拟态路由器 TCP 代理设计实现与形式化验证研究

金希文[1,2], 葛 强[1,2], 张 进[2], 丁 建[2], 江逸茗[2,3], 马海龙[2,3], 伊 鹏[2,3]

[1] 东南大学 网络空间安全学院 南京 中国 211100
[2] 紫金山实验室 内生安全研究中心 南京 中国 211100
[3] 中国人民解放军信息工程大学 郑州 中国 450000

摘要 拟态路由器基于拟态防御的动态异构冗余架构进行设计, 对于未知漏洞后门具有良好的防御能力。协议代理在拟态路由器中处于内外联络的枢纽位置, 协议代理的安全性和功能正确性对于拟态路由器有着重要意义。本文设计实现了拟态路由器的 TCP 协议代理, 并采用形式化方法, 对其安全性和功能正确性进行了验证。TCP 协议代理嗅探邻居和主执行体之间的 TCP 报文, 模拟邻居和从执行体建立 TCP 连接, 并向上层应用层协议代理提供程序接口。基于分离逻辑与组合思想, 采用 Verifast 定理证明器, 对 TCP 协议代理的低级属性, 包括指针安全使用、无内存泄露、无死代码等, 进行了验证; 同时, 还对 TCP 协议代理的各主要功能模块的部分高级属性进行了形式化验证。搭建了包含 3 个执行体的拟态路由器实验环境, 对实现结果进行了实际测试, 结果表明所实现的 TCP 代理实现了预期功能。TCP 协议代理实现总计 1611 行 C 代码, 其中形式化验证所需人工引导定理检查器书写的证明共计 588 行。实际开发过程中, 书写代码实现与书写人工证明所需的时间约为 1∶1。本文对 TCP 协议代理的实现与形式化验证工作证明了将形式化验证引入拟态路由器的关键组件开发中是确实可行的, 且证明代价可以接受。

关键词 拟态防御; 形式化验证
中图法分类号 TP393 DOI 号 10.19363/J.cnki.cn10-1380/tn.2023.09.01

Design, Implementation and Formal Verification of TCP Proxy in Mimic Defense Router

JIN Xiwen[1,2], GE Qiang[1,2], ZHANG Jin[2], DING Jian[2], JIANG Yiming[2,3], MA Hailong[2,3], YI Peng[2,3]

[1] School of Cyber Science and Engineering, Southeast University, Nanjing 211100, China
[2] Endogenous Security Research Center, Purple Moutain Laboratories, Nanjing 211100, China
[3] Information Engineering University, Zhengzhou 450000, China

Abstract The mimic defense router is designed based on the dynamic heterogeneous redundant architecture of mimic defense, which has good defense capability for unknown vulnerability backdoor. The protocol proxy is in the pivotal position of internal and external communication in the mimic defense router, and the security and functional correctness of the protocol proxy are of great importance to the mimic defense router. In this paper, we design and implement a TCP protocol proxy for the mimic defense router, and verify its security and functional correctness using a formal approach. The TCP protocol proxy sniffs TCP messages between neighbors and master entities, simulates the establishment of TCP connections between neighbors and slave entities, and provides a programmatic interface to the upper layer application layer protocol proxy. Based on the idea of separation logic and combination, the Verifast theorem provers are used to verify the low-level properties of the TCP protocol proxy, including the safe use of pointers, no memory leakage, and no dead code, etc. Also, some high-level properties of each major functional module of the TCP protocol proxy are formally verified. A mimic defense router experimental environment containing three entities was built and the results of the implementation were tested in practice, and the results showed that the implemented TCP proxy achieved the expected functionality. The TCP protocol proxy implementation totals 1611 lines of C code, of which a total of 588 lines of proofs written by a manually guided theorem checker are required for formal verification. In practice, the time required to write the code implementation and the manual proof is about 1∶1. The implementation and formal verification work of the TCP protocol proxy in this paper demonstrates that it is indeed feasible to introduce formal verification into the development of key components of mimic defense routers, and that the cost is provably acceptable.

Key words mimic defense; formal verification

通讯作者: 张进, 博士, 高级工程师, Email: zhangjin@pmlabs.com.cn。
本课题得到紫金山实验室自立课题“面向私有云的内生安全多模态基础网络”(No. ZL042301)资助。
收稿日期: 2022-02-15; 修改日期: 2022-05-03; 定稿日期: 2023-06-12

1 引言

拟态防御思想通过动态异构冗余(Dynamic Heterogeneous Redundancy, DHR)结构, 将相同的功能由多个异构的执行体并行执行, 并通过表决器对多个执行体的输出进行比对, 从而发现并修复由于随机故障或者人为攻击导致的系统异常, 提高了整个系统的可靠性与安全性[1]。拟态防御的 DHR 架构如图 1 所示。

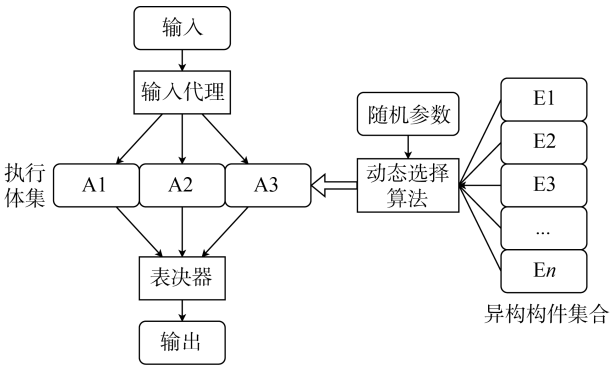

图 1 DHR 结构

Figure 1 DHR structure

输入代理、表决器、调度器等统称“拟态括号”。作为系统的基本功能组件, 拟态括号本身的安全性对于保证系统的整体安全尤为关键。传统的软件测试技术无法找到拟态括号实现中所有的程序缺陷和安全漏洞。本课题的基本动机是在拟态括号组件的开发中使用形式化验证技术, 得到经过形式化验证的拟态括号组件, 从而为设计实现高安全的拟态防御路由器等网络设备提供保障。

与其他软件领域的形式化验证工作相同, 在对拟态构造路由器的输入代理组件进行形式化验证时也会面临开发效率慢、验证所需成本高等问题。

本文基于符号执行、霍尔逻辑与分离逻辑, 提出了一种在拟态构造路由器中更好地运用形式化验证技术的方法, 并将这种方法成功应用于拟态构造路由器中 TCP 代理的设计、实现与形式化验证中。本文所使用的的形式化验证方法不会过多地影响实现时的开发效率, 同时验证阶段所需的人工成本也在可接受范围内。

拟态构造路由器[2]根据拟态防御思想, 将输入的报文由输入代理分配给不同的执行体, 由表决器模块进行表决, 最后输出报文。其中的 TCP 代理是输入代理的重要组件, 为多种应用层协议代理提供支撑, 因此, TCP 代理的安全性和正确性对于保证拟态路由器的安全性而言至关重要。

本文基于拟态路由器 DHR2 架构[2], 设计实现了 TCP 协议代理。如图 2, 在 DHR2 架构中, 执行体分为主执行体和从执行体两类。TCP 代理通过嗅探邻居节点和主执行体之间的 TCP 报文, 模拟邻居节点和从执行体建立 TCP 连接, 并向各类依赖 TCP 协议的应用层协议代理提供服务接口。

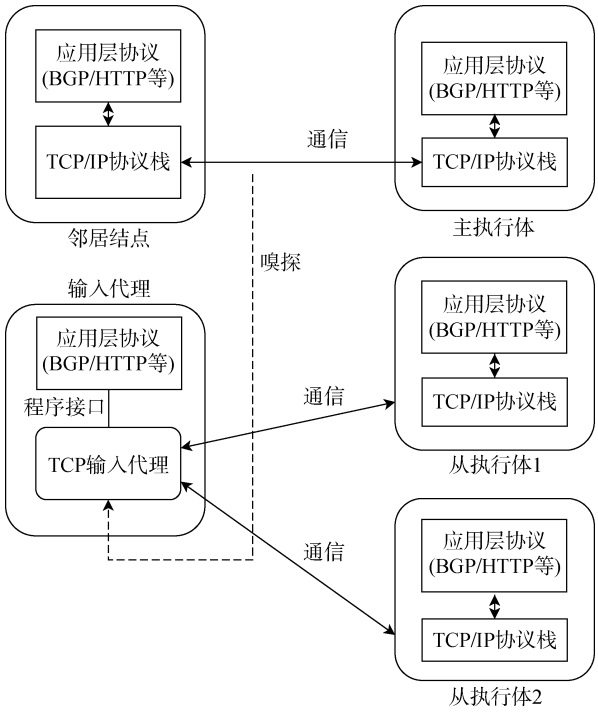

图 2 TCP 代理与执行体的结构

Figure 2 Structure of TCP proxy and entities

具体而言, TCP 代理实现了以下功能:

