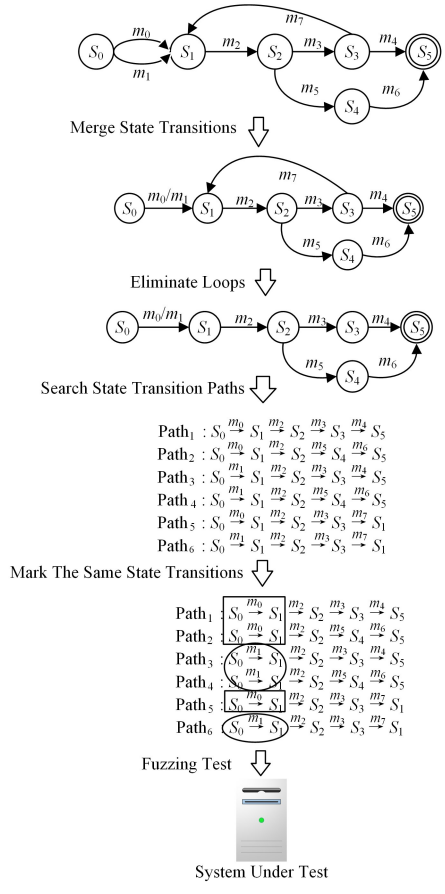

图 2　基于深度优先搜索的模糊测试流程

Fig. 2　Procedure of fuzzing test based on depth first search

首先根据被测系统协议的 RFC 文档，对被测系统的网络协议进行描述，构建协议状态机。协议状态机是对协议状态的形式化描述，现在很多针对协议状态机设计的 RFC 文档只有相关的语言描述，因此这一步骤需要专业人员在 RFC 文档的基础上建立状态机模型。除此之外，针对未公开的协议，也可以使用协议逆向工程的手段来获得协议的状态机信息，如使用 Netzob[16] 等逆向分析工具获取状态机模型。

在有向图中，关联一对顶点的有向边如果多于 1 条，并且这些边的始点与终点相同，则称这些边为平行边，关联同一个顶点的有向边称为自环。不包含平行边和自环的有向图称为简单图，简单图之外的有向图被称为多重图。深度优先搜索有利于寻找模糊测试需要的状态迁移路径，而深度优先搜索适用于简单图，因此需要将状态机转换成简单图。为了将状态机转换成简单图，需要将首尾状态相同的状态迁移合并。在合并状态迁移阶段，本文方法将合并当前状态和下一状态相同但输入符号不同的状态迁移。如在状态迁移函数 δ 中，有两个状态迁移$\langle s_0, m_0, s_1\rangle$和$\langle s_0, m_1, s_1\rangle$，需要将这两个状态迁移合并。对协议状态机进行路径搜索后，再将合并的状态迁移还原为两个独立的路径，以确保测试两个不同的状态迁移。

当有向图中存在循环时，模糊测试可能会在循环的状态迁移内绕圈，而忽略其他状态迁移。针对这一问题，现有的主流有状态协议模糊测试工具（如 Sulley 等）难以根据有循环路径的状态机生成会话图来对目标协议进行模糊测试。为了解决这个问题，可以消除状态机图中的循环，将状态机图转换成有向无回路图。深度优先搜索算法能够判断有向图中是否存在循环。在消除循环路径阶段，利用深度优先搜索算法，将部分路径从状态机迁移图中去除，使剩下的路径形成无循环的有向图，以便通过遍历图形成模糊测试路径。后续再对所去除的路径进行恢复，以确保模糊测试时能够对原状态机中所有的状态迁移进行测试。

在搜索状态迁移路径阶段，首先利用深度优先搜索算法，找到初始状态s_0到所有终止状态$s_{end} \in S$之间的路径，形成路径集合 $paths$。Floyd 算法[17] 是一种基于动态规划寻找图中两点之间最短路径的算法。为了能够充分测试原状态机图的所有状态迁移，利用 Floyd 算法求出从初始状态到删除边$\langle v_1, v_2\rangle$起点v_1的最短路径$Path_1$，再将删除边$\langle v_1, v_2\rangle$与该最短路径相连接形成新的路径$\langle Path_1, \langle v_1, v_2\rangle\rangle$，这样就能够形成需要最少引导报文来测试被删除边的状态迁移的路径。形成的新路径被加入到路径集合 $paths$ 中。然后将首尾状态相同的状态迁移路径重新分离成多个状态迁移路径，并将这些路径加入到路径集合 $paths$ 中。

