

基于状态引导的放大漏洞挖掘方法

蒋思康, 蔡瑞杰, 尹小康, 陈鸿羽, 刘胜利

(信息工程大学, 河南 郑州 450001)

摘要: 基于放大的分布式拒绝服务 (amplification-based distributed denial of service, ADDoS) 攻击对互联网是严重威胁。近年来, ADDoS 攻击事件显示该类攻击流量巨大且所利用的协议类型多样, 其中放大漏洞是导致 ADDoS 攻击的主要原因之一。然而, 迄今已知的放大漏洞主要是研究人员通过经验知识发现或者通过分析放大攻击事件的流量发现的, 缺乏主动挖掘放大漏洞的方法。已有的放大漏洞挖掘方法 AmpFuzz, 仅考虑单个请求的放大攻击模式, 并且局限于 UDP, 适用性不强。为此, 提出基于状态引导的放大漏洞挖掘方法 AFLAMP, 通过协议状态引导模糊测试, 并采用基于会话的带宽放大系数指导种子选择, 以更有效地挖掘放大漏洞。实验结果表明, 该方法可有效挖掘放大漏洞, 并在 5 个已知存在放大漏洞的协议服务程序 (OpenTFTP、OpenSLP、NTP、Memcached、Dnsmasq) 中发现了 11 个放大漏洞, 其中包括 6 个未知的放大漏洞, 相较于 AmpFuzz, 漏洞检出率提高 37.5%。此外, AFLAMP 还在基于 TCP 的协议服务程序 (LightFTP) 中发现了 12 个放大漏洞。

关键词: DDoS 攻击; 流量放大; 模糊测试; 协议状态; 漏洞挖掘

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2025012

Amplification vulnerability mining method based on state-guided

JIANG Sikang, CAI Ruijie, YIN Xiaokang, CHEN Hongyu, LIU Shengli

Information Engineering University, Zhengzhou 450001, China

Abstract: Amplification-based distributed denial of service (ADDoS) attacks have posed a persistent and severe threat to the Internet. Recent incidents revealed that these attacks not only generated substantial traffic but also exploited a diverse range of protocol types, with amplification vulnerabilities being identified as a primary cause. Traditionally, known amplification vulnerabilities were discovered either through empirical knowledge or by analyzing traffic from amplification attack incidents, highlighting a lack of proactive methods for identifying such vulnerabilities. Existing approaches, such as AmpFuzz, were limited to focusing solely on amplification patterns for individual requests and were restricted to the UDP protocol, which constrained their applicability. To address these limitations, a state-guided method for mining amplification vulnerabilities, referred to as AFLAMP, was proposed. This method

收稿日期: 2024-07-13; 修回日期: 2024-11-24

通信作者: 刘胜利, mr_shengliliu@163.com

引用格式: 蒋思康, 蔡瑞杰, 尹小康, 等. 基于状态引导的放大漏洞挖掘方法[J]. 网络与信息安全学报, 2025, 11(1): 165-177.

Citation Format: JIANG S K, CAI R J, YIN X K, et al. Amplification vulnerability mining method based on state-guided[J]. Chinese Journal of Network and Information Security, 2025, 11(1): 165-177.

leveraged protocol state to guide fuzz testing and employed session-based bandwidth amplification coefficients for seed selection, thereby enhancing the effectiveness of vulnerability discovery. Experimental results demonstrate that AFLAMP successfully identifies amplification vulnerabilities, uncovering 11 vulnerabilities in five services known to be susceptible (OpenTFTP, OpenSLP, NTP, Memcached, and Dnsmasq), including six previously unknown vulnerabilities. Compared to AmpFuzz, AFLAMP achieves 37.5% increase in the detection rate of vulnerabilities. Additionally, AFLAMP identifies 12 amplification vulnerabilities in a TCP-based service program (LightFTP).

Keywords: DDoS attack, traffic amplification, fuzz testing, protocol state, vulnerability discovery

0 引言

多年以来, 基于放大的分布式拒绝服务 (amplification-based distributed denial of service, ADDoS) 攻击^[1]正逐渐形成强大的DDoS攻击策略。根据清华大学联合华为等公司发布的《2023年全球DDoS攻击现状与趋势分析》报告, DDoS的攻击频次呈持续增长趋势, 在对超过100 Gbit/s流量的攻击进行分析后发现, UDP (user datagram protocol) 反射放大攻击和UDP Flood攻击是攻击者发起大流量攻击的主要手段, 并且TCP反射放大攻击和SYN Flood攻击也占据较大的比重。其中放大漏洞是导致ADDoS攻击的主要原因之一, 因此, 开展放大漏洞自动化挖掘技术研究具有重要的价值。

当前的漏洞挖掘研究工作主要聚焦于内存破坏型漏洞、命令注入漏洞等, 缺少针对放大漏洞的挖掘方法。究其原因, 放大漏洞相较于其他类型漏洞, 无法通过直接观察程序的运行状态发现漏洞, 漏洞发现难度较大。对于程序崩溃错误, 通常只关心某种请求类型是否会触发崩溃; 对于放大漏洞, 则更加关注响应报文的载荷与请求报文的载荷的比值。2022年, Krupp等^[2]首次提出在基于UDP的网络服务中挖掘放大漏洞的系统性方法AmpFuzz, 其基于最先进的灰盒模糊测试工具, 使用UDP状态感知技术显著提高了模糊测试的效率。

AmpFuzz为放大漏洞的挖掘提供了新的思路和解决办法, 然而在当前的研究中仍存在以下问题。

1) AmpFuzz在模糊测试过程中未考虑协议的状态信息, 难以发现基于状态转化的放大漏洞。

2) 当前研究通常将带宽放大系数 (bandwidth amplification factor, BAF) 作为放大漏洞的评价指标, 然而对于状态转化的放大漏洞, 将

BAF作为评价指标不够合理。

3) 对于协议模糊测试, 现有的方法大多数依赖手动设置的超时时间来开始或结束某个请求。这一方面可能会导致某些请求被服务端丢弃, 另一方面会导致模糊测试的效率降低。

4) AmpFuzz仅考虑基于UDP的放大漏洞, 并不支持基于TCP放大漏洞的挖掘。近年来的一些报告指出基于TCP的ADDoS攻击正在泛滥, 因此需要一个通用的放大漏洞挖掘框架来解决此问题。

为解决上述问题和挑战, 本文提出基于状态引导的放大漏洞挖掘方法。该方法利用静态分析和轻量级的插桩技术来获取协议服务程序的运行状态, 并通过添加放大向量最大化的种子变异策略, 使用基于会话的带宽放大系数来指导种子的选择, 从而提高放大漏洞的挖掘效果。

