一种新的轻量级安全代理协议

吕英豪, 陈嘉耕

华中师范大学计算机学院信息安全系 武汉 中国 430079

摘要 随着网络技术的发展和广泛应用, 在互联网环境下建立安全信道愈发显得重要。我们设计了一种采用 TLSv1.3 的握手协议框架的轻量级安全代理协议, 在安全性的基础上提供了更好的隐蔽性和性能。代理程序的用户接口基于 Socks5 协议, 保障了通用性。握手过程模拟 TLS, 将实际参数填充在 TLSv1.3 握手包内的随机区域和加密区域中来完成基于 ECDHE 密钥交换和挑战响应机制的握手。后续的代理转发过程中通过额外判断避免了加密数据的重复处理, 大大提高了通信效率。针对主动检测, 设计了基于 TCP 转发的主动对抗措施。健壮性方面, 可作为服务器的反向代理, 亦可作为基于 TCP 转发的反向代理服务器的后端, 可灵活构建冗余信道。依据实现原理, 命名为 FTLSocks, 意为 Fake TLS Socks。使用了协程池、空间重用、最少拷贝和无锁的设计, 实测高并发下资源消耗、吞吐量、响应时间等均优于现有流行工具。

关键词 代理; 流量整形; 流量混淆; TLS; Socks5; DPI
中图法分类号 TP309.2 DOI 号 10.19363/J.cnki.cn10-1380/tn.2021.05.07

A novel lightweight protocol of secure proxy

LV Yinghao, CHEN Jiageng

School of Computer, Central China Normal University, Wuhan 430079, China

Abstract With the development and wide application of network technology, establishing a secure channel has been used in various application scenarios. In this paper, we have designed a lightweight security proxy protocol taking advantage of the handshake protocol framework of TLSv1.3, which provides better performance given strong security. The user interface of the agent is based on the Socks5 protocol, which guarantees the versatility. The handshake process simulates TLS, and the true parameters are filled in the random area and the encrypted area in the TLSv1.3 handshake packet to complete the handshake based on the ECDHE key exchange and challenge-response mechanism. The subsequent proxy forwarding process avoids the repeated processing of encrypted data through additional judgments, which greatly improves the communication efficiency. For active detection, an active countermeasure based on TCP forwarding is implemented. In terms of robustness, it can be used as a reverse proxy for the server, or as the back end of a reverse proxy server based on TCP forwarding that can flexibly be used to construct redundant channels. The scheme is named Fake TLS Socks (FLTSocks), which applies a coroutine pool, space reuse, minimal copy, and lock-free design. The measured resource consumption, throughput, and response time under high concurrency are better than the existing tools.

Key words proxy; traffic shaping; traffic obfuscation; TLS; socks5; DPI

1 引言

自互联网诞生以来, 各种攻击和窃密技术层出不穷。在各类网络边界中, 小至家庭、企业, 大至 ISP(Internet Service Provider, 互联网服务提供商), 都存在不同形式和不同程度的劫持和窃密威胁[1]。为保护网络通信的机密性和完整性, 某种安全代理有时是必要的。为使讨论更加便利, 下文提到“网络攻击”均指网络劫持和窃密等安全代理保护范围的攻击, 不包括其他类型的攻击。同样的, 下文提到的“防御”即对上述攻击的防御。

安全代理的设计策略主要有六种[1], 分别为附带损害(Collateral Damage)、进入影响范围外(Outside Scope of Influence)、速率限制(Rate Limit)、通信解耦(Decoupled Communication)、淹没(Overwhelm)和无目标(No Target)。附带损害意味着攻击者必须面对由于封锁某一特定协议、服务或应用带来的巨大附带经济损失; 进入影响范围外通常指寻找跨越边界的实体或不受攻击的实体; 速率限制可能是对通信的限制(信道使用频率、吞吐量等)或对攻击效率的限制

通讯作者: 陈嘉耕, 博士, 副教授, Email: jiageng.chen@mail.ccnu.edu.cn。
本课题得到国家自然科学基金(No.61702212)资助。
收稿日期: 2020-08-02; 修改日期: 2020-11-09; 定稿日期: 2021-03-05

(验证码等); 通信解耦主要指非对称通信(可能是路由路径、时间、协议、应用等的非对称)和带外传输; 淹没通常是指大量部署服务端或通信节点; 无目标指使非法流量难以分辨(随机化或隧道化等)或隐藏网络地址(路由或人工传递等)。其中后四种的作用在于减少被发现的机会并使攻击者消耗更多资源, 直到攻击变得非常不经济, 从而迫使攻击者放弃攻击。在某一具体防御方案中通常会综合运用两到三种策略。

发展至今, 具有代表性的工具大约有二十余种, 但只有一种策略组合(附带损害+无目标)是广泛使用的、有效的和高性能的。代表协议有 Obfs4、Shadowsocks、TLS 隧道、Lampshade 和 Obfuscated SSH, 对应的工具为 Tor、Shadowsocks、Trojan、Lantern 和 Psiphon。让我们重点考察当前最为流行的 TLS 隧道代理技术。Trojan 完全把代理协议封装在 TLS 信道中, 并通过检测 TLS 握手阶段是否按照约定进行特殊处理来认证身份, 认证失败则提供正常的 HTTPS 服务。V2Ray 的生态完善, 同时支持多种协议, 但抛开锦上添花的额外特性不谈, 实际可用的协议只有 TLS 隧道一种, 经过配置也可实现类似 Trojan 的功能但不如 Trojan 隐蔽。基于上述讨论, 我们认为基于“附带损害+无目标”的思路最有利于打造出长期稳定有效的协议。

文章结构安排如下: 第 2 章阐述攻防现状并为本协议的设计策略作背景说明; 第 3 章详细描述本协议的设计方案和处理细节; 第 4 章对协议进行安全性分析; 第 5 章从安全性和性能两方面与其他工具进行比较分析; 第 6 章对应用场景和使用方法等进行进一步的探讨; 第 7 章总结全文。

2 背景

在互联网松散的结构下, 没有被保护的数据面临着非常大的被窃取机密信息或通信被劫持的风险。为保护网络通信而发展起来的安全代理技术与相应攻击技术之间的对抗由来已久, 现简述现有攻击技术和防御技术的原理和局限性。

2.1 假设、手段和原理

对攻击者的假设: 攻击者受限于计算资源和经济上的利益, 不会彻底阻断网络也不会采用白名单策略。一个更强的假设是, 攻击者的攻击设备是旁路的而非串接的[2]。

对防御者的假设: 可以通过某种合法途径不借助于已经建立的安全代理通信信道访问外部信息。

基本攻击手段: 攻击过程包含两个阶段, 发现和阻断。发现的基本手段是 DPI 和单包探针。阻断的基本手段是 DNS 投毒和 IP 黑名单。DPI 技术检查应用层数据; 单包探针查看服务响应; DNS 投毒对特定域名的 DNS 请求会抢先回复错误解析结果, 导致正确结果因迟到被丢弃; 凡是对在黑名单上的 IP 的请求都会被抢先用 TCP RST 包重置, 导致 TCP 连接断开。

基本防御手段: 代理。通过攻击范围外不受影响的主机协助转发请求的方式绕过攻击。

IP 黑名单的方法简单有效但有实际应用上的困难, 导致使用本地 DNS 可短暂绕过。具体来说, 困难有: 应当列入黑名单的 IP 数量众多、服务对应 IP 会发生变动、没有便捷的方法掌握黑名单服务的所有 IP、IPv6 配套设备和工具不到位等。Go Hosts 和 IPv6 隧道方案一度流行就是这种困难的具体体现。

最初使用的代理包括 HTTP 代理、Socks 代理、基于 Web 的代理(也称为在线代理)和 VPN 技术。常用的 VPN 有 IPsec、L2TP、PPTP、OpenVPN、WireGuard 等。但这四类代理都不是为规避攻击而生的。这些代理使得黑名单中的域名或 IP 不再直接出现在数据包外层, 但前三种易于通过拓展 DPI 的检查范围发现, VPN 技术具有重要的合法应用且安全性很高, 但两个原因使其在防御上无法起到有效的作用。第一个原因是, 通过包头和握手过程等易于被动发现活跃的 VPN 连接; 第二个原因是, 在某些攻击范围中, 使用 VPN 必须登记备案, 直接阻断未知 VPN 连接不会带来附带损害。这四种代理还有一些缺陷, 例如 HTTP 代理不能代理 UDP 流量、VPN 只能全局代理导致内部网站访问慢的同时暴露内部站点和自己在使用的 VPN 服务器地址[3]、在线代理不能保护用户的机密数据等。

通过拓展 DPI 技术检查范围, 上面提到的四种代理方法都很容易被封锁, 且任何基于域名的非法服务都不再能够稳定存续。例如, HTTP Header 中的 Host 字段、HTTPS 握手过程中的 SNI 字段和 TLSv1.3 的 ServerName 字段会被检查。尽管使用 DPI 技术进行大范围的攻击需要很多资源, 检查范围仍有可能包括每一个出口连接。剩余还有两种方法, 一是经常更换域名或添加新域名; 二是使用代理。前一条需要服务提供者经常维护, 后一条需要访问者主动使用。

发展至此, 后续相关技术对抗才正式进入本文讨论范围内。总结历史上出现过的防御技术原理, 大致可分为八种, 端到端(End to End)、端到中间(End to Middle)、分布式(Distributed)、中心化(Centralized)、中继(Relay)、TLS 相关(TLS Related)、侧信道(Side

Channel)、隐写(Steganography)。

端到端是大多数代理使用的技术, 指的是通过 IP 或域名直连代理服务端的情形。这就需要某种代理协议, 可以分为使用公共协议和使用私有协议两种。公共协议的使用又分为使用公共实现的和使用私有实现的。通常配合 HTTP 代理协议或 Socks5 协议作为对外服务接口。除了端到中间、中继和分布式外, 都是端到端的。

端到中间是一种需要把配合的服务端置于骨干网中或攻击范围外的 ISP 核心路由线路上的方法[4]。这种方法具有几个独特的性质: 不需用户部署、不通过访问特定网络地址与服务端接触、附带损害巨大。在设计中, 用户可通过访问目标ISP范围内任意未被封锁的 TLS 主机并在请求中包含隐藏的标签来尝试接触服务端。如果用户的请求报文恰经过部署了服务端的路由, 路由就会以一种隐蔽的方式声明自己的存在并劫持该连接到代理目标(有时要求该代理目标也支持 TLS 连接[5])。在这一过程中, 攻击者可以通过被动观察发现服务端的存在和大致路由位置, 但无法得知具体网络地址。文献[6]证明, 强有力的攻击者可以通过 BGP 劫持等技术改变路由路径绕过有端到中间服务端存在的 ISP, 但面临巨大的额外网络资源消耗。尽管这种方法的作者设计意图是使大多数 ISP 安装此服务以达到被规模化网络劫持的地区因为不得不承受无法承受的附带损害而被迫放弃攻击, 但没有哪个ISP愿意在自己的网络设备上安装这样的设备[7]。此外, 端到中间技术要求用户分享机密信息以便利用已建立的 TLS 信道, 这在有些情况下是不可接受的。端到中间的代表有: Telex、Decoy Routing、Crrippede、TapDance、MultiFlow、Waterfall。TapDance 是第一个旁路版本的端到中间方案, 并使用了新颖的选择密文隐写技术[7]。MultiFlow 是 TapDance 的通信解耦版本[5]。Waterfall 是第一个抵抗重路由攻击的端到中间方案[8]。

分布式可以分为中心辅助的分布式和全分布式。前者连接速度快, 性能高, 连接可靠性好, 但依赖于辅助中心服务器。后者连接速度慢, 整体网络性能低下, 有时无法连接, 但不依赖于中心服务器。前者的代表是 Tor, 后者的代表是 i2p。理论上全分布式还可分为受管理的和自组织的, 可参照 P2P 协议 ed2k 和 BT。由于全分布式网络面对攻击时(要求连接攻击范围内外的节点而非任意两个节点, 但攻击范围外节点少且发现困难)在连接速度和可靠性上较为脆弱, 目前还没有成熟的方案。为达到通信解耦的目的, 一些方案中会把协议的一部分做成分布式的[9-10]。

中心化可以分为批量提供的服务和分布式辅助的中心化。前者是最为简单直接的方式, 即一次性提供许多个节点供用户自由选择, 所有用户都是端到端直连节点。后者是一种复杂的相互辅助机制, 以中心化的服务集群为主, 但既可以通过预置的地址来直连也可以通过用户节点来发现、发布、共享新的服务地址, 还可以由用户承担少量的数据转发任务, 典型代表是 Lampshade 协议[11]。

