计算机科学
COMPUTER SCIENCE

基于测试用例自动化生成的协议模糊测试方法

徐威，武泽慧，王子木，陆丽

引用本文

徐威，武泽慧，王子木，陆丽. 基于测试用例自动化生成的协议模糊测试方法[J]. 计算机科学，2023，50(12)：58-65.

XU Wei, WU Zehui, WANG Zimu, LU Li. Protocol Fuzzing Based on Testcases Automated Generation [J]. Computer Science, 2023, 50(12): 58-65.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.221000225

基于测试用例自动化生成的协议模糊测试方法

徐 威[1] 武泽慧[1] 王子木[2] 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
(1150220930@qq.com)

摘 要 网络协议作为设备之间交互的规范,在计算机网络中发挥着至关重要的作用。协议实体中的漏洞会使设备遭受远程攻击,存在巨大的安全隐患。模糊测试是发现程序中安全漏洞的重要方法。在协议进行模糊测试之前需要对其进行逆向分析,在协议格式以及状态机模型的指导下生成高质量的测试用例。但上述过程中,测试用例生成需要手工构造,并且构造的测试用例难以覆盖深层次状态。针对上述问题,提出了一种自动化的测试用例生成技术。在模板中定义测试用例生成规则,基于状态迁移路径生成算法构建完备的测试路径,有效地对协议程序进行模糊测试。实验结果表明,与当前先进的协议模糊器 Boofuzz 相比,所提方法的有效测试用例生成数量增加了 51.8%。在 4 个真实软件中进行测试,验证了 3 个已公开漏洞,同时发现了一个新的缺陷并得到了开发人员的确认。

关键词:网络协议;模糊测试;测试用例生成;协议状态探测;漏洞挖掘
中图法分类号 TP393

Protocol Fuzzing Based on Testcases Automated Generation

XU Wei[1],WU Zehui[1],WANG Zimu[2] and LU Li[3]
1 State Key Laboratory of Mathematical Engineering and Advanced Computing,Zhengzhou 450001,China
2 Beijing Institute of Computer Technology and Applications,Beijing 100854,China
3 National Engineering Laboratory for Cyber Science and Technology,Zhengzhou 450001,China

Abstract As a specification for the interaction between devices,network protocols play an important role in computer networks. Vulnerabilities in protocol implementation can cause devices to be attacked remotely,which poses a huge security risk. Fuzzing is an important method to discover security vulnerabilities in programs. Before fuzzing of protocols,it is necessary to conduct reverse analysis on them,and generating high-quality testcases under the guidance of protocol format and state machine model. However,in the above process,testcase generation requires manual construction,and the constructed testcase is difficult to cover the deep level state. To solve these problems,this paper proposes an automated testcases generation technology. Defining testcase generation rules in the template,building complete test paths based on the state transition path generation algorithm,and effectively performing fuzzing on protocol programs. Experimental results show that compared with the current advanced protocol fuzzer Boofuzz,the number of effective testcases generated by the proposed method can be increased by 51.8%. It is tested in four real software to verify three open vulnerabilities. At the same time,a new flaw is found and confirmed by developers.

Keywords Network protocol,Fuzzing,Testcase generation,Protocol state detection,Vulnerability discovery

1 引言

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,其实体中会引入漏洞。一些黑客利用协议中的漏洞传播病毒,甚至能够在不需要访问物理主机的情况下发起远程攻击,导致成千上万的网络设备面临灾难性的威胁。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,结果显示,2022 年上半年协议中的高危漏洞占比超过 70%,因此,及时发现并修补协议中存在的安全漏洞极为重要。

当前主流的协议漏洞发掘方法有模糊测试(Fuzzing)、

到稿日期:2022-10-26 返修日期:2023-03-19
基金项目:国家重点研发计划(2019QY0501)
This work was supported by the National Key R&D Program of China(2019QY0501).
通信作者:武泽慧(wuzehui2010@foxmail.com)

污点分析、符号执行、补丁对比等。其中模糊测试[2]具有操作简单、效率高的特点，自 1989 提出以来已经被广泛用于多个领域的安全测试，如操作系统内核[3]、服务器[4]以及区块链[5]。模糊测试根据测试用例的生成方式可以分为基于突变的模糊测试技术和基于生成的模糊测试技术。基于突变的模糊测试技术不需要掌握协议的先验知识，减少了创建状态机需要的时间。然而，该方式生成的测试用例往往无法通过协议格式校验。基于生成的模糊测试技术，该方法通过对协议的分析获取协议的格式以及状态机模型，生成尽可能符合协议规范的测试用例。其生成的测试用例更容易被协议实体程序所接受，但是需要进行大量的人工分析以及对协议有足够的了解，且难以根据新的协议特性进行扩展。另一方面，现有的基于生成的模糊测试工具如 Peach[6]和 Boofuzz[7]不关心协议程序执行时的信息，很难探测到协议的深层次状态空间。由于基于突变的协议模糊测试工具不了解协议的状态转换关系，因此同样难以到达深层次的状态。此外，基于突变的协议模糊测试工具往往会选择更短的状态迁移路径，从而忽略了达到该状态的其他路径，造成测试不够完备的问题。