本文的贡献如下。

1) 提出基于会话的带宽放大系数 (BAFmsg), 用于评估请求序列的放大系数, 进而引导种子选择过程。

2) 引入程序运行状态感知技术来准确把握待测服务程序的运行状态, 用于解决对超时机制的依赖问题。

3) 提出并实现基于状态引导的放大漏洞挖掘框架AFLAMP。

4) 在5个已知存在放大漏洞的协议服务程序 (OpenTFTP、OpenSLP、NTP、Memcached、Dnsmasq) 中发现了11个放大漏洞, 其中包括6个未知的放大漏洞。此外, 还在基于TCP的协议服务程序 (LightFTP) 中发现了12个放大漏洞。

1 相关工作

1.1 基于放大的分布式拒绝服务攻击

传统上, 攻击者需要控制大量资源才能发起

DDoS 攻击，传统 DDoS 攻击场景^[1]如图 1 所示。但随着反射器和放大器的使用，攻击者可以利用某些易受攻击的服务来发起 ADDoS 攻击，典型的反射放大攻击场景如图 2 所示。攻击者通过欺骗或伪造 IP 数据包，将源地址修改为受害者的 IP，并将其发送给互联网中的反射放大服务器。当反射放大服务器接收到这些请求后，会向受害者发送比请求数据包更大的响应数据包，从而造成反射放大攻击。

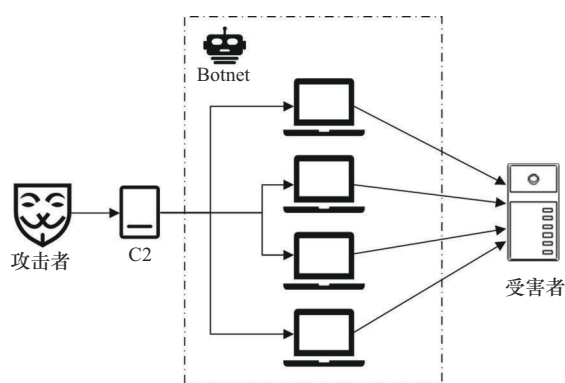


图1 传统 DDoS 攻击场景

Figure 1 Traditional DDoS attack scenario

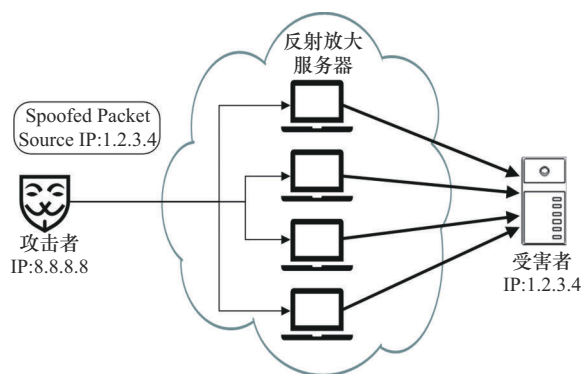


图2 反射放大攻击场景

Figure 2 Reflection amplification attack scenario

由于 UDP 是面向无连接的协议，其通常不验证请求来源是否合法而是直接发送响应数据包，因此这些基于 UDP 的网络服务节点可以作为天然的反射器，文献[3-7]大多围绕基于 UDP 的网络服务进行放大漏洞的研究，不过也有文献^[8-9]研究如何利用 TCP 发起反射放大攻击。

1.2 基于协议分析的放大漏洞挖掘

2014 年，Rossow 等^[3]提出了使用 BAF 和报文放大系数 (packet amplification factor, PAF) 来

衡量网络服务是否存在放大漏洞。同时，他们还对 14 种基于 UDP 的网络服务进行研究和测量。此后，研究者针对各种协议开展了进一步的研究。

Czyz 等^[4]深入研究了 NTP (network time protocol) 及其在放大攻击中的使用。Rijswijk-Deij 等^[5]对当时新引入的 DNSSEC 机制进行了详细的分析，发现通过向支持 DNSSEC 的域名发送 ANY 查询会得到更大的 BAF。Adamsky 等^[6]研究如何利用 P2P 网络发起放大攻击，也就是由攻击者首先将数据上传到分布式存储系统，然后欺骗受害者下载请求。在后来的许多放大攻击中发现了与此类似的攻击方式。例如，在 Memcache 协议上发现的放大漏洞，攻击者先向 Memcached 服务器发送一个添加超长键值对的请求，而后通过伪造受害者的 IP 地址不断地向服务器发送获取键值的请求，从而产生巨大的流量放大效果。Xu 等^[7]通过对 DNS 解析服务的研究，提出了新型 DNS 放大攻击 TsuKing，TsuKing 协调多个易受攻击的 DNS 解析器并精心构建查询语句，以形成强大的 DoS 放大器。

除了 UDP 外，Kührer 等^[8]研究了 TCP 3 次握手机制本身的放大能力，通过对互联网发起的 SYN/ACK、PSH 和 RST 请求进行扫描，从而了解这 3 个类别所对应请求的放大系数。2021 年，Bock 等^[9]提出利用中间盒 (middle box) 发起反射放大攻击，该方法利用 TCP 会话识别机制的漏洞，通过伪造受害者 IP 地址，发送 SYN 数据包与中间盒产生单向的 TCP 连接，然后发送一个带有未备案域名的请求数据包。接下来，攻击者利用一些中间盒没有 TCP 双向会话校验机制的设计缺陷欺骗中间盒，并利用中间盒下达对未备案域名的拦截。由于拦截时会给受害者返回包含阻断页面的响应报文，因此会产生流量放大的效果。

上述方法都是通过手动协议规范分析来发现放大漏洞的，本文的目标是使用自动化方法发现放大漏洞。

1.3 基于模糊测试的放大漏洞挖掘

模糊测试在查找和解决软件程序中的安全问题方面发挥着关键作用。协议模糊测试相对于一般应用程序模糊测试有着其独特的挑战。

首先, 需要遵守严格的协议规范; 其次, 需要保证消息发送和接收的顺序, 以及上下文符合要求。这使得其测试过程变得复杂, 因为需要深入了解这些通信协议如何运行, 以及如何随协议状态变化。

2020 年, AFLNET^[10]在 AFL 的基础上实现了第一个有状态的灰盒协议模糊测试工具。AFLNET 捕获客户端与服务端通信时的流量作为初始种子, 在模糊测试过程中将状态码及代码覆盖率作为反馈信息, 保留有价值的种子对其进行进一步测试。Natella^[11]在 AFLNET 的基础上设计并实现了 StateAFL, 通过适用性更强且粒度更细的长生命周期变量识别网络协议的状态。Liu 等^[12]认为不是所有的协议状态都同等重要, 为此在 AFLNET 的基础上实现了 AFLNETLegion 算法, 用来对协议的状态进行评估。SnapFuzz^[13]为了提高模糊测试的效率, 将标准的网络套接字 (TCP 和 UDP) 替换为 UNIX 域套接字。NS-Fuzz^[14]通过使用轻量级的状态变量来表示更细粒度的状态, 同时降低插桩带来的开销, 提高模糊测试效率。

先前的协议模糊器旨在寻找导致协议服务程序崩溃的输入, 而较少考虑其他的错误类型。AmpFuzz 是第一个在基于 UDP 的网络服务中挖掘放大漏洞的系统性方法, 但其没有考虑协议状