(1) 嗅探主执行体与邻居间的 TCP 报文, 处理分析后向上传递给应用层输入代理。

(2) 内部维护与从执行体的 TCP 连接状态信息, 并向上提供字节流接口, 与从执行体进行 TCP 通信。

本文将 TCP 协议代理的功能设计成低耦合的不同模块, 并基于 C 语言将其实现, 为每个模块都书写了形式化规约, 并使用形式化验证工具对模块进行形式化验证, 最后在搭建了测试环境测试了 TCP 协议代理的功能。

TCP 协议代理的实现和形式化验证为基于 TCP 的应用层协议代理的实现和形式化验证打下了基础。向上层提供的应用程序接口和形式化规约也方便了应用层输入代理的开发和验证, 为整个“拟态括号”的形式化验证提供了支撑。

2 相关工作

2.1 形式化验证理论与方法

形式化验证是指通过数理逻辑等数学方法, 证

明一段程序或一个系统是否能满足形式化规约描述的某个或某些属性。形式化验证工具主要可以分为 3 类: 模型检查、交互式定理证明器和基于可满足性模理论(Satisfiability Modulo Theory, SMT)的半自动化证明工具。

模型检查在一个有穷状态的系统上执行搜索, 如果这个系统的每种状态都符合搜索时设定的规约, 则这个有穷系统被验证符合这个规约。模型检查主要被用在协议验证、计算机硬件等方面。J Zhang 等[3]研究人员在 2021 年使用模型检查对 QUIC 协议进行了形式化分析。在软件领域, 由于大部分程序的状态无穷性, 导致模型检查工具并不能很好地运用在这些领域。

交互式定理证明器主要有 Coq[4]、Isabelle/HOL[5]和 Lean[6]。交互式定理证明器通过人工引导完成程序到形式化规约的正确性证明, 并用机器检查证明。虽然交互式定理证明器的表现力很强, 可以用来描述许多定理和属性, 但是, 证明这些定理和属性需要的人工工作十分繁琐, 需要大量的工作量[7-8]。

Verifast[9]、Dafny[10]等基于 SMT 的半自动化形式化验证工具使用 SMT 求解器对程序进行检查。SMT 求解器可以直接求解一些命题逻辑公式是否成立, 如果不成立, 还可以给出反例。这些基于 SMT 求解器的形式化验证工具在证明过程在仍然需要证明人员书写循环不变量、引理等注释帮助形式化验证工具进行定理证明, 但这些人工引导的工作比起交互式定理证明器少得多。R Pea 在 2020 年使用 Dafny 工具给出了红黑树的形式化规约和证明[11]。本文采用 Verifast 对拟态路由器的 TCP 代理进行形式化验证, 因此从方法分类角度, 属于基于 SMT 的半自动化证明。

2.2 网络功能的形式化验证

云计算的快速发展导致越来越多的网络功能(Network Function, NF)采用软件实现。虽然用软件实现 NF 在灵活性、实现代价方面具有优势, 但是随之而来也引入了软件缺陷和安全漏洞等问题。在实际生产环境中, 这些问题如果导致路由器等 NF 设备行为出错, 后果十分严重。因此, 通过形式化方法, 验证 NF 的安全性和功能正确性, 也逐渐成为当前的研究热点。

Yousefi 等人[12]在 2020 年形式化验证了网络功能设备的存活性。Abhashkumar 等[13]研究人员开发的工具 Tiramisu 可以快速验证网络中的控制平面。Yuan 等[14]研究人员开发的符号化模型检查器 NetSMC 能够更快速地验证有状态的网络功能设备。Beckett 等[15]研究人员为网络建模提出了一种新的中间表示语言。Yifan Li 等[16]研究人推出的通用数据平面验证框架 Mahjong 用模块化的方式重构了三种经典验证方法。Bingchuan Tian 等人[17]为阿里巴巴的可编程数据平面实现了第一个实际可用的验证系统 Aquila。Venkat 等人[18]开发能验证拥塞控制算法的某些属性的形式化验证工具。

Zaostrovnykh 等[19]研究人员们在 2019 年推出了能够自动验证 NF 产品的形式化验证工具 Vigor。开发人员使用 Vigor 开发并验证 NF 产品时, 只需要提供 C 语言的实现和 Python 语言描述的属性规范即可。Vigor 通过将 NF 开发过程中需要调用的数据结构预先实现为第三方库并手工形式化验证实现了整个 NF 产品的“一键验证”。因此, 使用 Vigor 开发时需要遵循 Vigor 提供的接口函数进行开发, 丧失了一定的灵活性。TCP 协议代理需要嗅探网卡上的数据并解析其中的 TCP 报文, 并直接通过网卡发送自己构造的 TCP 报文, 跨越了计算机网络中从数据链路层到应用层的多层报文。而 Vigor 提供的应用程序接口严格定义了 NF 产品位于计算机网络中的第几层, 如果用 Vigor 将 TCP 代理定义为数据链路层, 则无法获得解析 TCP 报文的能力, 如果将 TCP 代理定义为运输层, 则又无法获得直接操作网卡的能力。因此, Vigor 不能满足对 TCP 代理进行开发和形式化验证的需求。

本文使用 C 语言开发 TCP 协议代理, 并使用基于 SMT 求解器的形式化验证工具 Verifast 对程序进行验证, 在开发的灵活性和证明所需的人力上取得最大程度的平衡。

2.3 拟态设备与系统的形式化验证

拟态防御领域中已经存在一些形式化相关的工作。张兴明等[20]研究人员在 2018 年给出了拟态防御的马尔可夫博弈模型, 朱维军等[21]研究人员在 2019 年给出了动态异构冗余结构的拟态防御自动机模型, 王婷等[22]研究人员在 2020 年使用时间自动机模型证明了动态异构冗余结构的正确性和有效性。这些工作都给出了拟态防御理论动态异构冗余架构的特定模型, 形式化地描述了拟态防御的生效过程。然而, 目前缺少对拟态防御设备或系统的具体实现结果的形式化验证工作。特别是协议代理、裁决器、调度器等关键组件, 通过形式化方法, 验证其安全性和功能正确性, 对于提升拟态防御设备系统的整体安全能力具有重要意义。本文在此方面做了初步的研究探索。

3 设计

3.1 总体设计

拟态构造路由器中的 TCP 协议代理需要实现以下两个功能:

(1) 嗅探主执行体与邻居间的TCP报文, 处理分析后向上传递给应用层输入代理。

(2) 内部维护与从执行体的 TCP 连接状态信息, 并向上提供字节流接口, 与从执行体进行 TCP 通信。

如图 3 所示, 本文将第一个功能中的嗅探部分用嗅探模块实现; 第一个功能中的处理部分用报文解析模块实现; 第一个功能中的分析、向上传递部分用邻主连接管理模块实现; 第二个功能用类套接字模块实现。

