软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种基于轻量级链式验证的网络传输层安全性增强方法 *

冯学伟[1], 徐恪[1,2,4], 李琦[2,3,4], 杨宇翔[1], 朱敏[1], 付松涛[1]

[1](清华大学计算机科学与技术系 北京 100084)

[2](北京信息科学与技术国家研究中心 北京 100084)

[3](清华大学网络科学与网络空间研究院 北京 100084)

[4](中关村实验室 北京 100081)

通讯作者: 徐恪, E-mail: xuke@mail.tsinghua.edu.cn

摘 要: 传输层是网络协议栈的关键组成部分, 负责为不同主机间的应用程序提供端到端的服务. 已有的传输层协议如 TCP 等为用户提供了基本的差错控制和确认应答等安全保护机制, 在一定程度上保证了不同主机间应用程序收发报文的一致性. 但现有的传输层安全保护机制存在严重的缺陷, 如 TCP 报文的序列号容易被猜测推理, 报文校验和的计算依赖于有漏洞的补码求和算法等. 这导致现有的传输层安全机制并不能保证报文的完整性和安全性, 从而允许一个远程的攻击者伪造出一个报文, 注入到目标网络流中, 对目标网络流形成污染或攻击. 针对传输层的攻击发生在网络协议栈的基础层次, 可以旁路掉上层应用的安全保护机制, 对网络基础设施造成严重的危害. 本文深入研究了近年来针对网络协议栈的各种攻击和相关安全漏洞, 提出了一种基于轻量级链式验证的传输层安全性增强方法 LightCTL. 该方法基于哈希验证的方式, 使 TCP 连接双方能够对传输层报文形成彼此可验证的共识, 避免攻击者或中间人窃取和伪造敏感信息, 从而解决网络协议栈面临的典型安全威胁, 包括基于序列号推理的 TCP 连接重置攻击、TCP 劫持攻击、SYN 洪泛攻击、中间人攻击、报文重放攻击等. LightCTL 不需要修改中间网络设备如路由器等的协议栈, 只需对终端协议栈中的校验和相关部分进行修改, 因此方法易于部署, 同时显著提升了网络系统的安全性.

关键词: 计算机网络;网络传输层;链式验证;恶意报文注入

中图法分类号: TP311

中文引用格式: 冯学伟, 徐恪, 李琦, 杨宇翔, 朱敏, 付松涛. 一种基于轻量级链式验证的网络传输层安全性增强方法. 软件学报,2023. http://www.jos.org.cn/1000-9825/0000.htm

英文引用格式: Feng XW, Xu K, Li Q, Yang YX, Zhu M, Fu, ST. A Method for Enhancing Network Security of the Transport Layer by Leveraging the Lightweight Chain Verification. Ruan Jian Xue Bao/Journal of Software, 2023 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

A Method for Enhancing Network Security of the Transport Layer by Leveraging the Lightweight Chain Verification

FENG Xue-Wei[1], XU Ke[1,2,4], LI Qi[2,3,4], YANG Yu-Xiang[1], ZHU Min[1], FU Song-Tao[1]

[1](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[2](Beijing National Research Center for Information Science and Technology, Beijing 100084, China)

[3](Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China)

[4](Zhongguancun Lab, Beijing 100081, China)

* 基金项目: 国家重点研发计划课题(2022YFB3102300); 国家自然科学基金项目(61825204, 61932016, 62132011); 北京高校卓越青年科学家计划项目(BJJWZYJH01201910003011)

收稿时间: 2022-06-16; 修改时间: 2023-01-10; 采用时间: 2023-03-29

Abstract: The transport layer is a key component in network protocol stack, which is responsible for providing end-to-end services for applications between different end hosts on the Internet. Existing transport layer protocols such as TCP provide users with some basic protections, e.g., error controls and acknowledgements, which ensures the consistency of user datagram to a certain extent. However, these basic protections are not adequate to defend various attacks on the Internet. For example, the sequence number of TCP segments is easy to be guessed and inferred, and the calculation of the datagram's checksum depends on the vulnerable one's complement sum. As a result, the existing transport layer security mechanisms cannot guarantee the integrity and security of the datagram transferred on the Internet, which allows a remote attacker to craft a fake datagram and inject it into the target network stream, thus poisoning the target network stream. The attack against the transport layer occurs at the basic layers of the network protocol stack, which can bypass the security mechanisms enforced at the upper application layer (e.g., user name and password) and thus cause serious damages to the network infrastructure. In this paper, after investigating various prior attacks over network protocols and the related security vulnerabilities, we propose a security mechanism LightCTL based on the lightweight chain verification, which can be deployed at the transport layer to guarantee the integrity of the datagram transferred on the Internet. Based on the hash verification, LightCTL enables both peers of a TCP connection to create a verifiable consensus on transport layer datagrams, so as to prevent attackers from stealing and forging sensitive information. As a result, LightCTL can successfully foil various attacks against network protocol stack, including TCP connection reset attacks based on sequence number inferring, TCP hijacking attacks, SYN flooding attacks, Man-in-The-Middle attacks, replay attacks. Besides, LightCTL does not need to modify the protocol stack of intermediate network devices such as routers. It only needs to modify the checksum and the related parts of the end hosts' protocol stack. Therefore, LightCTL is easy to be deployed in the real world and significantly improves the security of networks.

Key words: Computer Networks; Transport Layer; Chain Verification; Malicious Packets Injection

TCP/IP 协议定义了网络空间数据交互和信息传递的基本准则规范，为整个互联网的正常运转提供了保障和支撑. TCP/IP 协议起源于上世纪 60 年代末美国政府资助的 ARPANET 网络. 在建设之初，ARPANET 网络主要面向特定高校和军事部门等小规模的可信组织，因此为了保持良好的开放性和高效性，TCP/IP 协议在设计之初并没有很好地考虑安全性和隐私性，这也导致了针对 TCP/IP 协议栈的网络攻击层出不穷[1][2].

在一次用户正常访问远程服务器的过程中，攻击者可能会在中间网络请求的多个环节和多个层次，发动恶意攻击. 如表 1 所示，首先在链路接入层，本地的攻击者可能会通过 ARP 污染，劫持用户的流量，或者通过链路层嗅探，监听其他用户的数据帧[3]；然后在网络层，攻击可以通过篡改源 IP 地址，伪装成其他主机欺骗通信对端，也可以通过观测目标主机的 IPID 分配情况，构建一个侧信道，推理猜测目标主机的通信状态[4] [5] [6]. 此外，攻击者还可以利用 IP 层的分片机制，伪造一个恶意的分片注入到目标网络流中，迫使合法分片和恶意分片错误重组，形成污染攻击[7] [8] [9]. 由于当前互联网缺乏对 ICMP 消息的安全验证，攻击者通过源 IP 地址伪造、伪装成中间路由器，发送一个恶意 ICMP 消息给目标主机，导致目标主机异常[10] [11]. 在控制平面，攻击者可以污染路由器的 OSPF 路由表或 BGP 路由表，劫持网络流量[12] [13] [14] [15] [16]. 在传输层，常见的攻击包括针对 TCP 协议的拒绝服务攻击和劫持攻击，例如最常见的 SYN Flooding 攻击和 TCP 序列号猜测攻击[17] [18] [19]. UDP 报文缺乏足够的安全验证能力，攻击者可以伪造出一个 UDP 报文，欺骗接收端，污染接收端的 DNS 缓存或者系统时间[20] [21] [22]. 针对应用层的攻击，攻击造成的危害随着应用的不同而变化，常见的包括通过 DNS 劫持操控用户的网络域名访问，通过 Web 攻击污染用户的 Web 缓存，通过邮件伪造欺骗用户点击造成信息泄露等[23] [24].

分析上述攻击，可以发现针对链路层的攻击，作用范围通常限定在局域网内，因此攻击造成的影响相对有限，可以通过局域网中的 MAC 地址绑定等手段解决[25]. 针对网络层的攻击，造成的影响通常是借由传输层来体现的，单独的在网络层难以形成直接的威胁破坏. 例如通过误用网络层的 IPID，形成侧信道，最终攻击的目标是 TCP 协议. 通过误用网络层的 IP 分片机制，最终污染攻击的目标是 UDP 协议. 而针对应用层的攻击，需要突破传输层的约束和限制，如 DNS 劫持需要突破传输层的源端口随机化机制，Web 攻击和邮件伪造攻击需要突破 TCP 层的序列号确认与应答机制. 因此，梳理发现传输层在多样化的网络攻击中，往往是攻击者需要突破的主要目标，它在网络协议栈安全中扮演着关键的角色.

传输层是整个互联网基础协议族的核心组成部分，基于 IP 层的功能原语，为上层的应用提供统一的服务，包括可靠传输（通过应答机制和超时重传机制保证）、按序送达（通过序号机制保证）、流量控制（通过滑动

窗口机制保证)、拥塞控制（通过慢启动、拥塞避免、快速恢复、以及指数回退机制保证）等. 不同于 IP 协议的无连接特性，传输层协议更加复杂，除 UDP 协议外，基本都具备面向连接的特征，提供上述 4 种基本服务，存在状态转换与控制等复杂机制，协议的形式化描述和具体实现也比较复杂，很容易产生安全问题. 但目前与传输层安全相关的研究很少，更多的研究都关注在 IP 层，包括地址校验、身份认证、流量审计等. 主要原因在于 IP 层是路由层，诸多路由协议都是在 IP 层实现和工作的，因此引起了更大的关注度. 但在当前复杂多元的网络环境下，随着终端计算能力的不断增强，端到端的安全性也变得更加重要和突出. 而且诸多的网络攻击效果，也都是借由传输层来体现的. 因此，加强传输层安全性的研究，保障端到端的安全可靠传输，在当前及未来的网络计算环境下，将是一个重要的需求和研究内容[33].

