Design of SOCKS Version 6

Vladimir Olteanu, Dragoş Niculescu - University Politehnica of Bucharest

1 INTRODUCTION

Mobile network operators deploy MPTCP [5] on mobile phones to "bond" LTE and WiFi and offer faster network speeds. Since there is very little adoption of MPTCP on the server side, a proxy is needed to terminate the MPTCP connections and communicate with the servers using regular TCP.

Versions 4 and 5 [3] of the SOCKS protocol were developed two decades ago and are still in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers.

Ensuring proxies give state-of-the-art transport performance is key. Indeed, in order to take advantage of the new features of transport protocols, such as TCP Fast Open (TFO) [2] or to enjoy better security against potential on-path attackers, Korea Telecom and other users have resorted to using proxies such as Shadowsocks [1], that use non-standard protocols and are unreviewed in terms of security.

This document describes our design of SOCKS version 6, now undergoing development at the IETF [4]. The key improvement is the elimination of extra round trips between the client and the proxy in most cases. Our design also works around key problems of both TFO and 0-RTT TLS Session Resumption [6] by making requests idempotent using a lightweight mechanism. The protocol is also extensible, allowing further features to be implemented without breaking backward compatibility.

2 BASIC PROTOCOL

When a client wishes to establish a connection to a server, it must open a TCP connection to the SOCKS v6 proxy. The key difference from version 5 is that the client optimistically sends as much information upfront as possible. The request contains the server's IP address (or FQDN) and port, along with an initial chunk of application-layer data (e.g. an HTTP GET). If the client and proxy have previously communicated, it is possible to perform 0-RTT authentication, in which case the SOCKS request will also carry the authentication information.

Next, the proxy sends an authentication reply. If the request did not contain the necessary authentication information, the proxy indicates an authentication method that must proceed. This may trigger a longer authentication sequence during which future 0-RTT authentications can be set up.

Finally, the proxy connects to the remote server and generates an operation reply. All further TCP data is relayed verbatim to and from the server.

In the regular case, when authentication is properly set up (see Fig. 1(b)), the proxy attempts to connect to the server immediately after the receipt of the request, thus incurring no extra delay.

Assuming that the SOCKS v6 proxy is on path, the time it takes to receive a data response from the server (e. g. an HTTP OK) is no worse than the one obtained when directly establishing a TCP connection to the server.

	TFO at proxy	TFO at server	Total RTT
TCP	-	No	$2P + 2S$
	-	Yes	$P + S$
SOCKSv6	No	No	$2P + 2S$
	Yes	No	$P + 2S$
	Yes	Yes	$P + S$

SOCKS v6 can outperform TCP when the remote server does not support TFO. Clients can still use TFO on the client-proxy leg, thus shaving off one client-proxy RTT. This is highly advantageous for mobile deployments, where the latency between the device and the base station can be quite high.

Further, when running SOCKS v6 over TLS, the timings remain unchanged if 0-RTT Session Resumption is used.

In contrast, SOCKS v5 requires two data round-trips (or at least 3, if authentication is used) before application data can pass through (see Fig. 1(a)), and offers no support for TFO on the proxy-server leg.

3 IDEMPOTENCE OF REQUESTS

Both TCP Fast Open and 0-RTT TLS Session Resumption have well-known issues that could lead to replays. Under rare circumstances, a resent TFO SYN could lead to the establishment of a duplicate connection at the server. This can be disastrous if, for example, the server is handling financial transactions and erroneously performs a transaction twice. As for TLS, the initial chunk of data sent out by the client

Posters-Demos '18, August 20–25, 2018, Budapest, Hungary

ACM ISBN 978-1-4503-5915-3/18/08…$15.00
https://doi.org/10.1145/3234200.3234212

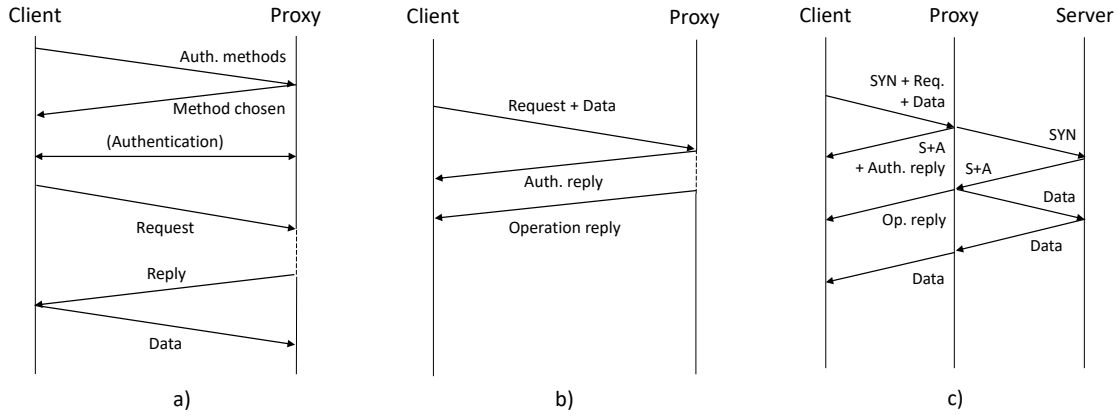

Figure 1: a) SOCKS v5; b) SOCKS v6; c) SOCKS v6 with TFO only between client and proxy