中继可以分为基于第三方网站中转、基于自建服务中转和基于 CDN 中转。基于第三方网站中转使用附带损害策略, 主要利用免费公共存储服务和云存储等攻击者通常不会封锁的服务作为通信中介, 典型代表有 CloudTransport[10]。基于 Web 的代理也属于基于第三方网站中转的类型, 但易于发现、可封锁、无附带损害、暴露了用户通信明文给代理服务提供方, 是防御技术发展早期曾经流行的一种方式。基于自建服务中转使用淹没策略和通信解耦策略并避免单点故障, 主要是简单的 TCP 端口转发, 典型代表为 DAIP(Distributed Anti-interference Proxy, 分布式抗干扰代理)[11]。基于 CDN 中转, 使用附带损害策略, 利用域名前置(Domain fronting)技术访问被禁止的服务。域名前置技术利用了 CDN 实现上的特性, 把未被封锁的域名写在 TLS 请求外部, 把实际要访问的写在内部 HTTP 协议的 HOST 头上, 则 CDN 会按照外部标识建立 TLS 连接并按照内部标识请求内容[9]。这样做等于把明文暴露给了 CDN 服务提供商, 但既然使用了 CDN 就意味着已经暴露了, 那么这种暴露或许也是可以接受的。

TLS 相关(TLS Related)是利用 TLS 协议的广泛性和安全特性衍生出的一系列防御技术。主要有三种类型, 分别是域名前置技术、SNI 改写、加密数据隐写。域名前置技术前文有述。SNI 改写会直接去除 SNI 拓展(根据 RFC 的要求, 没有 SNI 也要继续握手)或利用不同域名同一证书的情形建立连接[12], 实际用途不大。加密数据隐写则是通过某种方式把实际数据隐写在 TLS 通信过程中, 效率很低, 对抗流量分析也不够有力[1,7,13]

侧信道和隐写通常不会单独使用, 而是嵌入在其他方案的某一步骤用于临时传递少量信息。例如端到中间系列方案就采用了这种技术。

2.2 对抗场景

按照监测技术成本排序从低到高依次为包过滤、无状态被动单包 DPI、有状态同连接多包 DPI、无状态单包主动探测、有状态单包组主动探测、时

间或空间跨度上同类型流量关联分析、短跨度内特定类型组流量关联分析、网络连接元数据聚合分析[1,14-16]。需要消耗更多资源的方法在没有强制性需求的情况下, 短时间应该不会实际使用。

从攻击者视角来看, 所有的监测对象只有四种, 可被动发现的、可主动发现的、难以发现或假阳性概率高的、无法发现或正常的。前两种发现后封锁, 第三种在特殊时期或特殊情况下(损害和收益的平衡点升高)封锁, 第四种无法或不需处理。

现在从技术角度具体看一下特定主题下的特殊情况和处理方式。

可被动发现特征的代表有各类 VPN 协议、原始 Tor、Socks5 等, 特点是有明显特征, 可直接发现阻断。抗被动探测代理的代表是 Obfs2 和 Obfs3, 被动只能观察到随机数据, 但主动探针可以得到明确的反馈[17]。实际工作的一般模式是先分析流量特点, 如果怀疑程度达到阈值就发送探针进行确认[17]。

随之发展出了抗主动探测的代理协议, 其特点是无协议头、首包验证、无握手或带外握手、对不通过验证的请求不响应内容。典型代表有 Obfs4、Shadowsocks、Lampshade、Obfuscated SSH。

以 Shadowsocks 为例, 从最初发明到达到抗主动探测的水平, 经历了一些曲折。安全缺陷由以下几点引发: 没有防重放机制、没有完整性保护、使用流密码、服务端根据特定字节的值是否合法来决定是否马上断开连接。没有防重放机制意味着可反复发送某个数据包、没有完整性保护意味着可以篡改内容、使用流密码意味着可以精确修改特定位置的字节、服务端的行为保证了只需暴力尝试即可。随后 Shadowsocks 加入了随机延时抵抗和称为 OTA(One-Time-Auth, 一次性验证)的机制, 但由于几乎同样的原因导致仍然可以探测。之后引入了块密码, 但由于没有完整性保护导致了重定向攻击的可能性[18]。此后又引入了 AEAD 加密方法, 但由于交互过程属于 0-RTT 类型, 重放攻击始终存在。最终现有活跃版本引入了布隆过滤器来缓解重放攻击的威胁。

由于对探针没有回应, 看似无法发现, 但实际上除寻找设计缺陷外仍可通过七种指标的综合判断以很低的假阳性率判定[17]。七种指标具体为超时时间、触发特定关闭连接方式的字节数阈值(操作系统缓存中有未读取数据则以 FIN 包关闭, 否则以 RST 包关闭)、包头部字节熵值(无明文协议头在互联网中是罕见的[17])、流量突发(原本分布在多个主机中的资源现全部通过代理请求导致流量突增)、流量集中(原本指向不同网站的请求现全部指向代理)、泄露的非法 DNS 请求、未关联的 DNS 请求(仅本地解析而无实际访问, 实则通过代理访问)。其中后四种亦可在其他代理的流量分析中发挥作用。但在实际工作中似乎遇到了其他困难, 此类代理仍以相当高的概率存活。

隧道代理是一种将数据嵌入到另外一种协议中的代理方法, 外部协议必须是加密协议, 因此 SSH、TLS 都可以被利用, 但实时视频流更加受设计者欢迎。典型代表有, Trojan、OpenSSH 的端口转发工作模式、DeltaShaper、CovertCast、Facet 等[13,19-20]。前两种额外消耗较小, 后三种利用的是视频流, 额外消耗大, 吞吐量较低, 但噪声更大。隧道代理在实际攻击中较为困难, 以 Facet 为例, 使用最先进的攻击方法要封锁 90%的 Facet 流量需要封锁全部 Skype 视频通话流量的约 40%[20]。

模拟代理是一种用第三方实现模拟某著名协议的代理方法。典型代表有 SkypeMorph、StegoTorus、CensorSpoofer 等。文献[21]中提到, 由于必须实现目标协议的细节、统计特性、实现怪癖和甚至实现错误, 模拟代理是不可行的。但我们认为如果该协议具有泛用性、全加密、明文部分很少、是公有协议, 则是可行的。

中转代理是借由云存储服务、社交平台等可访问的数据中转点进行数据中转的代理方法。使用过程中上传下载数据均使用平台提供的标准方法, 因此不能在网络连接级别区分, 但出于附带损害的考虑也不能简单封锁整个服务。此种方法的性能接近 Tor 网络, 如果过程中涉及隐写则会进一步降低[10]。使用这种方法的很少, 例子有 CloudTransport、PSTA 等[10,16]。攻击难度较大。

端到中间前文已经介绍过, 容易发现但绕过或封锁的代价较大。较好的方式是通过技术以外的手段迫使服务提供者放弃承受继续提供服务可能带来的附带损害。现实当中还没有见到这种技术的实际应用。

TLS 相关前文也介绍过, SNI 相关的兼容性不好, 域名前置技术依赖于 CDN 但很多 CDN 都屏蔽了域名前置这种做法[22]。剩余正常工作的仍然是难以攻击的, 且攻击的价值也不大。

一个名为“西厢计划”的项目以文献[23]中逃避 IPS 系统的相关理论为依据, 致力于寻找拦截机制实现上的漏洞来绕过攻击。基本原理是故意在通信开始时包含重置包并忽略符合某种特征的响应包。具体做法有两种: 一, 故意发送不正确的数据包使得

正常的 TCP 握手中夹带 RST 包, 以此使攻击程序认为连接已经断开; 二, 依靠侦察收集的 DNS 投毒的拦截响应, 归纳特征并忽略这些符合特征的响应, 进而使得真实回复有效。这类做法短期是有效的, 但容易失效且局限性很大。

还有一类有政治背景的工具, 依靠资金支持购买大量域名和 IP, 依靠普通代理技术和动态更换代理网络地址的方式强行对耗。这些工具不仅不具有技术上的重大意义, 部分还有巨大安全漏洞和窥探用户隐私的行为[24], 在此略过。

另一股技术方向是引入许多新协议, 包括 HTTP、HTTP2、UDP、mKCP、MTProto、WebSocket 等。混淆方式除了直接做新协议隧道之外。还有许多组合用法, 例如 TCP + TLS + Web、TCP + TLS 分流器、WebSocket + TLS + Web、HTTP/2+TLS+WEB 等。在具体用法上, 有使用公共实现的也有重新自行实现的, 还有一类是只在数据包前后添加一些字节作为特征的。除了使用公共实现的方案, 都是容易通过流量分析识别的, 引入这些协议除了使花样变多之外没有什么实际意义[22]。事实上, 现实很快证明了这一点。

这一阶段的典型代表是 Shadowsocks 的混淆插件、V2Ray 和 Trojan。那些混淆插件只有效了很短一段时间, 没什么理论创新。V2Ray 可以通过配置和第三方软件搭配实现种类繁多的层层套壳和中转, Trojan 则是对 TCP + TLS + Web 的专门实现。从外部看来, 基于 TLS 隧道可利用 TLS 的全部安全特性, 再加上失败转发机制, 实际运行效果很不错。缺点有两个: 运行沉重, 配置繁琐。

机器学习在加密代理流量识别中是个常见策略, 但仍存在许多问题使得机器学习方法不够实用, 例如数据来源不合理、分类标准不合理、训练结果迁移性不好、运行成本高昂、最好的准确率仅约 94% 且只能在目标网络上训练、假阳性率过高无法承受等[25-32]。

另有几种方法不在上述框架中。例如文献[33]突破了混淆协议本身一并处理混淆、会话和数据传递的惯性思维, 为代理连接添加额外的会话层, 将代理协议划分为用户数据层、会话层和混淆层, 从而大大降低重复建立连接的开销并增加了在多个或多次连接中用户通信不中断的能力。文献[34]提出在 TLS 隧道中建立 Websocket 连接, 可传递任意二进制数据, 从而可承载任意代理协议。若未来 CDN 技术改共享私钥模式为代理信任链模式, 则基于 CDN 转发的代理技术或可焕发新生[35]。

3 协议设计

3.1 设计思路

本方法的设计初衷是提供一种兼具安全性和性能的代理。特点有: 在机密性和完整性的基础上加入隐蔽性、避免重复加密导致的性能问题、为主机被入侵和密钥泄漏两种情形提供更多保护。

据上文所述, 要做到规避网络攻击, 有六大策略和七大技术类别, 但由于前文所述的种种原因实际活跃的高性能选择只有一种搭配, 而这种搭配只有三种思路:

(1) 提供一个通用架构和最高的灵活性;

(2) 提供一个快速简单无特征的加密代理;

(3) 嵌套一个常见且不能简单禁止的协议, 并使两者理论上不可区分。

造成这种现象的原因有许多, 其中最重要的是威胁模型的选择、性能和初始部署难度。有些方法选择了信任提供服务的第三方这样一个不是很强的模型。有些方法为了实现完善的隐蔽性和匿名性过多牺牲了网络性能, 多数情况是不可忍受的。还有一些方法提供了额外的安全特性和可用性保证, 但是初始部署很复杂或不可能独自完成。

本方法选择的是思路三的一种折衷, 是文献[34]的一种特化实现, 在不牺牲性能和部署难度的前提下提供了尽可能多的安全特性, 易于通过拓展部署获得更多可用性和安全特性。包括主动探测对抗、负载均衡、反向代理、流量转发等在内的其他功能借由真正的 nginx 服务支持。当与 nginx 协同工作时, 只做简单的转发工作, 可实现理论上的最低性能损耗。

在本方法中, 代理连接建立之初携带完整 TLSv1.3 握手特征, 握手协议参数嵌入 TLS 握手中的随机区域和加密区域(例如 Random、Session ID、Application Data 数据段等), 随后借助 Key Exchange 扩展完成基于 X25519 的 ECDHE 密钥交换和挑战响应过程; 建立会话后对明文数据和特征数据加密, 对加密数据直接转发; 本地配置文件填充到固定长度后使用主密码导出密钥加密; 随机生成用于前期认证的长期密钥; 自动生成并加密保存配置文件, 无需手动传参。更详细的过程在下一节进行描述。