如果路径$Path_1$和$Path_2$中包含重复的状态迁移，在使用 Sulley 进行模糊测试时，会首先根据路径生成两个会话图。这两个会话图是相对独立的，而测试用例是针对输入符号 m 生成的符合 m 格式的报文，如果输入符号 m 相同，则按照基于输入符号格式的模糊测试方法所生成的测试用例也相同。在进行模糊测试时，Sulley 不会关联两个会话图中的重复状态迁移，因为这会使得相同的测试用例在重复的状态迁移上被测试两次，从而出现冗余测试的情况。为了解决这个问题，在标记重复状态迁移阶段，对路径集合 $PATH$ 中首尾状态相同且迁移的输入符号也相同的状态迁移进行标记，在模糊测试中根据标记避免在不同路径的重复的状态迁移中使用相同的测试用例，减少冗余测试。

在基于测试用例引导的模糊测试阶段，将对应各个输入符号的测试用例用于测试被测系统协议的状态迁移，调整测试用例输入顺序，使其符合路径集合 $PATH$ 中的状态迁移路径，这样的做法能够让测试用例既能对系统进行测试，又能使协议实体进行状态转换，并引导协议实体进入特定状态。如一个状态迁移路径中有两个状态迁移：$\langle S_i, m_a, S_j\rangle$和$\langle S_j, m_b, S_k\rangle$。针对状态迁移$\langle S_i, m_a, S_j\rangle$有一个测试用例$test_1$，针对状态迁移$\langle S_j, m_b, S_k\rangle$有一个测试用例$test_2$，对该状态机进行测试时按顺序输入测试用例序列$\langle test_1, test_2\rangle$。输入$test_1$时，如果收到标识正确的响应报文，则说明协议实体状态发生了正常的迁移，从S_i迁移到了S_j，此时即可输入$test_2$进行针对状态迁移$\langle S_j, m_b, S_k\rangle$的测试；如果收到标识错误的响应报文，则说明协议实体状态未发生迁移，此时需要再次输入针对$\langle S_i, m_a, S_j\rangle$的测试用例；若针对状态迁移$\langle S_i, m_a, S_j\rangle$事先构造好的测试用例集已空，则输入针对状态迁移$\langle S_i, m_a, S_j\rangle$的正常报文以引导协议实体状态迁移至下一状态。该方法能够增加测试用例在发送报文中所占的比例，减少前期引导性的报文占发送报文的比例，使输入符号的输入顺序符合被测协议的状态迁移路径，从而对协议实体进行模糊测试。

下文将对各阶段的主要工作进行细致分析。

3.2 合并状态迁移

由于存在当前状态和下一状态相同而输入符号不同的情况，如图 3 所示，为了将状态机转换成有向无回路图（Directed Acyclic Graph，DAG）以便进行深度优先搜索，需要合并这类状态迁移。

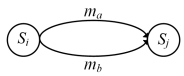

图 3 首尾状态相同的状态迁移

Fig. 3 State transitions of the same start states and end states

当出现首尾状态相同的状态迁移时，需要将这些状态迁移合并成一个，以将状态机图转换成有向无回路图。图 3 中出现了两个首尾状态相同的状态迁移$\langle S_i, m_a, S_j\rangle$和$\langle S_i, m_b, S_j\rangle$，将其合并后的状态迁移记作$\langle S_i, m_a/m_b, S_j\rangle$。

本阶段检查状态迁移函数 δ 中的所有状态迁移，合并首尾状态相同的状态迁移，得到协议状态机转换后的有向图 G。

3.3 消除循环路径

深度优先搜索能够在图中寻找回路。在使用深度优先搜索对有向图进行遍历时，如果发现某一个顶点的一个孩子是该顶点的祖先，则说明图中存在回路。基于这种想法，本阶段将使用基于深度优先搜索的路径去环算法，根据深度优先搜索全图，如果遍历路径形成循环，则删除该循环中的一条边以破坏循环，使协议状态机对应的有向图中不存在循环路径。这样做能够确保模糊测试过程不会陷入循环，从而提高模糊

测试效率。基于深度优先的路径去环算法如算法 1 所示。

算法 1 基于深度优先的路径去环算法

```
输入：G=(V,E)
输出：G'=(V',E'),remove_edges
1. def DFS_DAG(G,v)://以 v 为起点遍历图 G
2. pEdge=the first edge of v
3.   While (pEdge!=Null):
4.     v'=the other node of pEdge
5.       if v' is not visited://遍历到没有访问过的结点
6.         mark v' is being visited
7.         DFS_DAG(G,v')
8.       else if v' is being visited://遍历到了已经遍历过的祖先结点，说明存在环
9.       remove the edge (v,v') from G
10.      add the edge to remove_edges
11.    pEdge=the next edge
12.  mark v has been visited
13. G'=G
14. return G',remove_edges
```

