Bypassing deep packet inspection: Tunneling VPN traffic over TLS

***Abstract*—In some countries network operators employ deep packet inspection techniques in order to block certain types of traffic. For example, Virtual Private Network (VPN) traffic can be analyzed and blocked to prevent non-cooperative users from sending encrypted packets over such networks.**

By observing that HTTPS works all over the world (configured for extremely large number of web-servers) and cannot be easily analyzed (the payload is usually encrypted), we argue that in the same manner VPN tunneling can be organized: By masquerading the VPN traffic with TLS or its older version - SSL, we can build a reliable and secure network. Packets, which are sent over such tunnels, can cross multiple domains, which have various (strict and not so strict) security policies. Despite that the SSH can be potentially used to build such network, we have evidence that in certain countries connections made over such tunnels are analyzed statistically: If the network utilization by such tunnels is high, bursts do exist, or connections are long living, then underlying TCP connections are reset by network operators.

Thus, here we make an experimental effort in this direction: First, we describe different VPN solutions, which exist in the world; second, we describe how SSH traffic can be analyzed and blocked; and, finally, we describe our experimental effort with Python based software, which allows users to create VPN tunnel using TLS protocol.

I. Introduction

Virtual private networks (VPN) are crucial in modern era. By encapsulating and sending clients traffic inside protected tunnels it is possible for users to obtain network services, which otherwise would be blocked by network operator. VPN solutions are also useful when accessing companys Intranet network. For example, corporate employees can access internal network in a secure way by establishing a VPN connection and directing all traffic through the tunnel towards the corporate network. This way they can get services, which are otherwise impossible to get from the outside world.

We believe that VPN solutions make the Internet more liberal.

II. Background

There are various solutions which can be used to build VPNs. One example, is Host Identity Protocols (HIP) [1]. HIP is a layer 3.5 solution (it is in fact located between transport and network layers) and was originally designed to split the dual role of IP addresses - identifier and locator. For example, a company called Tempered Networks uses HIP protocol to build secure networks (for sampling see [6]).

Another solution is Secure Shell protocol (or SSH) [5]. SSH is application layer protocol which provides an encrypted channel for insecure networks. SSH was originally designed to provide secure remote command-line, login, and command execution. But in fact, any network service can be secured with SSH. Moreover, SSH provides means for creating VPN tunnels between the spatially separated networks. Unfortunately, SSH connection can be analyzed and blocked (would it be as widely spread as for example example TLS protocol, things could be different).

Like SSH, OpenVPN [4] runs on top of UDP or TCP transport protocols. We have evidence that in certain countries OpenVPN is successfully blocked by governments. Of course, it is harder to detect this protocols because the traffic is encapsulated inside TCP/UDP connections and deep packet inspection solutions are required in order to effectively block such tunnels.

Another widely used layer 3 protocol for building the VPNs is IPSec protocol [3]. In IPSec security association can be established using pre-shared keys or using Internet Key Exchange protocols (IKE and IKEv2) [2]. Because IPsec runs directly on top of IP protocol, it can be easily detected without usage of sophisticated packet inspection solutions.

III. Hardware and software

To implement the VPN client and server we have used Python framework. The implementation consists roughly 1.2K LOC and all the functions are realized in userspace.

During the experiments, however, we have considered the following setup. For hardware we have selected a micro instance from UpCloud. The instance had single core CPU, 25 GB of data storage, 1GB of random access memory, and was located in Frankfurt, Germany. The client machine was located in Tashkent, Uzbekistan. For software we have selected (on both ends) Ubuntu 18.04. We have also used iperf tool to measure the throughput, and we have used ping utility to measure the round trip times.

IV. Experimental evaluation

In Figure 1 we show the general view of the architecture and in Figure 2 we show working prototype.

We have made several experiments over the course of our work. The very first experiment was related to sending ping messages towards the VPN TLS server and observing the differences in round trip times (basically, we have compared the round trip times between the setting in which tunnel was present and the setting in which the tunnel did not exist). Next, we have sat down to measure the throughput between the local and remote machine. Basically, we have performed 50 measurements for a setting, in which the traffic was going inside the tunnel and the same number of measurements for