尽管文献[23]宣称, 模拟另一个协议, 尤其是另一个私有协议的某特定实现是困难的和实际不可能的, 但 TLSv1.3 使我们只需实现 TLS 协议的极小一部分且可参考工业标准实现源码。经过仔细设计使攻击者不能控制明文且多数情况可以利用真实实现,

进一步降低了模仿要求。通过充分利用 nginx 的优秀实现和细节特征, 可以避免处理不遵守协议的意外情况。最终, 只需自行处理 Client Hello 和对加密数据报文的破坏即可。

3.2 协议介绍

Fake TLS Socks 协议分为三部分: 简化 Socks5 协议(Fake TLS Socks Protocol, 简称 FTLSS)、伪装的 TLS 握手协议(Fake TLS Handshake Protocol, 简称 FTLSH)、传输协议(Fake TLS Transport Protocol, 简称 FTLST)。为方便讨论, 下面也称 FTLSocks 客户端为 Socks5 服务器。

FTLSS 协议只负责 FTLSocks 客户端与网络应用之间的通信过程, 后续使用 FTLSH 协议进行双向认证并产生会话密钥, 完成握手之后由 FTLST 协议负责转发过程。代理流量经过选择性加密后嵌入 Application Data 数据包的加密部分。加密和解密的过程会在本地维护一个递增的序列号作为 AEAD 的 Additional Data 输入。整个通信过程中根据具体情况 FTLSocks 服务端可能向 FTLSocks 客户端反馈五种不同的 TLS 警告消息。

通信过程中各实体间的关系如图 1 所示。若忽略模板部分, 整个过程有效数据如图 2 所示。

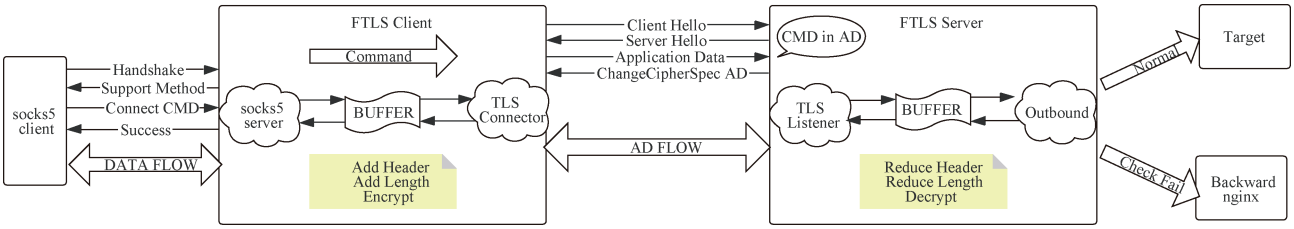


图 1 实体关系图

Figure 1 Entity Relationship Diagram

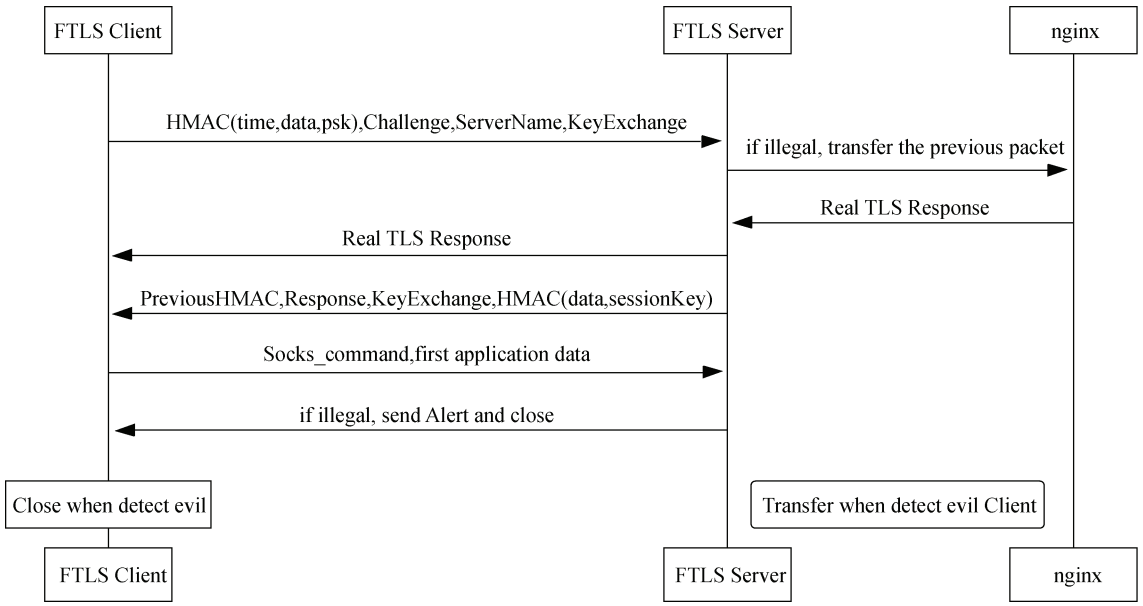


图 2 协议通信概览序列图

Figure 2 Protocol General View Sequence Diagram

基于本协议的代理通信系统部署需要前置要求如下, 否则无法达到设计安全性:

(1) 一个受控域名

(2) 一个合法的证书(可选)

(3) 一台专用 VPS

本协议的安全性建立在以下假设的基础上:

敌手拥有窃听和主动攻击的能力但不能遮断信道, 也不能直接获取会话密钥或长期密钥。合法的客户端和服务端预共享一个长期密钥。用户可短期建立与攻击范围外主机之间的安全信道用于初始部署和分发密钥。本地主机和自行部署的代理服务器是安全运行环境, 云服务提供商不会泄露网络通信数据。

下文 3.3 到 3.5 节分别详述 FTLS 协议的三个子协议, 即 FTLSS、FTLSH 和 FTLST。

3.3 FTLSS

FTLSS 协议用于在客户端提供基本的 Socks5 服务。Socks5 代理协议被广泛支持、简单易用、同时支持 TCP 和 UDP, 但对于我们的特定需求而言仍可进一步简化, 只保留必要的特性。简单来说, 这里只包含 Socks5 的无认证模式和 TCP Connect 指令。这部分内容发生在 Socks 客户端和 Socks 服务端之间, 对应图 3 的①。

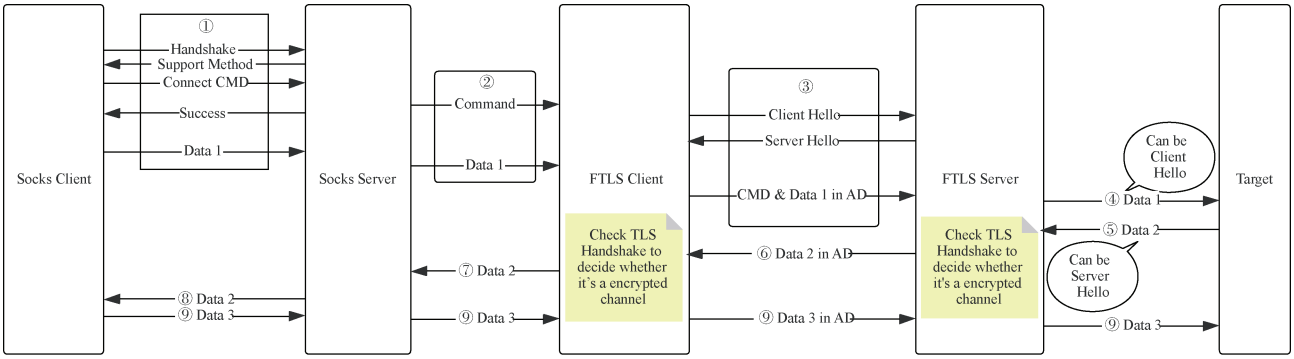


图 3 通信序列图

Figure 3 Comunication Sequence Diagram

握手阶段: Socks5 服务器在收到客户端的协商请求后, 会检查是否包含无认证方式, 返回给客户端的值只有两种可能:

- '0x05 0x00': 采用无认证的方式建立连接
- '0x05 0xff': 任意一种认证方式都不支持

建立连接: Socks 客户端发出指令, Socks 服务端收到后建立一条与 FTLSocks 服务端之间的新连接。这里总是回复: '0x05 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x10 0x10'表示接受代理请求。客户端收到这个请求后会发送第一个原始请求数据包, 即图 3 中的 Data1。FTLS 客户端此时表现与标准 Socks5 服务器不同, Socks 指令与第一个请求数据包被合并为一个数据包后加密处理再转发, 即图 3 的②。

3.4 FTLSH

FTLSH 协议用于建立安全会话。为理解 FTLSH 的设计思路, 先看在 TLSv1.3 握手中的 3 个阶段:

密钥交换: 建立共享密钥数据并选择密码参数, 之后所有的数据都被加密。

Server 参数: 建立其他的握手参数。

认证: 认证 Server(并且选择性认证 Client), 提供密钥确认和完整性。

为达到伪装目的, 我们只需关注第一个阶段即可。取 TLSv1.3 用于建立 HTTPS 连接时的惯用参数, 将 FTLSH 所需参数插入其中。具体以使用 Firefox 71 访问全面支持 TLSv1.3 的某知名网站取得的数据作为模板。伪装方法如图 2 所示, 对应图 3 的③, 具体描述如下:

(1) 客户端与服务端之间先以四个数据包作为开端, 依次为客户端发送 Client Hello, 服务端发送 Server Hello, Change Cipher Spec, Application Data, 客户端发送 Change Cipher Spec, Application Data, 随后双方均为 Application Data, 除非连接出现严重错误。

(2) 整个握手过程中, 若一切正常不应该出现任何警告消息。若遇到非常严重的网络错误或有攻击者插入恶意流量则所有四种致命警告都有可能被触发。错误与警告消息的对应关系遵守 RFC 8446 的规定。

(3) 未加密的 Client Hello 和 Server Hello 消息以模板填充的方式进行构造, 此后从 Application Data 开始即为加密通信。

(4) Change Cipher Spec 和 Alert 是固定内容, 可以在合适的时候直接发送, 发送之前不需要特殊处理。

(5) Client Hello 和 Server Hello 都有 Random 和 Session ID 共 64 字节空间。可供传输一个 HMAC-SHA256((timestamp, all remain data), PSK)值作为 Session ID。发送 32 字节的随机值作为挑战值填充到 Random。其中 timestamp 容许误差范围为[-1,0]。

(6) 双方利用 Key Exchange 字段使用 X25519 进行 ECDHE 密钥交换。

(7) 服务端和客户端分别进行身份验证。前者验 SessionID == HMAC ((timestamp, all remain data), PSK), 后者验证 First_AD_Payload == HMAC (all_remain_data, PSK) && response == HMAC (challenge,session_key), 均通过则建立连接。

(8) 对第一条消息, 客户端验证失败则直接断开连接, 目的在于辨别是否有恶意 Server。服务端验证失败即把数据包转发给 nginx 并回传 nginx 的响应, 目的在于辨别是否有恶意 Client。若服务端验证失败

且连接后端伪装服务失败, 则向对端发送 Close Notify Alert 并马上关闭 TCP 连接(配置一个可用的伪装服务是必要的)。

(9) 对第二条消息, 客户端验证失败则直接断开连接(说明出现尝试攻击的 Server 但 PSK 尚未泄露), 服务端验证失败(说明第一条消息被成功重放)则发送 BadRecordMAC 并马上关闭 TCP 连接。

需要特别说明的是握手包的大小问题:

(1) 根据 RFC 8446 要求, 不足 512 字节的 Client Hello 应该填充到 512 字节, 所以对本模板中域名长度不超过 160 个字符的情况, Client Hello 固定为 512 字节长度。为了简化协议提高速度, 不对超过 160 个字符的超长域名提供支持。

(2) 假设同一个 HTTPS 服务器对同样的 Client Hello 总是回复同样的 Server Hello, 则 Server Hello 也是固定长度。

(3) Server Hello 之后会有一个长度为 32 字节的 Application Data 包用于完成后续 TLS 握手。

(4) 这些包长度不变是符合规范要求的。并非所有采用了 TLSv1.3 的网站的握手过程都是这样的, 但这种方式是一种被推荐的和流行的方式。

3.5 FTLST

FTLST 协议用于提供转发和加密流量分流特性。有两种路径进入 FTLST 阶段。第一种路径是探测到第一个握手包非法, 此时被判定为非法的访问流量不经任何处理直接由 FTLSocks 服务端转发到伪装出口; 第二种路径是正确交互后正常提供服务。