算法 1 中，输入为图 G，输出为去环之后的图 G' 以及被去除的边的集合 *remove_edges*。图中的结点有 3 种状态：未访问、正在访问和已访问。第 1－4 行为遍历算法初始变量赋值；第 5－7 行说明深度优先搜索遍历到状态为未访问的结点，将结点标记为正在访问，再次使用 DFS_DAG 函数进行递归调用；第 8－10 行说明深度优先搜索遍历到状态为正在访问的结点，说明图中存在回路，此时删除最近遍历的一条边以破坏回路，并将删除的边保存到 *remove_edges* 中；第 11－12 行，将结点标记为已访问；第 13－14 行返回没有回路的图 G' 和被去除的边的集合 *remove_edges*，作为算法的输出。

本阶段通过调用函数 DFS_DAG(G,v)，完成对有状态协议状态机图的去环操作，使图变为一个有向无回路图。其中，G 是有状态协议的状态机图，v 是初始状态结点。

3.4 搜索状态迁移路径

模糊测试会话图是将状态迁移路径组合形成的状态迁移图，描述了状态迁移的起点与状态迁移的方向，在进行模糊测试时，利用会话图就能控制模糊测试流程。为了确定模糊测试路径，需获得有向无回路图中，以初始状态为起点、各个不同终止状态为终点的路径，建立符合状态迁移路径的模糊测试会话图。

本阶段利用上一阶段得到的有向无回路图 G'，通过基于深度优先搜索的两点之间的全路径搜索算法，得到状态迁移路径。除此之外，为了确保原状态机图中的每个状态迁移都能被测试到，需要还原之前删除的状态迁移所对应的状态迁移路径。为了补充上述状态迁移路径，在原始的状态机图 G 中使用寻找最短路径的 Floyd 算法，利用边集 *remove_edges* 中的边构成新的状态迁移路径，以确保边集 *remove_edges* 中的状态迁移在模糊测试时能够被覆盖。在合并状态迁移阶段合并的状态迁移，是由首尾状态相同但是输入符号不同的多个不同的状态迁移构成的。尽管首尾状态相同，但是由于其输入符号不同，这些状态迁移并不是相同的状态迁移。为了确保能够对状态机图中的所有状态迁移进行测试，需要将合并的状态迁移进行分离，还原出首尾状态相同但输入符号不同的状态迁移，以形成不同的测试路径。在本阶段，将把合并状态迁移阶段中所合并的状态迁移重新分离开，并形成最终的状态迁移路径集合 $STPS$。基于深度优先搜索的两点之间的全路径搜索算法如算法 2 所示。

算法 2 基于深度优先搜索的两点之间的全路径搜索算法

```
输入：G'=(V,E)
输出：paths
1. stack=[] //初始化一个栈，保存当前搜索的路径
2. paths=[] //初始化变量，保存搜索到的路径
3. def DFS_PATH(start,index,end,G'):
4.   stack.push(index)
5.   if index==end:
6.     paths.append(stack.copy()) //找到终点
7.     stack.pop() //往前回溯一位，查看是否有其他路径
8.   else://依次搜索不在栈中的其他结点
9.     for v in V:
10.      if v not in stack:
11.        DFS_PATH(start,v,end,G')
12.      stack.pop()
13. DFS_PATH(start_node,start_node,end_node,G')
14. return paths
```

算法 2 中，第 1－2 行分别初始化了一个栈变量和一个搜索到的路径变量；第 3 行定义了使用深度优先搜索的寻找路径函数；第 4 行将寻找到的第一个结点压入栈中；第 5－7 行检查是否找到终点，如果是终点，则将栈中的结点保存至 *paths* 变量中；第 8－12 行如果检查到不是终点，则继续寻找其他路径；第 13－14 行调用 DFS_PATH 函数，返回搜寻到的路径集合 *paths*。

首先，使用基于深度优先搜索的两点之间的全路径搜索算法，输入有向无回路图 G'，即可得到在有向无回路图 G' 中，从初始状态 s_0 到终止状态集合 F 中各终止状态的路径。

然后，为了将边集 *remove_edges* 中的边包含到模糊测试的范围内，在原有状态协议的状态机图 G 中使用 Floyd 算法，求出覆盖 *remove_edges* 中各边的最短路径。设 *remove_edges* 如式(1)所示：

$$remove\_edges=(\langle v_{11},v_{12}\rangle,\langle v_{21},v_{22}\rangle,\cdots,\langle v_{n1},v_{n2}\rangle) \quad (1)$$

其中，$v_{ik}(i\in(1,n),k\in\{1,2\})$ 是图的结点，$\langle v_{i1},v_{i2}\rangle$ 是从 v_{i1} 到 v_{i2} 的有向边。