表 1 针对网络协议栈的多样化攻击

网络	攻击类型	攻击位置
链路层	ARP 污染	本地
	数据帧监听与嗅探	本地
网络层	IP spoofing	远程
	IPID 误用	远程
	IP 分片攻击	远程
	ICMP 误用	本地/远程
	路由劫持	远程
传输层	TCP 拒绝服务	远程
	TCP 劫持	远程
	UDP 报文伪造	远程
应用层	DNS 劫持	远程
	Web 攻击	远程
	邮件伪造	远程

在当前的协议栈设计和实现中，传输层初步具备了一些安全防御机制. 但这些机制并不足以应对当前日趋复杂隐蔽的多样化网络攻击. 例如,在 TCP 协议中,TCP 目前主要有 3 种安全机制来保证报文的安全性.（1）源端口随机化. 在 TCP 报文头中，源端口是一个 16 位的字段. 在早期的 TCP 协议实现中，源端口号的分配都是顺序分配的，因此攻击者通过累加推理的方式，可以很容易地猜测出一个目标连接的源端口号，对目标连接造成威胁.（2）TCP 校验和. TCP 校验和主要用于保证报文在传输过程中，不会出现传输差错. 校验和的计算采用补码求和的方式，覆盖 TCP 首部和 TCP 数据部分，以及一个 12 字节的伪首部. 由于补码求和不能保证报文的完整性，攻击者可以很容易地构造出一个等价体，欺骗 TCP 校验和机制.（3）TCP 序列号的应答号机制. 32 位的 TCP 序列号和应答号字段，保证了 TCP 报文的可靠传输，同时也实现了对报文的验证机制. 如果当前报文的序列号没有在可接受范围内，那么接收端就会丢弃掉该报文. TCP 协议的序列号和应答号机制，一定程度上缓解了伪造报文注入攻击. 但当前 TCP 序列号和应答号的实现，随机化程度不高，容易被猜测推理，允许一个远程的攻击者伪造出一个报文，注入到目标网络流中，对目标网络流形成污染或攻击.

针对当前协议栈传输层安全能力的不足，旨在解决互联网场景下多样化的网络攻击，本文提出了一种基于轻量级链式验证的传输层安全性增强方法 LightCTL（Lightweight-Verification Chain for Transport Layer）. 该方法基于哈希验证的方式，使 TCP 连接双方能够对传输层报文形成彼此可验证的共识，避免攻击者或中间人窃取和伪造敏感信息，从而解决网络协议栈面临的典型安全威胁. TCP 通信对端在建立连接之初，基于随机化的初始序列号 ISN（Initial Sequence Number），双方协商一个密钥，然后基于该密钥对连接上的 TCP 报文进行链式哈希验证，计算报文的哈希校验值，替换掉原有的不安全的 TCP 校验和字段. 当报文到达接收端后，接收端再基于相同的过程，计算报文的哈希校验值，并将其与报文中携带的校验和字段进行匹配，如果二者

相等，则说明报文在传输过程中未受到篡改，并且报文的发送源端是安全可信的. LightCTL 可以有效抵御多样化的网络攻击，包括基于 IP 分片的 TCP 流量污染攻击，基于 IPID 侧信道的 TCP 连接重置攻击及劫持攻击、ACK 报文伪造攻击、中间人攻击、报文重放攻击等. LightCTL 不需要修改中间网络设备如路由器等的协议栈，只需对终端协议栈中的校验和相关部分进行修改，因此方法易于部署. 通过在实际网络中进行部署评估，结果表明 LightCTL 引入的性能损失（即 LightCTL 的部署给网络数据传输带来的额外时延）也非常小，在 2.84%以内.

本文第 1 节分析当前网络传输层的安全性. 第 2 节介绍本文提出的基于轻量级链式验证的网络安全性增强方法. 第 3 节介绍了 LightCTL 的实现，以及从安全性和性能的测试结果.最后总结全文.

1 网络传输层安全性分析

现有的传输层相关安全防御方法主要有两种. 一是 TLS 协议族，二是 QUIC 协议族. TLS 协议族需要工作在可靠传输层协议之上（目前主要是工作在 TCP 协议之上），而 QUIC 则工作在 UDP 协议之上. QUIC 协议可靠性相关的需求主要由协议自身实现，需要注意的是 QUIC 的序列号不再是针对字节编号，而是针对一个报文整体编号，而且这个序列号是单向递增的，即使有报文丢失，重传报文也会得到一个新编号，这个重传包主要是依靠 Stream Offset 变量来识别的，即 QUIC 的按序送达是依靠 Stream Offset 来保证的，重复报文的 Stream Offset 值不变. TLS 和 QUIC 提供了端到端的通信安全，包括保证通信数据的机密性、完整性以及对通信双方的安全认证. 在数据加密过程中，TLS 首先通过公钥机制，促使通信双方进行安全交换，生成一个共享密钥，然后利用该密钥对流量进行对称式加密. QUIC 的密钥交换过程则主要靠 Diffie Hellman 算法保证[26]，最终也是生成一个共享密钥，进行对称式加密.

传输层的加密机制可以保证报文载荷的机密性、阻止中间人攻击等，在很大程度上提高数据传输的安全性. 但并不能完全抵御针对传输层的恶意攻击. 例如针对 TCP 连接的 RST 攻击、SYN 洪泛攻击、序列号推理攻击等依然可以作用到 TLS 协议上，因为 TLS 协议只是保证了上层数据的机密性，对于下层的 TCP 协议、以及 TCP 报文的头部，都不能进行安全性校验或保护，而这些头部信息，恰恰是传输层攻击和利用的主要目标，攻击者只需对两个序列号进行猜测破解，就可以进行多种攻击. 甚至，攻击者可以利用 IP 层的分片机制，伪造出一个恶意分片注入到目标 TCP 连接中，迫使合法分片和该恶意分片进行错误重组，从而污染目标数据流. 在这一过程中，攻击者可以绕开 TCP 序列号和应答号的猜测，直接攻击目标连接.

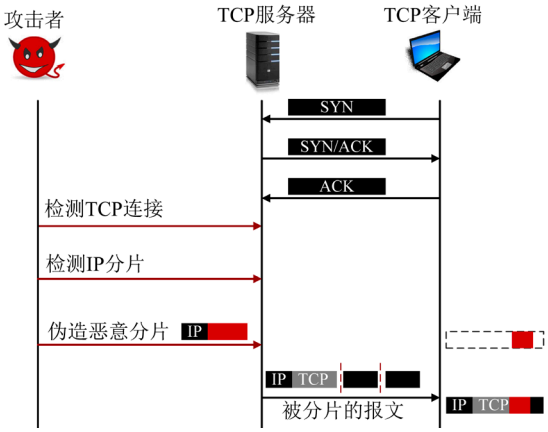

图 1 基于 IP 分片的 TCP 流量污染

如图 1 所示，在 TCP 服务器和客户端之间存在一条目标连接，攻击者想要通过 IP 分片机制来污染该连接的数据流. 首先在第一步，攻击者通过侧信道和恶意 JavaScript 等工具的辅助，检测出目标 TCP 连接的存在[27] [28]. 然后第二步，攻击者通过探测和利用 ICMP Fragmentation Needed 消息，检测到目标 TCP 连接存在

IP 分片的情况. IP 分片是 IP 层依据不同网络最大传输单元（MTU）对分组进行调整处理的基础机制，在网络中经常发生. 第三步，攻击者伪造一个恶意的分片，然后利用 IP 地址欺骗技术，将分片的源 IP 地址指定成服务器的 IP 地址，冒充成服务器，将分片发送给 TCP 客户端. 最后，当服务器的合法分片到达客户端后，会和攻击者的恶意分片进行错误重组，实现对原始 TCP 报文流的污染. 而在这一过程中，原始 TCP 报文的序列号和应答号被携带在第一个合法分片中，攻击者只需污染其他数据段即可，绕过了 TCP 序列号和应答号的猜测.

分析已有的网络攻击，包括上述基于 IP 分片的 TCP 连接污染攻击，我们发现，他们均能够绕过 TCP 协议已有的安全验证机制，包括基于序列号的报文认证和基于校验和的报文验证. 尤其是在 TCP 报文头中，16 位的 TCP 校验和字段 checksum 不能起到有效的安全校验作用. TCP 协议在设计之初，当时受限于终端计算能力和资源，TCP 协议的校验和采用了简单的补码求和的方式，主要用于验证报文在传输过程中是否出现错误. 传统校验算法中，该字段主要由发送方依据采用补码求和的方式计算. 发送方把一个 12 字节的伪首部（4 字节源 IP 地址、4 字节目的 IP 地址、1 字节保留置 0、1 字节传输层协议号、2 字节 TCP 报文长度）、TCP 报头以及 TCP 数据，分为 16 位的字（如果总长度为奇数个字节，则在最后增添一个各位都为 0 的字节）. 然后，用反码相加法累加所有的 16 位字（进位也要累加），再对最后的计算结果取反，作为 TCP 的校验和，填充到 TCP 报文头的 checksum 字段，发送给对端. 报文到达接收端后，接收端再次计算该值，如果计算结果与原有填充的字段值相等，则说明报文在传输过程中没有出现差错.

这种采用补码求和的方式，并不能保证报文的完整性，攻击者可以很容易地伪造出一个拥有相同校验和的恶意载荷，轻易的绕过校验机制. 而校验和字段又是 TCP 头部字段中的必选项，鉴于当前终端计算资源和能力的极大提升，我们提出基于哈希的链式验证，对 TCP 校验机制进行增强改进，从而抵御多样化的网络攻击.