第二种路径具体可分为两种情况: 首先, 客户端和服务端分别独立运行分流算法, 通过记录握手后的前两个包是否是 TLS 协议的 Client Hello 和 Server Hello 来判断当前代理连接是否是 TLS 连接。若为 TLS 连接, 针对之后通信中的 Application Data 类型数据包, FTLSocks 客户端直接转发, 服务端通过试解密来判断当前数据包是经客户端处理后得到的 Application Data 还是原始 Application Data。针对其他类型的数据包, 客户端不需判断, 直接加密后嵌入 Application Data 数据域转发。若非 TLS 连接, 客户端对后续所有数据包直接加密; 服务端解密后转发到代理目标。

即依据前面连接的状态决定可能采取三种不同的转发策略, 其思路主要有两点: 不重复加密、不对 TLS 协议进行实际处理。对应图 3 中的④~⑨。

不重复加密: 分流策略如图 4 所示。“不重复加密”降低了两端的计算成本, 有限算力下可以提供更高的流量处理能力。非加密流量处理过程如图 5 所示, 加密流量处理过程如图 6 所示。

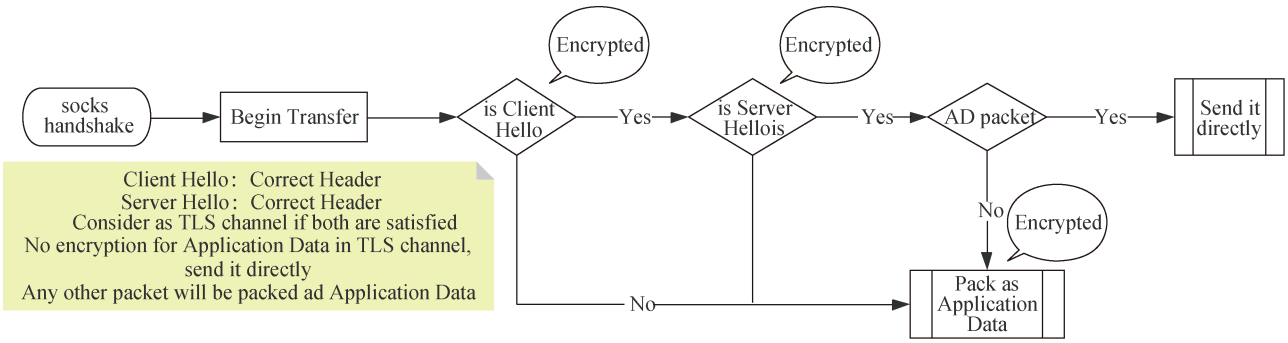


图 4 分流流程图

Figure 4 Traffic Shunt Flow Diagram

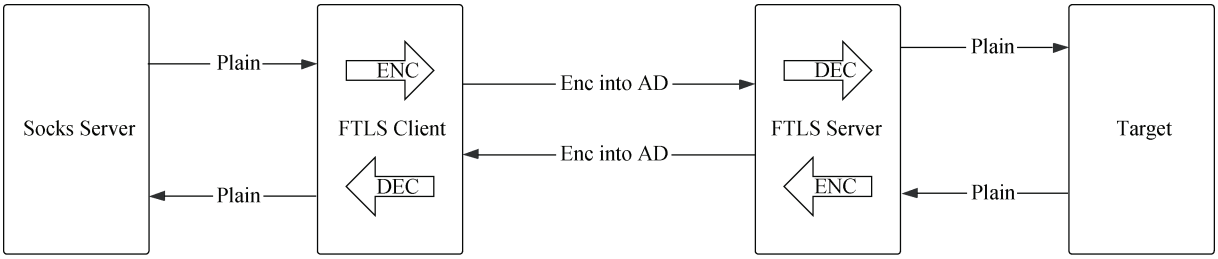


图 5 非加密通信示意图

Figure 5 Non-encrypted Communication

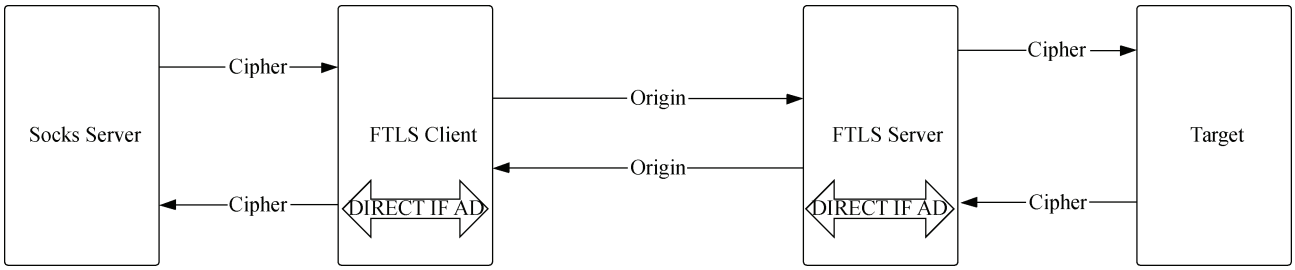


图 6　加密通信示意图

Figure 6　Encrypted Communication

不实际处理 TLS: “不实际处理 TLS”在减少服务器计算的同时为更低的响应时间带来了可能性。相比于完整的 TLS 协议复杂的处理过程, FTLST 仅包含加密流量分流、数据包序号滚动和五种警告消息。每次成功加密序列号递增一, 同时 12 字节随机值作为 nonce 随加密数据传输。后续 4.2.3 和 4.4.1 节还会详述。

4　安全性分析

下文分别分析 FTLS 协议面对窃听攻击(通过监听通信过程获取机密信息)、中间人攻击(分别欺骗双方使其各自认为攻击者是对方)、重放攻击(重新发送监听得到的数据以获取信任或造成未授权的操作)、数据篡改(截获通信数据后修改再发送给原接收方)、主动攻击(攻击者假装是通信的某一方与另一方交互, 然后利用交互过程得到的信息进一步进行攻击)的安全性, 并探讨密钥泄露情形下的安全性。最后, 对攻击者利用流量分析技术(分析网络流量的长度、内容和特点等来识别特定类型的网络活动)识别伪装的可能性和对策进行了简要说明。

4.1　窃听与中间人

主密钥长度 256bit, 使用 CSPRNG 生成。采用 ECDHE 机制交换密钥, AEAD 加密处理, 可保证抵抗窃听攻击。

握手过程中包含一次双向挑战响应, 双方都必须同时使用 PSK 和当次协商的会话密钥证明身份。对服务端来说, PSK 用于生成 HMAC, 会话密钥第一次用于计算响应; 对客户端来说, PSK 用于生成 HMAC, 会话密钥第一次用于请求数据加密。所以中间人攻击也是不可行的。

4.2　重放

一般来说, 正常的真实 TLS 握手(非 0-RTT 且非恢复会话)中的 Client Hello 和 Server Hello 不包含机密信息且即使重放也不能正常建立 TLS 会话, 不必担心重放攻击。但 FTLS 协议把这两个数据包作为身份验证信息的来源, 所以必须充分考虑重放带来的影响。

4.2.1　Client Hello

使用时间戳的 HMAC-SHA256 值作为防重放的基本措施, 有效时间 2 s。可以看到 Client Hello 是有可能重放成功的。但由于后面的第二轮交互之前随即完成了密钥交换, 之后的会话需要使用会话密钥进行验证和加密, 进行重放的攻击者不能计算出正确的会话密钥, 所以不可能再次发送合法的数据包, 所以重放不可能成功, 第二个包必定错误并被 BadRecordMAC 类型的 Alert 消息拒绝。攻击者能够重放 Client Hello 并不会带来实际风险, 因为其即使重放成功也不会得到与正常服务器不同的反馈。

4.2.2　Server Hello

由于 Server Hello 本身具有的性质, 不可能被重放。Server Hello 中包含了一个 HMAC 值保护其完整性, 同时其中填充的 Session ID 必须与对应 Client Hello 一致, 不破坏完整性的同时保持 Session ID 一致对实施重放的攻击者来说是不可能做到的。

4.2.3　其后任意数据包

其后的每个数据包都涉及 AEAD 加密。防止重放的关键在 nonce 和序列号的使用。nonce 取 12 字节随机值并随加密载荷传输; 序列号不传输, 作为 AEAD 的 Additional Data 在本地递增, 每成功加密一次或解密一次递增一。重放、截断、篡改、重排序都将导致 AEAD 验证失败(BadRecordMAC Alert)并中断连接, 必须重新握手才能恢复通信。

4.3　篡改

与上一节中描述的类似, TLS 中的 Client Hello 和 Server Hello 也不必担心篡改, 但在 FTLS 协议中为了简化对 TLS 握手过程的模拟并避免对异常握手过程的处理, 必须对两者进行完整性保护。完成密钥交换前的第一轮交互两个数据包使用 HMAC 保护完整性, 完成密钥交换后使用 AEAD 保护完整性, 非加密数据部分是固定值并被检查。综上, 篡改是可以

立即发现的。

若验证失败说明对方未遵循FTLS协议, 此时交由 nginx 进程处理收到的握手包可保证对异常握手数据包正确响应。因此, 攻击者不能通过构造特殊握手包来辨别当前是否为FTLS服务。

4.4 主动攻击

从发起连接到握手完成, 许多步骤都可能遭遇主动攻击, 接下来分别做出分析。由于协议本身的性质, 被察觉与正常 TLS 服务的不同即丧失隐蔽性可认为是“没有达到设计目标”。

4.4.1 主动探测

由于众所周知的原因, 先来谈主动探测的应对措施。主动探测指攻击者伪装为客户端主动向可疑服务端发送探针, 并通过响应判断可疑服务是否为特定目标服务的过程和手段。

失败转发: 针对握手过程, 目标是使探测者或任意未持有正确访问凭证的访问者无法通过直接访问觉察活跃的FTLS代理服务与TLS服务的差异。

具体来说, 第一个握手包(Client Hello)验证失败则不经任何处理直接转发到伪装服务; 若第一个握手包验证成功(必定是有效时间内被重放成功)则来到第二个握手包的验证, 但任意非法对端由于无法计算出正确的会话密钥不可能验证通过。一旦验证失败, 则按照 TLS 协议规定发送 BadRecordMAC 消息后关闭 TCP 连接。这里关闭连接的动作不会牺牲隐蔽性, 因为任何一个 HTTPS 服务收到不符合协议的数据包时都是这样处理的。

检测并关闭遇到攻击的连接: 为了不在主动攻击下显示出与常规 TLS 服务的区别, 需要在连接发生错误的时候给出恰当的警告消息并正确处理是继续连接还是断开连接。根据前面的分析, 攻击者不可能直接与代理服务进行可控明文的合法交互, 所以一旦插入攻击探针必定导致致命错误。可能的情形共有五种, 分别为篡改握手、篡改明文协议头、篡改密文、重放握手、重放 Application Data 和重排序。对应的警告消息只能是四种会导致 TLS 连接断开的致命错误警告 Illegal_parameter、BadRecordMAC、UnexpectedMessage 和 RecordOverflow 之一。为应对这种探测, 只需根据恶意攻击探针插入的时间和方式的不同, 恰当地给出警告并关闭连接即可。

收到致命错误警告时关闭连接: 虽然攻击者不能注入合理的密文, 却有可能注入明文的致命错误警告消息。根据 RFC 文档的规定, 收到致命错误警告时应该关闭连接, 则此时遵守协议关闭连接即可。

4.4.2 非法客户端

假设攻击者没有得到长期密钥, 则当攻击者伪装为客户端与服务端通信时, 会由于缺失有效的预共享长期密钥, 无法正确构造第一个包, 不能伪装成功。握手验证失败后, 攻击者被指向伪装服务, 无法得到任何有效信息。

4.4.3 非法服务端

当攻击者伪装为服务端与客户端通信时, 由于缺失有效的预共享长期密钥,不能正确计算 Server Hello 的 MAC 值, 所以不能伪装成功。失败后客户端不会继续握手而是发送 UserCanceled 后发送 CloseNotify, 随之关闭连接。攻击者得到了一个有效的 Client Hello 但既不能重放也不能得到长期密钥, 无助于进一步攻击。

这里客户端的行为与正常的客户端不同, 可能会被发觉异常, 但正常客户端也偶尔直接关闭连接。一段时间的静默可能有助于避免暴露客户端的存在, 但考虑到对客户端 IP 的动态性以及对服务可用性的巨大影响, 并未实现此主动静默的功能。