TCP 代理内部的功能通过相互独立的多个模块实现, 模块之间通过函数调用相互耦合。

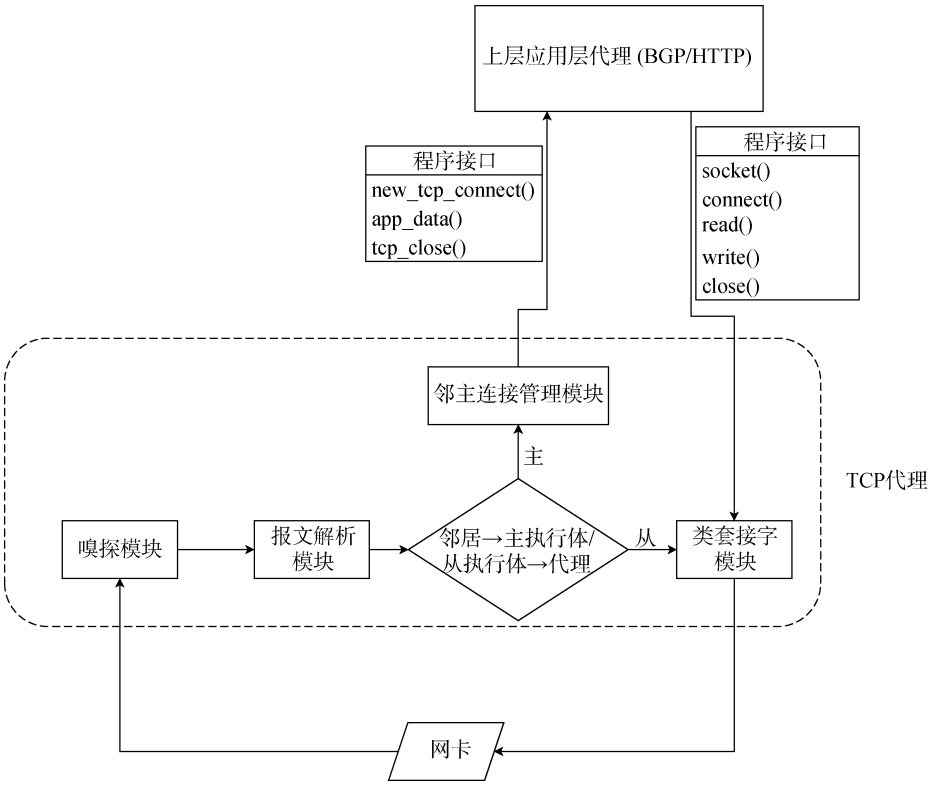

图 3 总体设计

Figure 3 Overall design

嗅探模块嗅探网络中的流量, 既可以嗅探到 TCP 邻居与主执行体之间的报文, 也可以嗅探到从执行体发送给 TCP 协议代理的报文。

报文解析模块将嗅探到的 TCP 报文解析成程序内部使用的 C 语言结构体, 方便后续模块使用相关 TCP 头部字段。

邻主连接管理模块管理“邻居-主执行体”之间交互的数据, 根据内部保存的状态分析当前 TCP 报文处于建立连接、交换数据、断开连接等 TCP 状态的哪个状态。因为邻居与主执行体之间可能存在超时重传等导致报文乱序的现象, 该模块也会对乱序的 TCP 报文进行重新排列。邻主连接管理模块实现了“邻居-主执行体”这条 TCP 连接上的相关程序接口。

类套接字模块完成与从执行体的数据交互工作。拟态防御中的输入代理需要模拟邻居与从执行体交换数据, 这一步骤需要输入代理修改报文中的 TCP 字段, 与从执行体建立连接、交换数据、断开连接, 管理 TCP 连接时的序列号等信息。类套接字实现了这些功能, 并向上层应用层输入代理提供了类似 socket 套接字的程序接口, 方便上层输入代理的开发。

3.2 嗅探模块

嗅探模块嗅探网卡中的报文数据, 得到一个字符串, 传递给报文解析模块使用。嗅探模块基于 pcap 库实现, 设置要捕获的网卡和捕获规则, 即可捕获网卡上对应的报文。

如图4, 嗅探模块同时也是TCP协议代理程序本身的主循环, 在事件驱动模型下, 每嗅探到网卡中的一个报文就驱动程序本身的一次循环, 完成对这个报文的操作, 并进入下一次嗅探。

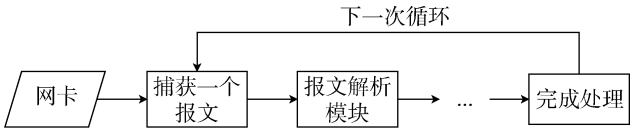

图 4　嗅探模块主循环

Figure 4　Main loop of the sniffer module

3.3　报文解析模块

报文解析模块的输入是嗅探模块输出的字符串, 输出是解析后得到的结构体, 方便程序后续处理。

如图 5, 报文解析模块在解析时根据 TCP 头部结构, 对嗅探得到的字符串指针做偏移操作, 得到 TCP 报文中每个字段的值, 并填入结构体的字段中。

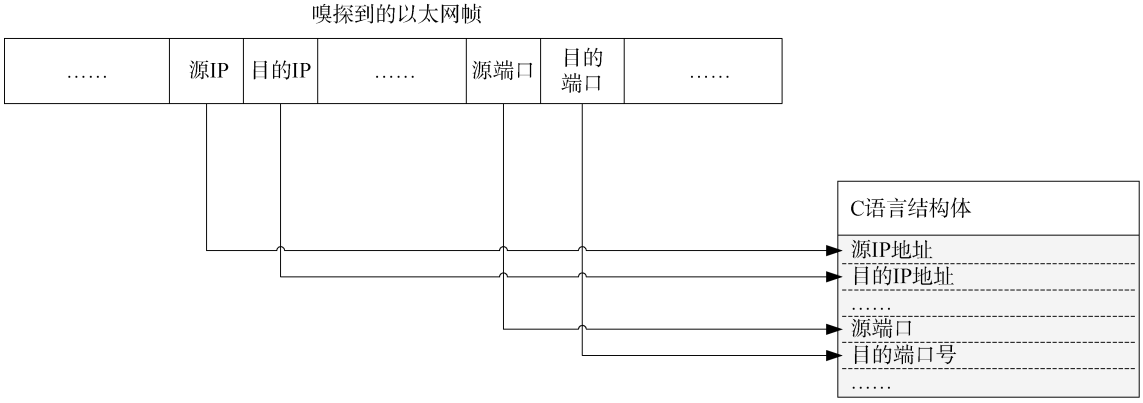

图 5　解析字符串

Figure 5　Parsing strings

3.4　邻主连接管理模块

邻主连接管理模块的输入是报文解析模块解析得到的结构体, 输出是一些预定义好的操作码。程序主循环通过这些状态码调用相应的程序接口将报文向上传递给上层应用层代理。

表 1　邻主连接管理模块操作码

Table 1　Operation code of neighbor-master connection mnodule

操作码	解释	与上层应用层代理的接口
0	不做处理	
1	邻居-主执行体已完成 TCP 建立连接, 向上传递该信息	new_tcp_connect()
2	邻居→主执行体的数据, 向上传递该信息	app_data()
3	邻居-主执行体的 TCP 连接已断开, 向上传递该信息	tcp_close()

TCP 协议中用(源 IP, 目的 IP, 源端口号, 目的端口号)的四元组唯一标识一条 TCP 连接, 邻主连接管理模块中也使用这样的四元组唯一标识一条“邻居-主执行体”间的 TCP 通信。如图 6, 邻主连接管理模块内部使用链表保存“邻居-主执行体”间的每一条 TCP 连接的相关信息。每一个链表结点是一条 TCP 连接, 保存了 TCP 连接的状态、序列号、确认号等信息, 以及一个保存了应用层数据的链表。

保存应用层数据的链表用来处理“邻居-主执行体”间的乱序报文, 因为 TCP 协议代理嗅探“邻居-主执行体”间的报文, 所以无法通过重复 ACK 等手段引发邻居或主执行体的重传。如图 7, 当 TCP 连接中某个报文丢失时, 超前的报文要先在链表中缓存起来, 等待后续重传报文来“填补空洞”。

3.5　类套接字模块

类套接字模块实现了与从执行体之间的 TCP 连接管理, 并向上层应用层输入代理提供程序接口, 方便上层输入代理向执行体分发数据。

在拟态防御中, 输入代理需要模拟客户端向从执行体建立连接、发送数据。在模拟客户端时, TCP 四元组的源 IP、源端口号有两种填写方法:

(1) 使用输入代理自身的源 IP, 源端口号使用随机的不重复值。如果使用系统套接字接口向从执行体建立连接, 则只能使用这种方法。

(2) 源 IP 和源端口号都是用监听到的邻居的值。这种方法需要重写 TCP 报文头中的字段, 所以只能由输入代理构造 TCP 报文, 用最底层的网卡发包机制实现。

虽然计算机网络中 TCP/IP 层对应用层做了封装, 但许多协议还是用到了 TCP/IP 层中的字段。比如 BGP 协议用 IP 地址唯一地标识每个 BGP 路由器结点, HTTP 服务器也会根据 IP 地址做一些防 DDoS 攻击的处理。所以, 在类套接字模块中, 我们使用了第 2 种方法来模拟邻居和从执行体通信。类套接字模块完整的实现了从链路层-IP 层-TCP 层的协议栈, 并对上层应用层输入代理提供了类套接字接口。