通过改进传输层的检验机制，系统化的解决安全问题，一方面可以直接保护上层的所有应用，避免为不同的应用程序定制冗余复杂的专有安全防御机制；另一方面也可以很好的兼容现有的网络基础设施，通过在终端节点上部署传输层安全防御机制，避免改动数据传输路径上的中间路由器、中间盒子等设备，易于部署. 此外，不同于 IP 层的语义信息有限（关注于 IP 分组的路由和转发），在传输层进行安全保护，能够实现对 IP 层安全问题的覆盖和兼顾，还可以保证端到端的细粒度语义控制，有效降低性能损失.

2 基于轻量级链式验证的网络安全性增强方法

由于现有的传输层校验机制存在安全缺陷，不能保证传输层报文的安全性，本章我们详细阐述本文提出的一种基于轻量级链式验证的传输层安全校验机制 LightCTL. LightCTL 通过改进传统传输层协议的校验和机制，使报文校验和的计算，不再只依赖于当前报文，而是依赖于整个报文流，从而对抗攻击者对报文的预测、窃取、伪造等. LightCTL 安全机制的部署，不依赖于 TLS、QUIC 等复杂加密协议，仅对传输层的校验和字段进行重新生成，一方面保证了轻量级的性能开销；另一方面可以有效认证报文的发送源端、保护整个报文的完整性（包括头部字段），从而抵御当前多样化的网络攻击. TLS、QUIC 等安全协议工作在传输层之上，而 LightCTL 机制工作在传输层本身. LightCTL 通过改进传输层的校验机制，使传输层本身具备了内生的安全防御能力. 需要说明的是，LightCTL 机制并不旨在取代已有的安全协议，LightCTL 机制可以和已有的安全协议相互补充、协作防护，比如 LightCTL 可以有效抵御攻击者对传输层本身的破坏（例如阻止攻击者伪造 RST 报文从传输层直接恶意阻断网络连接），从而为上层的协议及应用提供安全的基础层，弥补 TLS 等上层安全协议对传输层的防御不足. 但 LightCTL 是通过改进传输层的校验机制实现的轻量级安全防御机制，它并不具备数据加密的功能，因此用户数据的加密保护，还需依赖于 TLS、QUIC 等协议.

2.1 设计需求

LightCTL 面临的设计需求主要包括以下几个方面：

（1）可抵御多样化网络攻击. 受 LightCTL 机制保护的主机，可以有效抵御恶意报文注入、网络连接恶

意阻断、报文重放等各种攻击威胁，显著提升主机的安全性. 这一设计需求在我们的方案中可以很好的满足. 通过采用新型链式哈希验证的方式，通信双方能够对传输的报文形成彼此可验证的共识，避免了攻击者对报文的恶意篡改和注入.

（2）易于部署. LightCTL 的部署仅需改动终端主机的协议栈，而无需对中间网络设备，如路由器等进行改动，这样易于 LightCTL 在实际中的广泛部署. 我们通过仅改动终端主机协议栈中校验和部分的代码，来满足这一设计需求.

（3）预防密钥泄露攻击. LightCTL 要能够有效缓解密钥泄露引起的安全风险. 不同于 TCP MD5 机制，LightCTL 采用了链式验证的方式保证了整个 TCP 流的完整性，这样即使通信双方的密钥泄露，那么对于旁路条件下的攻击者（即攻击者不在通信路径上）而言，它很难伪造出一个恶意报文注入到目标连接中，因为每个报文的验证还依赖于前面的历史报文. LightCTL 的这一安全特性优于 TCP MD5 机制.

（4）低开销. LightCTL 引入的性能损失必须在可接受的范围内，不能明显降低传输吞吐. 我们在 LightCTL 中计算链式哈希校验和时，采取了轻量级的 DJB 哈希算法，因此方法开销小. 此外，我们为每一条传输层连接维护一个逻辑上的校验链. 针对每一个校验链，LightCTL 会为其分配一个 16 位字的变量，存储该连接上前序数据包的校验和. 该变量值也参与后续报文校验和的计算，以此来实现链式校验和的传递和维护. 因此，存储校验和的变量数目和主机上并发的 TCP 连接数量一致，维护开销和存储开销不会给主机引入过多的负载. 例如，如果主机并发有 10,000 条 TCP 连接，实际的存储开销仅为 10,000*16 bit=20KB.

下面，我们给出 LightCTL 的详细设计方案，以及在 Linux 系统上的实现与评估. 我们的方案可以很好的满足上述 4 个需求.

2.2 方法设计

如图 2 所示，我们重新设计传输层报文的校验和机制，采用链式哈希计算的方式，生成报文中可验证的 checksum 字段. 每一个传输层报文校验和的计算，是根据当前报文数据和上一个报文的校验和计算获得，形成一个完整的校验链，从而对抗攻击者的伪造和破坏.

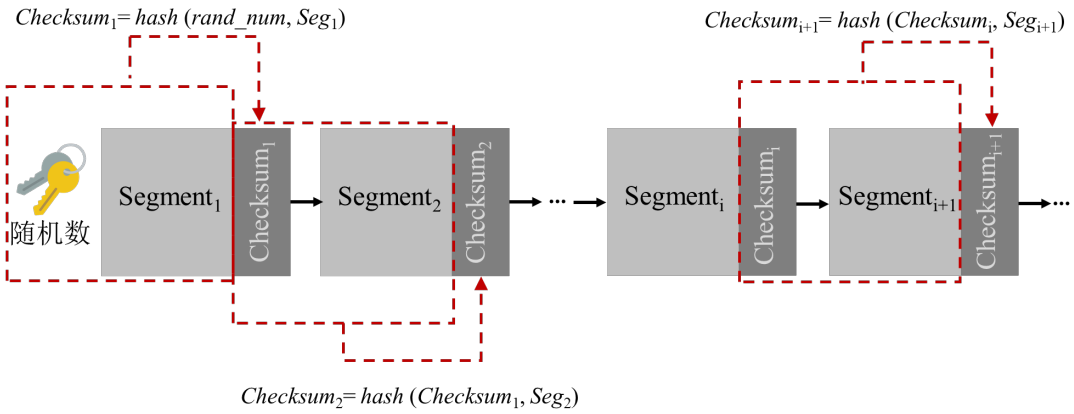

图 2 基于链式哈希传递的传输层报文校验机制

例如图 2 中，第 i+1 个传输层报文的校验和 $checksum_{i+1}$ 是由上一个报文（即第 i 个报文）的校验和 $checksum_i$ 和当前报文共同计算生成的. 计算生成校验和后，将该值填充到报文中，并随报文一起发送给接收方. 接收方接收到报文后，首先取出本地存储的上一报文的校验值，然后结合当前接收到的报文一起计算校验和，如果计算结果和报文中的填充值相同，则说明该报文确实是连接的对等端发送过来的，不是恶意或伪造的报文，接收该报文，否则丢弃.

对于攻击者而言，即使其成功猜测出 TCP 连接的序列号和确认号、或者成功构造出一个恶意的 IP 分片注入到目标 TCP 流中，但其很难获得初始的随机数以及当前 TCP 数据流中上一报文的校验值，因此攻击者

伪造出的恶意报文校验和字段，经接收方验证，必定错误，报文会被丢弃. 这种新型的传输层报文验证方式，使传统报文的校验和字段信息不再孤立，具备了链式完整性传递和验证的功能，可以有效抵御攻击者针对报文的破解、伪造等威胁，实现传输层安全能力的增强.

此外，由于 LightCTL 在链式计算、验证报文校验值时，不需存储整个数据流上所有报文的校验值，而是仅存储上一报文的校验值（即一个 16 位字），因此不会花费太多额外的存储开销. 同时，校验值的计算复杂度与当前的计算方法完全相同，主要是移位计算操作，也没有额外的计算开销. 因此，LightCTL 是轻量级的，不会给原生的 TCP 协议引入过多的性能损失.

2.3 算法流程

图 3 给出了通信双方在一次具体的数据传输过程中，LightCTL 是如何工作的，以及如何有效保证数据报文的安全性. 假设通信网络环境是不安全的，Alice 和 Bob 之间的数据通信可能会遭到恶意攻击者的破坏. Alice 和 Bob 之间的传输层协议为 TCP 协议，LightCTL 的工作流程如下：

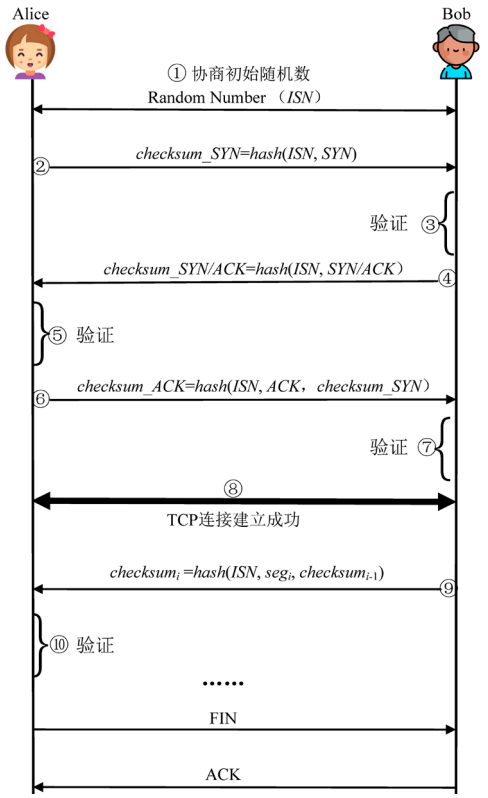

图 3 LightCTL 工作流程

①通信双方 Alice 和 Bob 通过某种方式如可信第三方等，协商出一个随机数 Random Number，作为共识源点，用于后续计算链式哈希校验值. 在实际的实现过程中，简单起见，双方可以事先确定好选择 TCP 连接的初始化序列号 ISN 作为随机数. LightCTL 会为每一条连接维护一个逻辑上的校验链，而每一个校验链都有自己的哈希密钥. 当采用连接的初始化序列号 ISN 作为哈希密钥时，不同连接如果发生 ISN 复用，那么这会影响哈希算法的抗碰撞性. 在 LightCTL 中，ISN 复用的概率等同于在 2^{32} 空间内发生碰撞的概率，已有工作表明，在系统没有实现漏洞的情况下，随机化序列号的碰撞和猜测非常困难[1]. 例如，Linux 系统中，每一条连接的 ISN 是由函数 siphash_3u32 ()随机生成的，该函数的显著特点之一就是可以有效缓解 hash flooding 碰撞问题[30].