态变化导致的放大漏洞, 并且其仅局限于 UDP。本文在 AFLNET 基础上实现了 AFLAMP, 通过协议状态引导模糊测试, 并使用基于会话的 BPA 来指导种子的选择, 从而提高放大漏洞的挖掘效果。

2 基于状态引导的放大漏洞挖掘框架

2.1 系统架构设计

为了实现自动化寻找放大漏洞, 本文提出基于状态引导的放大漏洞挖掘框架 AFLAMP。图 3 展示了 AFLAMP 的系统架构。

AFLAMP 主要由静态分析模块、插桩编译模块和模糊测试模块 3 部分组成。其中静态分析模块使用 LLVM 编译器将源代码转化为 LLVM 中间指令, 通过静态分析得到程序的终止分支位置信息, 以供编译时插桩使用。插桩编译模块将覆盖率反馈及程序运行状态感知的相关代码插入程序中, 最终得到被测协议服务程序。模糊测试模块负责在模糊测试循环中根据被测协议服务程序所反馈的覆盖率、状态和放大系数等信息构建状态模型, 并进行种子选择和变异, 从而发现放大漏洞。

2.2 静态分析模块

协议服务程序通常是 C/S 架构的程序, 在使用 C/C++ 等语言开发服务程序过程中会用到相应

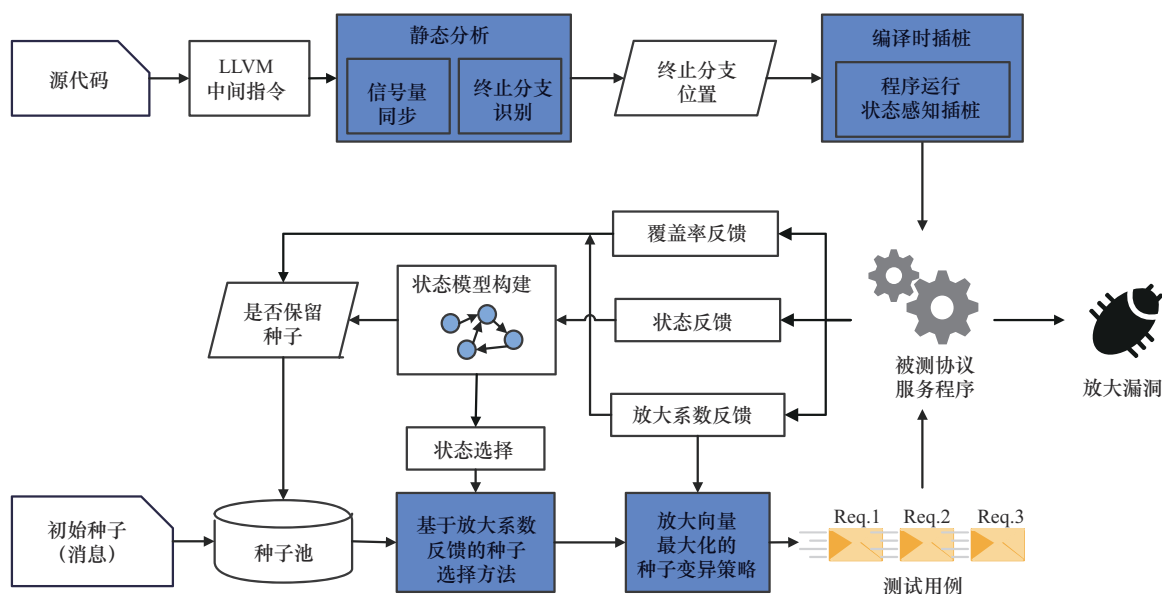


图3 AFLAMP 系统架构

Figure 3 AFLAMP system architecture

的网络 API 来创建和管理网络连接。以 TCP 为例，一个完整的网络通信过程包括：创建套接字、绑定套接字、监听连接、接收连接、读取和发送数据、关闭连接。当协议服务程序运行后，刚开始会阻塞在接收连接的状态中，等待来自客户端的连接，当客户端成功连接后，协议服务程序通常会阻塞在读取数据的状态中，直到接收到来自客户端的输入。如果协议服务程序尚未处在相应的阻塞状态中，而接收到来自客户端的请求，那么这些请求会被无情地丢弃。

在网络编程中，为了解决多路复用问题，开发人员会使用与 `select` 和 `poll` 函数相关的网络 API 来编写网络服务程序，其中 `select` 函数允许程序监视多个文件描述符，等待一个或多个文件描述符变为“就绪”状态，即它们有数据可读、可写或者有错误待处理。`poll` 函数是 `select` 函数的一个替代，它打破了文件描述符数量的限制。为了便于说明，本文将网络函数分为 Source、Sink 和 Block 3 类，如表 1 所示。①Source 类：所有具有接收数据包功能的函数。②Sink 类：所有具有发送数据包功能的函数。③Block 类：所有可以在等待接收数据时阻塞当前进程的函数，其中 `accept` 函数虽然并不用于等待接收数据，但仍然将其划分到 Block 类。

表 1 网络函数分类

Table 1 Network function classification

类别	网络函数
Source	<code>recv</code> , <code>recvfrom</code> , <code>recvmsg</code> , <code>recvmsg</code>
Sink	<code>send</code> , <code>sendfrom</code> , <code>sendmsg</code> , <code>sendmsg</code>
Block	<code>select</code> , <code>pselect</code> , <code>poll</code> , <code>ppoll</code> , <code>epoll_wait</code> , <code>epoll_pwait</code> , <code>accept</code>

2.2.1 基于共享内存的信号量同步机制

为了提高模糊测试的效率、减少服务程序被阻塞的时间，以及避免客户端的请求被协议服务程序丢弃，本文提出了基于共享内存的信号量同步机制，用来控制模糊器和被测服务程序之间的交互过程。模糊器在执行一个测试用例前，首先初始化共享内存，并将该共享内存与信号量绑定，而后使用环境变量将共享内存 ID 传递给被测协议服务程序，使用 `LD_PRELOAD` 机制来

HOOK 相关网络函数，以使被测协议服务程序可以执行自定义的代码。当被测协议服务程序执行到 Source 类或 Block 类函数时，此时会通过共享内存 ID 获取当前信号量的值，并使信号量增加，以便告知模糊器“我已准备完毕”。图 4 展示了模糊器与被测协议服务程序之间的信号量同步过程。

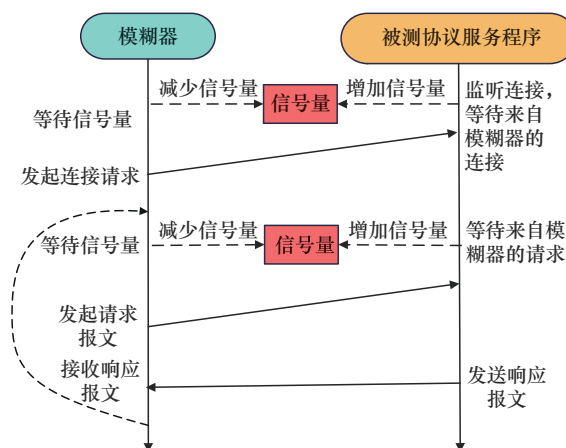


图 4 信号量同步过程

Figure 4 Semaphore synchronization process

当被测协议服务程序执行后，首先会处在监听状态，等待来自模糊器的连接，同时将共享内存中的信号量增加。模糊器在信号量增加后，便会向被测协议服务程序发起连接请求，同时消耗信号量。