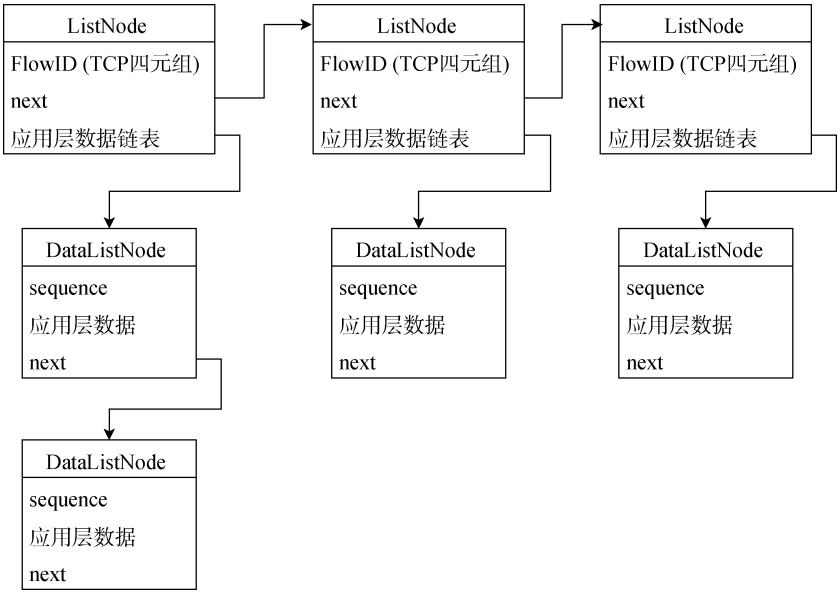

图 6　邻主连接管理模块链表

Figure 6　Linked list of neighbor-master connection management module

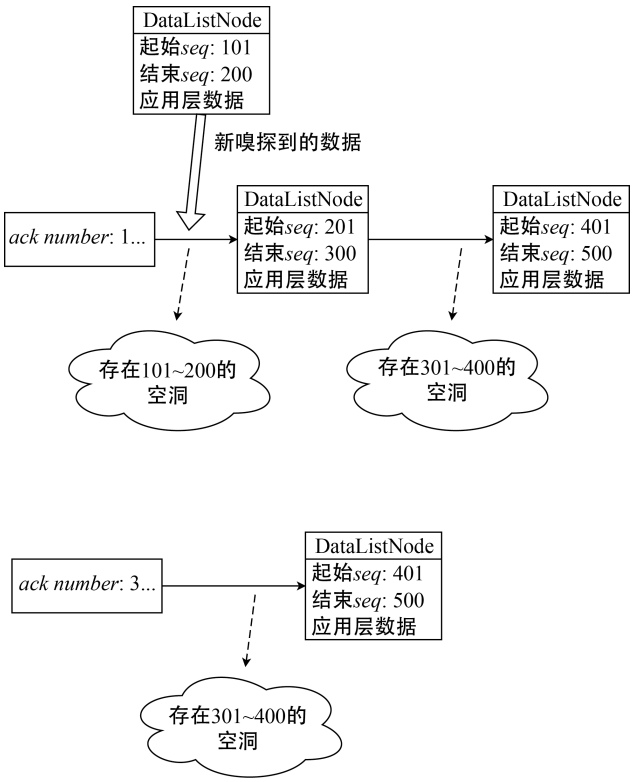

图 7　填充空洞

Figure 7　Filling the hollow

通过这些接口, 上层应用层输入代理无需实现 TCP 相关部分的逻辑, 直接调用类套接字接口即可与从执行体之间收发数据。

如图 8, 类套接字模块内部使用了一个链表保存 TCP 连接信息, 链表中的每个结点都是一条 TCP 连接, 保存了这条 TCP 连接上的源 IP、目的 IP、源端口号、目的端口号、序列号、确认号、发送缓冲区、接收缓冲区等信息。

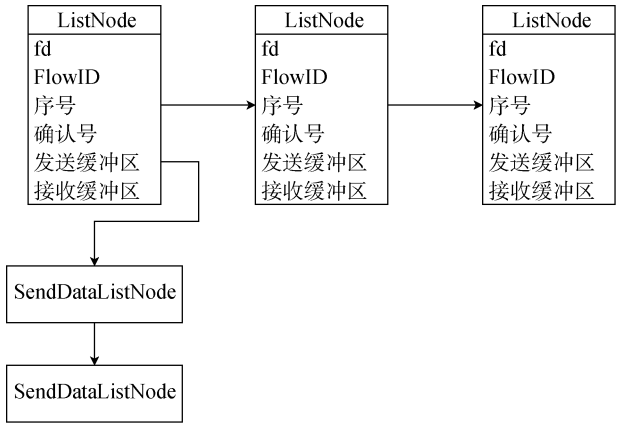

图 8　类套接字模块链表

Figure 8　Linked list of socket-like module

网卡嗅探到“从执行体→输入代理”的报文和上层应用层输入代理调用类套接字接口时都会与读取该链表, 并根据链表结点中保存的 TCP 连接信息完成功能。

如图 9, 应用层输入代理调用建立连接接口时, 类套接字模块会先发送第一次握手的报文, 当网卡嗅探到第二次握手的报文时, 再发送第三次握手的报文, 完成 TCP 连接。

应用层输入代理调用断开连接接口时, 类套接字模块会先发送第一次挥手的报文, 当网卡嗅探到第二次和第三次挥手的报文时, 再发送第四次挥手的报文, 断开 TCP 连接。