②Alice 向 Bob 发出 TCP 连接建立请求. Alice 首先发送一个 TCP SYN 报文，报文的校验和 checksum 字

段值为 checksum_SYN=hash(ISN, SYN)，其中函数 hash()是对报文数据部分和共识源随机数 ISN 进行哈希计算的函数，可采用 DJB 等经典哈希算法[29]. 至此，得到客户端 Alice 到服务器 Bob 数据流方向上，第一个报文的校验和 checksum_SYN.

③服务器端 Bob 在接收到第一个 SYN 报文后，进行验证. 利用函数 hash()对本地存储的共享随机数（或者报文中直接携带的 ISN）和接收到的 SYN 报文数据部分，进行哈希计算. 如果计算结果等于报文中携带的 checksum_SYN，则报文认证合法，接收报文，同时存储 checksum_SYN 到本地.

④Bob 发送应答报文 SYN/ACK，报文的校验和 checksum 字段值 checksum_SYN/ACK=hash (ISN, SYN/ACK)，checksum_SYN/ACK 表示服务器端到客户端数据流方向上，第一个报文的校验和.

⑤Alice 进行验证，利用本地存储的共享随机数和接收到的 SYN/ACK 报文，进行哈希计算. 如果计算结果等于报文中携带的 checksum_SYN/ACK，则报文认证合法，接收报文. 同时存储 checksum_SYN/ACK 到本地.

⑥Alice 发送 ACK 报文，报文的校验和字段值为 checksum_ACK=hash (ISN, ACK, checksum_SYN）.

⑦Bob 进行验证，利用本地存储的 checksum_SYN 和接收到的 ACK 数据部分进行哈希计算，若结果等于 checksum_ACK，则接收报文，并更新存储 checksum_ACK.

⑧至此，TCP 连接建立完毕.

⑨服务器端 Bob 向客户端 Alice 推送数据. Bob 发送报文 seg_i，报文的校验和字段值 $checksum_i$=hash(ISN, seg_i, $checksum_{i-1}$)，其中 $checksum_{i-1}$ 表示服务器端存储的发给客户端的上一个报文的校验和.

⑩客户端 Alice 进行验证，利用哈希算法对本地存储的 $checksum_{i-1}$ 和接收到的报文数据部分 seg_i 进行哈希计算，如果计算结果等于 $checksum_i$，则报文认证合法，接收报文，同时更新存储 $checksum_i$ 到本地. 重复过程⑨-⑩，直到 Bob 和 Alice 之间的通信过程结束. Alice 请求关闭 TCP 连接，Bob 确认后，连接关闭.

需要注意的是，传输层的连接是全双工的，Bob 给 Alice 发送数据的同时，Alice 可能也会给 Bob 发送数据，对于 Alice 到 Bob 方向的报文流，链式验证的方法与上述过程完全相同，区别仅在于发送方是 Alice，验证方是 Bob.

算法 1: 基于 *LightCTL* 的 TCP 报文校验和生成

```
输入:   包含伪首部的 TCP 报文 seg
输出:   一个 16 位字的哈希值
初始化: key = ISN
1:  if seg is the first seg:
2:      seed = seg.length()
3:  else:
4:      seed = getlastchecksum()
5:  seed = odder (seed)
6:  seg.checksum = 0
7:  if seg contains an odd number of octets:
8:      seg = padding (seg)
9:  end if
10: hash = 5381
11: foreach 16-bit word term_i in seg:
12:     hash = hash*seed*key+ term_i
13: end foreach
14: write (seg.checksum, hash)
15: store (seg.checksum)
```

算法 1 给出了基于 LightCTL 生成 TCP 报文校验和的过程. 算法的输入是当前需要计算链式哈希校验和的一个 TCP 报文 seg，seg 中包含了 12 字节的伪首部. 算法的输出是一个 16 位字的哈希值，作为当前报文 seg 的 checksum 值，填充到报文中. 注意在生成该报文的 checksum 值之前，该字段初始的填充值为 0.

我们对 13 种常见的哈希算法进行了实际测评，测评发现经典的 DJB 哈希算法在计算报文校验和的场景下，其抗碰撞能力和计算开销等方面都优于其他算法，因此我们采用了 DJB 算法作为 LightCTL 的哈希算法

（哈希函数的具体选择和测评结果详见 3.1.1 小节）. 首先，进行算法的初始化. 我们设置 TCP 连接的初始序列号 ISN 作为哈希密钥 key，然后，选择缓存的上一报文的校验和作为哈希种子 seed，注意，如果当前报文为该方向上的第一个报文，我们选择当前报文的长度作为哈希种子. 根据 DJB 哈希算法的要求，我们对哈希种子进行奇数化处理，选择离其最近的奇数，作为实际的哈希种子. 接着，我们初始化报文中 checksum 字段为 0.

接着进入到当前报文的链式校验和计算流程. 如果当前报文的大小为奇数个字节，那么我们对报文进行填充，在报文末尾填充 1 个字节的 0. 然后，我们扩展了 DJB 哈希算法，针对报文 seg 中的每一个 16 位字 $term_i$，通过将哈希种子 seed 和密钥 key 同当前的 hash 变量值迭代相乘，并与 $term_i$ 相加，最终计算出当前报文的哈希校验和，保存在 hash 变量中. 最后将写其入到报文 checksum 字段中，同时本地保存，供链式计算下一个报文的哈希校验和.

报文到达接收端以后，接收端采用算法 2 对报文进行验证. 如果计算出的当前报文哈希校验和与报文中携带的校验和字段匹配，那说明报文确实是由通信对端发送过来的，并且没有被污染或篡改过，则接收该报文，并本地存储报文校验和供校验下一个报文用. 否则，丢弃该报文，等待发送端超时重传，直到一个可通过验证的报文被接收，发送 ACK 报文给接收端. ACK 报文的校验和计算，同算法 1.

算法 2: 基于 *LightCTL* 的 TCP 报文校验和验证

```
输入:   接收到的 TCP 报文 seg
输出:   是否接收该报文
初始化: key = ISN
 1:  if seg is the first seg:
 2:      seed = seg.length()
 3:  else:
 4:      seed = getlastchecksum()
 5:  seed = odder (seed)
 6:  checksum=seg.checksum
 7:  seg.checksum = 0
 8:  if seg contains an odd number of octets:
 9:      seg = padding (seg)
10:  end if
11:  hash = 5381
12:  foreach 16-bit word term_i in seg:
13:      hash = hash*seed*key + term_i
14:  end foreach
15:  if hash == checksum:
16:      accept (seg)
17:      store (checksum)
18:  else:
19:      drop (seg)
```

需要说明的是，在实际网络环境中（网络未发生断网等故障），TCP 连接上的报文可能会发生 3 种异常情况，即：报文丢失、重复报文到达、报文到达乱序. 针对这 3 种异常情况，借助于 TCP 协议的面向连接、可靠传输等特点，LightCTL 能够很好的完成链式校验功能. 首先，针对第一种情况，当报文丢失时，TCP 协议的超时重传机制会触发发送端、重新发送丢失的报文. 所以，我们在实现 LightCTL 时，只需等待丢失的报文到达后，然后在 TCP 的接收缓存中再顺序计算链式校验和即可. 针对第二种情况，如果有重传引起的重复报文到达，那么 TCP 协议会依据序列号机制对重复报文丢弃，这也不会影响到 LightCTL 的链式计算过程. 针对第三种情况，如果数据包发生了乱序，LightCTL 会在 TCP socket 的接收缓存中先缓存乱序报文，然后再等待前序报文的到达. 等前序报文到达后再依次链式计算各个报文的校验和，然后提交给上层应用，这也与 TCP 协议的按序接收特征相吻合.

2.4 应用场景

LightCTL 可以被广泛部署于各种应用场景，为通信双方提供安全性保护. 针对两种典型的网络攻击模型（即 On-Path 攻击者和 Off-Path 攻击者），我们对方法的应用场景及防御效果进行分析说明.

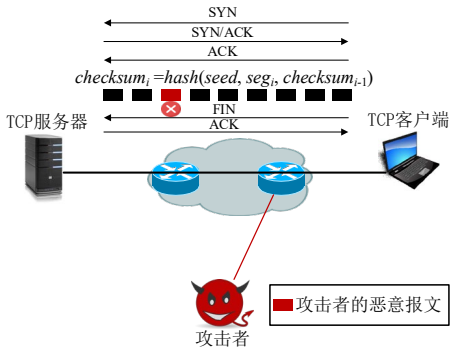

图 4 On-Path 网络攻击模型

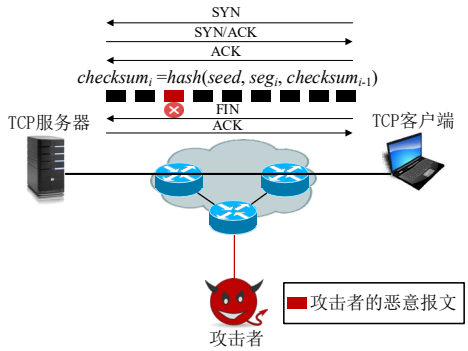

图 5 Off-Path 网络攻击模型

如图 4 所示，我们在 TCP 服务器和客户端上部署我们的 LightCTL 安全防御机制. TCP 服务器为客户端提供各种各样的服务，如消息订阅、视频点播、邮件推送等等.