对于 *remove_edges* 中的每条边，使用 Floyd 算法在原状态机图 G 中寻找从初始状态结点 s_0 到结点 v_{i1} 的最短路径，该路径记作 $floyd\_path_i$。将路径 $\langle floyd\_path_i,\langle v_{i1},v_{i2}\rangle\rangle$ 也加入到路径集合 *paths* 中。

最后，搜索路径集合 *paths* 中合并的状态迁移，并将合并的状态迁移分离。如果某路径中只有一个合并的状态迁移，如路径 $\langle\langle S_0,m_0,S_i\rangle,\langle S_i,m_a/m_b,S_j\rangle\rangle$，则该路径能够被分离成路径 $\langle\langle S_0,m_0,S_i\rangle,\langle S_i,m_a,S_j\rangle\rangle$ 和路径 $\langle\langle S_0,m_0,S_i\rangle,\langle S_i,m_b,S_j\rangle\rangle$。如果某路径中有多个合并的状态迁移，则需要对

合并状态迁移中的输入符号m_i作笛卡尔积，分离得到多条路径。如有一条两个合并的状态迁移路径$\langle\langle S_0, m_0, S_i\rangle, \langle S_i, m_a/m_b, S_j\rangle, \langle S_j, m_c/m_d, S_k\rangle\rangle$，则该路径可被分离成$\langle\langle S_0, m_0, S_i\rangle, \langle S_i, m_a, S_j\rangle, \langle S_j, m_c, S_k\rangle\rangle, \langle\langle S_0, m_0, S_i\rangle, \langle S_i, m_b, S_j\rangle, \langle S_j, m_c, S_k\rangle\rangle, \langle\langle S_0, m_0, S_i\rangle, \langle S_i, m_a, S_j\rangle, \langle S_j, m_d, S_k\rangle\rangle, \langle\langle S_0, m_0, S_i\rangle, \langle S_i, m_b, S_j\rangle, \langle S_j, m_d, S_k\rangle\rangle$。分离状态迁移能够将合并的状态迁移还原成原状态机图中的首尾状态相同、输入符号却不同的状态迁移，将这些状态迁移加到测试中，从而保证被测系统的每个状态迁移都得到了充分测试。

3.5 标记重复状态迁移

该阶段需要对重复状态迁移的状态迁移进行标记，以减少模糊测试阶段针对相同的状态迁移进行冗余测试。

定义 3 状态迁移$\langle S_{11}, m_1, S_{12}\rangle, \langle S_{21}, m_2, S_{22}\rangle, \cdots, \langle S_{n1}, m_n, S_{n2}\rangle$为重复状态迁移，当且仅当$S_{11}=S_{21}=\cdots=S_{n1} \wedge m_1=m_2=\cdots=m_n \wedge S_{12}=S_{22}=\cdots=S_{n2}$。

重复状态迁移指初始状态和迁移之后的状态相同、且输入符号相同的状态迁移。一般而言，测试用例都是针对某个状态迁移进行测试。传统的模糊测试方法需要根据状态迁移路径创建会话图，对每个会话图进行独立测试。在展开循环后的情况下，根据多个状态迁移路径创建的多个会话图中可能包含重复状态迁移。由于传统的模糊测试方法中各会话图之间是相互独立的，因此会话之间的重复状态迁移没有联系。现有基于生成的模糊测试方法在生成测试用例时是根据输入符号的格式进行填充的，如果输入符号 m 相同，则生成的测试用例也相同，因此会对多个会话图中的重复状态迁移进行重复测试，从而出现冗余。对重复状态迁移进行标记，可以使同一个测试用例不会出现在不同会话图下的重复状态迁移中，减少了模糊测试过程中的冗余工作。

如图 4 所示，为了说明标记重复状态迁移能够减少冗余，提出以下假设：现有两条状态迁移路径$Path_1$和$Path_2$，其中各有两个状态迁移，并且都包括$\langle S_0, m_0, S_1\rangle$这个状态迁移。针对状态迁移$\langle S_0, m_0, S_1\rangle$有两个测试用例$test_0$，$test_1$，针对状态迁移$\langle S_1, m_1, S_2\rangle$有一个测试用例$test_2$，针对状态迁移$\langle S_1, m_2, S_3\rangle$有一个测试用例$test_3$。