为了解决上述问题，本文使用嗅探工具捕获协议程序正常通信时产生的流量，逆向推断出协议格式和状态机模型。依据协议格式在模板中定义生成规则，自动化生成符合规范的测试用例。该方法可在缺少协议先验知识以及不干扰协议程序执行过程的前提下，快速开始一个模糊测试进程，并且具有较好的可扩展性。此外，为探测深层次的状态空间，本文通过状态机指导测试路径的生成以及消息链的发送。本文的贡献如下：

1）提出了一种测试用例自动化生成方法，通过基于关键字的协议逆向技术来推断出协议格式以及状态机模型。根据推断出的协议格式构建模板，依据模板自动化地生成测试用例。

2）提出了一种协议状态深层次覆盖方法，根据构建的状态机生成测试路径，实现深层次的协议状态覆盖以及执行到达该状态的所有路径。

3）基于上述方法，构建了自动化测试用例生成的模糊测试工具 NAPfuzz。实验结果表明，NAPfuzz 在没有协议先验知识的情况下，能够基于网络流量准确推断出协议的格式以及状态模型，通过构建模板自动生成符合协议规范的测试用例。与当前广泛使用的协议模糊测试工具 Boofuzz 以及 AFLnet[8]相比，NAPfuzz 的漏洞发现能力更强。在 HTTP 协议程序 aiohttp[9]中，发现了一个新的缺陷，并得到了开发人员的确认。

2　相关背景

2.1　协议分类

根据请求和响应数据是否独立，网络协议可以分为有状态协议和无状态协议。对于无状态协议如 HTTP，UDP，客户端逐个发送的请求数据之间没有依赖关系，服务器只响应接收到的客户端数据，而不依赖于之前的请求和响应数据。测试无状态的协议时，通过构建符合格式的测试用例，可以覆盖到较多的代码空间，测试难度较低。有状态协议的输出不仅与输入有关，还与协议程序当前的状态有关，大多数应用层协议都是有状态的，如 RTSP，FTP 协议。测试有状态协议时，不仅需要考虑协议的格式，还需考虑协议的状态转换关系。通常可以使用如图 1 所示的协议状态机模型，描述一个协议的输入输出以及状态之间的对应关系。

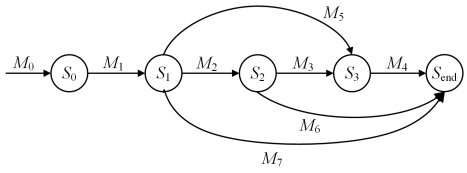


图 1　协议状态机模型

Fig. 1　Protocol state machine model

2.2　协议模糊测试

模糊测试是当前检测漏洞最有效的方法，Google 一项调查显示，超过 37%的漏洞是通过该方法发现的[10]。相较于传统的覆盖率导向的模糊器如 AFL[11]，Libfuzzer[12]，Honggfuzz[13]等，协议模糊测试更多地关注输入的格式以及被测试程序的状态信息。协议模糊测试工作流程如图 2 所示，主要包括预处理、变异策略选择、测试用例生成、模糊测试执行、异常判断，以及结果分析 6 个步骤。

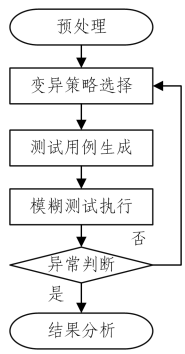


图 2　协议模糊测试流程

Fig. 2　Protocol fuzzing process

根据测试用例生成方式的不同，可以将模糊测试分为基于生成和基于突变两类。基于突变的协议模糊测试技术使用特定的策略以及算法对符合格式的输入（通常是捕获的 pcap 包）进行变异，生成新的测试用例；基于生成的协议模糊测试技术根据获取的协议规范，生成能够通过协议格式校验的测试用例，对协议实体程序的代码空间进行测试。

基于生成和基于突变的模糊测试技术在上述某些步骤中关注的目标有所不同。在预处理阶段，基于生成的模糊测试技术需要获取协议规范，用于指导测试用例的生成，代表性的工作有 Peach，Boofuzz 以及 SNOOZZ[14]。这类模糊器在测试前需要研究人员掌握协议的先验知识，手动编写脚本定义的报文格式以及状态模型。基于突变的灰盒模糊器在此阶段主要通过对协议程序的源码进行插装，来获取程序执行过程中的反馈信息，用于指导测试用例的变异，以及保留有价值的测试用例对其进一步测试。此技术代表性的工具有 AFLnet[8]，Sateafl[15]，TCP-fuzz[16]等。AFLnet 通过对源码进行插装来获取覆盖率信息，以及解析服务端应答报文中的响应码来反馈协议的状态信息。Statafl 在内存分配/释放处插入一套新

设计的桩代码，通过获取长周期变量信息来反映协议的状态。TCP-fuzz 沿用了 AFL 的插装方式，使用覆盖率信息的转变来表示协议状态的转变。然而，这类模糊器仅能对开源协议实体程序进行测试。此外，插桩会增加额外的计算开销。

在异常判断的过程中，如果测试程序崩溃或无法收到测试程序的响应，协议模糊器就会将这些异常信息以及造成此次异常的测试用例保存下来，通过人工分析判断是否为一个真正的漏洞。除了上述方式外，TCP-fuzz 同时执行不同的 TCP 协议程序，通过比较相同输入下这些程序的输出来判断是否存在语义错误。