首先在 On-Path 的攻击场景中，攻击者位于客户端和服务器的通信路径上，因此攻击者可以拦截、监听和修改客户端与服务器之间的通信报文. 在这种场景下，攻击者即使捕获到了服务器和客户端之间的 TCP 报文，此时如果它想冒充服务器注入一个报文到客户端，假定它填充了正确的端口号、序列号和应答号，但由于攻击者不知道服务器和客户端之间的报文校验和生成算法、也不知道二者的哈希密钥，所以攻击者伪造的报文不能顺利通过接收端的校验插入到哈希链中，会被丢弃.

如果攻击者想篡改正常报文的载荷，那么篡改后的报文到达接收端后，经接收端再次哈希校验后，会发现基于当前报文计算出的校验值和报文中携带的校验值不匹配，导致报文被丢弃. 最后，如果攻击者想要丢弃合法报文，那么 TCP 本身的超时重传机制，保证了发送端总会把正常报文传递到接收端. 因此，对于 On-Path 类型的攻击者，LightCTL 能够很好的保护 TCP 通信双方的数据传输.

如图 5 所示，对于 Off-Path 类型的攻击者而言，要想破坏服务器跟客户端之间的 TCP 连接更加困难. 在这种威胁模型下，攻击者不在客户端和服务器的通信路径上，也无法拦截、监听和修改客户端与服务器间的报文. 但攻击者对客户端和服务器均连通可达，可以向二者发送报文. 同理，由于攻击者不能伪造出可通过接收端验证的报文校验和，因此，不论攻击者注入的 TCP 报文、还是利用 IP 分片篡改后的 TCP 报文，都将被接收端丢弃. 因此，LightCTL 也可以很好的抵御 Off-Path 类型的网络攻击.

3 实现与评估

我们在 Linux 内核版本 5.0.1 上实现了 LightCTL，并从安全性和性能两方面对 LightCTL 进行了评估.

3.1 LightCTL实现

3.1.1 Hash函数选择

哈希函数的基本功能是将任意长度的数据映射到有限长度的域上，形成数据的摘要. 整体上哈希函数分为 2 大类，密码学哈希函数和非密码学哈希函数[32]. 密码学哈希函数（如 SHA256）的输入长度可变，并且相比较非密码学哈希函数具有更强的抗碰撞特性，但通常操作和运算也更加复杂和耗时. 在本文提出的 LightCTL 机制中，由于哈希计算的输入长度固定（为受路径 MTU 约束的 TCP 报文长度），并且计算引入的额外开销不能产生明显的网络延迟，同时我们通过实际测评发现在网络报文传输场景下非密码学哈希函数也具有良好的抗碰撞特性，因此基于以上三个原因，本文采用了非密码学的哈希函数作为 LightCTL 的密码函

数. 需要说明的是，LightCTL 机制中哈希函数的选择不是固定唯一的，管理员也可以按需替换哈希函数，甚至采用密码学的哈希函数.

我们对 13 种常用的非密码学哈希函数，即 BOB，OAAT，Simple，SBOX，TWMX，Hsieh，RSHash，JSHash，BKDR，DJBHash，NDJBHash，DEKHash，APHash 进行了分析，我们从性能和抗攻击能力 2 个方面入手，对这 13 种函数进行了仔细对比. 我们对 10000 个随机生成的字符串进行哈希操作，这些字符串的长度在 700-1400 字节之间（TCP 报文的长度通常不超过 1500 字节），然后分析哈希计算时间和碰撞的概率.

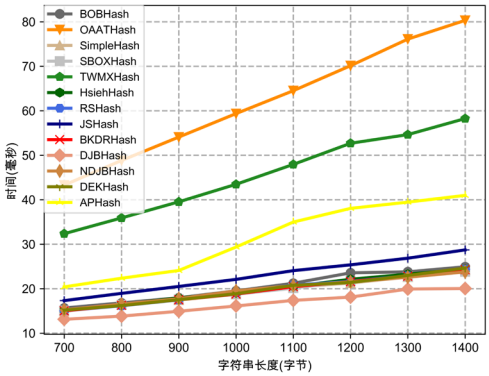

图 6 哈希函数性能对比

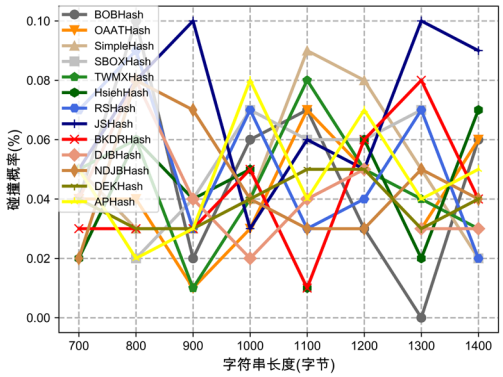

图 7 哈希函数冲突概率对比

具体而言，我们首先分别随机生成长度为 700-1400 字节的字符串，每种长度各 10000 个字符串，然后开始测试. 我们创建一个线性哈希表，逐个对字符串执行哈希运算，装填进哈希表中，发生哈希冲突时，我们以链表形式装载该字符串. 最后我们统计整个过程所消耗的时间，以及发生碰撞的概率（即碰撞字符串的个数/总字符串数）.

如图 6 所示，随着字符串长度从 700 个字节增长到 1400 个字节，几乎所有哈希函数消耗时间也相应增加，其中 DJB 哈希函数增长得最少，为 52.7%，而 AP 哈希函数增长得最多，为 100.9%. 对于 700 到 1100 字节长度的输入，大多数哈希函数的计算时间都在 15~30ms 之间，其中 DJB 哈希函数的计算时间均为最短，而 OAAT 哈希函数的计算速度则最慢.

表 2 Hash 函数综合评估

哈希函数	时间(ms)	碰撞概率(%)	性能得分	抗碰撞性得分	综合得分
DJB	16.67	0.0413	1	0.939	0.97
DEK	19.68	0.0388	0.846	1	0.92
BKDR	19.58	0.0413	0.851	0.939	0.90
Hsieh	19.85	0.0413	0.840	0.939	0.89
NDJB	19.69	0.045	0.846	0.861	0.85
Simple	19.50	0.0488	0.855	0.795	0.82
SBOX	19.52	0.0488	0.854	0.795	0.82
BOB	20.42	0.0475	0.816	0.816	0.82
RS	19.68	0.0525	0.847	0.738	0.79
AP	31.21	0.0475	0.534	0.816	0.67
JS	22.98	0.07	0.725	0.554	0.64
TWMX	45.60	0.045	0.366	0.861	0.61
OAAT	62.08	0.0413	0.268	0.939	0.60

在图 7 中，随着字符串输入长度从 700 个字节到 1400 个字节，哈希表的碰撞概率在 0%至 0.1%之间波动，对于长度在 700-1400 字节的所有字符串中，DEK、DJB、BKDR、Hsieh、OAAT 哈希函数的平均碰撞概率较小，在我们的实验中，使用这五个哈希函数时，10000 个字符串平均仅发生三到四次碰撞.

我们对这 13 种哈希函数进行了综合评价，结果如表 2 所示，每个哈希函数的综合得分由其计算时间、碰撞概率按照百分制比例计算得分后，求平均获得. 结果表明 DJB 哈希函数的抗碰撞能力等指标最优，并且计算开销小. 因此我们采用了 DJB 算法作为 LightCTL 的哈希算法.

3.1.2 Linux系统实现

在 Linux 5.0.1 内核中，TCP 发送报文的校验和，主要由函数 csum_partial_copy_from_user ()计算生成. 在该函数中，会调用函数 csum_partial ()，对用户提交的数据进行累加和校验计算，生成校验和保存到数据结构中. 函数的部分代码示例如下.

```
unsigned int csum_partial_copy_from_user (
    const unsigned char *src,
    unsigned char *dst,
    int len,
    int sum,
    int *err_ptr
)
{
    if (copy_from_user (dst, src, len))
    {
        /*拷贝用户空间数据到内核空间*/
        *err_ptr = -EFAULT;
        /* bad address */
        return (-1);
    }
    /*计算用户数据的校验和，并将其写入到 skb->csum
    中*/
    return csum_partial(dst, len, sum);
}
```

基于前述的算法 1，我们对函数 csum_partial ()进行扩展，形成对 TCP 报文的链式哈希校验，生成报文的校验和，并填充到 TCP 报文头部的 checksum 字段中.

```
static int tcp_v4_checksum_init (
    struct sk_buff *skb
)
{
    /*验证 TCP 报文的伪首部是否正确*/
    skb->csum= csum_tcpudp_nofold (
        skb->nh.iph->saddr,
        skb->nh.iph->daddr,
        skb->len,
        IPPROTO_TCP,
        0
    );
    /*验证数据部分是否正确*/
    if (skb->csum)
        return __skb_checksum_complete (skb);
    /* bad TCP segment*/
        return (-1);
}
```

对于接收到的 TCP 报文，接收方主要依据函数 tcp_v4_checksum_init ()对报文的校验和再次计算，判断其是否等于报文中携带的 checksum 字段，相等则接收该报文，否则丢弃. 函数示例代码如下所示. 接收校验的第一部分，主要是对 TCP 报文的伪首部进行初步校验. 当伪首部的校验和正确时，再调用函数 __skb_checksum_complete ()完成对数据部分的校验. 分两步验证校验和，可以提高报文接收处理的效率. 如果第一步伪首部的验证不通过，那么就避免了第二步数据处理部分的计算，提高了处理速度. 基于上述我们

提出的算法 2，我们扩展了函数__skb_checksum_complete ()，使得接收端能够依据我们提出的链式哈希校验的方式，对报文的发送源端和完整性进行安全验证，避免了攻击者的注入攻击.