2.2.2 基于污点分析的终止分支识别技术

一方面，被测协议服务程序在接收到请求后，会根据当前协议状态结合接收到的请求内容判断是否发送响应。无论发送与否，在这之后被测协议服务程序通常会执行日志记录或清理数据的操作，这些操作对于模糊测试来说没有意义。另一方面，对于放大漏洞的挖掘来说，本文更关注能到达 Sink 类函数的分支，对于无法到达 Sink 类函数的分支则应该主动提前终止，而不应该依赖于代价昂贵的超时，以此来提高模糊测试的执行效率。本文使用污点分析技术结合程序的控制流程图和函数调用图来识别这些终止分支。

由于在后续的插桩过程中，通常需要比较指令的位置信息来找到插桩位置，而指令的位置信息通常为类似于“`file:source.c,loc:222`”的字符串，但是字符串的比较往往慢于数值的比较。因

此, 为了便于后续插桩过程中更快地比较指令位置信息, 本文在静态分析过程中为每条中间指令分配唯一的 ID 值, 并将其存储在该指令的元数据中, 以便后续插桩时使用。

本文将所有调用 Source 类函数的基本块 (Block) 视为 source 点, 将所有调用 Sink 类函数的基本块视为 sink 点。为了找到终止分支, 首先需要确定哪些基本块可以在 source 点之后到达 sink 点, 并将这些基本块称为具有 sink 能力的基本块 (sink-capable block), sink 点本身也被视为具有 sink 能力的基本块。总体来说, 具有 sink 能力的基本块包括以下两类: ①调用 Sink 类函数的基本块; ②不调用 Source 类函数, 且至少有一个具有 sink 能力的基本块作为后继。从具有 sink 能力的基本块到不具有 sink 能力的基本块所组成的边为终止分支。

为了找到程序中具有 sink 能力的基本块, 需要遍历函数调用图的所有强连通分量 (strongly connected component, SCC), 而后遍历每个强连通分量中的每个函数和基本块。函数内的分析如图 5(a) 所示, 由于 Block_A 中调用了 recvfrom 函数, Block_E 调用了 send 函数, 所以 Block_A 为 source 点, Block_E 为 sink 点, 迭代遍历 Block_A 的所有后继节点, 若存在到达 sink 点的路径, 则将这些基本块标记为具有 sink 能力的基本块 (如 Block_A、

Block_B、Block_C、Block_D)。函数间的分析如图 5(b) 所示, Block_A 为 source 点, Block_H 为 sink 点, Block_C 调用了函数 B。在函数 B (Func_B) 中, 函数 B 的第一个 Block_F 可以达到 sink 点, 因此调用函数 B 的基本块都存在到达 sink 点的路径, 所以 Block_C 为具有 sink 能力的基本块。

在找到程序中具有 sink 能力的基本块后, 判断哪些分支为终止分支, 而后将终止分支的位置信息输出, 以供后续的插桩编译模块使用。图 6 展示了如何在一个典型的协议服务程序上识别终止分支的过程。

协议服务程序首先会执行一些初始化操作 (如初始化套接字), 而后主函数进入一个循环体, 持续等待来自客户端的连接, 并对客户端的每个请求调用相应的处理函数, 处理函数检查请求以确定是否发送响应, 最后经过终止分支识别后, 图 6 中的红色分支将被标记为终止分支。

2.3 插桩编译模块

在通过静态分析模块对目标协议服务程序的中间指令进行分析后, AFLAMP 的插桩编译模块负责对目标代码进行进一步的处理, 并生成最终的被测协议服务程序供模糊器使用。插桩编译模块将根据静态分析所生成的终止分支位置信息对目标进行轻量级插桩, 最终得到具有程序运行

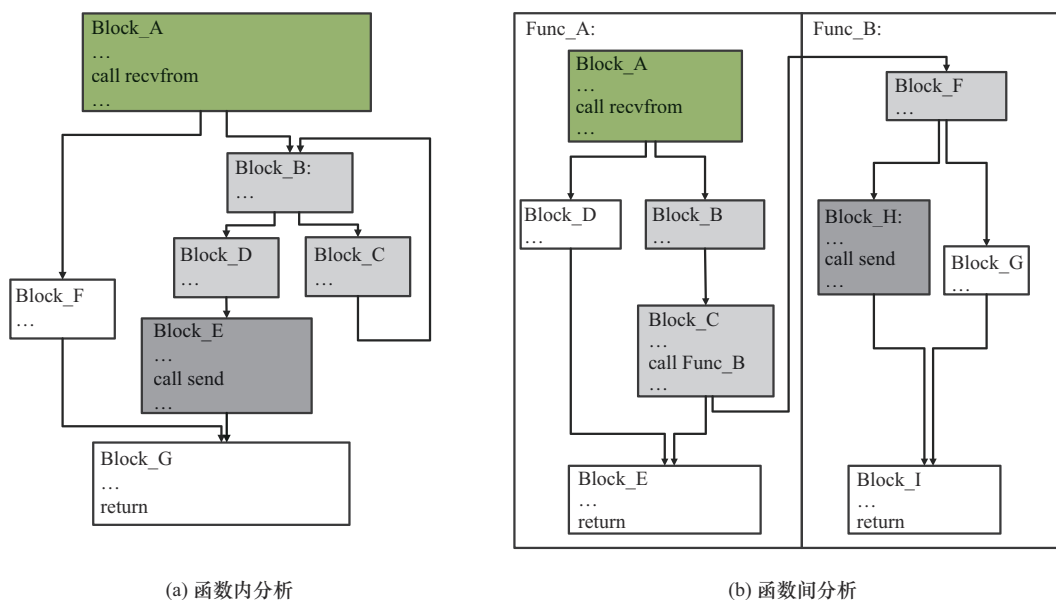


图 5 污点分析过程

Figure 5 Taint analysis process

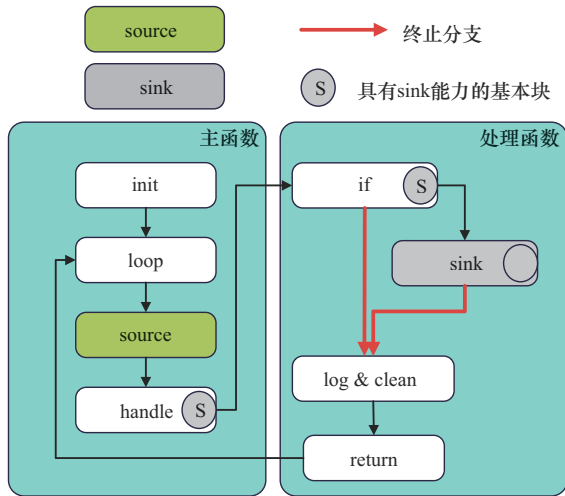


图6 基于污点分析的终止分支识别过程
Figure 6 Termination branch recognition process
based on stain analysis