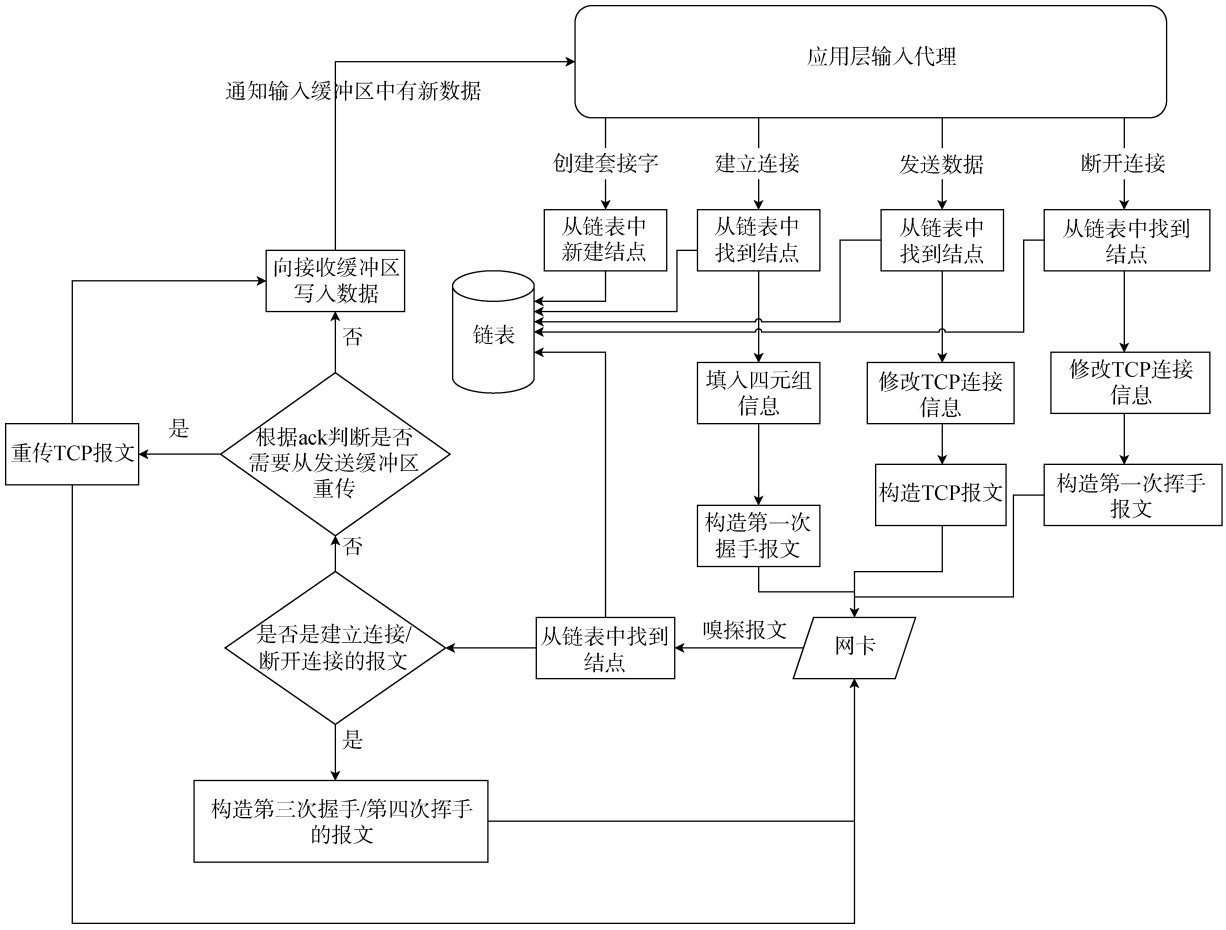

图 9　类套接字模块处理逻辑

Figure 9　Processing logic of socket-like module

应用层输入代理调用发送数据接口时, 既会构造出 TCP 报文用网卡发送, 也会复制一份保存在发送缓冲区中等待确认。

当网卡嗅探到“从执行体→输入代理”的报文时, 根据从执行体发过来的 TCP 报文的确认号字段删除发送缓冲区中缓存的数据。如果出现了重复 ack, 则说明发送给从执行体的报文出现了丢包现象, 需要重新发送一份。对于从执行体发送给自己的数据, 缓存到接收缓冲区中, 等待上层应用层输入代理读取。

4　实现与形式化验证

4.1　实现平台

如图 10, TCP 协议代理的编码用 C 语言完成, 基于 Linux 平台开发, 开发过程中用到了 C 语言函数标准库以及 pcap 网卡抓包和 libnet 网卡发包这两个第三方库。

4.2　形式化验证方法概述

如图 11, TCP 协议代理基于形式化验证工具 Verifast 进行形式化证明。Verifast 基于符号执行[23-24]和 SMT 求解器实现对 C 和 Java 程序的形式化验证, Verifast 验证器本身的可靠性证明已被 Coq 检查[25]。被验证的程序需要对每个函数书写带有前置条件和后置条件的形式化规约。Verifast 会检查每个函数是

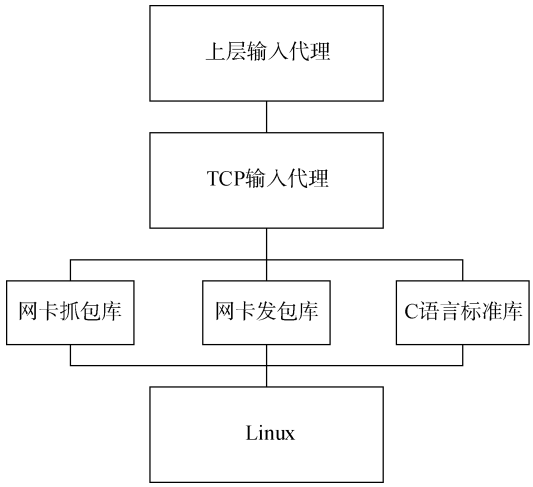

图 10　实现结构

Figure 10　Implementation structure

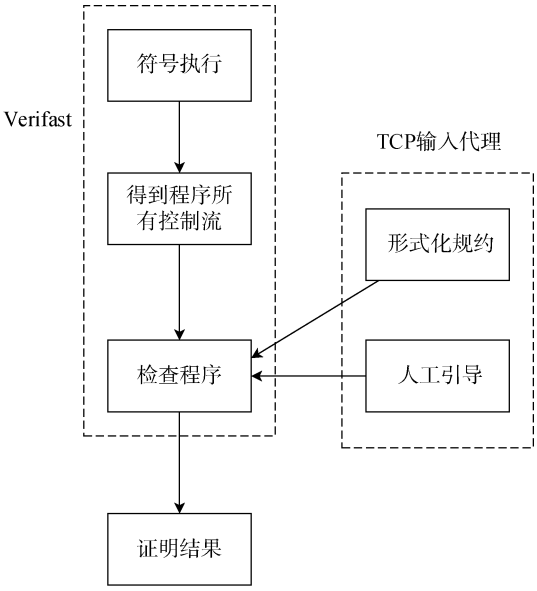

图 11 证明结构

Figure 11 Proof structure

否满足书写的形式化规约, 对于函数之间的相互调用, Verifast 会在调用函数前检查程序的状态中是否有满足被调用函数的前置条件, 调用函数后将被调用函数的后置条件添加到程序的状态中, 从而将程序的验证过程分成多个部分, 减少证明时需要检查的路径。

表 2 计划验证的内存安全性目标特性

Table 2 Memory safety properties for verification

特性名称	特性内涵	验证方法
指针访问的内存安全	不会发生数组越界、访问野指针等问题导致的段错误	Verifast 在对程序进行符号执行时会保存每个指针的符号值, 符号值又对应了程序中的一块内存空间。通过检查当前函数是否拥有对这块内存空间的访问权限检查指针的解引用操作是否会引发错误。
无内存泄漏	所有运行时申请的动态内存空间都会被释放	程序验证时每次发现申请动态内存空间都会将这块内存保存在表中, 释放时从表中删除。函数退出时如果发现没有释放也没有在形式化规约中声明的内存空间说明发生内存泄漏。
无重复释放错误	运行时申请的动态内存空间只会被释放一次, 不会产生“double free”的运行时错误	程序验证时每次发现申请动态内存空间都会将这块内存保存在表中, 释放时从表中删除。如果释放一块已经释放过的空间, 则表中无法查询到, 无法验证通过。
不存在死代码	程序中不会存在永远不会被运行到的代码	Verifast 在符号执行时会检查程序所有的控制流, 如果存在死代码, 则无论哪条控制流都不会运行到这条代码, 会被检查出来。

TCP 协议代理在形式化验证时只假设了在实现中用到的网卡抓包库和网卡发包库的正确性, 其余模块均通过形式化验证, 最大化地减少了由于底层依赖库的错误导致上层出现错误的可能性。同时, 上层输入代理在开发完后可以基于 TCP 协议代理的形式化规约快速对上层输入代理进行形式化验证。

4.3 嗅探模块的形式化验证

嗅探模块使用了 pcap 库进行开发, pcap 库本身只有非形式化的描述, 没有形式化规约。因此我们首先根据 pcap 库的非形式化描述书写了 pcap 库的形式化规约, 然后再书写了嗅探模块的形式化规约。

嗅探模块的每次程序主循环中都会得到一段精确定义长度的字符串, 后续模块对该字符串进行处理, 在循环的最后嗅探模块会对这段字符串的内存进行释放, 防止内存泄漏。

本文为嗅探模块的主循环书写了形式化规约的循环不变量。在形式化验证时, 该循环不变量会在第一次进入循环和每一次进入循环时被检查, 从而解决了程序循环导致的无穷状态的问题。Verifast 定理证明器检查了嗅探模块的实现, 证明了本文书写的循环不变量在循环中始终成立。

4.4 报文解析模块的形式化验证

如图 12, 报文解析模块的形式化规约定义了输入是一段定义长度的字符串, 在报文解析模块内部只会读取字符串的内容, 不会对这段字符串做任何修改。同时输出一个 C 语言结构体, 结构体的内存空间由报文解析模块向系统申请, 将 TCP 报文中相应的字段填入结构体中, 由于用形式化规约描述了对这个结构体的输出, 该结构体的内存空间由后续的模块拥有和释放, 无需报文解析模块进行管理, 从而防止了内存泄漏、段错误、重复释放等错误。