3 测试用例及状态迁移路径生成

3.1 系统设计

NAPfuzz 的目标是在没有协议先验知识的情况下，通过协议格式逆向技术构建模板，自动化生成符合协议规范的测试用例；依据协议状态机，生成覆盖所有协议状态的测试路径；最后基于网络流量探测，判断测试程序的异常状态。如图 3 所示，NAPfuzz 的模糊测试流程如下。

首先通过协议嗅探工具如 Wireshark，Tcpdump 等，捕获协议实体程序通信时的网络流量。本文选取的是 Wireshark，因为其不仅适用于不同平台，而且具有便于操作的用户图形界面。在捕获到足够的流量后，由于这些数据通常由多种协议通信产生，因此需要根据 IP 地址以及端口号等信息对其进行过滤。然后，使用优化的多序列比对算法对齐报文序列，通过确定协议的关键词进行聚类分析，进而推断出协议的格式以及状态机模型。接着根据推断出的协议格式构建模板，定义报文的可变字段，以及字段之间的关系等语义信息。最后依据模板自动化生成丰富的测试用例，并根据推断出的协议状态机模型，指导测试路径的生成以及消息序列的发送。在模糊测试执行时，使用网络流量检测工具，对测试程序进行监测。

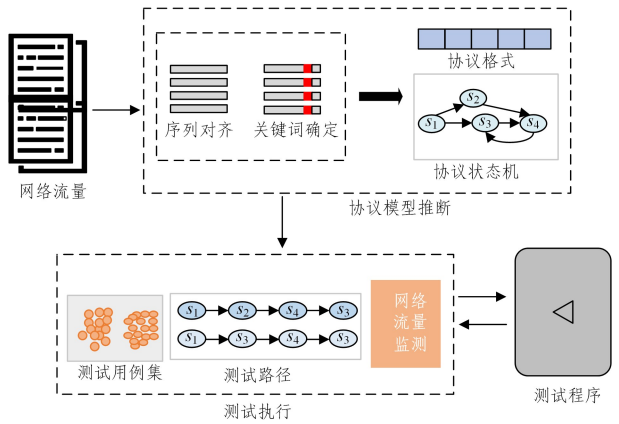


图 3 NAPfuzz 系统架构图

Fig. 3 NAPfuzz system architecture diagram

3.2 协议格式以及状态机模型推断

现有的协议逆向技术可分为两类：一类是基于协议实体程序的指令序列[17-19]，另一类是基于嗅探工具捕获的网络流量[20-25]。虽然前者可以在逆向过程中得到较高的精度，但在实践中的可行性通常较低。后者实行起来更为方便，因此本文采用基于网络流量的协议逆向技术，根据 Wireshark 捕获到的网络流量，推断出协议格式以及状态机模型，用于指导测试用例及测试路径的生成，实现深层次状态及其迁移路径的覆盖。在此步骤中，我们主要沿用了当前最先进的开源协议逆向工具 Netzob 的方法，并在其基础上进行了改进，以提高协议格式推断的准确性。Netzob 利用基于语义的 Needleman-Wunsch(NW)算法[26]对齐报文序列并计算相似性分数，之后根据这些分数进行聚类，通过分析集群内报文的共性得到协议格式。然而，Netzob 在语义收集阶段需要测试人员进行手工操作且对待测协议有一定了解。此外，由于报文内容的多样性大大降低了对齐的质量，因此造成了错误的聚类结果。在实际的通信过程中，客户端或者服务端在接收到消息时，通过关键字确定报文的类型。本节将进一步完善 Netzob 的格式推断方法，在识别关键词的基础上提高聚类结果的准确性，进而得到更准确的协议格式以及状态机模型。

在前文的描述中，作为报文聚类的依据，关键词的推断尤为重要。关键词通常在报文序列中的动态字段中产生，简单的协议报文格式具有相同的结构，关键词所在的偏移位置也相同。如图 4 所示的复杂协议中，报文具有不同的长度，关键词所在的偏移位置会有偏差。

m_{c0}	2f 31 2e 30 73 44 53 32 6f 93 e7 f2 53 00
m_{c1}	2f 31 2e 30 73 44 53 32 2f 95 1d d3 00 00
m_{c2}	2f 31 2e 30 65 44 53 36 6d 93 e3 43 00 00 02 11 5c 42 d5
m_{c3}	2f 31 2e 30 45 44 53 4c 6f 93 e4 4f 00 00
m_{c4}	2f 31 2e 30 73 44 53 0d 6f 93 e0 4d 63 00 02 11 5d 43 d5
m_{c5}	2f 31 2e 30 73 44 53 32 6f 95 2d d2 83 00
m_{c6}	2f 31 2e 30 73 44 53 32 5f 95 ed 44 00 00
m_{c7}	2f 31 2e 30 41 44 53 3c 3d 95 28 41 00 00 00 00 0c 2f 6f

图 4 协议报文序列

Fig. 4 Protocol messages

因此，本文首先使用改进的多序列对比算法将所有的报文序列通过插入“-”符号对齐。相较于传统的 Needleman-Wunsch 算法，本文提出的算法更加简单高效，具体过程如算法 1 所示。

算法 1 改进多序列比对算法

输入：$M_1, M_2, M_3, \cdots, M_n$

/* 初始报文序列 */

输出：$R_1, R_2, R_3, \cdots, R_n$

/* 对齐后的报文序列 */

1. /* 计算初始序列中最长序列长度 l_{max} */
2. /* 计算初始序列中最短序列的长度 l_{min} */
3. j←2

/* 从第 1 个字节开始初始化字段划分 */

4. while j<l_{min} do
5. if g⟨j−2,j⟩ and g⟨j,j+2⟩ is flag(s)
6. then g⟨j−2,j+2⟩←flag(s)
7. end
8. j←j+2 /* 继续划分下一字节 */
9. end while
10. g⟨l_{min},l_{max}⟩←data