during 0-RTT Session Resumption can be replayed by an on-path attacker, or even unwittingly by the client itself, if it is using TFO.

In order to safely use TFO and 0-RTT Session Resumption on the client-proxy leg, we have implemented a mechanism whereby SOCKS requests can be made idempotent[1]. The proxy will honor an individual request at most once, regardless of how many times it has been replayed.

To protect against duplicate SOCKS requests, authenticated clients can request, and then spend, idempotence *tokens*. A token can only be spent on a single SOCKS request.

Tokens are 4-byte unsigned integers in a modular 4-byte space. Therefore, if x and y are tokens, x is less than y if $0 < (y - x) < 2^{31}$ in unsigned 32-bit arithmetic.

Proxies grant contiguous ranges of tokens called *token windows*. Token windows are defined by their base (the first token in the range) and size. Windows are shifted (i. e. have their base increased, while retaining their size) by the proxy.

Well-behaved clients attempt to spend their tokens in order. This, however, does not guarantee that the requests will be received or processed in order by the proxy. Requests can also be dropped by the network.

The proxy tracks the status of the tokens in the window, always shifting it past spent tokens. To prevent low-order unspent tokens from stalling the window's advancement, they can be forfeited subject to a parameter called the *high water mark*. If the highest spent token exceeds the high water mark, the window is shifted. Increasing this parameter makes the proxy more tolerant to request reordering.

Our chosen solution has the following properties: *a) Light weight*: The memory usage is constant, the proxy performs only one memory allocation per client, and all operations are cheap. This mechanism does not open up an avenue for DoS by itself. *b) Resilient to proxy crashes*: Restarting the proxy would ultimately result in a new token window being allocated. Since the new window is very unlikely to overlap with the old one, the odds of honoring an old request are very low. *c) Resilient across time*: To successfully replay a request, an attacker would have to wait until the window shifts back to the same position (i. e until the client makes 4 billion requests). *d) Timely*: Token window advertisements are sent immediately after receiving the request, as part of the authentication reply. *e) Unrestrictive*: The only restriction placed on the client is that it can make a limited number of requests per client-proxy RTT, which is not an issue if the window size is large enough. Token expenditure requests and window advertisements can be received out-of order.

4 PROJECT STATUS AND DEMO

SOCKS v6 is being used for a trial MPTCP deployment on mobile phones at Orange Romania.

The demo will showcase how mobile phones can use SOCKS v6 on top of MPTCP to take advantage of the combined bandwidth of the mobile and WiFi interfaces.

We will also be demonstrating a TLS early data replay attack, and how enabling the idempotence mechanism protects against the attack.

A SOCKS v6 prototype, along with libraries that ease the development of other apps that use the protocol, can be found at https://github.com/45G.

Acknowledgements

This work was supported in part by a grant from the Romanian National Authority for Scientific Research and Innovation, UEFISCDI project PN-III-P2-2.1-PED-2016-0756.

[1]TFO is used on the proxy-server leg only if the client explicitly requests it.

REFERENCES

[1] 2018. Shadowsocks website. (20 May 2018). http://www.shadowsocks.org

[2] Yuchung Cheng et al. 2014. TCP Fast Open. RFC 7413. (Dec. 2014). https://doi.org/10.17487/RFC7413

[3] Marcus D. Leech. 1996. SOCKS Protocol Version 5. RFC 1928. (March 1996). https://doi.org/10.17487/RFC1928

[4] Vladimir Olteanu and Dragoș Niculescu. 2018. *SOCKS Protocol Version 6.* Internet-Draft draft-olteanu-intarea-socks-6-02. Internet Engineering Task Force. Work in Progress.

[5] Costin et al. Raiciu. 2012. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12).* USENIX Association, Berkeley, CA, USA, 29–29.

[6] Eric Rescorla. 2018. *The Transport Layer Security (TLS) Protocol Version 1.3.* Internet-Draft draft-ietf-tls-tls13-28. Internet Engineering Task Force. Work in Progress.