4.4.4 传输阶段

传输阶段本质上是 TLS 头与 AEAD 加密数据的组合。TLS 头检测为非法时发送 UnexpectedMessage 警告。若被插入巨大数据帧或篡改长度字段为超长则发送 RecordOverflow。篡改和重放的情形前面已经覆盖, 不再重复。

4.5 密钥泄露

也许有些防御人员相比通信内容的机密性更注重连接的可用性, 但如果在不同程度的密钥泄露场景中都能够提供额外的保护对很多场景是非常有吸引力的。

下面分别叙述两种不同场景下对代理使用者和代理数据的额外保护措施。

4.5.1 会话密钥泄露

由于软件缺陷, 可能导致内存内容泄露进而泄露当前会话密钥。会话密钥是通过 ECDH 协商而来, 并且只用于当前连接, 因而具有前向安全性。此时对任意其他的会话、此前的会话和主密钥都不能构成任何威胁。

4.5.2 本地入侵

如果被入侵, 则存储在本地的配置文件包含的机密信息可能被泄露, 进而导致长期密钥泄露。采用了五种措施来应对这种场景:

(1) 创建配置文件时时强制使用较长和较复杂的主密码, 防止暴力破解。

(2) 使用参数为 time=100, memory=32*1024,

threads=4, keylen=32 的 Argon2id 算法从主密码导出主密钥来加密配置文件。密钥导出过程加盐, 长度 32 字节, 每次生成配置文件的时候随机生成并附在配置文件末尾, 防止暴力破解。Argon2id 算法的推荐参数中 memory=64*1024, 我们使用的是推荐参数的一半, 但把参数 time 扩大到了推荐值的 100 倍, 总体安全性并没有降低。同时因为只在开启时运行一次所以不会对性能表现造成消极影响。参考消耗时间: 926.12ms on Intel Core i5-8300H。

(3) 不从命令行参数接受主密码, 而是以无回显的方式在后续步骤手动输入, 防止在命令历史中泄露密码。

(4) 载入配置文件后立即覆盖内存中的主密码。

(5) 生成配置文件时分为两个部分, 客户端配置文件不包含服务端参数(伪装出口地址, 超时时间, 协程池大小等)。两个配置文件主密码相同、大小相同, 但盐值不同。

(6) 配置文件加密前先填充到固定长度, 保证不会因为文件大小泄露关于配置的任何信息, 例如域名长度。

综上可以得出结论, 即使客户端或服务端被恶意用户取得本地访问权限也不大可能获得长期密钥或利用本工具留下的信息帮助寻找服务端、解密通信内容。

4.6 流量分析

超时时间、连接持续时间和包长度分布等是流量分析的主要内容。其中超时时间是可配置的, 后面两者取决于通过代理的连接。超时时间默认值 60 s 是互联网中普遍的选择但也可以根据需要修改。

后面两个因素若不加额外处理(这正是我们选择的做法, 更加激进的隐蔽性会牺牲大量性能)则直接反映了代理服务的使用情况且可与我们所声称的服务可能提供的流量作对比来判断是否足够可疑。但这既没有让 FTLSocks 相比现存的稳定工具更可疑, 也可以通过选择更合理的伪装服务来减少可疑的程度。实际上, 即使加入随机字节填充也不会让包长度分布看起来更加正常。

综上, 安全性分析大致可分为四种情况: 第一个包被发现攻击; 第一个包重放成功, 第二步被合理拒绝或超时断开; 被动观察无区别; 主动探查被合理响应并断开。不可观察性的要点在于: 从头到尾彻底的完整性保护确保了交互在约定的框架内进行而无需考虑过多细节, 只需关注代理功能; 超出框架外的由 TLS 的真实公共实现应对。

5 比较与分析

下文从安全性和性能两个方面与各个流行软件比较, 分析 FTLS 协议的优势。前向安全性、性能、单点故障抵抗和部署简单是 FTLSocks 的突出优势。

5.1 安全性比较

下面与较为流行的几个同策略工具和较具有代表性的其他工具进行分析对比。

V2Ray: 是一个通用工具集, 也是一个通用的网络通信框架, 提供了多样的协议支持和功能特性, 具有很强的可定制性。通过修改配置文件可以实现非常丰富的功能, 也可以使得外部流量具有各种形态, 还具有内置的路由和 DNS, 稍作更改就可以实现大部分新的混淆方案。官方发布的核心工具集由 go 语言实现, 但同时存在很多第三方实现。其缺点在于当定制多层通信协议的时候, 额外开销较大, 有时存在双重三重加密的情况。整体而言, V2Ray 选择的是大而全的路线, 新方案可以最大程度上受益于内置 DNS、连接复用、分流引擎等公共模块, 使得新方案也可以立刻得到较好的用户体验。

表 1 特性对比

Table 1 Feature Comparison

	分布式	对称模型	灵活性	抗主动探测	前向安全性	部署复杂性	效率	单点故障
V2Ray	否	混合	高	混合	混合	中高	低	可缓解
Trojan	否	混合	低	是	是	中	中低	是
Shadowsocks	否	是	无	是	否	低	高	是
Tor with Obfs4	是	否	低	是	是	中	极低	否
Lantern	是	否	低	否	未知	高	中	否
FTLSocks	否	混合	中	是	是	中低	高	否

Trojan: 完全以使用 HTTPS 流量发送一个类 Socks 代理协议来实现网络攻击规避为目标的工具。其实质是, 以“收到的数据包能否解析为合法协议”为规则进行流量分发的反向代理服务器。官方发布的工具以 C++实现, 使用 OpenSSL 1.1.1d 作为 HTTPS 的实现。其缺点在于真实建立的 TLS 连接开

销较大, 新连接响应速度较慢, 存在对大量 HTTPS 流量的重复加密、不能放在反向代理后做负载均衡。若采用本地运行一套完整网站的方法来提供主动检测对抗基础措施对服务器是个不小的压力, 不利于大规模部署。若采用静态或不更新内容的动态网站则失败转发机制失去了意义, 固定的内容使得容易被怀疑和判断。其也未提供便捷安全的配置方式, 只能直接编辑明文的 json 文件。其目前(此处指 1.14.0 版本)的实现有几个缺陷: 无良好的默认密码策略、对不合法请求的处理较慢、多核计算能力较弱、反向代理功能不完善(特别的, 不能反向代理 HTTPS)。Trojan 抛弃了 V2Ray 的臃肿架构, 将 TLS 嵌套模式单独取出再对 TLS 握手过程第一步进行修改。配合首包验证和失败转发, 最终得到了更加隐蔽高效的代理模式。但同时也失去了反向代理、多级代理、负载均衡、单点故障抵抗等诸多特性和网络架构上的灵活性。

Shadowsocks: 一个功能简单, 目标单一, 快速的加密 Socks 代理。其传输一个类 Socks 协议但把传统的 Socks 服务器拆分成客户端和服务端, 两端之间加密处理。服务端收到数据后进行试解密, 解密后协议解析成功则执行功能, 否则什么也不做。除此之外还可以在转发之前根据访问地址进行分流。支持混淆插件为流量进行预处理以便添加额外特征, 但时至今日受限于插件的工作方式这些混淆只是聊胜于无。Shadowsocks 无官方版本, 目前流行的有 C 语言实现的 shdowsocks-libev、go 语言实现的 shadowsocks-go2 和停留在 08/10/2015 的 2.8.2 版本的 Python 实现。各种实现的功能特性不一, 但得益于 Shadowsocks 的设计思想始终保持快速的处理能力和无明显特征的流量。其优点在于简单快速, 其缺点在于过于简单粗暴容易引起怀疑。在 2.4 节中, 我们介绍了此类代理的弱点和探测方法, 但不得不承认这种方法仍然有着一定的生存空间和其他方法难以比拟的性能优势。

Tor with Obfs4: 这是一种隐蔽地连接 Tor 网桥的方式, 是连接到 Tor 网络的一种方法。这里我们关注的重点是 Obfs4。它和 Shadowsocks 一样属于抗主动探测代理, 但专用于连接到 Tor 网桥。许多防御方式最终都连接到了 Tor 网络并由 Tor 网络提供剩余的服务, 充分利用了 Tor 网络的匿名性。凡事有利有弊, 匿名性与效率是一对无法调和的矛盾, Tor 网络特殊的运作方式使得获得更高的性能非常困难。通过一个并不臃肿的工具和不同的使用方式来获取灵活配置的匿名性、安全特性和性能可能是一种更好的方式。

Lantern: 一个跨平台的高可用性商业代理工具。Lantern 的协议 Lampshade 和其运作方式是介绍的几种协议中最为复杂的。其结合了中心化和分布式相关代理技术并由维护者部署的中心服务器提供主要服务, 用户可作为 P2P 节点提供辅助服务。为了提高性能表现和可用性, Lantern 抛弃了匿名性, 但也因此使服务地址更容易被攻击者得知[12]。由于部署的复杂性、运营的商业性、极弱的可配置性, 我们不认为 Lantern 是一个具有潜力的和值得信赖的工具。

FTLSocks: 对 FTLS 协议的实现, 由本文作者开发。为了提供一个安全、快速、轻量化、可配置、特性可选择的代理工具而被提出。最初的目标是减少 Trojan 实现方案带来的负担并提供相似的防御特性。伪装思路是伪 TLS 隧道代理, 并非完整 TLS 实现, 只实现了一半假的握手过程和一些警告的处理。结合首包验证机制和失败转发机制, 不能通过验证的连接由 nginx 处理, 从而节省了 TLS 相关处理过程的实现并充分利用了 nginx 的高性能实现和完善的反向代理功能。协议被设计为通过可通过不同部署方式或与第三方软件灵活配合来实现分流、负载均衡、单点故障抵抗等特性, 本身仅在高性能的前提下提供前向安全、机密性等基本安全特性。此外, 提供了更好的默认密钥策略和配置向导。

5.2 性能对比

设计新方法并创建本工具的主要目的是提高性能并避免不必要的计算资源消耗, 因此与流行工具相互比较的一次性能测试是必要的。

表 2 系统资源消耗

Table 2 System Resource Consumption

	FTLSocks	shadowsocks-go2	shadowsocks-libev	V2Ray	Trojan
大小(MB)	3.1	1.1	5.0	9.1	1.7
空闲内存占用(MB)	34.4	3.2	1.6	55.5	4.1
压测后内存占用(MB)	54.6	9.6	2.0	78.1	17.0
HTTP 压测中单位数据 CPU 消耗	21.73	19.68	22.29	139.43	18.59
HTTPS 压测中单位数据 CPU 消耗	23.23	25.32	22.32	143.64	19.81

(注: FTLS 刨除 Argon2id 计算导出密钥需要消耗的 32M 仅占 2.4M。这里列出的均为客户端数据, 服务端相差很小(除了 HTTP 情形下 FTLS 客户端有 0.33 的额外 CPU 消耗外, 其他情况差别绝对值均小于 0.01)。)

5.2.1 测试方法

测试使用 siege 进行, 分别使用大中小(147MB/22MB/74KB)三种大小的文件和低、中低、中、高(2、5、20、150 个并发线程, 分别代表个人轻度使用, 个人重度使用, 多人轻度使用, 多人重度使用)四种并发程度进行持续一分钟, 随机延时 0.5s(siege 的默认值, 更好地模拟真实访问的分布)的压力测试。

加密方法全选 AES-256-GCM, 保护方式可选时选 TLS 模式。

5.2.2 测试环境

所有测试数据和程序在 tmpfs 上保存, 确保本机 IO 不会成为测试瓶颈。为确保硬件规格一致且环境温湿度不会使硬件性能发生漂移导致测量不准确, 测试在云主机的环回网络进行, 云主机的型号为阿里云密集计算型 ic5 / ecs.ic5.2xlarge(8vCPU 8GiB)。

测试的硬件环境为 Intel Xeon(Skylake) Platinum 8163 8vCPU 主频 2.5 GHz, 睿频 2.7 GHz, 8 GB 内存, 内网带宽 2.5 Gbps, 内网收发包 80 万 PPS。

软件环境为 CentOS 8.1 x86_64 Kernel 4.18.0, nginx/1.17.10。

其中 nginx 采用 Mozilla Guideline v5.4 推荐中级配置, worker_processes=2, worker_connections=1024, keepalivetimeout=60, sendfile=on。

5.2.3 测试结果

参与测试的软件有 FTLSocks/0.1.0、shadowsocks-go2/0.1.0、shadowsocks-libev/3.3.4.0、Trojan/1.14.0 和 V2Ray/4.23.1。测试项目包括响应时间(s), 吞吐量(MB/s)、最长请求完成时间(s)、最短请求完成时间(s)。