/* 大于 l_{min} 的长度划分为 data 字段 */

11. return $R_1, R_2, R_3, \cdots, R_n$

使用上述算法将图 4 中的报文序列对齐后得到图 5 所示的结果。在对齐过程中，如果报文当前偏移位置字段为空，将使用“-”符号进行填充。依据对齐后的结果可以将其划分为不同的字段，并且将连续的多个静态字段合并成为一个长静态字段。关键词一定是在动态字段中产生，因此将图 5 中的 f1，f3，f4，f5，f8 作为候选的关键词。对于 f10 字段，mc0，mc1 等报文序列对应的值为空，不可能是关键词，同样可以将其忽略。

	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}
m_{c0}	2f 31 2e 30	73	44 53	32	6f	93	e7	f2	53	00	-- -- -- -- --
m_{c1}	2f 31 2e 30	73	44 53	32	2f	95	1d	d3	00	00	-- -- -- -- --
m_{c2}	2f 31 2e 30	65	44 53	36	6d	93	e3	43	00	00	02 11 5c 42 d5
m_{c3}	2f 31 2e 30	45	44 53	4c	6f	93	e4	4f	00	00	-- -- -- -- --
m_{c4}	2f 31 2e 30	73	44 53	0d	6f	93	e0	4d	63	00	02 11 5d 43 d5
m_{c5}	2f 31 2e 30	73	44 53	32	6f	95	2d	d2	83	00	-- -- -- -- --
m_{c6}	2f 31 2e 30	73	44 53	32	5f	95	ed	44	00	00	-- -- -- -- --
m_{c7}	2f 31 2e 30	41	44 53	3c	3d	95	28	41	00	00	00 00 0c 2f 6f

图 5　经过多序列算法对齐后的报文序列

Fig. 5　Message aligned by multiple sequence algorithm

1）报文相似度评分

对于复杂协议包含较多的字段，使用简单的报文序列比较可能会得到错误的报文间的相似度。因此本文引入编辑距离，来确定报文间的相似度。两个报文 m_k 和 m_t 之间的相似性得分 S_{kt} 的计算式如式(1)所示：

$$\begin{cases} S_{kt}=1-\dfrac{edit\_distance(m_k,m_t)}{\max\_len(m_k,m_t)} \\ P_1=\dfrac{\sum_i^n S_{kt}}{N} \end{cases} \tag{1}$$

其中，$edit\_distance(m_k,m_t)$ 表示 m_k 和 m_t 转变成两个相同的序列时需要的最少的增加、删除或者替换操作次数。$\max\_len(m_k,m_t)$ 表示 m_k 和 m_t 中最长的报文序列的长度。

2）集群内报文间结构相似性评分

在同一集群内，所有的报文应当具有相同的结构，在报文对齐阶段会尽可能少使用“-”符号。此外，集群的总数量不应过多。因此，本文使用平均每个集群中“-”符号的数量来确定结构相似度 P_2。

$$P_2=\frac{\sum_i^N number\_gap}{N} \tag{2}$$

其中，$\sum_1^N number\_gap$ 表示所有集群中“-”符号的总数量，N 表示集群的数量。

3）关键字位置字段评分

在客户端与服务端进行序列对齐后，分别根据上述两个度量指标选择得分最高的关键词，并计算字段偏移值 d_c 和 d_s，以及字段长度 l_c 和 l_s。通过式(3)计算关键词位置字段评分 P_3。

$$P_3=\begin{cases} \dfrac{d_c\times l_c}{d_s\times l_s}, & d_c\times l_c\leqslant d_s\times l_s \\ \dfrac{d_s\times l_s}{d_c\times l_c}, & d_c\times l_c>d_s\times l_s \end{cases} \tag{3}$$

候选关键词作为实际关键词的概率为上述 3 个度量结果的乘积，数值最大的即为最可能的关键词。在关键词确定完成后，将关键词相同的报文聚类，并推断出协议的格式以及状态机模型。

这一步骤是粗粒度的选择，之后将对候选关键字进行度量，进一步确定这些字段作为关键词的概率，概率最大的字段将被确定为协议的关键字段。度量主要基于以下几个方面：

1）基于关键词聚类后，同一集群中报文序列应当具有高度的相似性。

2）同一集群内的报文之间应当具有相同的结构。

3）关键词在所有的报文序列中同时出现，且长度固定。

在协议格式推断方面，将同一个集群内的报文序列对齐后分别标记为动态字段以及静态字段，并记录其长度。图 5 中的报文序列经过上述方法可以判断出 f5 为关键词，因此可以将报文聚类为两个集群。分别是集群 1{mc0，mc2 mc3，mc4 }和集群 2{mc1，mc5 mc6，mc7 }。集群 1 的协议格式如图 6 所示。状态机的推断使用 Ládi G[27] 的研究成果，本文不再赘述。