状态感知能力的被测协议服务程序。插桩编译模块的工作流程如图7所示。

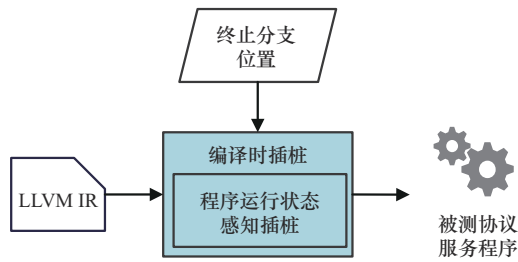


图7 插桩编译模块的工作流程
Figure 7 Instrumentation compilation module workflow

为了实现被测协议服务程序的运行状态感知能力，插桩编译模块根据程序的终止分支位置信息将额外的 `terminate` 代码注入相应的终止分支上。算法1描述了程序运行状态感知插桩的过程。

算法1 程序运行状态感知插桩算法

输入 Source, Loc

输出 null

```

1) ins_id ← 0;
2) for each Function F in Source do
3)   for each BasicBlock BB in F do
4)     cur_id ← BB.front().getMetadata();
5)     for each Successor S in successors(BB)
do
6)       next_id ← S.front().getMetadata();

```

```

7)       if tuple(cur_id, next_id) == Loc then
8)         Instrument call to terminate func-
tion
9)       end if
10)    end for
11)  end for
12) end for

```

在上述插桩算法中，插桩编译模块会遍历协议服务程序中的每一个基本块，获取基本块第一条指令的ID信息，而后遍历该基本块的所有后继节点，并获取所有后继节点的ID信息，将每条边与终止分支位置信息进行比较。当插桩编译模块通过这种方法找到终结分支所在的指令位置时，其会在该条指令后插入一条调用 `terminate` 函数的指令来完成插桩过程。在完成程序运行状态感知插桩后，得到的被测协议服务程序可以在处理完每个请求消息后决定是否结束当前线程，从而提高模糊测试效率。

2.4 模糊测试模块

在通过插桩编译模块对目标协议服务程序的中间指令进行插桩编译后，AFLAMP的模糊器就可以开始对插桩后的程序进行模糊测试。相比现有的灰盒模糊器，AFLAMP通过基于放大系数反馈的种子选择方法和放大向量最大化的种子变异策略实现了放大漏洞的挖掘。本节主要介绍模糊器中上述两个部分的设计方案。

2.4.1 基于会话的带宽放大系数

为了专注于基于状态转化的放大向量，必须进一步使模糊器能够衡量多个请求序列报文与多个响应报文大小之间的比率，为此本文在BAF的基础上提出基于会话的带宽放大系数 (BAFmsg)。在一次模糊测试过程中，假设模糊器向被测协议服务程序发送了 m 个报文，每个报文的大小为 $\text{len}(\text{Request payload}_i)$ ， i 表示第 i 个请求报文；模糊器接收到来自被测协议服务程序的 n 个报文，每个响应报文大小为 $\text{len}(\text{Response payload}_j)$ ， j 表示第 j 个响应报文，其计算方法如下。

$$\text{BAFmsg} = \frac{\sum_{j=0}^n \text{len}(\text{Response payload}_j)}{\sum_{i=0}^m \text{len}(\text{Request payload}_i)} \quad (1)$$

2.4.2 基于放大系数反馈的种子选择方法

AFLAMP 使用多种启发式方法来进行种子选择, 这些启发式方法根据构建的协议状态模型中相关的统计数据及被测协议服务程序所反馈的放大系数进行选择, 以帮助模糊器在指定的目标状态下选择更有可能使放大系数增大的种子。AFLAMP 的每个状态中记录了所有能到达该状态的种子, 在每个种子中记录了该种子所能触发的 BAFmsg。对于放大系数越大的种子, 在经过放大向量最大化的种子变异策略之后越有可能得到更大放大系数的放大向量, 应当给予这些种子更高的被选择到的概率。为此, 本文在 AFLNET 的种子选择方法的基础上, 引入了基于放大系数反馈的方法。遍历选定状态下的所有种子, 根据 BAFmsg 对种子进行递减排序, 计算种子下标与所有指定状态下种子数量的比值, 得到种子被选择的概率。与此同时, 为了专注于排序结果中前 20% 的种子, AFLAMP 参考 AFLNET 的启发式方法通过设置概率下限来避免经常选择到 BAFmsg 较小的种子。值得注意的是, 只有当模糊测试过程运行了足够长的时间来积累统计数据后, AFLAMP 才会开始应用这些启发式方法。一开始, 种子选择器使用循环调度的方法选择目标种子。算法 2 描述了基于放大系数反馈的种子选择算法。

算法 2 基于放大系数反馈的种子选择算法

输入 State

输出 Seed

```

1) ins_id ← 0;
2) seeds_count ← State.seeds_count;
3) if seeds_count > 10:
4)   Sort(State.BAFmsg)
5)   for each Seed  $S$  in State do
6)      $p \leftarrow (1 - \text{Index}(S) / \text{seeds\_count}) * 100$ ;
7)     if  $\text{UR}(100) < p \ \&\& \ p > 80$ :
8)       return  $S$ 
9)     end if
10)  end for
11) else
12)  Send ← State.seeds[idx++%seeds_count]
13)  return  $S$ 

```

2.4.3 放大向量最大化的种子变异策略

由 BAFmsg 的计算公式可知, 为了使 BAFmsg 增大, 一种方法是在保持响应报文的数量和大小不变的情况下缩减请求报文的数量或大小; 另一种方法是修改请求报文以增加响应报文的数量或大小。本文分析了常见的协议服务程序, 得到以下观察: 许多协议服务程序是通过指定的分隔符来获取请求内容的, 并且在命令结束后通常会有一段额外的可选字段。AFLAMP 使用一种简单且有效的种子变异策略来实现放大向量最大化, 即一旦模糊器找到一个 $\text{BAFmsg} > 1$ 的种子, 就通过从候选种子的结尾或指定分隔符的位置逐个删除字节来实现种子的变异。这对于放大向量最大化很有帮助, 可以帮助模糊器找到在当前状态下能够收到相同响应的最小请求。

举例来说, FTP 是一种常用的文件传输协议, FTP 的客户端与服务端进行通信时, 通常以 CRLF 作为行分隔符。当客户端发送 “HELP\r\n” 请求时, 服务端会响应相关的帮助信息, 此时 $\text{BAFmsg} > 1$, 那么在接下来的种子变异过程中会优先逐个删除分隔符之前的字节, 最终会发现 “H\r\n” 请求也能导致相同的响应, 从而使放大向量最大化。

3 实验验证

本文设置了实验来评估 AFLAMP, 以展示其在挖掘放大漏洞方面的能力, 以及基于状态引导的放大漏洞挖掘方法的优势。本文试图回答以下研究问题。

RQ1: AFLAMP 能否发现未知的放大漏洞/放大向量?