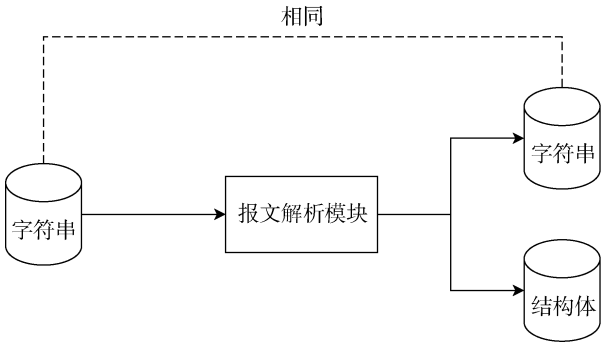

图 12 输入与输出的字符串相同

Figure 12 Identicality of input and output strings

如图 13, 在报文解析模块内部, Verifast 保证了程序只会对输入的字符串的内存空间做访问, 任何

超出这段内存空间的访问都无法通过形式化验证, 从而确保了程序不会在运行时出现段错误。

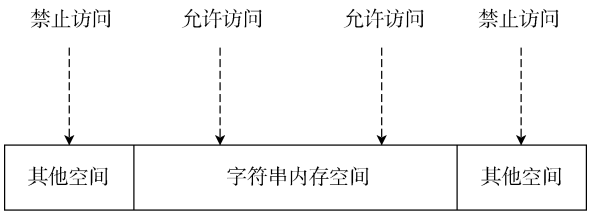

图 13 访问不会越界

Figure 13 Access do not cross bounds

4.5 邻主连接管理模块的形式化验证

邻主连接管理模块的形式化规约定义了在返回值不同的情况下, 所返回的内存数据的不同。

在返回值为 0 时, 不会返回任何内存空间。在返回值是 1 时, 这时是一个新的 TCP 连接的建立信息, TCP 协议规定建立连接时的最后一个报文可以“捎带”应用层数据, 邻主连接管理模块的形式化规约中也描述了这一行为, 会根据 *data_len* 是否为 0 决定返回的是空字符串还是带有应用层数据的字符串。返回值为 2 时会有应用层数据, 后续交给上层输入代理处理, 因此该应用层数据的释放也由上层输入代理控制。返回值为 3 时断开连接, 没有应用层数据需要交给上层。

通过这些精确描述的形式化规约而不是非形式化的对函数接口的语言描述, 可以清晰地让不同模块的不同开发者明白程序中内存的所有权, 尤其是 C 语言这类没有垃圾回收的语言, 每个函数只能对形式化规约中声明的内存空间进行访问、修改与释放, 释放内存空间后其他函数就无法访问这段地址, 从而防止了段错误和重复释放等错误。

如图 14, 在对邻主连接管理模块进行证明时, Verifast 会追踪内部链表的每一个结点, 如果某个结点错误地断开连接又没有释放, 则在检查证明时会出现一个链表以及一个单独的结点, 与我们书写的邻主连接管理模块的形式化规约不符, 定理证明器就会指出这一错误。

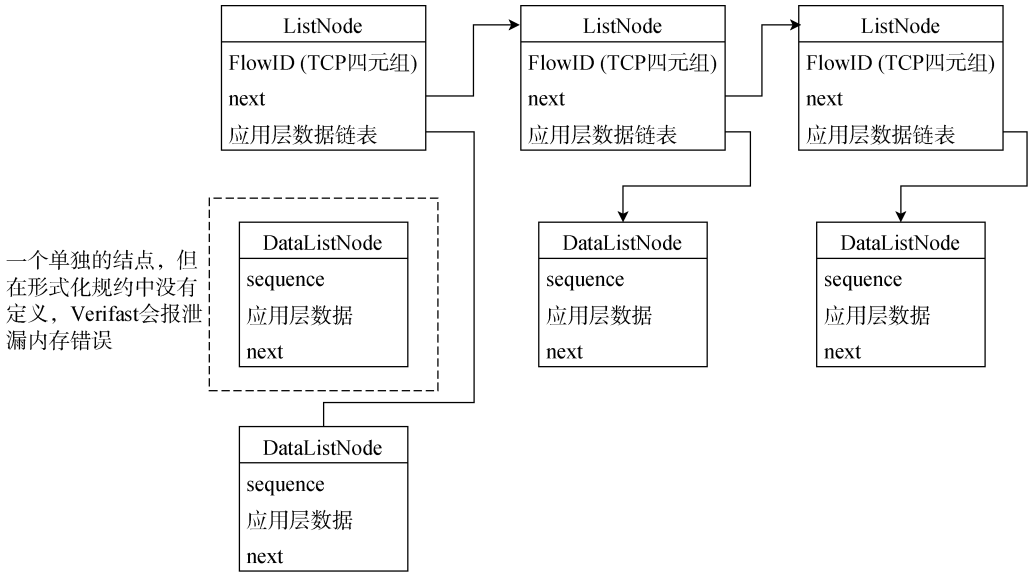

图 14 检查邻主连接管理模块的内存泄漏

Figure 14 Checking for memory leaks in neighbor-master connection management module

4.6 类套接字模块的形式化验证

类套接字模块定义了每个类套接字接口的形式化规约。从而精确定义了程序接口，避免因为含糊不清的非形式化语言描述造成不同模块间的调用错误。

创建套接字接口 socket()无需传入参数, 会返回一个文件描述符。

建立连接接口 connect()需要传入文件描述符以及建立连接的 TCP 四元组, 返回连接是否成功, 在接口的内部会对类套接字模块内部的链表进行修改。

发送数据接口 write()传入文件描述符以及要发送的字符串和长度, 在调用返回后这个字符串的内存空间将不存在。通过这一形式化规约上层输入代理开发者就能准确地明白这个传入的字符串是不需要自己来释放的, 会在发送数据接口内部被释放。防止了内存泄漏和重复释放的问题。

接收数据接口 read()传入文件描述符和要写入的字符串缓冲区, 调用返回后这个字符串的内容会被修改成接收到的内容。形式化规约定义了这个字符串空间还是由上层输入代理自己进行管理与释放。

断开连接接口 close()传入文件描述符, 返回断开连接是否成功。

如图 15, 在对类套接字模块进行形式化验证时, Verifast 会追踪类套接字模块内部用到的链表的每一个结点的内存空间。如果接收到从执行体发过来的确认报文, 改变了发送缓冲区链表的链接, 却没有回收掉链表结点的内存空间, 就会在 Verifast 的证明中存在一个单独的结点的内存空间, 而我们的形式化规约中没有声明这段内存空间, 就会导致验证不通过。对于接收缓冲区用到的字符串内存空间, Verifast 也会检查读取与写入时的边界条件, 防止出现段错误。

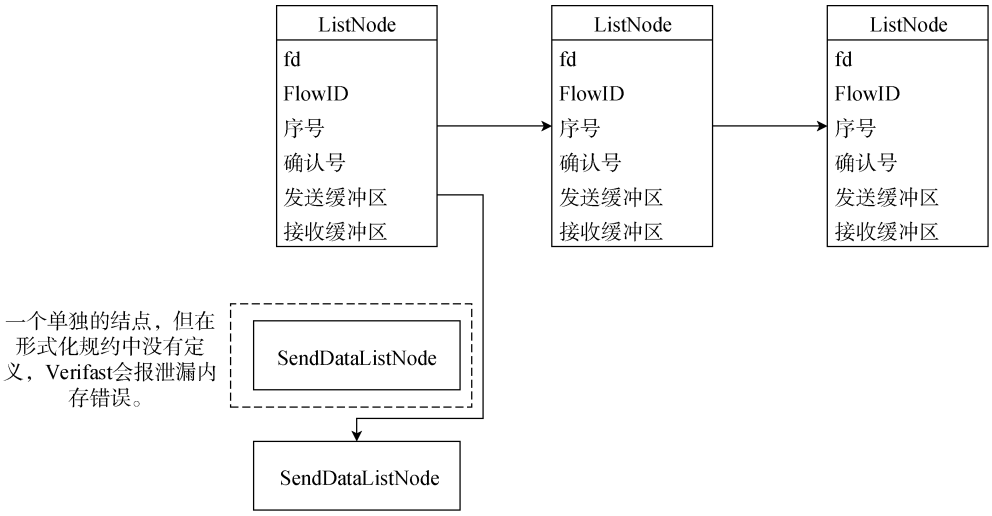

图 15 检查类套接字模块的内存泄漏

Figure 15 Checking for memory leaks in socket-like module

4.7 形式化验证结果及分析

如表 3 所示, 除了低级的内存安全属性, 本文还形式化验证了各模块的部分高级属性。本文对 TCP 代理书写的形式化规约均已通过 Verifast 定理证明器的检查。在形式化证明时, 本文在实现中发现了多个不符合形式化规约的程序缺陷, 且这些缺陷在证明前并未被编译器和单元测试发现。这说明本文使用的形式化验证技术相比传统的软件测试技术寻找拟态防御基本组件中程序缺陷的能力更强。