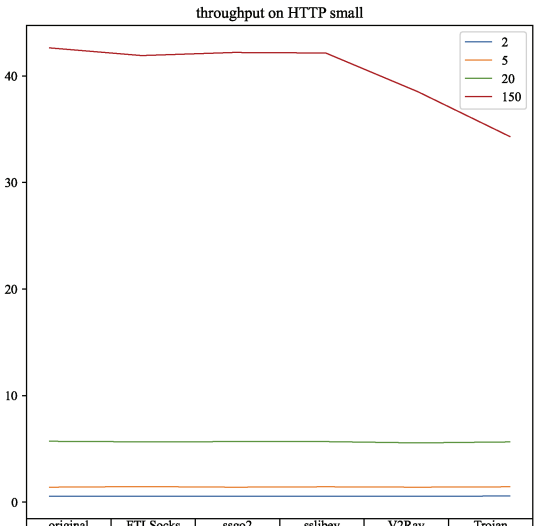
为节约空间并使图表更具代表性, 吞吐量选取所有测试结果, 最长/短请求时间选取大文件测试结果, 响应时间选取中等文件大小测试结果。除请求完成时间因加密和非加密的测试结果极度相似而只选取加密情形进行展示外, 都同时展示了两种情形。

概览所有性能测试结果图示可以发现, 性能表现从优到劣的大致顺序为直接访问、FTLSocks、shadowsocks-go2、shadowsocks-libev、V2Ray with TLS、Trojan。与预期不同的是 Trojan 的表现在高并发下低于 V2Ray, 低并发下高于 V2Ray。

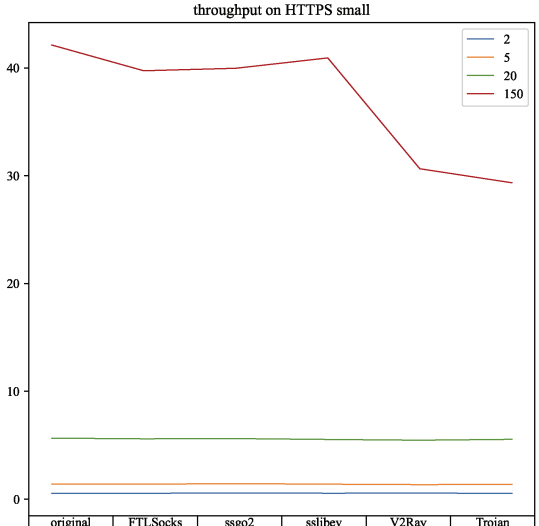
吞吐量的测试中, 高并发情况下小文件较低, 中等文件情形正常, 大文件时轻微下降; 测试压力较低时不同程序差别较小; 随着文件的增大, 即使并发压力较低时也可以看出明显的性能差距。无论哪种情形, FTLSocks 与 shadowsocks-go2 都是代理程序中表现最好的两个, 且远超第三名。

在最长/短请求完成时间和响应时间的测试中, 六张图显示的趋势基本一致, FTLS 协议都表现出了优势。横向对比可以看出 FTLSocks 的代理过程对响应时间的影响最小且高并发下阻滞时间也最短。

比较 HTTP 和 HTTPS 通信可以发现, HTTPS 通信在所有情形下性能表现都有所降低, 但各被测对象性能表现的相对关系几乎不变, FTLSocks 始终占据较大优势。虽然 FTLSocks 的 HTTPS 流量转发过程减少了对加密和解密的需求, 但增加了选择判断的需求, 结果上看性能增加可观却不及预期。

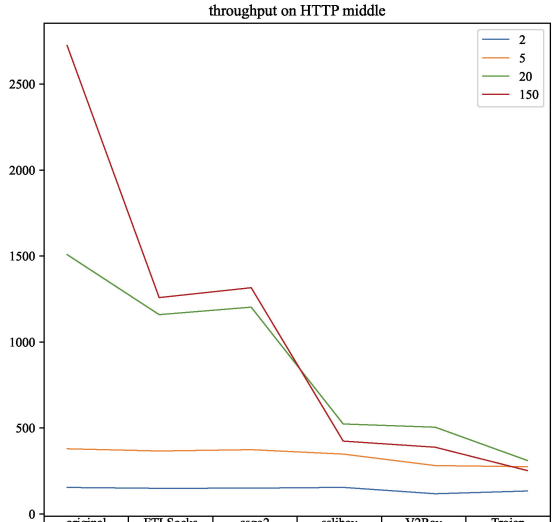


	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	0.56	0.56	0.55	0.55	0.55	0.57
5	1.4	1.46	1.4	1.44	1.4	1.44
20	5.71	5.66	5.68	5.68	5.56	5.65
150	42.64	41.92	42.21	42.16	38.53	34.32



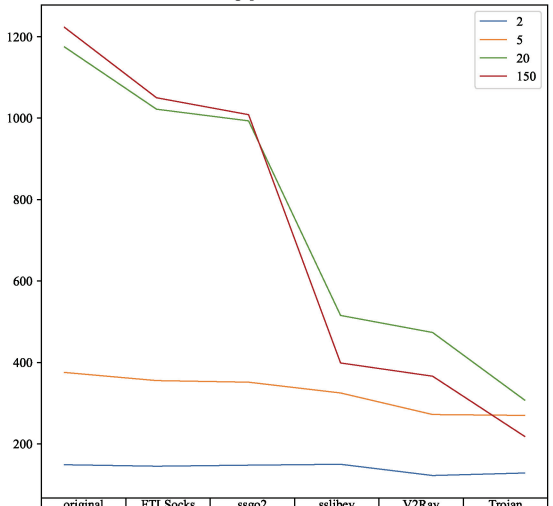
	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	0.55	0.54	0.58	0.55	0.56	0.54
5	1.41	1.39	1.44	1.39	1.35	1.37
20	5.66	5.6	5.62	5.54	5.47	5.55
150	42.13	39.74	39.97	40.93	30.65	29.35

throughput on HTTP middle



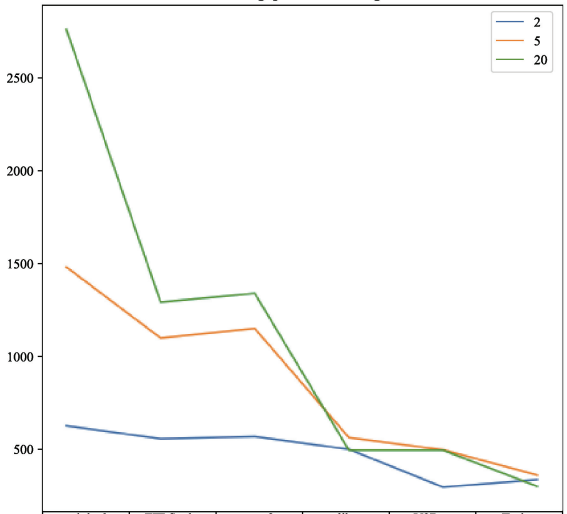
	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	153.79	148.71	149.8	154.15	117.18	133.14
5	378.67	365.98	373.59	347.84	281.05	274.57
20	1508.15	1159.03	1202.55	523.03	504.25	310.84
150	2723.96	1258.19	1315.7	423.35	387.74	252.13

throughput on HTTPS middle



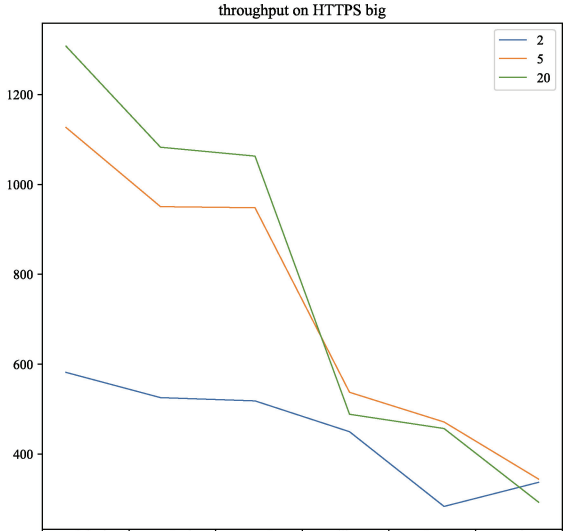
	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	148.74	145.11	147.65	149.82	122.25	128.42
5	375.41	355.4	351.47	324.99	272.03	269.86
20	1174.82	1021.93	993.3	515.41	473.54	307.58
150	1222.9	1050.05	1008.51	398.69	366.16	218.39

throughput on HTTP big



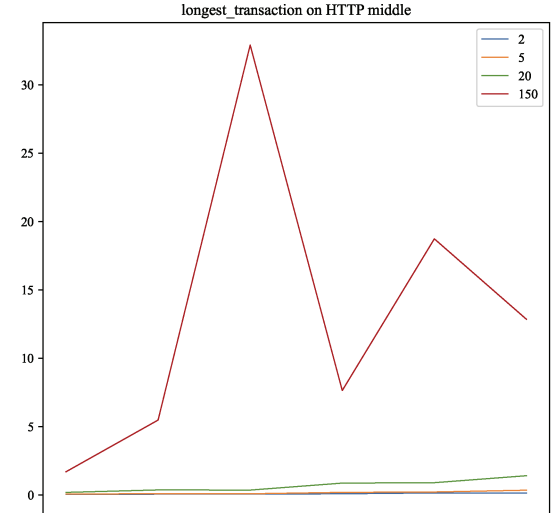
	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	621.2	551.19	562.97	494.02	289.02	330.11
5	1481.08	1096.82	1148.7	557.24	491.25	354.38
20	2768.17	1290.88	1339.11	488.81	488.81	293.28

throughput on HTTPS big



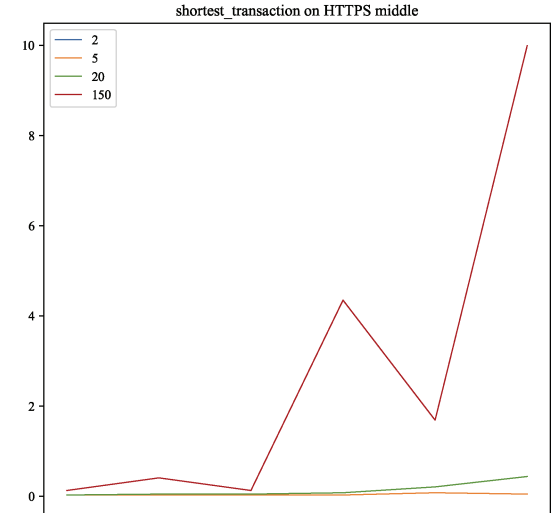
	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	581.97	525.82	518.57	450.0	283.84	337.45
5	1126.7	950.57	948.13	537.69	471.62	344.61
20	1307.56	1082.71	1062.98	488.81	457.11	293.28

longest_transaction on HTTP middle

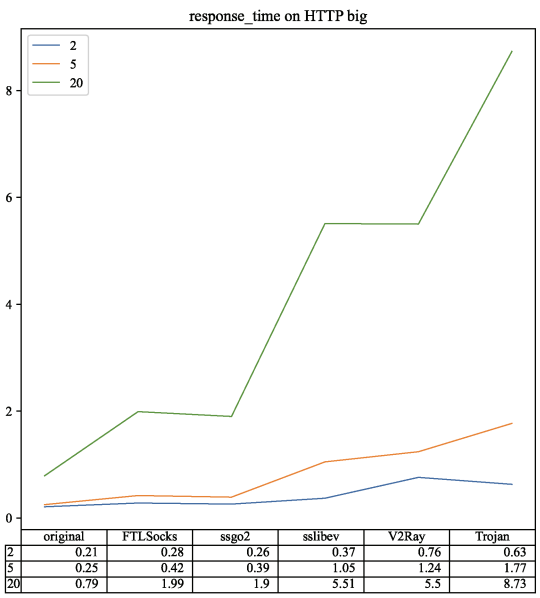


	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	0.05	0.06	0.06	0.09	0.15	0.14
5	0.06	0.1	0.09	0.19	0.22	0.35
20	0.19	0.37	0.36	0.87	0.9	1.41
150	1.7	5.48	32.9	7.64	18.73	12.85