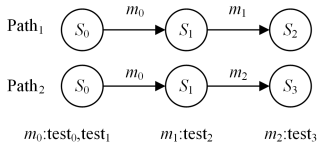

图 4 标记重复状态迁移减少冗余测试

Fig. 4 Mark repetitive state transitions to reduce test redundancy

传统的模糊测试方法根据两条状态迁移路径生成独立的会话图时，针对$Path_1$将形成测试用例序列$\langle test_0, test_2\rangle$，$\langle test_1, test_2\rangle$，其中$test_0$和$test_1$用于状态迁移$\langle S_0, m_0, S_1\rangle$的测试，$test_2$用于状态迁移$\langle S_1, m_1, S_2\rangle$的测试。针对$Path_2$将形成测试用例序列$\langle test_0, test_3\rangle$，$\langle test_1, test_3\rangle$，其中$test_0$和$test_1$用于状态迁移$\langle S_0, m_0, S_1\rangle$的测试，$test_3$用于状态迁移$\langle S_1, m_2, S_3\rangle$的测试。从上述分析可以看出，由于$Path_1$和$Path_2$形成的会话图相对独立，因此在对这两个路径进行模糊测试时，重复对两个路径中的状态$\langle S_0, m_0, S_1\rangle$进行了测试。

如果对重复状态迁移进行标记，并将针对重复状态迁移的测试用例分散到这些重复的状态迁移上，针对$Path_1$形成测试用例序列$\langle test_0, test_2\rangle$，针对$Path_2$形成测试用例序列$\langle test_1, test_3\rangle$。相比传统的模糊测试方法形成的测试用例序列$\langle test_0, test_2\rangle$，$\langle test_1, test_2\rangle$，$\langle test_0, test_3\rangle$，$\langle test_1, test_3\rangle$，将用于同一个状态迁移$\langle S_0, m_0, S_1\rangle$的测试用例$test_0$和$test_1$分别用在两条路径中，这样对重复状态迁移进行标记的测试用例序列只有$\langle test_0, test_2\rangle$和$\langle test_1, test_3\rangle$，其能够将相对独立的会话图关联起来以减少冗余测试。

按照定义 3，对路径集合 $paths$ 中的所有路径进行搜索，如果存在互为重复状态迁移的状态迁移，则将其标记出来，在下一模糊测试阶段中，将针对某一状态迁移生成的测试用例分散到各个路径上的与其互为重复状态迁移的状态迁移中，以减少冗余测试，从而达到提高测试效率的目的。

3.6 基于测试用例引导的模糊测试

定义 4(引导报文) 在针对有状态协议的模糊测试过程中，测试一个状态迁移时，需要输入一系列报文，先将协议实体的状态从初始状态迁移到待测状态下，这一系列报文被称为引导报文。

如果模糊测试过程中需要产生过多的引导报文，则将直接影响模糊测试的测试时间，并降低测试效率。因此，使用既能够对被测系统进行测试，又能够引导协议实体发生状态迁移的测试用例，可以减少引导报文的数量，降低模糊测试耗费的时间，提升模糊测试效率。

协议实体针对输入的报文存在 3 种响应方式。第一，输入的报文完全符合当前状态预期的输入报文，因此将返回表示正确的响应报文，且协议实体的状态发生迁移。第二，输入的报文不符合当前状态预期的输入报文，将返回表示错误的响应报文，且协议实体状态不发生迁移。第三，输入报文后，协议实体可能出现错误或异常，也有可能在进行程序设计实现时没有考虑该种输入情况，不做任何响应，且协议实体不发生迁移[2,13]。为了便于区别，本文将第一种响应方式称为发生迁移的输入，将第二种和第三种响应方式称为不发生迁移的输入。一般而言，引导报文都是能够使协议实体发生迁移的输入，而在测试用例中，一部分测试用例是发生迁移的输入，一部分测试用例是不发生迁移的输入。测试用例不能确保被测协议实体会发生状态迁移，因此在输入一个测试用例后，如果收到表示错误的响应报文或超时没有收到报文，则可以认为协议实体的状态没有变化，然后输入针对当前状态的下一个测试用例。如果收到表示正确的响应报文，则可以认为协议实体跳转到了下一状态，然后输入针对相应状态的测试用例。

模糊测试的具体过程如下。首先，根据各输入符号m_i的格式，生成相应的测试用例。然后，根据路径集合 $paths$ 建立测试会话图，使模糊测试能够按照路径集合 $paths$ 中的路径执行。其次，检查路径集合 $paths$，如果发现某些状态迁移有多个，则说明这些状态迁移为重复状态迁移。生成与m_i相关

的测试用例并分散应用在这些不同路径的重复状态迁移中，这些测试用例在测试中不仅能够对被测系统进行测试，还可以作为引导协议实体状态迁移的报文，以减少引导报文的占比，提高模糊测试效率。最后，运行监视代理、日志记录代理和模糊测试程序，对目标有状态协议实体进行模糊测试。当监视代理收到响应的正确报文时，说明当前的测试用例输入是发生迁移的输入，则模糊测试程序针对路径中的下一个状态迁移发送测试用例；当监视代理收到响应的错误报文或没有接收到任何响应时，说明当前的测试用例输入是不发生迁移的输入，则模糊测试程序继续针对当前状态迁移发送测试用例，该过程如图 5 所示。