RQ2: 与现有的放大漏洞挖掘方法 AmpFuzz 相比, 本文所提方法 AFLAMP 效果如何?

为了回答 RQ1 和 RQ2, 本文在一组目标协议服务程序上 (这组目标协议服务程序中包括存在已知放大漏洞的服务程序) 运行 AFLAMP 和 AmpFuzz, 以评估两者的放大漏洞挖掘能力。

RQ3: 协议状态引导能否帮助发现更多的放大漏洞?

为此, 本文比较了 AFLAMP 在启用和未启用状态引导时所能发现的放大漏洞数量。

3.1 实验设置

由于AFLAMP是基于状态引导的模糊器，因此在选择实验目标协议服务程序时会优先考虑有状态的协议服务程序。基于UDP实现的协议服务程序通常被认为是无状态的，但也有例外。TFTP是一个基于UDP、有状态的文件传输协议。它通过内部的状态管理和简单的控制协议，确保文件可以在不可靠的UDP上实现可靠传输。Memcached是一个高性能的分布式内存对象缓存系统，在获取某个键值前，需要先设置其对应的数据，这种行为也可以认为是一种状态。

本文主要选取ProFuzzBench^[15]及AmpFuzz数据集中的部分程序作为测试目标。ProFuzzBench数据集中包含了基于不同传输层协议（TCP和UDP）的多种应用层协议服务程序，AmpFuzz数据集中包含已知的存在放大漏洞的协议，这样方便进行横向比较。目标协议服务程序的具体信息如表2所示。为了减少模糊测试结果中的噪声^[16]，每个目标重复模糊测试5次，并使用Tcpdump工具捕获模糊测试过程中的所有流量数据。所有实验在1台配置为Intel Core i7-12700H 2.30 GHz处理器和64 GB内存的服务器上进行。

表2 目标协议服务程序及版本

Table 2 Target protocol serve application and version

目标协议 服务程序	传输层 协议类型	协议类型	端口	程序版本 ^注
LightFTP	TCP	FTP	21	5980eal
OpenTFTP	UDP	TFTP	69	00921e7
OpenSLP	UDP	SLP	427	4bb7543
NTP	UDP	NTP	123	4.2.8p17
Memcached	UDP	Memcache	11 211	1.6.9
Dnsmasq	UDP	DNS	53	2.73rc6

注：程序版本使用版本号或git commit id表示。

3.2 放大漏洞挖掘能力评估

为了评估AFLAMP挖掘放大漏洞的能力，分别使用AFLAMP以及AmpFuzz对每个目标协议服务程序模糊测试24 h，重复运行5次。总体而言，AFLAMP在5个已知存在放大漏洞的协议服

务程序（OpenTFTP、OpenSLP、NTP、Memcached、Dnsmasq）中发现了11个放大漏洞，其中包括6个未知的放大漏洞，相较于AmpFuzz发现的8个放大漏洞，漏洞检出率提高了37.5%。此外，AFLAMP还在基于TCP的协议服务程序（LightFTP）中发现了12个放大漏洞。

3.2.1 实验结果

在模糊测试运行结束后，手动分析放大向量识别放大漏洞产生的原因，以找到不同的放大漏洞触发点，本文将这些不同的放大漏洞触发点视为不同的放大漏洞，漏洞列表及成因如表3所示。

在表3中，bug id表示在某个协议中发现的漏洞编号，amps表示放大向量的数量，max(BAF)表示最大的带宽放大系数，max(BAFmsg)表示最大的基于会话的带宽放大系数。与此同时，在best中列出了所有（5次）模糊测试结果中的最大值，mean表示结果的平均值，所有基于状态转化的漏洞为高亮显示。AmpFuzz只能发现单个请求导致的放大漏洞，其BAF与BAFmsg的结果相同，因此不再重复统计AmpFuzz的BAFmsg值。

本文主要根据响应报文的内容或者请求报文的的操作类型来区分不同的放大漏洞。例如，LightFTP中的#1和#2漏洞主要是根据响应报文的内容进行区分的。对于LightFTP#1漏洞，当请求中含有字符Q时，服务端会将其视为退出命令进行解析，进而回复“Goodbye”响应。对于LightFTP#2漏洞，当请求中包含字符串“SY”时，服务端会将其视为系统类型查询命令进行解析，进而回复“UNIX Type: L8”响应。

与此同时，为了进一步比较AFLAMP与AmpFuzz的放大漏洞挖掘能力，本文统计了两者的针对各个协议服务程序所挖掘到的放大漏洞数量(bugs)、放大向量的数量(amps)和最大带宽放大系数(maxBAF)，结果如表4所示。

从表4中可以看出，AFLAMP在6种目标协议服务程序上共发现了23个放大漏洞，其漏洞挖掘能力整体高于AmpFuzz，对于同一目标AFLAMP能够发现更多且放大系数更大的放大向量。

表3 漏洞列表及成因
Table 3 Vulnerability list and causes

目标协议 服务程序	bug id	AmpFuzz				AFLAMP						漏洞成因
		amps		max (BAF)		amps		max (BAF)		max (BAFmsg)		
		best	mean	best	mean	best	mean	best	mean	best	mean	
LightFTP	#1	—	—	—	—	40	29.60	15.66	12.37	15.66	12.37	221 Goodbye!
	#2	—	—	—	—	34	20.60	13.00	11.96	13.00	11.96	215 UNIX Type: L8
	#3	—	—	—	—	44	32.40	17.33	17.33	17.33	17.33	200 Command okay
	#4	—	—	—	—	12	8.20	8.62	7.77	8.62	7.77	530 Invalid user name or password
	#5	—	—	—	—	6	3.80	7.13	6.11	7.13	6.11	500 AUTH unsuccessful
	#6	—	—	—	—	37	28.40	50.33	42.78	50.33	42.78	211-Extensions supported
	#7	—	—	—	—	44	30.40	85.00	72.25	85.00	72.25	214-The following commands are recognized
	#8	—	—	—	—	2	2.00	16.3	9.59	7.68	5.37	226 Transfer complete
	#9	—	—	—	—	3	3.00	37.3	19.95	11	6.94	150 File status okay
	#10	—	—	—	—	4	4.00	14.66	9.33	7.42	5.03	250 Requested file action okay
	#11	—	—	—	—	3	3.00	4.8	4.80	5.76	5.76	550 Permission denied
	#12	—	—	—	—	4	4.00	6.88	6.88	5.76	5.76	350 REST supported
OpenTFTP	#1	5	5.00	21.40	21.40	12	12.00	51.60	51.60	51.60	51.60	Read file
	#2	7	7.00	3.00	3.00	45	45.00	6.00	6.00	6.00	6.00	No such file
	#3	—	—	—	—	22	22.00	9.00	9.00	4.40	4.40	invalid message
OpenSLP	#1	—	—	—	—	10	10.00	18.30	18.30	1.70	1.70	get service
NTP	#1	5	3.50	3.66	2.43	62	58.20	3.66	3.66	3.66	3.66	request nonce
	#2	16	15.00	33.66	33.66	148	98.00	33.66	33.66	33.66	33.66	read variables
Memcached	#1	20	15.00	2.00	2.00	95	45.00	3.00	2.78	3.00	2.78	Error message
	#2	6	4.50	34.80	22.60	8	4.50	51.20	32.61	51.20	32.61	get settings
	#3	8	4.10	67.52	45.60	56	30.88	128.6	72.88	128.6	72.88	get stats
	#4	—	—	—	—	5	5.00	3.00	2.80	1.50	1.43	get keys
Dnsmasq	#1	32	32.00	1.70	1.70	56	40.40	1.70	1.70	1.70	1.70	Standard query