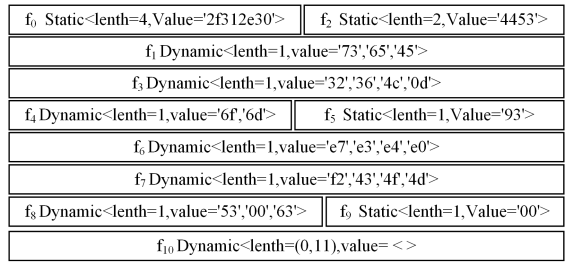


图 6　集群 1 协议格式

Fig. 6　cluster 1 protocol format

3.3　基于模板的测试用例自动生成

根据前文阐述，测试用例的质量直接影响到最终的模糊测试结果。高质量测试用例有更大的的概率触发目标中存在的漏洞。协议程序往往对输入有着严格的校验，其通信过程如图 7 所示。当协议程序收到一个外部输入时，首先会对输入进行解析，并依据规则进行匹配。如果解析之后匹配错误，输入会被直接拒绝并响应 Error。在匹配成功后，根据输入数据字段匹配执行的功能，执行完成后向客户端发送相应的响应。本小节的目的是根据协议的格式以及语义构建模板，在模板中将字段分为静态字段以及动态字段。同时，根据语义信息定义动态字段的长度等。在测试用例生成时，不改变协议格式和静态字段或者进行微小的改动，防止模糊器发送的测试报文在开始阶段被待测程序拒绝而无法起到漏洞挖掘的作用。

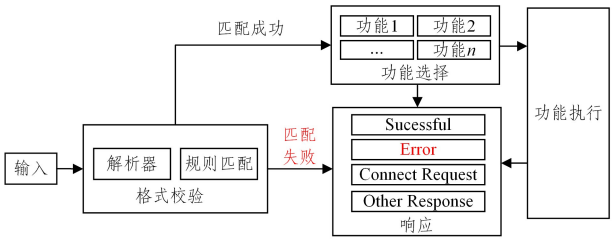


图 7 协议程序工作过程

Fig. 7 Protocol program workflow

根据缓冲区溢出、格式化字符串、整形溢出等漏洞的触发条件，在模板中定义如表 1 所列的测试用例的生成规则。

1)对于静态字段，在通信过程中几乎保持原值，但是该字段在解析异常情况下仍有触发漏洞的可能性，因此使用边界值比如$-1,0,2^{n-1},-2^{n}$对其进行填充。

2)对于动态字段，生成方式更加丰富，可以为模糊测试构建大量的测试报文，增强漏洞触发能力，具体如下：

(1)文本类型动态字段：对分隔符进行替换、扩充以及减少操作。替换指将分割符替换为其他特殊字符，比如将“/”“r/n”替换为“%”“/n”等；扩充是指对分隔符进行 n 倍重复，这样可以检测到越界错误；减少指将分隔符随机删除。此外设置一个字典，用预定义的字符串如“true”“play”“%d”对其进行替换，以扩大测试数据的覆盖范围。

(2)二进制类型动态字段：对长度较短的二进制字段进行逐位取反；对长度较长的二进制字段进行字节级的取反、右移、与其他动态字段互换等操作。这种方式可以检测解析器以及功能代码中潜在的漏洞。

表 1 测试用例生成规则

Table 1 Testcase generation rules

字段类型	类别	类型	规则概述
静态字段	字符串/二进制	—	使用如$-1,0,2^{n}-1,-2^{n}$这样的边界值进行填充
动态字段	字符串	分隔符	替换、扩充、删除
		其余部分	对其进行 n 倍重复，使用字典定义字符串进行填充
	二进制	短字段	逐位取反、空值
		长字段	字节级取反、右移、互换
变长字段	字符串/二进制	—	使用灵活的方式进行随机填充

对于变长字段，可以使用灵活的方式对其进行随机的数据填充。

由于漏洞的触发条件比较复杂，因此需要同时对多个不同的数据字段进行填充。上述的字段生成方案一次只填充一个片段，然而，多字段变异在消息中随机选择一些动态字段，并对每个选择的字段执行上述生成方案。在接收到新的响应或目标协议程序崩溃之前，多字段变异将会一直进行下去。

3.4 状态迁移路径生成

当前的协议模糊测试工具难以测试到深层次的协议状态，且会忽略较长的协议状态迁移路径。在图 1 所示的协议状态机模型中，到达状态 S_3 的状态迁移路径有 $L_1\langle S_0,S_1,S_2,S_3\rangle$以及 $L_2\langle S_0,S_1,S_3\rangle$。状态迁移路径越长，需要发送和接收报文的时间开销就越大。大多数模糊测试工具倾向于发送更快到达目标状态的报文序列，以减少消息的发送和接收时间。因此现有方法很难覆盖深层状态，同样也无法找到深层次路径下的 bug。另一方面，模糊测试执行的时间通常有限，应当将主要的时间和精力花费在更有可能发现漏洞的状态迁移路径上。因此，本文根据协议的状态机模型，首先对协议状态迁移路径进行删减，将几乎不可能存在漏洞的状态迁移路径删除，此路径主要包含两种：直接迁移到终止状态的路径和回退到初始状态的路径。

根据上述方法删除图 1 中安全的状态迁移路径后得到如图 8 所示的状态机模型。之后，本文基于深度优先算法遍历状态机中的所有状态，得到完成的状态迁移路径。具体的测试路径生成算法如算法 2 所示。