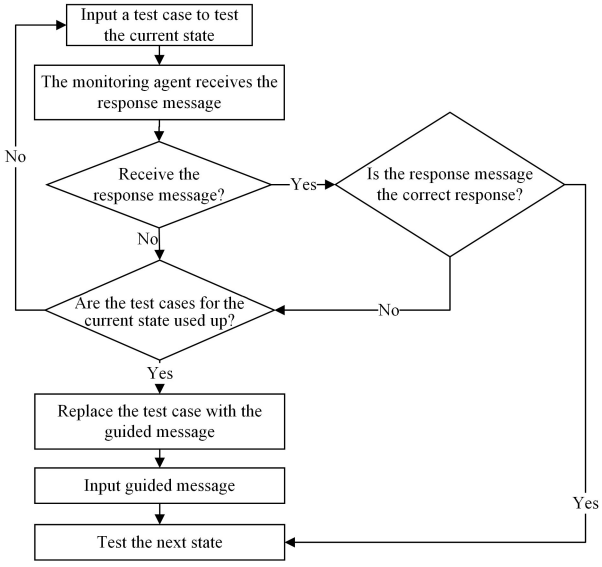

图 5 基于测试用例引导的模糊测试阶段

Fig. 5 Phases of fuzzing test based on test cases guidance

4 评估

在评估阶段，本文将使用 FTP 和 SMTP 作为评估本文方法有效性的对象，将本文方法（FTMBDFS）与 Sulley、Li 等[8]的方法（TPSG）、Zhang 等[11]的方法（VMNPBF）就有效测试用例比例（Proportion of Valid Test Cases，PoVTC）指标和模糊测试时间指标进行对比。下面将介绍两个评估指标。

有效测试用例比例是衡量漏洞挖掘有效性的重要指标，它指模糊测试方法生成的能够使被测系统产生异常的测试用例数量占所有发送报文数量的比例。由于该占比数值一般较小，为了便于观察，数值采取乘 100 的方式，如式(2)所示：

$$PoVTC=\frac{a}{\frac{n}{100}}=\frac{a}{n}\times 100 \tag{2}$$

其中，a 表示测试用例中能够使被测系统发生异常或错误的测试用例数量，n 表示发送报文总数。该指标越高，说明能够使被测系统发生异常或错误的测试用例在总发送报文数中的占比越高，模糊测试效率就越高。

模糊测试时间是衡量模糊测试效率的一个重要指标。它主要是由模糊测试时发送引导报文、测试用例的时间和协议实体处理时间构成，能够有效反映模糊测试方法生成的测试用例数量的多少。该指标越高，说明该模糊测试方法生成的测试用例越多，该指标越低，说明该模糊测试方法生成的测试用例越少。

4.1 对 FTP 进行实验评估

文件传输协议（File Transfer Protocol，FTP）是一种应用范围广泛的应用层协议。客户端能够通过控制端口对服务器发送控制指令，通过数据端口上传或下载响应的文件。其输入符号列表和状态机图如表 1 和图 6 所示[2]。

表 1 FTP 输入符号列表

Table 1 Incoming symbol list of FTP

message type	incoming symbol	message type	incoming symbol
USER	m_1	RETR	m_7
PASS	m_2	RMD	m_8
PWD	m_3	CWD	m_9
TYPE	m_4	DELE	m_{10}
STOR	m_5	CDUP	m_{11}
MKD	m_6	QUIT	m_{12}

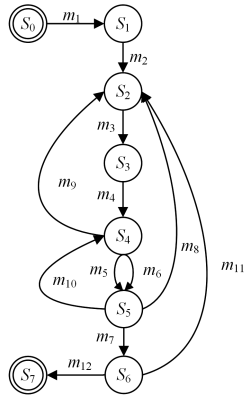

图 6 FTP 有限状态机

Fig. 6 Finite state machine of FTP

在本实验中，将使用 Quick'n Easy FTP Server V4.0.0 作为 FTP 协议被测协议实体，其满足 RFC 文档中对 FTP 协议状态机的描述。实验所用的各方法的发送报文数、测试时间和测试用例数如表 2 所列，各方法的 PoVTC 指标数值如图 7 所示。

表 2 FTP 发送报文数和测试时间

Table 2 FTP messages number and testing time

Method	The number of messages	Testing Time	The number of test cases
Sulley	66177	13 h 17 min	13476
VMNPBF	71270	15 h 22 min	17968
TPSG	42957	8 h 48 min	10107
FTMBDFS	30499	6 h 7 min	13476