表 3 各模块形式化验证的部分高级属性

Table 3 Some advanced properties of the formal verification of each module

模块	形式化验证的高级属性
嗅探模块	每次嗅探报文的结束后都不会留下未释放的内存
报文解析模块	输入与输出的字符串相一致, 并输出一个结构体; 内部不会发生对字符串数组的越界访问
邻主连接管理模块	返回值一定在规定的范围内; 对不同的返回值, 返回的字符串也与返回值对应
类套接字模块	创建套接字接口返回的文件描述符一定递增; 发送数据接口传入的字符串会在模块内部释放, 不会造成内存泄漏

如表 4 所示, 不统计一些胶水代码与工具代码, TCP 协议代理总计用 1611 行 C 语言代码实现, 形式化验证时需要人工引导定理检查器书写的证明共计 588 行。实际开发过程中, 书写代码实现与书写人工证明所需的时间约为 1∶1。考虑到形式化验证保证了整个 TCP 协议代理的内存安全和部分高级属性的正确, 这些人力投入是完全值得的。

表 4 各模块实现行数与证明行数

Table 4 Number of implementation lines and proof lines for each module

模块	代码行数	人工书写的证明行数	证明结果
嗅探模块与程序主循环	410	74	
报文解析模块	196	172	通过
邻主连接管理模块	356	166	通过
类套接字模块	649	176	通过
总计	1611	588	

TCP 协议代理的形式化验证为整个应用层输入代理的形式化验证提供了基础。如果能将整个“拟态括号”的组件都进行形式化验证, 就能够进一步增强拟态防御的安全性与可靠性。

5 测试

5.1 测试环境

echo 协议是一个基于 TCP 协议的“请求-响应”式

的应用层协议, 常被用于测试 TCP 连接是否正常。本文基于 TCP 协议代理实现了一个简易的 echo 服务输入代理, 用于测试 TCP 协议代理的功能正确, 并作为示例代码展示 TCP 协议代理的应用程序接口的使用方法。

如图 16, 测试环境搭建了 2 个客户端和 3 个执行体, 以及 echo 输入代理。

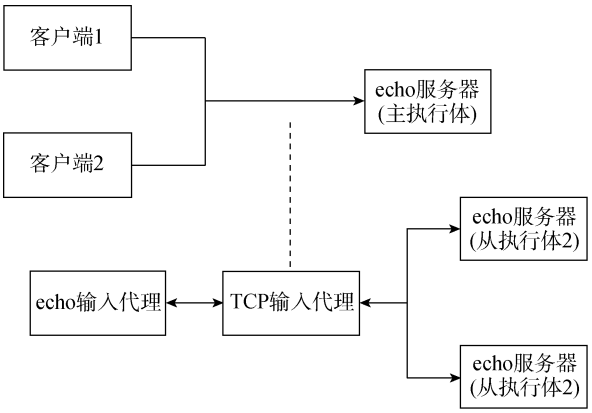

图 16 测试环境

Figure 16 Test environment

5.2 报文解析、邻主连接管理模块功能测试

构造一些 TCP 报文, 观察通过报文解析模块和邻主连接管理模块后能不能给出预期的结果。为了测试邻主连接管理模块中对嗅探到的乱序数据的处理功能是否正确, 构造的报文也模拟了由于丢包导致的乱序报文的现象。

表 5 构造 TCP 报文顺序

Table 5 Constructing TCP message sequences

TCP 报文	邻主连接管理模块的输出	应用层数据	输出值解释
第一次握手	0	-	不做处理
第二次握手	0	-	不做处理
第三次握手	1	-	建立连接完成, 向上报告
发送数据 1	2	1	应用层数据, 向上报告
发送数据 1 的 ack	0	-	不做处理
主执行体的 echo	0	-	不做处理
客户端的 ack	0	-	不做处理
(假设这里发送数据 2, 丢失)	-	-	-
发送数据 3	0	-	超前的数据, 不做处理
发送数据 2 (超时重传)	2	23	与 3 合并, 向上报告
……	……	……	……
第四次挥手	3	-	断开连接, 向上报告

如表 5, 在邻居与主执行体完成 TCP 建立连接操作时, 测试结果输出 1; 在邻居结果发送数据 2 时, 测试结果输出 2; 在发送数据 3 时, 由于发生了丢包, 序列号不连续, 所以测试结果输出 0; 在发送数据 2 后, 测试结果输出 2, 并将第 2 次发送的数据和第 3 次发送的数据合并上报。

该实验结果表明, 报文解析模块和邻主连接管理模块功能实现正确, 行为与预期一致。

5.3 类套接字模块功能测试

测试类套接字模块提供的类套接字接口是否实现了 TCP 协议的相关功能。

首先测试能否与从执行体机器上的 TCP 协议栈相互连接并交换数据。编写测试代码调用类套接字模块的创建套接字、建立连接、发送数据、接收数据、断开连接等接口, 与从执行体的 TCP 协议栈交互数据。然后测试类套接字模块是否实现了可靠传输功能。报文的丢失在双方向上都有可能发生。如图 18, “从执行体→TCP 协议代理”方向上的报文丢失通过在“嗅探模块→类套接字模块”之间多加一层丢弃报文的测试代码实现; “TCP 协议代理→从执行体”方向上的报文丢失通过在“类套接字模块→网卡发包”之间多加一层丢弃报文的测试代码实现。

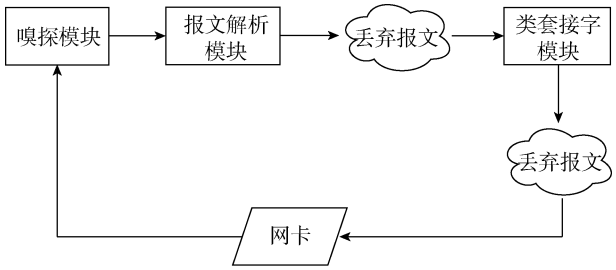

图 17 模拟丢失报文

Figure 17 Simulation of lost messages

如图 18, 编号 12 的报文被测试代码丢弃, TCP 代理未对该报文做确认, 编号为 14 的重传报文是该丢包的重传, 编号 15 与 16 的报文之间存在一次 TCP 代理发出的请求报文, 被测试代码丢弃, 因此收到了从执行体的重复确认, 编号为 18 的报文是对该丢包的重传。测试结果表明, 对于“从执行体→TCP 代理”方向和“TCP→从执行体”方向上的丢包, TCP 协议代理都正确地实现了重传机制与可靠性传输。

6 结论

TCP 协议代理实现了输入代理中嗅探报文、解析报文、邻主连接管理、与从执行体建立 TCP 连接、向上提供类套接字接口等功能, 并基于 Verifast 定理证明器书写了形式化规约。通过形式化验证, 证明了

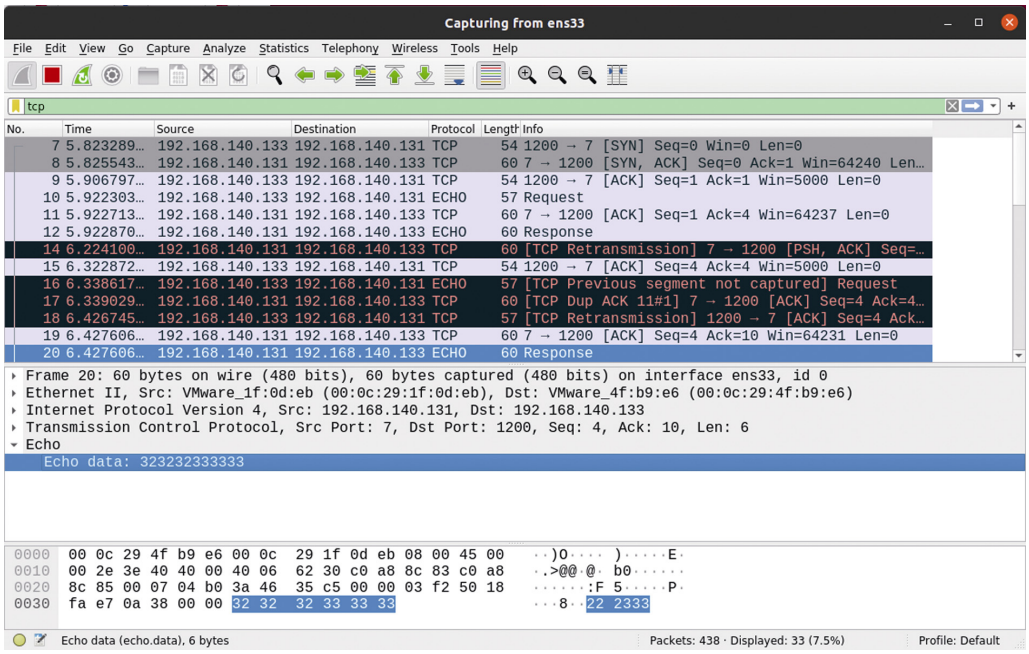

图 18　模拟丢失报文结果

Figure 18　Results of simulating lost messages

代码的内存安全性, 以及邻主连接管理、类套接字接口等内部模块功能的正确性。

TCP 协议代理实现总计 1611 行 C 代码, 其中形式化验证所需人工引导定理检查器书写的证明共计 588 行。实际开发过程中, 书写代码实现与书写人工证明所需的时间约为 1∶1。本文对 TCP 协议代理的实现与形式化验证工作证明了将形式化验证引入拟态路由器的关键组件开发中是确实可行的。下一步的研究工作是将本文所采用的形式化验证方法应用到拟态路由器的调度器、裁决器等关键软件模块的设计实现中, 进一步提升拟态路由器的整体安全能力。