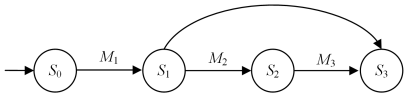


图 8 删除安全路径后的状态机

Fig. 8 State machine after deleting secure paths

算法 2 测试路径生成算法

输入：协议状态图 Statemap

输出：状态迁移路径 Mesqueue

1. S_{INIT} == GetInit(Statemap)/ * 定义初始状态 * /
2. S_{END} == GetFin(Statemap)/ * 定义终止状态 * /
3. Mesqueue. push(Sinit)
   / * 初始状态放入状态迁移路径 * /
4. While state∈Getstate(Statemap)do
   / * 遍历协议状态图中所有的状态 * /
5. If state != S_{INIT}
   / * 判断当前状态是否为一个新的状态 * /
6. Set S_{INIT} == state
7. Mesqueue. push(S_{INIT})
   / * 初始状态放入状态迁移路径 * /
8. End if
9. If state == S_{END}or state ==Φ/ * 遍历完成 * /
10. Get. push(Mesqueue)
    / * 获取完整的状态迁移路径 * /
11. End if

算法 2 首先获取协议的初始以及终止状态，并将初始状态存放到状态迁移路径中，然后根据协议状态图遍历所有的协议状态。在遍历过程中，算法首先选择初始状态的后继状态，并判断此状态是否被遍历过。如果没有标记该状态，则算法会先标记该状态并将其放到状态会话链中，随后选择其后继状态。否则，算法将跳过此状态并重复迭代，直到后继状态未标记。在终止状态遍历完成或不存在后继状态时完成遍历，获取到完成状态迁移路径。图 8 中生成的两条状态迁移路径分别为：$\langle S_0,S_1,S_2,S_3\rangle$以及$\langle S_0,S_1,S_3\rangle$。

3.5 网络异常监测

此步骤用于探测目标程序是否存活，其通过监控设备的

网络通信并设置超时，确定设备是否崩溃。对设备网络通信的监控并不是一个单一的步骤，而是发生在整个模糊过程中。在超时的情况下，NAPfuzz 将继续发送相同的消息序列 3 次，减少由网络波动而不是程序崩溃造成的误报。如果 3 次均超时，NAPfuzz 将使用脚本重新启动测试程序，并再次发送相同的消息序列到测试程序。如果测试仍然没有按时返回消息，NAPfuzz 将记录崩溃和相应的消息序列。

4　实验评估

4.1　试验设置

本文根据上述模糊测试框架，构建了 NAPfuzz。首先测试 NAPfuzz 协议格式逆向的准确性，然后与当前广泛使用的协议模糊测试工具 Boofuzz 和 AFLnet 进行对比。Boofuzz 作为 Sulley 工作的继承，是一个基于生成的协议模糊测试工具，不过其没有协议逆向功能，需要手动定义协议的格式。本文将 NAPfuzz 的逆向结果输入 Boofuzz，对比二者测试用例生成的数量，以及测试用例的质量。此外，本文对真实的协议实体软件进行了测试，与 Boofuzz 和 AFLnet 进行了漏洞发现能力的对比实验。

实验硬件设置为：Intel(R)Core(TM) i7-10700 CPU @ 2.90GHz，运行内存 256GB 的 Ubuntu18.04 服务器。

4.2　格式准确性评估

本文选取广泛使用的网络协议 RTSP 和 ICMP 来测试 NAPfuzz 协议格式的逆向能力。推断出的协议报文格式以及实际的报文格式如图 9 所示。图 9(a)中，RTSP 协议报文中 Method 被识别为动态字段；空白字段指包括“空格”“换行符”在内的分隔符，URL、版本、首部字段名是固定的，因此被识别为静态字段。序列号的值是递增的动态字段；其他首部字段的值如 session 值是相同的，因此被识别为静态字段。图9(b)中，ICMP 协议报文中类型、校验和被识别为动态字段，代码、标识符、选项字段被识别为静态字段。