3.2 安全评估

LightCTL 可以作为一种基本的安全防御机制，部署在终端主机操作系统的 TCP/IP 协议栈中，对每一个 TCP 连接上的报文进行安全校验，保证通信双方的安全性. 我们对 LightCTL 的安全性首先进行验证，评估其是否能够有效抵御当前多样化的网络攻击. 同时我们将 LightCTL 与其他传输层安全机制进行对比，如通信时开启 TCP MD5 选项、部署 TLS 机制等，综合对比评估 LightCTL 的安全防御效果，评估结果如表 3 所示.

实验拓扑设置如图 8 所示. Web server 为客户 Client 提供在线的 Web 服务. SSH server 为客户 Client 提供远程 SSH 连接服务. Web server 和 SSH server 的主机配置为 ARM，Ubuntu 20.04 操作系统，以及 Apache 服务器软件和 SSH 服务器软件. Client 的主机配置为 ARM，Ubuntu 20.04 操作系统. 在与 TCP MD5 机制进行对比时，由于 TCP MD5 选项在 Linux 内核中没有默认开启，我们重新配置了内核编译选项并在通信时将其开启；在与 TLS 机制进行对比时，我们对 Apache 服务器配置了相关证书，模拟用户通过 HTTPS 加密访问 Web 数据.

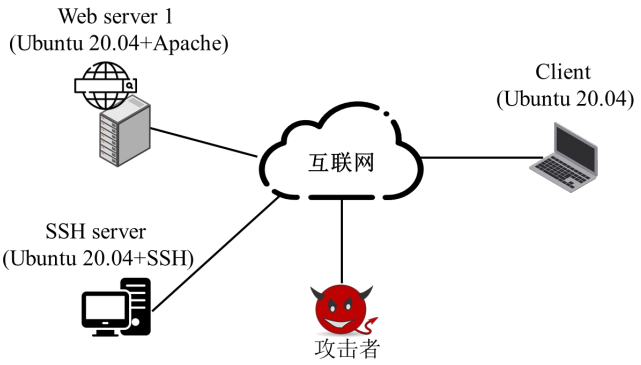

图 8　LightCTL 安全性评估实验拓扑图

依据两种攻击模型，我们分别在 On-Path 和 Off-Path 场景下，进行了实验验证，评估 LightCTL 的安全性. 首先，我们在 Web server 服务器和 Client 上部署了我们的 LightCTL 安全机制，即两端的主机配备的是 Linux 5.0.1 内核，内核中的 TCP 报文校验和机制已被我们扩展替换. 在 TCP 客户端和 TCP 服务器之间建立一条 TCP 连接，在我们的实验中，我们建立了一条应用层的 HTTP 通信会话，在传输层由一条 TCP 连接承载. 一个恶意的攻击者企图破坏服务器和客户端之间的通信流量. 我们假定攻击者已经具备了很强的攻击能力，包括：攻击者可以进行源 IP 地址伪造，伪装成客户端或服务器欺骗对方；攻击者可以检测出服务器和客户端之间 TCP 连接的存在；攻击者可以检测到服务器和客户端之间 TCP 报文的分片情况；攻击者可以通过侧信道等漏洞，推理出 TCP 报文的序列号和应答号等.

在原生 Linux 内核中，如果攻击者具备这些攻击能力，可以利用 Scapy 库发送伪造 TCP 段. 对于 Off-Path 攻击，如在基于控制报文的连接阻断攻击中，攻击者只需要向 SSH server 发送一条 TCP RST 段即可将 SSH 连接阻断. 如果我们采用同样手段对部署了 LightCTL 防御机制的主机进行了攻击，无论双方的通信密钥是否丢失，此时攻击均会失败，因为攻击者注入的 RST 报文会被连接丢弃掉. 对于开启了 TCP MD5 选项的 TCP 会话而言，其只有在密钥未发生泄漏的情况下能进行防御，一旦通信密钥被窃取，攻击者仍然可以伪造出 TCP RST 报文阻断该连接；对于开启了 TLS 加密机制的 TCP 会话而言，由于 TLS 协议只是保证了上层数据的机密性，而 TCP RST 报文作用于传输层本身，仍然可以直接阻断该连接，因此无法防御该攻击.

在基于数据报文的流量污染攻击中，我们对开启了 Apache 服务的 Web server 进行攻击，攻击者需要发送一条伪造 TCP 数据段，即可将伪造消息注入到客户端，从而篡改客户端看到的 Web 数据内容. 如果我们采用同样手段对部署了 LightCTL 防御机制的主机进行了攻击，无论通信密钥是否泄漏，此时攻击均会失败，

因为攻击者注入的 TCP 报文会被丢弃掉，导致攻击无法成功实施，这是由链式校验的特性所保障；TLS 加密机制也能抵御该攻击，而 TCP MD5 机制则只能在密钥未泄漏的情况下能够抵御.

在伪造 ACK 操控流量速率攻击中，我们对开启了 Apache 服务的 Web server 进行攻击，攻击者发送多条伪造 TCP ACK 段，提前确认数据，从而影响通信链路 RTT 的测量. 我们同样对部署了 LightCTL 防御机制的主机进行了攻击，无论通信密钥是否泄漏，此时攻击均会失败. 和基于控制报文的连接阻断攻击中情况相同，TLS 加密机制无法防御该攻击，而 TCP MD5 机制只有在密钥未泄漏的情况能够防御.

在基于 IP 分片的流量污染攻击中，我们对开启了 Apache 服务的 Web server 进行攻击，攻击者不需要知道当前 TCP 连接的源端口号、序列号、应答号等，只需通过发送多个伪造的 IP 分片数据包、迫使合法分片和伪造分片在客户端发生错误重组，即可将伪造消息注入到客户端. 相反，如果我们采用同样手段对部署了 LightCTL 防御机制的主机进行了攻击，此时攻击会失效，无论通信密钥是否泄漏，因为攻击者不知道当前报文的上一个报文的校验和，污染的 TCP 报文无法通过 LightCTL 的校验. TLS 加密机制也能抵御该攻击，而对于 TCP MD5 机制，一旦发生密钥泄漏则无法抵御该攻击.

对于 On-Path 攻击，攻击者的能力模型强于 Off-Path 类型的攻击者，攻击者能够监听到服务器和客户端之间的 TCP 会话报文，即他自然而然地具备我们先前对 Off-Path 攻击者较强能力的假设. 我们同样进行了上述攻击. 我们还在此情景下进行了报文重放攻击（中间攻击者缓存 Server 和 Client 之间的报文，一定时间后再次发送）、报文恶意丢弃攻击（中间攻击者随机丢弃 Server 和 Client 之间的报文）. 此外，我们对部署了 LightCTL 的 Server 发送大量 SYN 握手请求，进行 SYN 洪泛攻击.

即使在攻击者具备这些攻击能力之后，我们实验发现，在 TCP 连接受到 LightCTL 保护之后，由于攻击者无法获取 TCP 通信双方共享的哈希密钥，或者难以获得链式数据流上的之前数据段的校验和，因此攻击者发送的报文不能逃逸我们的防御机制，将被终端用户丢弃，即攻击者很难破坏通信双方的流量或内容. 表 3 列举了我们的评估效果，针对不同的网络攻击，可以看出 LightCTL 可以起到很好的保护效果. 我们测试了 8 种攻击，针对表 3 中的前 7 种典型恶意攻击，LightCTL 能够成功的防御.

表 3 LightCTL 安全防御效果评估（“是”代表可以抵御对应攻击类型，“否”代表不能抵御）

编号	攻击类型	攻击场景	LightCTL		TCP MD5		TLS
			密钥泄漏	密钥未泄露	密钥泄漏	密钥未泄漏	
1	基于 IP 分片的流量污染	Off-Path	是	是	否	是	是
2	基于数据报文的流量污染	Off-Path	是	是	否	是	是
3	基于控制报文的连接阻断	Off-Path	是	是	否	是	否
4	伪造 ACK 操控流量速率	Off-Path	是	是	否	是	否
5	中间人攻击	On-Path	否	是	否	是	是
6	报文重放	On-Path	是	是	是	是	是
7	报文恶意丢弃	On-Path	是	是	是	是	是
8	SYN 洪泛攻击	Off-Path	否	否	否	否	否

当攻击者位于 TCP 双方通信路径上、或者攻击者是一个中间人时，它能够成功监听到服务器发送给客户端的 TCP 报文. 但如果我们将 LightCTL 中的哈希密钥设置为双方提前已知的一个随机数，那么攻击者注入的报文或分片，将不能逃逸掉 LightCTL. 因为在 LightCTL 中，报文校验和的计算，不仅依赖于整个 TCP 的传输报文链，还依赖于哈希密钥或种子. 而一旦哈希密钥发生泄漏，由于中间人天然能获取到整条链路上的报文，因此当加密算法公开时，攻击者能够通过伪造注入恶意报文，同样 TCP MD5 机制也无法防御，而 TLS 机制由于其基于混合加密的协议特性，能够保护通信双方不受中间人攻击的威胁.