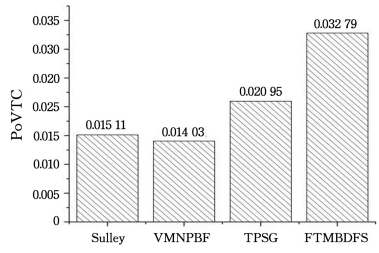

图 7 各方法的 PoVTC 指标数值

Fig. 7 PoVTC of each method

从表 2 可以得出，本文提出的方法 FTMBDFS 将生成更少的测试用例，相比对比方法中用时最少的 TPSG 方法减少了 30%以上，相比的模糊测试工具 Sulley 减少了 50%以上。从图 7 可以看出，本文方法的有效测试用例比例指标相比对比方法中最高的 TPSG 提高了 50%以上。

Sulley 在 13476 个测试用例中发现了 10 个能够使协议实体产生异常的测试用例，VMNPBF 在 17968 个测试用例中发现了 10 个能够使协议实体产生异常的测试用例，TPSG 在 10107 个测试用例中发现了 9 个能够使协议实体产生异常的测试用例。本文方法在 13476 个测试用例中发现了 10 个能够使协议实体产生异常的测试用例。除此之外，本文方法的测试用例占发送报文数的比例达到 44.19%，高于其他方法。上述数据能够有效说明，针对 FTP 协议，无论是在模糊测试时间方面还是有效测试用例比例方面，本文方法相比现有的其他方法都更有优势。

4.2 对 SMTP 进行实验评估

简单邮件传输协议(Simple Mail Transfer Protocol, SMTP)是用于传输邮件的应用层协议，目前被广泛应用于各种邮件应用程序中。客户端和服务器端通过 25 号端口进行通信。利用命令、客户端将邮件的原地址、目的地址和邮件的具体内容传递给服务器端，服务器端进行相应的响应并接收邮件。其输入符号列表和状态机图如表 3 和图 8 所示[18]。

表 3 SMTP 输入符号列表

Table 3 Incoming symbol list of SMTP

message type	incoming symbol	message type	incoming symbol
helo	m_1	rcpt to	m_8
ehlo	m_2	rset	m_9
mail from	m_3	data	m_{10}
saml	m_4	〈CRLF〉	m_{11}
soml	m_5	rset	m_{12}
rcpt to	m_6	quit	m_{13}
rcpt to	m_7		

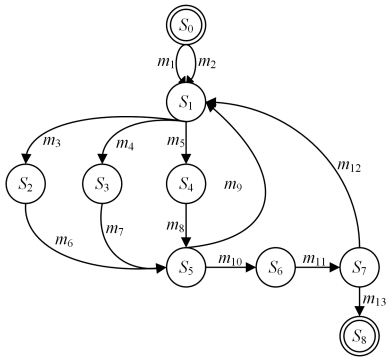

图 8 SMTP 有限状态机

Fig. 8 Finite state machine of SMTP

在该测试实验中，将使用 Quick'n Easy Mail Server V3.3.0 作为 SMTP 协议的被测协议实体，其满足 RFC 文档中对 SMTP 的状态机描述。实验所用各方法的发送报文数、测试时间和测试用例数如表 4 所列，各方法的 PoVTC 指标数值如图 9 所示。

表 4 SMTP 发送报文数和测试时间

Table 4 SMTP messages number and test time

Method	The number of messages	Testing Time	The number of test cases
Sulley	40428	7h48min	12831
VMNPBF	47166	9h0min	14850
TPSG	26952	5h27min	10870
FTMBDFS	21 230	4h17min	12831

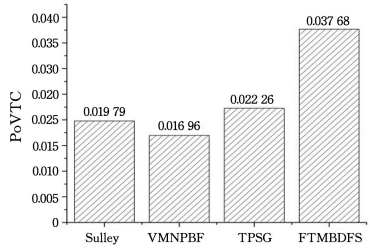

图 9 各方法的 PoVTC 指标数值

Fig. 9 PoVTC of each method

从表 4 可以看出，本文方法在发送报文数上远小于其他方法，且测试时间远低于传统的模糊测试工具 Sulley。从图 9 可以看出，本文方法在有效测试用例比例指标上相比其他方法也有显著提高，在针对 SMTP 协议实体的漏洞挖掘中，数值高达 0.037 68。Sulley, VMNPBF, TPSG 方法生成的能够使协议实体产生异常的测试用例数分别为 8, 8, 6，其生成的测试用例总数分别为 12831, 14850, 10870。本文方法生成的能够使协议实体产生异常的测试用例数为 8，生成的总测试用例数为 12831。除此之外，本文方法的测试用例占发送报文数的比例达到 60.44%，高于其他方法。从上述情况来看，本文方法能够在更短的测试时间内测出数量与其他方法相同的引发协议实体异常的测试用例。