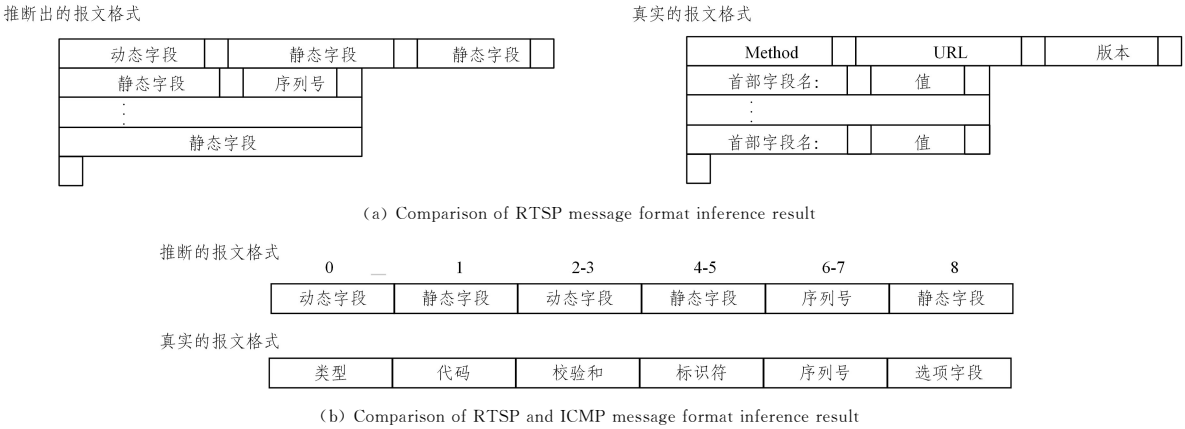


图 9　RTSP 和 ICMP 报文格式推断结果比较

Fig. 9　Comparison of RTSP and ICMP message format inference results

4.3　生成测试用例的数量以及准确性

测试用例的质量会直接影响测试的结果，能够通过协议格式检查的测试用例才会被协议程序所接受，起到漏洞挖掘的作用。本文将 NAPfuzz 与 Boofuzz 进行对比，通过比较相同时间内生成有效测试用例的数量以及所占比例来判断其测试用例的质量。图 10 给出了 5h 内 NAPfuzz 与 Boofuzz 生成的测试用例总数量以及有效测试用例(能够触发协议实体程序正常响应的测试用例)数量的情况。

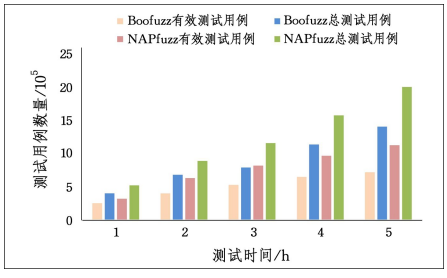


图 10　测试用例生成结果

Fig. 10　Testcase generation results

从图 10 中可以看出，NAPfuzz 生成的测试用例数量以及质量都优于 Boofuzz。在 5h 内 NAPfuzz 生成的有效测试用例数量比 Boofuzz 提高了 51.8%。在测试时间超过 4h 后，NAPfuzz 生成测试用例的速度加快，但是有效测试用例占比降低。这是由于测试后期进入多字段变异阶段，生成测试用例的数量增加，且其可能会破坏协议格式，造成无效的测试用例也增多。即便如此，NAPfuzz 有效测试用例的数量以及比例依然高于 Boofuzz。

4.4　漏洞发现能力评估

本文选取常用的有状态以及无状态网络协议实体程序作为测试对象。选择的具体协议及实体程序如表 2 所列。其中，FTP，RTSP 和 DNS 都是经典的有状态网络协议，并且具有较为复杂的状态转换关系。HTTP 作为无状态网络协议在现实世界中也有着广泛的应用。因此，本文选择上述协议的实体程序作为测试对象，来评估 AFLnet，Boofuzz 和 NAPfuzz 的漏洞发现能力。

表 2　待测协议信息

Table 2　Protocol information to be tested

Protocol	Program	Stateful/Stateless network protocol
HTTP	aiohttp v3.8.1	Stateless network protocol
DNS	BIND 9 v9.9.6	Stateful network protocol
FTP	OpenTFTP 1.65	Stateful network protocol
RTSP	Live555 v 0.93	Stateful network protocol

对表 2 中的 4 个程序分别使用 AFLnet, Boofuzz, NAPfuzz 测试 24h 后的漏洞发现情况如表 3 所列。表 3 中, type 表示漏洞的类型, detected 表示是否检测到该漏洞, time 表示发现该漏洞的时间, 大于 24 h 表示在测试的 24 h 内没有发现该漏洞。在漏洞发现数量上, NAPfuzz 能够发现 AFLnet 及 Boofuzz 发现的所有历史漏洞, 漏洞类型包括缓冲区溢出、格式化字符串、释放后重利用。在漏洞发现的时间上, NAPfuzz 所用的时间少于另外两种工具。此外, 我们在 aiohttp v3.8.1 中发现了一个新的缺陷, 目前已经报告给开发人员, 并得到了开发人员的确认。AFLnet 随机变异生成的测试用例畸形度过高, 而 Boofuzz 生成策略比较简单, 导致生成了大量类似的测试用例。因此二者都没有发现 aiohttp 中的缺陷。

表 3 漏洞发现情况

Table 3 Vulnerabilities discovery

Rrogram	CVE number	Type	NAPfuzz		AFLnet		Boofuzz	
			detected	time	detected	time	detected	time
Aiohttp v3.8.1	—	crash	Y	25 min 12 s	N	大于 24 h	N	大于 24 h
BIND 9 v9.9.6	CVE-2021-25216	buffer overflow	Y	5 min 9 s	Y	9 min 4 s	Y	12 min 32 s
Lightftp v1.1	CNVD-2020-22689	format string	Y	8 min 46 s	Y	11 min 20 s	Y	20 min 12 s
Live555 v0.93	CVE-2019-15232	Use-After-Free	Y	1 h 23 min	Y	2 h 12 min	N	大于 24 h

结束语 本文研究网络协议模糊测试自动化生成符合协议规范的测试用例问题, 提出了一种基于报文序列关键词的逆向方法, 推断出协议格式以及状态机模型; 在模板中定义测试用例生成的规则以及报文的格式, 生成器依据模板生成测试用例; 基于协议的状态机模型, 使用深度优先算法生成测试路径, 使模糊器尽可能测试到所有的状态, 以及状态迁移路径。实验结果表明, NAPfuzz 可以生成高质量的测试用例, 且能够有效地发现协议实体程序中的漏洞。本文的不足之处在于需要收集足够的通信流量, 才能保证推断结果的准确性。此外, 本文方法无法应用于加密的协议。另一方面, 本文实现的是一个黑盒协议模糊器, 下一步将研究通过执行时的反馈信息对报文中不同的可变字段实现不同的变异策略, 同时保留有价值的种子并对其进一步进行测试。

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] SCHUMILO S, ASCHERMANN C, GAWLIK R, et al. {kAFL}: {Hardware-Assisted} Feedback Fuzzing for {OS} Kernels[C]// 26th USENIX Security Symposium (USENIX Security 17). 2017: 167-182.