对于第 6 种报文重放攻击，攻击者将之前捕获的报文再次发给 TCP 接收端，企图干扰接收端对正常报文的接收. 在这种情况下，LightCTL 所依赖的 TCP 协议，会缓解这种攻击. 因为 TCP 协议识别到重复报文后，会对其直接丢弃，导致报文重放攻击失败.

在第 7 种报文恶意丢弃攻击中，一个路径上的攻击者，它无法注入恶意载荷，但它选择丢弃原始的 TCP

报文，造成 DoS 攻击的效果. 这种情况下，LightCTL 仍然能够有效规避，因为 TCP 机制本身具备超时重传机制，如果攻击者丢弃掉了部分报文，发送端在接收不到应答报文、发生超时后，会重传报文给接收端. 这在一定程度上缓解了攻击者的恶意丢弃，但如果恶意攻击者是 ISP 粒度的，它持续的丢包，这种攻击可能需要互联网策略管理层面的更高粒度的防御办法来解决.

对于第 8 种 SYN 洪泛攻击，由于攻击者对服务器的访问请求和合法的访问请求一样，LightCTL 并不能对这种攻击进行有效防御，但已有的防御技术如 SYN cookie 等，已经能够较好的防御这种攻击.

需要注意的是，网络层的安全机制 IPsec 也可以实现对网络数据流的安全保护. 不同于 IPsec，本文研究提出的 LightCTL 是一种传输层的安全防御机制，二者位于 TCP/IP 协议栈的不同层次. 相比较 IPsec，LightCTL 是一种轻量级的安全机制，只需对终端主机传输层原有的校验和机制进行改进，不需路由器的支持. 此外，LightCTL 可以作为 IPsec 的补充，为没有部署 IPsec 的主机提供安全保护.

作为一种传输层安全补充防御机制，LightCTL 由于其链式验证的特性，能够抵御多种攻击. 但出于性能的考虑，LightCTL 采用了轻量级的哈希算法，因此其也存在一定的局限性，比如极端情况下，攻击者可能会尝试破解哈希算法，进而破坏双方的通信过程. 我们通过实验测试分析了攻击者破解 DJB 哈希算法的可行性. 在实验中，我们设定 DJB 哈希算法的密钥是 32 位的随机数，假设一个 On-Path 攻击者捕获了客户端和服务器间的通信报文，试图通过枚举 32 位长度的空间来破解密钥. 我们在一台内存 16GB 的 Apple M1 Pro 电脑上进行了破解时间测试. 首先，我们基于 DJB 哈希算法和 32 位随机密钥计算获得了 TCP 报文的哈希值；然后，我们模拟攻击者枚举 32 位长度的密钥空间计算相同 TCP 报文的哈希值，记录攻击者破解出正确密钥所需的时间. 多次重复实验结果表明，攻击者破解出 32 位随机密钥平均需要约 2.23 小时. 图 9 给出了破解密钥所需时间的 CDF（Cumulative Distribution Function，累计分布函数）图.

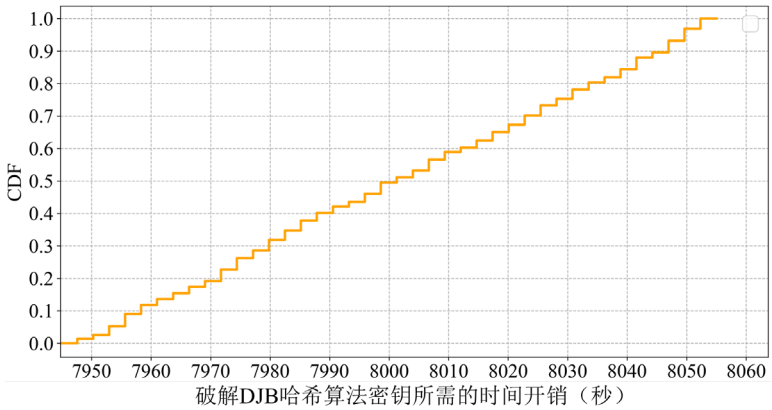

图 9　破解 DJB 哈希算法 32 位密钥所需的时间累计分布图

这种破解复杂度在实际中通常不容易实现，因为这要求目标连接的持续时间至少要在 2.23 小时以上. 此外，如果我们将密钥扩展到 64 位，甚至 128 位，那么破解将变得更加困难，甚至理论上不可行. 需要说明的是，为了实现性能和安全性的折中，本文中我们选取了 DJB 哈希算法和 32 位的随机数作为密钥. 但在实际中，管理员在部署 LightCTL 机制时，如果追求更高的安全性，则可以采用更长的密钥，甚至在 LightCTL 中采用密码学的哈希函数，进一步提升攻击者的破解难度.

3.3 性能评估

我们在实际环境中，对 LightCTL 的性能也进行了评估，验证 LightCTL 的引入是否会对 TCP 通信带来明显的性能损失. 我们从两个方面对 LightCTL 的性能进行了评估. 首先我们对比了 LightCTL 和原生 TCP 协议栈，然后我们对比了 LightCTL 和 TCP MD5 机制在性能方面的差异.

3.3.1 LightCTL与原生TCP校验机制的对比

我们以文件传输为测试场景，评估了 LightCTL 和原生 TCP 协议栈性能上的差别. 首先我们采用原始的 Linux 5.0.1 内核部署了两台主机，一台充当文件服务器，另一台充当有文件下载需求的客户端. 两台主机的配置参数如下：服务器和客户端均为 ARM，Ubuntu 20.04 操作系统，二者的传输带宽平均值为 112.84MB/s.

我们记录了从客户端下载不同大小文件的时延. 文件的下载基于 FTP 协议，底层的传输协议是系统原生的 TCP，采用的报文校验机制是基于补码求和的校验机制. 然后，我们利用 LightCTL 机制扩展了两台主机的 TCP 校验和机制，重写了校验和部分的内核代码，并对两台主机重新进行了编译. 此时通信两端对传输报文的校验，不再是传统的补码和计算方式，而是基于我们提出的轻量级链式哈希校验的方式. 作为对比，我们在客户端向服务器请求下载同样大小的文件，然后记录时延，判别 LightCTL 引入的性能损失.

如图 10 所示，在两种情况下，我们分别从服务器端下载了 20 次不同大小的文件. 文件大小从 1KB 到 4GB 不等，然后记录下载每份文件的平均时延. 绿色曲线代表了采用原生 TCP 协议栈时，不同大小文件的下载时延，红色曲线代表了采用 LightCTL 时的下载时延. 平均而言，相比较原生的 TCP 协议栈实现，LightCTL 引入的性能损失在 2.84%以内.

3.3.2 LightCTL与TCP MD5的对比

TCP MD5 是 RFC 2385 提出来的一项规范标准，用于增强 TCP 协议的安全性. TCP MD5 是 TCP 协议的各个安全扩展中与本文研究方案最接近和最具对比性的，因为二者都采用了增强报文校验和的方式来提高数据传输安全. TCP 协议的其他一些安全扩展，如源端口随机化（RFC 6056）、Challenge ACK 机制（RFC 5961）等，则侧重于提高 TCP 连接的随机化程度，从而对抗攻击者的暴力猜测和报文注入，与本文方法的防御角度不同. 本节，我们对 LightCTL 和 TCP MD5 二者进行对比评估.

当前，TCP MD5 是作为 TCP 协议头部的选项之一来定义的（Kind=19，Length=18）. TCP MD5 值的计算是依据 TCP 伪首部、TCP 头、TCP 报文数据、以及 TCP 双方的共享密钥 4 个元素实现的. TCP MD5 机制在各个操作系统中并没有得以广泛的实现，例如在 Linux 操作系统中，TCP MD5 机制没有被默认支持. 如果用户想启动 TCP MD5 机制，必须重新配置编译选项. 我们在分析了多样化的网络攻击后，发现 TCP 协议现有的校验和机制作为必选项，但又存在严重的安全漏洞，所以我们提出了新的轻量级链式哈希验证机制，来替换 TCP 的校验和机制，使得对报文的安全验证成为必要功能，提高通信的安全性.

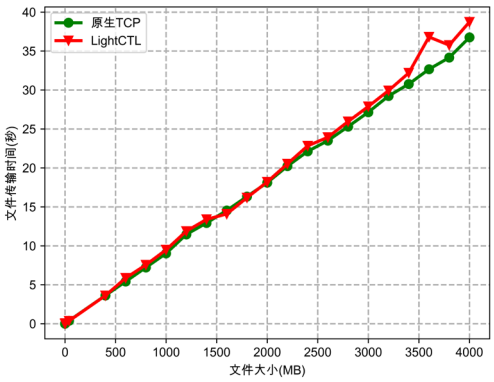

图 10 LightCTL 与原生 TCP 的对比图

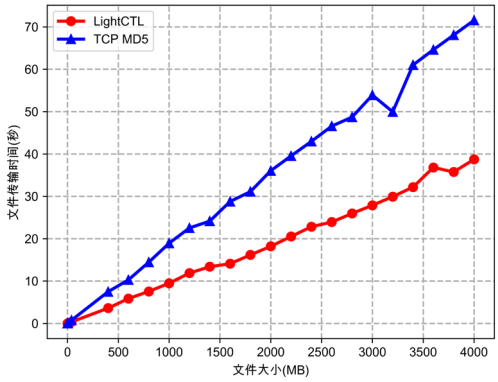

图 11 LightCTL 与 TCP MD5 的对比

此外，在攻击者不知晓客户端与服务器通信密钥的情况下，我们的基于哈希的链式验证校验方法可以使接收端保证整个链上的数据都是可信的，而 TCP MD5 机制只能保证当前报文的可信. 在通信双方密钥泄漏的情况下，即使采取 TCP MD5 机制，如果攻击者掌握了密钥，可以利用如 IPID 侧信道等方法构造出合法报文

完成攻击，但如果采用我们的基于哈希的链式验证校验方法，即使 Off-Path 攻击者掌握了密钥，但却不能获取通信双方上一个数据包的校验和，所以无法成功攻击. 已有研究工作表明，密钥泄露是一种常见的安全问题[31]，此外网络管理员为了便于记忆，可能会采用弱密钥，导致攻击者可以破解出 TCP MD5 密钥[32].