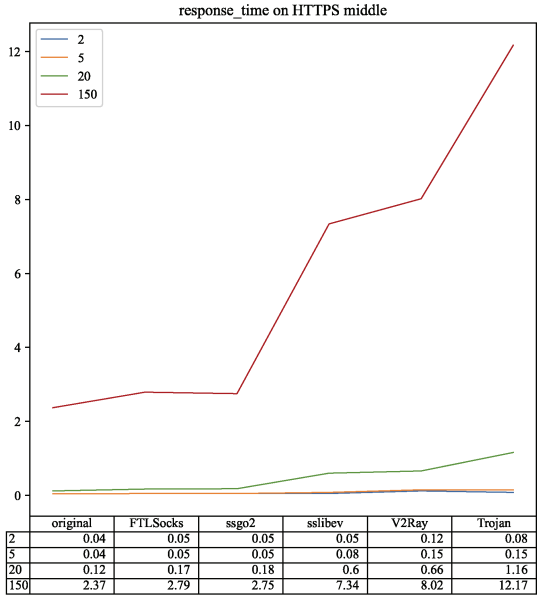
shortest_transaction on HTTPS middle



	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	0.03	0.03	0.03	0.03	0.08	0.05
5	0.03	0.03	0.03	0.03	0.08	0.05
20	0.03	0.05	0.05	0.08	0.21	0.44
150	0.13	0.41	0.13	4.35	1.69	9.99



	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	0.21	0.28	0.26	0.37	0.76	0.63
5	0.25	0.42	0.39	1.05	1.24	1.77
20	0.79	1.99	1.9	5.51	5.5	8.73



	original	FTLSocks	ssgo2	sslibev	V2Ray	Trojan
2	0.04	0.05	0.05	0.05	0.12	0.08
5	0.04	0.05	0.05	0.08	0.15	0.15
20	0.12	0.17	0.18	0.6	0.66	1.16
150	2.37	2.79	2.75	7.34	8.02	12.17

图 7　性能测试组图

Figure 7　Performance Test Group Chart

比较不同并发情况可以发现, 当并发线程数达到 150 时, shdowsocks-libev、V2Ray 和 Trojan 的吞吐量都随着并发数上升而下降, 说明触及了并发性能瓶颈。反观 FTLSocks 和 shadowsocks-go2 则吞吐量仍有少量上升且 FTLSocks 的吞吐量上升幅度略大, 充分证明了 FTLS 协议的性能优势。

5.2.4　系统资源占用比较

这里主要比较客户端数据。如表 2 所示。服务端一般来说与客户端相差很小, 且计算资源和空间资源相对充足, 为简洁起见不在表 2 列出。

CPU 资源消耗的详细信息如表 3-4 所示。文件大小的比较方法为去除调试信息和符号表再用 UPX 压缩处理。对非单个可执行文件的计算所有文件, 数据文件按照 7z 常规压缩后大小计算。静态链接的只计算可执行文件本身, 动态链接的同时计算动态链接库文件。

内存占用的比较方法为, 分别比较客户端默认配置下启动状态和压测后。用于比较的压测方法为 20 线程中等文件大小 HTTP 方式压测一分钟。选择这种方法是因为从性能测试数据来看这种情形下最能充分发挥性能。

CPU 占用率的比较方法为, 首先采用 20 线程中等文件大小压测 1 min, 随后统计出平均 CPU 占用率, 最后分别比较 HTTP 和 HTTPS 两种情况下单位数据量消耗的 CPU 资源。CPU 占用率数据使用 pidstat 工具收集, 吞吐量使用 siege 工具收集, 数据独立于性能测试环节。计算方法为平均 CPU 占用率(%)除以吞吐量(GB/s)。考虑到吞吐量更大时加解密必定会消耗更多计算资源, 这种计算方法是比较公平的。

从表中可以看出, FTLSocks 在内存占用大小和文件大小上与用 C/C++编写的 shadowsocks-libev 和 Trojan 或相对逻辑简单的 shadowsocks-go2 都不占有优势。从 CPU 占用率可以看出, FTLSocks 所需资源相对较少。但从另外一个角度来说, 无论是哪一个都相当小且足够小, 都可以在资源受限的嵌入式设备上良好工作并得到较好的性能表现。

表 3　HTTP 情形下的 CPU 消耗

Table 3　CPU Consumption on HTTP

分类	客户端			服务端			吞吐量
	用户空间	内核空间	总占用率	用户空间	内核空间	总占用率	
FTLSocks	9.78	11.95	21.73	8.98	12.41	21.40	1.11
shadowsocks-go2	8.32	11.36	19.68	8.32	11.36	19.68	1.15
shadowsocks-libev	11.44	10.85	22.29	11.44	10.85	22.29	0.49
V2Ray	93.50	45.93	139.43	93.50	45.93	139.43	0.30
Trojan	52.70	12.00	18.59	6.59	12.00	18.59	0.47

表 4 HTTPS 情形下的 CPU 消耗

Table 4 CPU Consumption on HTTPS

分类	客户端 e			服务端			吞吐量
	用户空间	内核空间	总占用率	用户空间	内核空间	总占用率	
FTLSocks	10.62	12.61	23.23	10.62	12.61	23.23	0.92
shadowsocks-go2	10.70	14.62	25.32	10.70	14.62	25.32	0.89
shadowsocks-libev	11.46	10.86	22.32	11.46	10.86	22.32	0.49
V2Ray	96.32	47.31	143.64	96.32	47.31	143.64	0.29
Trojan	7.02	12.79	19.81	7.02	12.79	19.81	0.44

可以看出除 Trojan 的 HTTP 情形之外雷达图上对称维度差别不大, 这说明客户端和服务端的 CPU 资源消耗是几乎对称的。需要特别指出的有三点: FTLSocks 客户端消耗比服务端略大、V2Ray 消耗最大且远超其他几种、客户端与服务端总体持平但 Trojan 在 HTTP 情形下用户空间 CPU 占用较突出。具体如表 3~4 和图 8 所示。

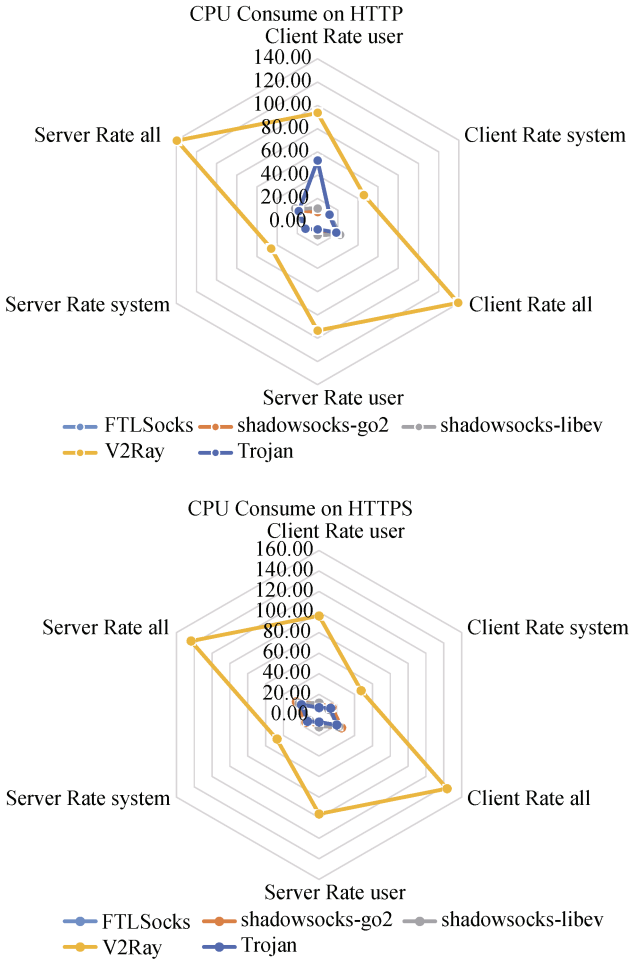


图 8 CPU 消耗雷达图

Figure 8 CPU Consumption Radar Chart

此外, 值得一提的是, 即使反复用大量数据和高并发来测试, FTLSocks 也不会占用比仅压测一分钟后更多的内存(多次测试中误差大约为 1M)。

5.2.5 分析

让我们结合源代码和性能测试数据来进行具体原因的分析。首先需要说明的是, 本工具用于导出加解密配置文件的主密钥是 Argon2id。该算法具备内存消耗机制来抵抗 ASIC、GPU 和高性能计算集群的攻击。在本实现中选择了 32M 作为内存参数, 因此可以认为空闲情况下实际消耗内存为 2.4M。这部分空间在密钥导出结束后将作为堆空间由 go runtime 管理。

Shadowsocks 的协议结构简单, 且属于预置密钥的 0-RTT 协议, 各方面超出本工具的表现是理所当然的, 此处仅为提供一个流行工具性能标准的参考。所以分析的重点是 Trojan 和 V2Ray 的 TLS 模式与本工具的比较。

对比本工具和 V2Ray, 可以发现本工具的 CPU 占用率较低, 但性能表现却更好。虽然内存占用相对较高, 但仍在可接受的范围内。分析 V2Ray 的 TLS 伪装实现可以发现, 在这种场景下 V2Ray 进行了两次加密, 完整使用了 TLS, 为保持通用性还做了包括路由、DNS、连接复用等许多额外的工作。这就是 V2Ray 消耗 CPU 资源较多的原因了。

Trojan 虽然在内存占用和 CPU 占用上具有优势, 但同时性能表现也大大下降了。从性能测试数据可以发现只在并发量较低时 Trojan 的表现才接近或略高于其他比较对象。

由此也可以看出网络上流传的大部分性能测试之所以得出 Trojan 性能更高的结论的原因是: 只测试了较低并发负载(单用户单代理目标)下的使用场景。

阅读源代码可以发现, 这种现象本质上是由两个原因造成的:

(1) 对多核计算能力利用较弱;

(2) 完整的 TLS 栈。

FTLSocks 使用了更简化的协议设计, 从根本上保证了更高的吞吐量和更优的性能表现。从测试数

据可以看出, 无论是吞吐量还是响应时间抑或最短最长请求完成时间都占有相当大的优势。

在实现上, FTLSocks 使用了协程池来增强并发性能, 对象池来减少内存分配, 无竞争无锁的设计, 所以能够充分利用 CPU 性能, 实现同等压力下更高的吞吐量。缺点就是占用了略多的内存。在长时间大规模的并发场景中, 采用协程池无论是计算资源还是内存资源消耗上都是有优势的。

虽然 FTLSocks 在单位数据 CPU 消耗上略有劣势但仍大致处于同等水平, 考虑到提供的丰富安全特性和极佳的性能表现, 应当认为 FTLSocks 是一种更优的设计。

此外, 由于 FTLS 协议的特性, 从未建立真正的 TLS 连接, 因而可以得到两个额外的好处。(1)可以结合 nginx 的 HTTPS 反向代理功能利用世界上其他 HTTPS 网站作为反向代理目标, 节省本地搭建和运行服务的精力和系统资源消耗; (2)可以作为反向代理的后端, 在负载均衡的帮助下实现更优的性能表现和更高的可用性。

6 应用场景和讨论

6.1 场景

快速部署: FTLSocks 的最简部署方式只需一个域名和一台服务器, 部署过程只需要生成配置文件、上传和运行三步即可提供代理服务, 合法证书和伪装服务都不是必须的, 但这样做同时也是隐蔽性和可用性最差的方式。通过下面介绍的使用方式可以充分发挥 FTLS 协议的威力, 极大增强隐蔽性、可用性和安全性。

服务集群: 由于 FTLSocks 设计上的极简性, 使得规模化部署服务集群非常容易。可以把快速部署的方法简单推广到服务集群的部署中。

快速恢复: 通过组建转发网络, 可以抵抗单点故障并在 FTLS 服务器被阻断访问时迅速恢复访问。该网络至少需要两台转发器和一台服务器, 较好的结构应该有至少三台转发器和两台服务器, 有条件的情况下转发器越多越好。使用过程中应当始终保持至少一台转发器从不在紧急情况以外使用, 留作恢复访问的紧急入口。该转发网络的大致结构和通信过程如图 9 所示, URGE 连接从不在紧急情况外使用, Relay 转发流量到真正的 FTLS 服务从而在攻击者观察下隐藏代理服务器的 IP 地址。这种使用方式并不是 FTLS 协议的一部分, 但协议的设计中考虑了这一部分, 从而保证了转发网络可以顺畅运行。