注：“—”表示未发现相关漏洞。

表4 AmpFuzz与AFLAMP模糊测试结果
Table 4 Fuzz testing results of AmpFuzz and AFLAMP

目标协议 服务程序	AmpFuzz			AFLAMP		
	bugs	amps	maxBAF	bugs	amps	maxBAF
		best	best		best	best
LightFTP	—	—	—	12	187	85.00
OpenTFTP	2	12	21.40	3	67	51.60
OpenSLP	—	—	—	1	10	18.30
NTP	2	24	33.66	2	180	33.66
Memcached	3	24	67.52	4	137	128.6
Dnsmasq	1	32	1.70	1	56	1.70

注：“—”表示未发现相关漏洞。

3.2.2 漏洞案例分析

SLP（service location protocol）是一种网络服务发现协议，允许计算机和其他设备在局域网

中找到所需的网络服务。其最近曝光的 CVE-2023-29552 漏洞，允许未经身份验证的攻击者向 SLP 协议服务程序注册任意服务，而后当攻击者向 SLP 服务器发送获取默认服务的请求时，就会触发流量放大攻击。

在表3展示的实验结果中，AFLAMP能够发现相同的放大漏洞 OpenSLP#1，而 AmpFuzz 无法发现该漏洞。这主要是因为该漏洞的发现需要先进行服务的注册，从而使服务列表的内容增加，而后向服务器请求默认服务列表，此时服务器会返回所有已注册的服务信息，若不进行服务注册，只发送获取服务列表的请求，则无法发现此放大漏洞。OpenSLP#1 流量放大过程如图8所示。这说明 AFLAMP 在发现基于状态转化的放大漏洞方面具有优势。

Protocol	Length	Info
SRVLOC	618	Service Registration, V2 XID - 57706
SRVLOC	60	Service Acknowledge, V2 XID - 57706
SRVLOC	71	Service Type Request, V2 XID - 33103
SRVLOC	573	Service Type Reply, V2 XID - 33103
SRVLOC	71	Service Type Request, V2 XID - 11880
SRVLOC	573	Service Type Reply, V2 XID - 11880

bits) on interface 1	0000	00 00 00 00 00 00 00 00	00 00 00 00 08 00 45 00
00:00:00_00:00:00	0010	00 39 32 fe 40 00 40 11	09 b4 7f 00 00 01 7f 00
	0020	00 01 a6 e5 01 ab 00 25	fe 38 02 09 00 00 1d 00
	0030	00 00 00 00 81 4f 00 02	65 6e 00 00 ff ff 00 07
	0040	64 65 66 61 75 6c 74	

图8 OpenSLP#1 流量放大示意图
Figure 8 OpenSLP#1 traffic amplification schematic

通过深入分析该放大漏洞的成因发现，在进行服务注册时可自定义服务 URL 的长度及服务的有效期，因此可以不断地向服务端注册服务直到达到服务的最大缓存，经过测试，该漏洞所导致的最大放大系数可以达到 2 235。

Memcached 是一种广泛使用的高性能分布式内存对象缓存系统。根据公开报道^[7]，攻击者可以利用该协议服务程序发起核弹级的 ADDoS 攻击，其最大放大系数高达 50 000，这也是目前已知的放大系数最大的协议。Memcached 放大漏洞形成的原因主要是当攻击者请求服务器上的缓存数据时，服务器会不经验证地将已有的缓存数据返回，从而导致放大。在实验结果中，AFLAMP 发现了此放大漏洞（Memcached#6），而 AmpFuzz 未发现，这主要得益于 AFLAMP 采用了基于状态引导的模糊测试方法，可以进一步挖掘基于状态转化的放大漏洞。

除此之外，AFLAMP 还发现了 5 个未知的放大漏洞，其中 Memcached#4 漏洞的最大放大系数可以达到 128.6，而 AmpFuzz 所发现的 Memcached#4 漏洞的最大放大系数仅有 67.52。通过对比两者的放大向量，如图 9 所示，可以看到 AmpFuzz 的放大向量多了一段 8 字节的填充，而 AFLAMP 可以通过放大向量最大化的种子变异策略，逐个删除分隔符之前的字符，从而找到最大的放大向量。

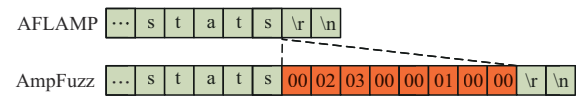


图9 AFLAMP 与 AmpFuzz 关于 Memcached#4 放大向量对比
Figure 9 Comparison of amplification vectors for Memcached#4 between AFLAMP and AmpFuzz

除了上述基于 UDP 的协议服务程序外，AFLAMP 还扩展了对基于 TCP 的协议服务程序

的支持。在 LightFTP 中，AFLAMP 发现了 12 个放大漏洞，其中漏洞 LightFTP#8~#12 需要先进行登录，才能被进一步触发，其流量放大过程如图 10 所示。

Length	Info
99	Response: 220 LightFTP server v2.0a ready
74	Request: USER a
100	Response: 331 User a OK. Password required
74	Request: PASS a
96	Response: 230 User logged in, proceed.
83	Request: A\000@dS >@AP dS\002a
115	Response: 226 Transfer complete. Closing data
99	Response: 220 LightFTP server v2.0a ready
74	Request: USER a
100	Response: 331 User a OK. Password required
74	Request: PASS a
96	Response: 230 User logged in, proceed.
109	Request: 1\000\000\000\177\003
120	Response: 150 File status okay; about to open
124	Response: 451 Requested action aborted. Local

图10 LightFTP 流量放大过程
Figure 10 LightFTP traffic amplification process

通过对实验结果的分析，可以发现许多协议产生放大漏洞的原因并不唯一，也就是说服务端会放大多种请求类型的流量。这进一步凸显了使用系统化方法挖掘放大漏洞的必要性，因为仅阻止固定类型请求的简单防御是不完整的，攻击者仍可以使用不同的放大向量发起攻击。