[4] ZHAO W, XIE F, PENG Y, et al. Security testing methods and techniques of industrial control devices[C]// 2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing. IEEE, 2013: 433-436.

[5] ASHRAF I, MA X, JIANG B, et al. GasFuzzer: Fuzzing ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities[J]. IEEE Access, 2020, 8: 99552-99564.

[6] MICHAEL E. PEACH FUZZER[EB/OL]. (2021-03-30) [2022-10-13]. https://peachtech. gitlab. io/peach-fuzzer-community.

[7] JOSHUA P. Boofuzz. [EB/OL]. (2022-2-12) [2022-10-13]. https://github. com/jtpereyda/bo-ofuzz.

[8] 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 (ICST). IEEE, 2020: 460-465.

[9] ANDREW S, SVIATOSLAV S, NIKOLAY K, et al. aiohttp [EB/OL]. (202-9-16) [2022-10-13]. https://github. com/aio-libs/aiohttp.

[10] HAWKES B. Project zero five years of 'make 0day hard' [EB/OL]. (2019-07-15) [2022-10-13]. https://i. blackhat. com/USA-19/Thursday/us-19-Hawkes-Project-Zero-Five-Years-Of-Make-0day-Hard. pdf.

[11] ZALEWSKI M. American fuzzy lop [EB / OL]. (2014-08-08) [2022-10-13]. http://lcamtuf. coredump. cx/afl.

[12] MAX M, FRANCISCO O, JULIAN V, et al. Libfuzzer [EB/OL]. (2021-12-19) [2022-10-13]. https://github. com/Dor1s/libfuzzer-workshop.

[13] ANESTIS B, DAVID C, KAMIL R, et al. Honggfuzz [EB/OL]. (2021-12-19) [2022-10-13]. https://github. com/google/honggfuzz.

[14] NEVES N, ANTUNES J, CORREIA M, et al. Using attack injection to discover new vulnerabilities[C]// International Conference on Dependable Systems and Networks (DSN'06). IEEE, 2006: 457-466.

[15] NATELLA R. Stateafl: Greybox fuzzing for stateful network servers[J]. Empirical Software Engineering, 2022, 27(7): 1-31.

[16] ZOU Y H, BAI J J, ZHOU J, et al. {TCP-Fuzz}: Detecting Memory and Semantic Bugs in {TCP} Stacks with Fuzzing [C]// 2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021: 489-502.

[17] NEWSOME J, BRUMLEY D, FRANKLIN J, et al. Replayer: Automatic protocol replay by binary analysis[C]// Proceedings of the 13th ACM Conference on Computer and Communications Security. 2006: 311-321.

[18] LIN Z, ZHANG X, XU D. Automatic reverse engineering of data structures from binary execution[C]// Proceedings of the 11th Annual Information Security Symposium. 2010.

[19] MA R，ZHENG H，WANG J，et al. Automatic protocol reverse engineering for industrial control systems with dynamic taint analysis[J]. Frontiers of Information Technology & Electronic Engineering，2022，23(3)：351-360.

[20] BOSSERT G，GUIHÉRY F，HIET G. Towards automated protocol reverse engineering using semantic information[C]// Proceedings of the 9th ACM Symposium on Information，Computer and Communications Security. 2014：51-62.

[21] LEITA C，MERMOUD K，DACIER M. Scriptgen：an automated script generation tool for honeyd[C]// 21st Annual Computer Security Applications Conference(ACSAC'05). IEEE，2005.

[22] CUI W，KANNAN J，WANG H J. Discoverer：Automatic Protocol Reverse Engineering from Network Traces[C]// USENIX Security Symposium. 2007：1-14.

[23] KLEBER S，VAN DER HEIJDEN R W，KARGL F. Message type identification of binary network protocols using continuous segment similarity[C]// IEEE Conference on Computer Communications(INFOCOM 2020). IEEE，2020：2243-2252.

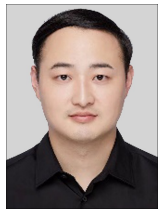
[24] LUO J Z，YU S Z. Position-based automatic reverse engineering of network protocols[J]. Journal of Network and Computer Applications，2013，36(3)：1070-1077.

[25] KARIM F，MAJUMDAR S，DARABI H，et al. LSTM fully convolutional networks for time series classification[J]. IEEE Access，2017，6：1662-1669.

[26] NEEDLEMAN S B，WUNSCH C D. 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.

[27] LÁDI G，BUTTYÁN L，HOLCZER T. GrAMeFFSI：Graph Analysis Based Message Format and Field Semantics Inference For Binary Protocols，Using Recorded Network Traffic[J]. Infocommunications Journal，2020，12(2)：25-33.

XU Wei，born in 1997，postgraduate. His main research interests include reverse engineering and vulnerability mining.



WU Zehui，born in 1988，Ph. D. His main research interests include software vulnerability and software-defined networking.

（责任编辑：何杨）