参考文献

[1] Wu J X. Research on Cyber Mimic Defense[J]. *Journal of Cyber Security*, 2016, 1(4): 1-10.
(邬江兴. 网络空间拟态防御研究[J]. *信息安全学报*, 2016, 1(4): 1-10.)

[2] Ma H L, Yi P, Jiang Y M, et al. Dynamic Heterogeneous Redundancy Based Router Architecture with Mimic Defenses[J]. *Journal of Cyber Security*, 2017, 2(1): 29-42.
(马海龙, 伊鹏, 江逸茗, 等. 基于动态异构冗余机制的路由器拟态防御体系结构[J]. *信息安全学报*, 2017, 2(1): 29-42.)

[3] Zhang J J, Yang L, Gao X M, et al. Formal Analysis of QUIC Handshake Protocol Using Symbolic Model Checking[J]. *IEEE Access*, 2021, 9: 14836-14848.

[4] Bertot Y, Castéran P. Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions[M]. Berlin: Springer, 2004

[5] Nipkow T, Paulson L C, Wenzel M. *Isabelle/HOL: a proof assistant for higher-order logic*[M]. Berlin: Springer, 2002.

[6] de Moura L, Kong S, Avigad J, et al. The Lean Theorem Prover (System Description)[C]. *International Conference on Automated Deduction*, 2015: 378-388.

[7] Klein G, Elphinstone K, Heiser G, et al. seL4: Formal Verification of an OS Kernel[C]. *The ACM SIGOPS 22nd symposium on Operating systems principles*, 2009: 207-220.

[8] Gu R H, Shao Z, Chen H, et al. CertiKOS: An Extensible Architecture for Building Certified Concurrent OS Kernels[C]. *The 12th USENIX conference on Operating Systems Design and Implementation*, 2016: 653-669.

[9] Jacobs B, Smans J, Philippaerts P, et al. VeriFast: A Powerful, Sound, Predictable, Fast Verifier for C and Java[C]. *NASA Formal Methods Symposium*, 2011: 41-55.

[10] Leino K R M. Developing Verified Programs with Dafny[C]. *2013 35th International Conference on Software Engineering*, 2013: 1488-1490.

[11] Peña R. An Assertional Proof of Red-Black Trees Using Dafny[J]. *Journal of Automated Reasoning*, 2020, 64(4): 767-791.

[12] Yousefi F, Abhashkumar A, Subramanian K, et al. Liveness Verification of Stateful Network Functions[C]. *The 17th Usenix Conference on Networked Systems Design and Implementation*, 2020: 257-272.

[13] Abhashkumar A, Gember-Jacobson A, Akella A. Tiramisu: Fast Multilayer Network Verification[C]. *The 17th Usenix Conference on Networked Systems Design and Implementation*, 2020: 201-220.

[14] Yuan Y, Moon S J, Uppal S, et al. NetSMC: A Custom Symbolic Model Checker for Stateful Network Verification[C]. *17th USENIX Symposium on Networked Systems Design and Implementation*, 2020: 181-200.

[15] Beckett R, Mahajan R. A General Framework for Compositional Network Modeling[C]. *The 19th ACM Workshop on Hot Topics in Networks*, 2020: 8-15.

[16] Li Y F, Jia C J, Hu X H, et al. Mahjong: A Generic Framework for

Network Data Plane Verification[C]. *The Symposium on Architectures for Networking and Communications Systems*, 2021: 52-58.

[17] Tian B C, Gao J Q, Liu M Q, et al. Aquila: A Practically Usable Verification System for Production-Scale Programmable Data Planes[C]. *The 2021 ACM SIGCOMM 2021 Conference*, 2021: 17-32.

[18] Arun V, Arashloo M T, Saeed A, et al. Toward Formally Verifying Congestion Control Behavior[C]. *The 2021 ACM SIGCOMM 2021 Conference*, 2021: 1-16.

[19] Zaostrovnykh A, Pirelli S, Iyer R, et al. Verifying Software Network Functions with no Verification Expertise[C]. *The 27th ACM Symposium on Operating Systems Principles*, 2019: 275-290.

[20] Zhang X M, Gu Z Y, Wei S, et al. Markov Game Modeling of Mimic Defense and Defense Strategy Determination[J]. *Journal on Communications*, 2018, 39(10): 143-154.
(张兴明, 顾泽宇, 魏帅, 等. 拟态防御马尔可夫博弈模型及防御策略选择[J]. *通信学报*, 2018, 39(10): 143-154.)

[21] Zhu W J, Guo Y B, Huang B H. A Mimic Defense Automaton Model of Dynamic Heterogeneous Redundancy Structures[J]. *Acta Electronica Sinica*, 2019, 47(10): 2025-2031.
(朱维军, 郭渊博, 黄伯虎. 动态异构冗余结构的拟态防御自动机模型[J]. *电子学报*, 2019, 47(10): 2025-2031.)

[22] Wang T, Xiang L L, Chen T M. Time Automata Model and Verification of Mimic Defense System[J]. *Journal of Chinese Computer Systems*, 2020, 41(8): 1718-1724.
(王婷, 项露露, 陈铁明. 拟态防御系统的时间自动机模型和验证[J]. *小型微型计算机系统*, 2020, 41(8): 1718-1724.)

[23] Cadar C, Sen K. Symbolic Execution for Software Testing[J]. *Communications of the ACM*, 2013, 56(2): 82-90.

[24] Baldoni R, Coppa E, D'elia D C, et al. A Survey of Symbolic Execution Techniques[J]. *ACM Computing Surveys*, 2019, 51(3): 1-39.

[25] Vogels F, Jacobs B, Piessens F. Featherweight verifast[J]. *Logical Methods in Computer Science*, 2015, 11(3): 1-57.

金希文 于 2019 年在杭州电子科技大学计算机科学与技术专业获得学士学位。现在东南大学计算机技术专业攻读硕士学位。研究领域为网络空间安全、形式化方法。Email: jinxw@seu.edu.cn

葛强 于 2019 年在浙江工商大学信息安全专业获得学士学位。现在东南大学网络空间安全专业攻读硕士学位。研究领域为网络空间安全、形式化方法。Email: geqiang@seu.edu.cn

张进 于 2008 年在解放军信息工程大学通信与信息系统专业获得博士学位。现为网络通信与安全紫金山实验室路由器架构工程师。主要研究领域为软硬件协同设计、网络安全。Email: zhangjin@pmlabs.com.cn

丁建 于 2011 年在南京邮电大学信号与信息处理专业获得硕士学位。现为网络通信与安全紫金山实验室路由交换协议研发工程师。主要从事协议安全方面的研究。Email: dingjian@pmlabs.com.cn

江逸茗 于 2012 年在解放军信息工程大学通信与信息系统专业获得博士学位, 国家数字交换系统工程技术研究中心副研究员, 主要研究方向为网络安全与新型网络体系架构。Email: jiangyiming@ndsc.come.cn

马海龙 于 2011 年在信息工程大学通信与信息系统专业获得博士学位。现任信息工程大学信息技术研究所研究员。研究领域为网络安全、路由工程。研究兴趣包括: 创新网络体系、网络安全管控等。Email: longmanclear@163.com

伊鹏 于 2007 年在信息工程大学通信与信息系统专业获得博士学位。现任信息工程大学技术研究所研究员。研究领域为新型网络体系结构、网络安全管控和主动防御技术研究。Email: 15238363586@139.com