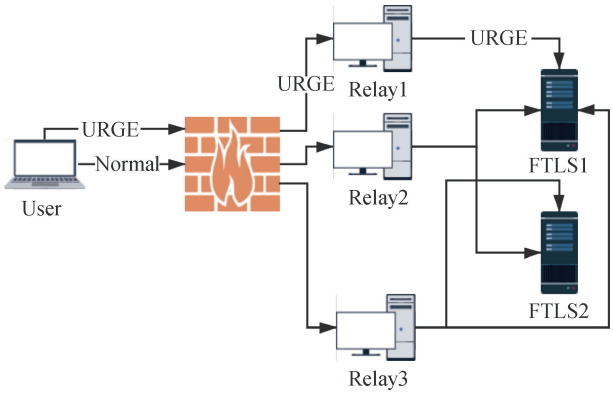


图 9 转发网络拓扑图

Figure 9 Forwarding Network Toplogy Diagram

本地入侵: 通过申请证书、部署伪装服务、使用较高强度的主密码、专用的非特权用户来运行、仅允许非特权用户使用带密码保护的私钥进行 SSH 登录、不运行其他服务, 可以减少攻击者入侵后从内存得到主密钥的风险。

流量分流: 通过将 AutoProxy 类似的软件置于 FTLSocks 前可以起到分流的作用, 进而避免不必要的代理动作, 提升用户体验。

6.2 讨论

FTLSocks 提供的配置向导可以对一些参数进行调节, 这些参数对性能、流量负载能力造成一定影响的同时还会给流量带来一些特定的特征(例如性能下降曲线、超时时间等)。为避免基于流量指纹的流量分析技术的攻击, 可以通过不使用参数默认值进一步增强隐蔽性。此外, 配置向导提供的 KeepAlive 和 Deadline 参数值是常用默认值, 可以不改变。

FTLSocks 中使用的“首包验证”结合“失败转发”的机制在设计上需要一个伪装服务, 这个服务的选择对隐蔽性有很大的影响。虽然基于 TLS 的伪装无法在应用层辨别但仍有可能结合具体场景进行区分。为缓解这一问题可以采用使攻击者无法确定流量性质和伪装出特定场景的统计特征两种策略。以前者为例, 通过在代理的伪装出口使用合理的伪装服务(例如, 一个有身份验证的 API 服务), 可以减少可疑程度。这里的关键点在于, API 服务的非授权用户只能得到很短的错误消息, 不能确定后续内容长度和性质的变化规律; 基于 Web 的服务存在大量静态资源或变化频率很低的动态资源, 而这些资源的大小和访问时机都是可预测的。此外, 长期连接某特定 Web 服务是可疑的, 但长期使用某 API 服务则是可以接受的。后者的代表如 NaïveProxy, 直接使用 Chromium 开源项目的网络堆栈模块, 可以获得

Chrome 浏览器的统计特征。

从攻击的角度来看, 辨别其伪装性最好的办法是使用基于用户画像的方法长时间观察目标网络活动[36], 利用 2.2 节中提到的代理七种指标进行综合判断。这些指标是代理使用者的网络流量必然出现的特征。有节制地使用代理并对本地网络和伪装服务进行恰当配置可以有效挫败这种尝试。

除代理本身的部署情况以外, 使用的环境和方式也会对隐蔽性有所影响。例如, 在浏览器中使用代理时阻止 WebRTC 可以防止在你访问某些网站时泄露你的本地 IP 地址。使用 CDN 来让代理流量先经过一条合法路径也可以增强隐蔽性但现有 CDN 要求使用源网站的证书的私钥才能提供服务, 这种退让从根本上说是 CDN 技术设计的安全缺陷。若采用文献[35]提出的允许 CDN 提供商和源网站各自独立维护自己的证书和私钥的 CDN 方案就可以更放心地使用 CDN 作为代理服务了。

综上, FTLSocks 通过配合特定的使用方式可以达到更强的隐蔽性、安全性和可用性。未来的工作中可尝试通过整合 Turbo Tunnel 思想[33]在复杂网络环境中提供更高可靠性, 并通过恰当填充降低通过侧信道识别[37]的可能性。

7 结论

FTLSocks 通过设计新的伪装协议兼顾了吞吐量、响应时间和安全性, 最终实现了性能大幅提升、安全性大幅提高的同时, CPU、内存资源消耗提高不明显的效果。新协议得到了类似于 Trojan 的安全性和近似 Shadowsocks 的性能, 同时避免了 Trojan 在反向代理上的天然缺陷。FTLS 协议作为一种新的伪装代理协议是相比现有流行协议更优的协议。

致 谢 感谢陈嘉耕教授在我写作本文的过程中给予的悉心指导。

注: 文中图形均为作者原创, 使用英文进行文字标注是受空间所限。特此说明。

参考文献

[1] Elahi T, Goldberg I. CORDON—A Taxonomy of Internet Censorship Resistance Strategies[EB/OL].

[2] Ensafi R, Fifield D, Winter P, et al. Examining how the Great Firewall Discovers Hidden Circumvention Servers[C]. *The 2015 Internet Measurement Conference*, 2015: 445-458.

[3] Berger T. Analysis of Current VPN Technologies[C]. *First International Conference on Availability, Reliability and Security*, 2006: 8pp.-115.

[4] Karlin J, Ellard D, Jackson A W, et al. Decoy Routing: Toward Unblockable Internet Communication[C]. *FOCI*, 2011.

[5] Manfredi V, Songkuntham P. MultiFlow: Cross-Connection Decoy Routing using {TLS} 1.3 Session Resumption[C]. *8th {USENIX} Workshop on Free and Open Communications on the Internet*, 2018.

[6] Schuchard M, Geddes J, Thompson C, et al. Routing around Decoys[C]. *CCS '12: The 2012 ACM conference on Computer and communications security*. 2012: 85-96.

[7] Wustrow E, Swanson C M, Halderman J A. Tapdance: End-to-middle anticensorship without flow blocking[C]. *23rd {USENIX} Security Symposium*, 2014: 159-174.

[8] Nasr M, Zolfaghari H, Houmansadr A. The Waterfall of Liberty: Decoy Routing Circumvention that Resists Routing Attacks[C]. *CCS '17: The 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017: 2037-2052.

[9] Chen J L, Trang Nguyen U. A Robust Protocol for Circumventing Censoring Firewalls[C]. *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018: 1798-1805.

[10] Brubaker C, Houmansadr A, Shmatikov V. Cloudtransport: Using cloud storage for censorship-resistant networking[C]. *International Symposium on Privacy Enhancing Technologies Symposium. Springer, Cham*, 2014: 1-20.

[11] Fifield D, Lan C, Hynes R, et al. Blocking-Resistant Communication through Domain Fronting[J]. *Proceedings on Privacy Enhancing Technologies*, 2015, 2015(2): 46-64.

[12] Shbair W M, Cholez T, Goichot A, et al. Efficiently Bypassing SNI-Based HTTPS Filtering[C]. *2015 IFIP/IEEE International Symposium on Integrated Network Management*, 2015: 990-995.

[13] Barradas D, Santos N, Rodrigues L. DeltaShaper: Enabling Unobservable Censorship-Resistant TCP Tunneling over Videoconferencing Streams[J]. *Proceedings on Privacy Enhancing Technologies*, 2017, 2017(4): 5-22.

[14] Deri L C, Martinelli M, Bujlow T, et al. NDPI: Open-Source High-Speed Deep Packet Inspection[C]. *2014 International Wireless Communications and Mobile Computing Conference*, 2014: 617-622.

[15] Gancheva Z, Sattler P, Wüstrich L. TLS Fingerprinting Techniques[J]. *Network*, 2020, 15.

[16] Zhao J Y. *Research on Censorship Circumvention Anonymous Techniques*[D]. Beijing: Beijing University of Posts and Telecom, 2016.

(赵进洋. 审查规避匿名技术研究[D]. 北京: 北京邮电大学, 2016.)

[17] Frolov S, Wampler J, Wustrow E. Detecting Probe-resistant Proxies[C]. *Network and Distributed System Security. The Internet Society*, 2020.

[18] Zhiniang Peng. Redirect attack on Shadowsocks stream ciphers. https://github.com/edwardz246003/shadowsocks, Mar. 2019.

[19] McPherson R, Houmansadr A, Shmatikov V. CovertCast: Using Live Streaming to Evade Internet Censorship[J]. *Proceedings on Privacy Enhancing Technologies*, 2016, 2016(3): 212-225.

[20] Li S, Schliep M, Hopper N. Facet: Streaming over Videoconferencing for Censorship Circumvention[C]. *The 13th Workshop on Privacy in the Electronic Society*, 2014: 163-172.

[21] Houmansadr A, Brubaker C, Shmatikov V. The Parrot is Dead: Observing Unobservable Network Communications[C]. *2013 IEEE Symposium on Security and Privacy*, 2013: 65-79.

[22] Frolov S, Wustrow E. The use of TLS in Censorship Circumvention[C]. *NDSS*, 2019.

[23] Ptacek T H, Newsham T N. Insertion, evasion, and denial of service: Eluding network intrusion detection[R]. *Secure Networks inc Calgary Alberta*, 1998.

[24] Appelbaum J. Technical analysis of the Ultrasurf proxying software[J]. *The Tor Project*, 2012.

[25] Lotfollahi M, Jafari Siavoshani M, Shirali Hossein Zade R, et al. Deep Packet: A Novel Approach for Encrypted Traffic Classification Using Deep Learning[J]. *Soft Computing*, 2020, 24(3): 1999-2012.

[26] Saber A, Fergani B, Abbas M. Encrypted Traffic Classification: Combining Over-and Under-Sampling through a PCA-SVM[C]. *2018 3rd International Conference on Pattern Analysis and Intelligent Systems*, 2018: 1-5.

[27] Zeng X M, Chen X S, Shao G L, et al. Flow Context and Host Behavior Based Shadowsocks's Traffic Identification[J]. *IEEE Access*, 2019, 7: 41017-41032.

[28] Liu C, Cao Z G, Xiong G, et al. MaMPF: Encrypted Traffic Classification Based on Multi-Attribute Markov Probability Fingerprints[C]. *2018 IEEE/ACM 26th International Symposium on Quality of Service*, 2018: 1-10.

[29] Zhang Y D, Chen J G, Chen K M, et al. Network Traffic Identification of Several Open Source Secure Proxy Protocols[J]. *International Journal of Network Management*, 2021, 31(2): e2090.

[30] Wang L, Dyer K P, Akella A, et al. Seeing through Network-Protocol Obfuscation[C]. *CCS '15: The 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015: 57-69.

[31] Deng Z Y, Liu Z H, Chen Z G, et al. The Random Forest Based Detection of Shadowsock's Traffic[C]. *2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics*, 2017: 75-78.

[32] Yang Y, Kang C C, Gou G P, et al. TLS/SSL Encrypted Traffic Classification with Autoencoder and Convolutional Neural Network[C]. *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems*, 2018: 362-369.

[33] Fifield D. Turbo Tunnel, a good way to design censorship circumvention protocols[C]. *10th {USENIX} Workshop on Free and Open Communications on the Internet*, 2020.

[34] Frolov S, Wustrow E. {HTTPT}: A Probe-Resistant Proxy[C]. *10th {USENIX} Workshop on Free and Open Communications on the Internet*, 2020.

[35] Wang Z, Li W Q, Cai Q W. Delegation Transparency for HTTPS with CDNS[J]. *Journal of Cyber Security*, 2018, 3(2): 16-30.
(王泽, 李文强, 蔡权伟. 面向 HTTPS 的内容分发网络代理关系透明化[J]. *信息安全学报*, 2018, 3(2): 16-30.)

[36] Han Z H, Chen X S, Zeng X M, et al. Detecting Proxy User Based on Communication Behavior Portrait[J]. *The Computer Journal*, 2019, 62(12): 1777-1792.

[37] Gao P, Guang H, Chen X and Li G S. Traffic Identification based on Side-channel Features for Security Proxy [J]. *Computer Engineering*, 2020: 1-10.
(高平, 广晖, 陈熹, 李光松. 基于侧信道特征的安全代理流量分类[J/OL]. *计算机工程*, 2020: 1-10)

吕英豪 现在华中师范大学计算机学院信息安全专业攻读学士学位。研究领域为网络信息安全、安全协议设计。Email: huraway@mails.ccnu.edu.cn

陈嘉耕 于 2012 年 3 月在日本国立大学北陆先端科学技术大学院大学获得信息科学专业博士学位。现任华中师范大学计算机学院副教授。研究领域为网络安全协议, 算法分析等。Email: jiageng.chen@mail.ccnu.edu.cn