4.3 测试小结

本节分别使用 Sulley, VMNPBF, TPSG 和本文提出的 FTMBDFS 方法对 FTP 和 SMTP 协议实体进行模糊测试。测试结果显示，本文方法在漏洞的挖掘方面更具优势，不仅能够充分地对每个状态转换进行遍历测试，还可以针对各条遍历路径生成更少的测试用例。因此，本文方法在生成更少报文的情况下能够挖掘更多的漏洞。

结束语 本文提出了一种基于深度优先搜索的模糊测试用例生成方法。相比之前的工作，该方法能够在更短的测试时间内挖掘出更多的漏洞。

该方法在合并状态迁移阶段合并了首尾状态相同的状态迁移。在消除路径循环阶段，使用基于深度优先的路径去环算法将状态机图中的环路去除，以将状态机图转换成一个有向无回路图。在搜索状态迁移路径阶段，首先使用基于深度优先搜索的两点之间的全路径搜索算法寻找状态机迁移路径，然后将前一阶段去除的边利用 Floyd 算法求出初始状态到该状态迁移的最短路径，并将这条路径和该状态迁移合并加入到待测路径集合中。在标记重复状态迁移阶段，将各重复的状态迁移标记，以便在模糊测试时不会生成重复的测试用例。在基于测试用例引导的模糊测试阶段，对协议实体程序进行模糊测试以挖掘漏洞。

参 考 文 献

[1] ZHANG X, LI Z J. Survey of Fuzz Testing Technology[J]. Computer Science, 2016, 43(5): 1-8, 26.

[2] ZHANG H Z, HONG Z, ZHOU S L, et al. A fuzzing optimization method based on protocol state migration traversal[J]. Computer Engineering and Applications, 2019, 56(4): 82-91.

[3] Oulu University Secure Programming Group. The PROTOS Project[EB/OL]. [2020-08-09]. https://www.ee.oulu.fi/research/ouspg/Protos.

[4] EDDINGTON M. PeachFuzzer [EB/OL]. [2020-08-09]. https://sourceforge.net/projects/peachfuzz.

[5] GitHub. Sulley[EB/OL]. (2013-06-11)[2020-08-09]. https://github.com/OpenRCE/sulley.

[6] Spike fuzzing platform [EB/OL]. [2017-08-11]. http://www.immunitysec.com/resourcesfreesoftware.shtml.

[7] SONG C, YU B, ZHOU X, et al. SPFuzz: A Hierarchical Scheduling Framework for Stateful Network Protocol Fuzzing[J]. IEEE Access, 2019, 7: 18490-18499.

[8] LI J L, CHEN Y L, LI Z, et al. Mining RSTP Protocol Vulnerabilities Based on Traversal of Protocol State Graph[J]. Computer Science, 2018, 45(9): 171-176.

[9] MA R, WANG D, HU C, 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.

[10] RONG F, CHANG Y. Machine Learning for Black-Box Fuzzing of Network Protocols[C]// Proceedings of the International Conference on Information and Communications Security. 2017: 621-632.

[11] ZHANG S D, ZHANG L Y. Vulnerability Mining for Network Protocols Based on Fuzzing[C]// Proceedings of the 2014 2nd International Conference on Systems and Informatics. 2014: 644-648.

[12] HUANG Y, ZOU Q W, FAN K F. Fuzzing test-based vulnerability mining for industrial control network protocol[J]. Journal on Communications, 2018, 39(S2): 181-188.

[13] WANG G B, ZHAO J L, CUI B J. Fuzzing method based on field filter and packet repair for GTPv2 protocol[J]. Internet of Things, 2019, 8(100104): 1-7.

[14] ZHANG W Y, ZHANG L, MAO J L, et al. An Automated Method of Unknown Protocol Fuzzing Test[J]. Computer Science, 2020, 43(4): 653-667.

[15] NEMINATH H, JONATHAN S. Detecting TCP ACK storm attack: a state transition modelling approach[J]. IET Networks, 2018, 7(6): 429-434.

[16] BOSSERT G, HIET G, HENIN T. Modelling to simulate botnet command and control protocols for the valuation of network intrusion detection systems[C]// 2011 Conference on Network and Information System Security (SAR-SSI). La Rochelle: IEEE, 2011: 1-8.

[17] ZUO X F, SHEN W J. An Improved Algorithm for Multiple Shortest Path Problem Based on Floyd Algorithm[J]. Computer Science, 2017, 44(5): 232-234, 267.

[18] MA R, REN S, MA K, et al. Semi-valid Fuzz Testing Case Generation for Stateful Network Protocol[J]. Tsinghua Science and Technology, 2017, 22(5): 458-468.

LI Yi-hao, born in 1996, postgraduate. His main research interests include cyberspace security and protocol reverse engineering.

HONG Zheng, born in 1979, Ph.D, associate professor. His main research interests include cyberspace security and protocol reverse engineering.