3.3 消融实验

为了评估基于状态引导的放大漏洞挖掘方法能否帮助发现更多的放大漏洞，本文比较了 AFLAMP 在启用状态引导和未启用状态引导时所能发现的放大漏洞数量，结果如表 5 所示。

表5 AFLAMP 消融实验结果			
Table 5 Results of AFLAMP ablation experiment			
目标协议 服务程序	AFLAMP 放大漏洞数量	AFLAMP w/o SA 放大漏洞数量	漏洞检出 提高率
LightFTP	12	7	71.43%
OpenTFTP	3	2	50.00%
OpenSLP	1	0	100%
NTP	2	2	0
Memcached	4	3	33.33%
Dnsmasq	1	1	0
总计	23	15	53.33%

注：“AFLAMP w/o SA”表示未启用状态引导的方法。

从表 5 展示的实验结果来看，启用状态引导的方法可以发现更多的放大漏洞，整体的漏洞检

出率提高了 53.33%，说明基于状态引导的放大漏洞挖掘方法能帮助模糊器挖掘更多的放大漏洞。对于 NTP 和 Dnsmasq 协议服务程序，无论是否开启状态引导的方法，其发现放大漏洞的数量相同。这主要是因为这几个协议服务程序的状态比较单一，在通常情况下不将其认为是具有状态协议。LightFTP 的协议实现较复杂，具有较多的协议状态，因此在启用状态引导方法后能发现更多的放大漏洞。

由于 AFLAMP 是基于 AFLNET 实现的，因此在状态表征上仍采用传统的基于响应码的方案。例如，在 LightFTP 中，响应码 220 表示服务处在正常运行状态，响应码 230 表示用户登录成功，这种简单的状态表征方法对于部分采用响应码的协议有较好的效果，但也存在局限性。例如，RTSP 无论服务处在何种状态，只要请求的格式正确，服务器便会返回状态码 200。因此，探索新的状态表征方法，在状态表征上进行更细粒度的区分，寻找能更准确表征协议服务程序所运行的状态，可以帮助提高漏洞挖掘效果。

4 结束语

本文提出并实现了一种基于状态引导的放大漏洞挖掘框架 AFLAMP，将状态引导的方法应用于放大漏洞的挖掘，解决了现有放大漏洞挖掘方法中通用性不强、漏洞挖掘能力有限的问题。实验结果表明，本文方法具备较强的放大漏洞挖掘能力。

目前有许多关于协议状态表征的相关研究，如 NSFuzz 使用程序变量来表示协议状态，通过将协议状态进行更细粒度的区分，从而进一步提高状态覆盖率。在后续研究工作中，可以考虑探索新的协议状态表征方法来提高放大漏洞挖掘能力。

参考文献：

- [1] ISMAIL S, HASSEN H R, JUST M, et al. A review of amplification-based distributed denial of service attacks and their mitigation[J]. Computers & Security, 2021, 109: 102380.
- [2] KRUPP J, GRISHCHENKO I, ROSSOW C. AmpFuzz: fuzzing for amplification DDoS vulnerabilities[C]//Proceedings of the 31th USENIX Security Symposium (USENIX Security 2022), Berkeley: USENIX Association, 2022: 1043-1060.
- [3] ROSSOW C. Amplification hell: revisiting network protocols for DDoS abuse[C]//Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS 2014). Virginia: the Internet Society, 2014: 1-15.
- [4] CZYZ J, KALLITSIS M, GHARAIBEH M, et al. Taming the 800 pound gorilla: the rise and decline of NTP DDoS attacks[C]//Proceedings of the 2014 Conference on Internet Measurement Conference. New York: ACM Press, 2014: 435-448.
- [5] RIJSWIJK-DEIJ R V, SPEROTTO A, PRAS A. DNSSEC and its potential for DDoS attacks: a comprehensive measurement study[C]//Proceedings of the 2014 Conference on Internet Measurement Conference. New York: ACM Press, 2014: 449-460.
- [6] ADAMSKY F, KHAYAM S A, JAGER R, et al. P2P file-sharing in hell: exploiting BitTorrent vulnerabilities to launch distributed reflective DoS attacks[C]//Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT 15). Berkeley: USENIX Association, 2015.
- [7] XU W, LI X, LU C, et al. TsuKing: coordinating DNS resolvers and queries into potent DoS amplifiers[C]//Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2023: 311-325.
- [8] KÜHRER M, HUPPERICH T, ROSSOW C, et al. Hell of a handshake: abusing TCP for reflective amplification DDoS attacks[C]//Proceedings of the 8th USENIX Workshop on Offensive Technologies (WOOT 14). Berkeley: USENIX Association, 2014.
- [9] BOCK K, ALARAJ A, FAX Y, et al. Weaponizing middleboxes for TCP reflected amplification[C]//Proceedings of the 30th USENIX Security Symposium (USENIX Security 21). Berkeley: USENIX Association, 2021: 3345-3361.
- [10] PHAM V T, BOHME M, ROYCHOUDHURY A. AFLNET: a grey-box fuzzer for network protocols[C]//Proceedings of the 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST). Piscataway: IEEE Press, 2020: 460-465.
- [11] NATELLA R. Stateafl: greybox fuzzing for stateful network servers[J]. Empirical Software Engineering, 2022, 27(7): 191.
- [12] LIU D, PHAM V T, ERNST G, et al. State selection algorithms and their impact on the performance of stateful network protocol fuzzing[C]//Proceedings of the 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Piscataway: IEEE Press, 2022: 720-730.
- [13] ANDRONIDIS A, CADAR C. SnapFuzz: high-throughput fuzzing of network applications[C]//Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis. New York: ACM Press, 2022: 340-351.
- [14] QIN S, HU F, MA Z, et al. NSFuzz: towards efficient and state-aware network service fuzzing[J]. ACM Transactions on Software Engineering and Methodology, 2023, 32(6): 1-26.
- [15] NATELLA R, PHAM V T. Profuzzbench: a benchmark for stateful protocol fuzzing[C]//Proceedings of the 30th ACM SIGSOFT Inter-

national Symposium on Software Testing and Analysis. New York: ACM Press, 2021: 662-665.

- [16] KLEES G, RUEF A, COOPER B, et al. Evaluating fuzz testing[C]// Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York: ACM Press, 2018: 2123-2138.

[作者简介]



蒋思康（1997-），男，浙江义乌人，信息工程大学硕士生，主要研究方向为模糊测试与漏洞挖掘。



陈鸿羽（2001-），男，辽宁辽中人，信息工程大学博士生，主要研究方向为模糊测试与漏洞挖掘。



蔡瑞杰（1990-），男，河南开封人，信息工程大学讲师，主要研究方向为网络安全。



刘胜利（1973-），男，河南周口人，信息工程大学教授、博士生导师，主要研究方向为网络设备安全和网络攻击检测。



尹小康（1993-），男，河南周口人，信息工程大学讲师，主要研究方向为网络安全、二进制代码分析和机器学习。