在性能方面，我们对比了引入 TCP MD5 机制和 LightCTL 给 TCP 通信带来的性能损失. 如图 11 所示，我们记录了在二者分别开启的情况下，从 TCP 服务器端下载不同大小文件到客户端的时延. 蓝色曲线代表了启用 TCP MD5 时的文件下载时延，红色曲线则代表了采用 LightCTL 时的下载时延. 整体而言，相比较 TCP MD5，LightCTL 能够较快的完成文件传输任务，性能平均提升 44.28%. 主要原因在于 LightCTL 采用了更加轻量级的 DJB 哈希算法.

4 总 结

本文在深入分析了当前 TCP/IP 协议栈面临的多样化网络攻击后，提出了一种基于轻量级链式验证的网络安全性增强方法. 针对当前 TCP/IP 协议栈传输层安全能力的不足，我们提出了 LightCTL，对可被攻击者轻易绕过的传统 TCP 校验和机制进行扩展和增强. 该方法基于哈希验证的方式，使 TCP 连接双方能够对传输层报文形成彼此可验证的共识，避免攻击者或中间人窃取和伪造敏感信息，从而解决网络协议栈面临的典型安全威胁. 我们在 Linux 内核版本 5.0.1 上实现了 LightCTL，并从安全性和性能两方面对 LightCTL 进行了验证和评估. 实验结果表明 LightCTL 可以有效抵御多样化的网络攻击，包括基于 IP 分片的 TCP 流量污染攻击，基于 IPID 侧信道的 TCP 连接重置攻击及劫持攻击、中间人攻击、报文重放攻击等. LightCTL 不需要修改中间网络设备如路由器等的协议栈，只需对终端协议栈中的校验和相关部分进行修改，因此方法易于部署. 通过在实际网络中进行部署评估，结果表明 LightCTL 引入的性能损失也非常小. 相比较原生的 TCP 协议栈，LightCTL 引入的性能损失在 2.84%以内. 与 TCP MD5 机制对比，LightCTL 可以将传输性能平均提升 44.28%.

References:

[1] Feng X W, Fu C, Li Q, et al. Off-path tcp exploits of the mixed ipid assignment//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. Virtual event. 2020: 1323-1335.

[2] Chou Y F, Yi B, Wang X W, et al. A rapid defense mechanism based on mf-dl for ddos attack in ipv6 networks. Chinese Journal of Computers, 2021, 44(10): 2047-2060.

[3] Nam S Y, Jurayev S, Kim S, et al. Mitigating arp poisoning-based man-in-the-middle attacks in wired or wireless lan. EURASIP Journal on Wireless Communications and Networking, 2012, 89(1): 1-17

[4] Luckie M, Beverly R, Koga R, et al. Network hygiene, incentives, and regulation: deployment of source address validation in the internet//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. London, UK. 2019: 465-480.

[5] Ensafi R, Knockel J, Alexander G, et al. Detecting intentional packet drops on the internet via tcp/ip side channels//Proceedings of International Conference on Passive and Active Network Measurement. Los Angeles, United States. 2014: 109-118.

[6] Ensafi R, Park J C, Kapur D, et al. Idle port scanning and non-interference analysis of network protocol stacks using model checking//Proceedings of USENIX Security Symposium. Washington, DC, United States. 2010: 257-272.

[7] Herzberg A, Shulman H. Fragmentation considered poisonous, or: one-domain-to-rule-them-all.org//Proceedings of IEEE Conference on Communications and Network Security (CNS). Washington, DC, United States. 2013: 224–232.

[8] Brandt M, Dai T, Klein A, et al. Domain validation++ for mitm-resilient pki//Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York, United States. 2018: 2060–2076.

[9] Zheng X F, Lu C Y, Peng J, et al. Poison over troubled forwarders: a cache poisoning attack targeting dns forwarding devices//Proceedings of 29th USENIX Security Symposium. Boston, United States. 2020: 577–593.

[10] Bellovin S M. Security problems in the tcp/ip protocol suite. ACM SIGCOMM Computer Communication Review. 1989, 19(2): 32–48.

[11] Bellovin S M. A look back at “security problems in the tcp/ip protocol suite"//Proceedings of the 20th Annual Computer Security Applications Conference. Washington, DC, United States. 2004: 229–249.

[12] Nakibly G, Kirshon A, Gonikman D, et al. Persistent ospf attacks// Proceedings of the 2012 Network and Distributed System Security Symposium. San Diego, United States. 2012: 1-12.

[13] Nakibly G, Sosnovich A, Menahem E, et al. Ospf vulnerability to persistent poisoning attacks: a systematic analysis//Proceedings of the 30th Annual Computer Security Applications Conference. New Orleans, United States. 2014: 336–345.

[14] Nordstrom O, Dovrolis C. Beware of bgp attacks. ACM SIGCOMM Computer Communication Review. 2004, 34(2): 1–8.

[15] Sermpezis P, Kotronis V, Dainotti A, et al. A survey among network operators on bgp prefix hijacking. ACM SIGCOMM Computer Communication Review. 2018, 48(1): 64–69.

[16] Cho S, Fontugne R, Cho K, et al. Bgp hijacking classification//Proceedings of 2019 Network Traffic Measurement and Analysis Conference. Paris, France. 2019: 25–32.

[17] Cao Y, Qian Z Y, Wang Z, et al. Off-path tcp exploits: global rate limit considered dangerous//Proceedings of the 25th USENIX Security Symposium. Austin, United States. 2016: 209-225.

[18] Cao Y, Qian Z Y, Wang Z, et al. Off-path tcp exploits of the challenge ack global rate limit. IEEE/ACM Transactions on Networking, 2018: 26(2): 765–778.

[19] Chen W T, Qian Z Y. Off-path tcp exploit: how wireless routers can jeopardize your secrets//Proceedings of the 27th USENIX Security Symposium. Baltimore, United States. 2018: 1581–1598.

[20] Man K Y, Qian Z Y, Wang Z J, et al. Dns cache poisoning attack reloaded: revolutions with side channels//Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. Virtual event. 2020: 1337–1350.

[21] Man K Y, Zhou X N, Qian Z Y. Dns cache poisoning attack: resurrections with side channels//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. Virtual event. 2021: 3400–3414.

[22] Philipp J, Haya S, Michael W. The impact of dns insecurity on time//Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Valencia, Spain. 2020: 266—277.

[23] Chen J, Jiang J, Duan H, et al. Host of troubles: multiple host ambiguities in http implementations//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York, United States. 2016: 1516-1527.

[24] Pandove K, Jindal A, Kumar R. Email spoofing. International Journal of Computer Applications, 2010, 5(1): 27-30.

[25] Rahman F, Kamal P. Holistic approach to arp poisoning and countermeasures by using practical examples and paradigm. International Journal of Advancements in Technology, 2014, 5(2):82–95.

[26] Boneh D. The decision diffie-hellman problem//Proceedings of International Algorithmic Number Theory Symposium. Berlin, Heidelberg. 1998: 48-63.

[27] Qian Z Y, Mao Z M, Xie Y. Collaborative tcp sequence number inference attack: how to crack sequence number under a second//Proceedings of the 2012 ACM conference on Computer and communications security. New York, United States. 2012: 593-604.

[28] Qian Z Y, Mao Z M. Off-path tcp sequence number inference attack-how firewall middleboxes reduce security//Proceedings of 2012 IEEE Symposium on Security and Privacy. Oakland, United States. 2012: 347-361.

[29] Henke C, Schmoll C, Zseby T. Empirical evaluation of hash functions for multipoint measurements. ACM SIGCOMM Computer Communication Review, 2008, 38(3): 39-50.

[30] Aumasson J P, Bernstein D J. SipHash: a fast short-input PRF[C]// Proceedings of International Conference on Cryptology in India. Springer, Berlin, Heidelberg, 2012: 489-508

[31] Fumy, Walter, and Peter Landrock. "Principles of key management." IEEE Journal on selected areas in communications 11(5) 1993: 785-793.

附中文参考文献:

[32] 孙泽民, 芦天亮, 周阳. 基于 BGP 协议的 TCP MD5 加密认证的破解技术分析. 信息网络安全. 2015: (9), 37-40.

[33] 徐恪, 付松涛, 李琦, 刘冰洋, 江伟玉, 吴波, 冯学伟. 互联网内生安全体系结构研究进展. 计算机学报. 2021:44(11), 2149-2172.

冯学伟(1985－),男,博士,主要研究领域为网络安全与程序安全性分析.

徐恪(1974－),男,博士,教授,博士生导师,主要研究领域为互联网体系架构、网络安全.

李琦(1979－),男,博士,副教授,博士生导师，主要研究领域为互联网安全、人工智能安全.

杨宇翔(1999－),男,博士研究生,主要研究领域为网络安全.

朱敏(1979－),女,博士,高级工程师,主要研究领域为网络体系结构与网络安全.

付松涛(1982－),男,博士研究生,主要研究领域为网络安全和路由故障定位.