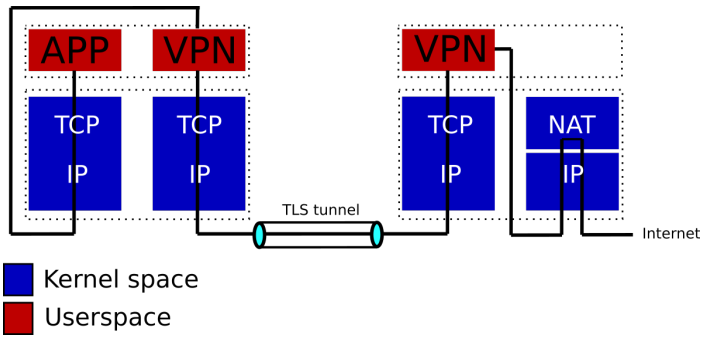

Fig. 1: Architecture of the prototype

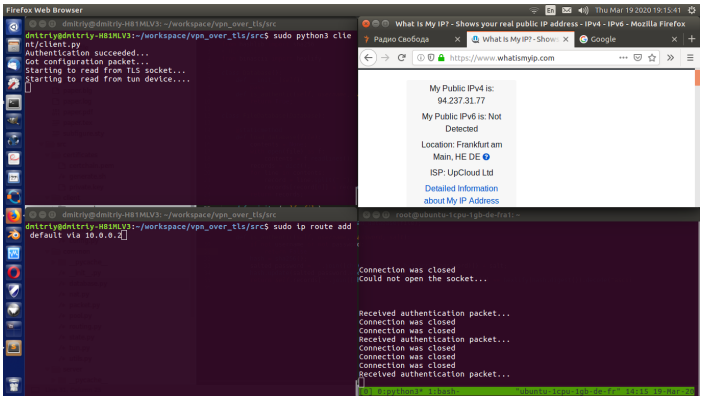

Fig. 2: Working prototype

the setting, in which the traffic was going normally (meaning, unencrypted and not encapsulated in TLS packets).

In Figure 3 we show the distribution of round-trip times (RTT). Obviously, the distributions are similar, and RTT for plain ICMP packets just slightly smaller.

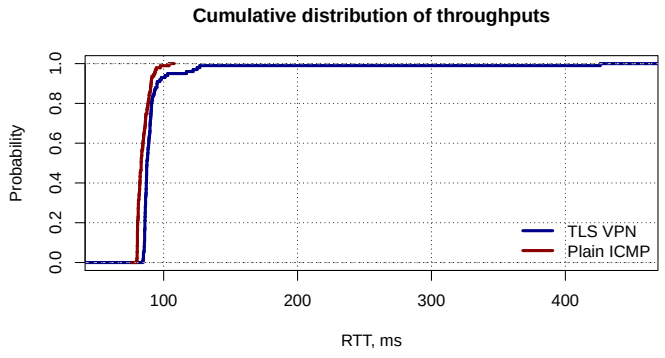

Fig. 3: Distribution of RTTs

In Figure 4 we show the distribution of the obtained throughput for both TLS protected tunnel and regular TCP connections. Mean value for the VPN connection was 6.4 Mb/s, and mean throughput for plain TCP connection was 12.06 Mb/s. Obviously, TCP inside TCP and userspace implementation of VPN client and server, decreases the throughput considerably.

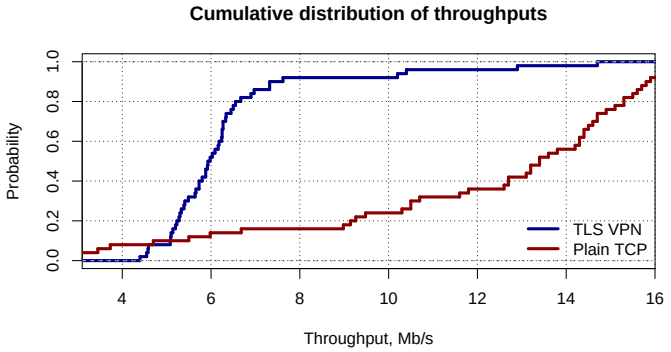

Fig. 4: Distribution of throughputs

Finally, we have made a successful experiment in which the tunnel was used for a long period of time without interruption (the tunnel was used for about 6 hours).

V. Conclusions

In this short report we have made an attempt to describe how VPN solutions can be blocked by network operators. We argued that VPN traffic can still be hided from observers, by masking the tunnels using TLS protocol. Our experiments suggested that this approach is handy when there is a need in secure channel, but network operators are eager to block VPN traffic.

Although it is hard to make a business with this solution (on the long run, network operators can simply block all IP addresses which belong to VPN servers, hosted by a business), we argue, however, that people can use this software and deploy it on their own cloud machines (they can keep the IP addresses of the servers in secret). In this way there is a little chance that network operators can pinpoint the IP addresses of the servers: (i) the traffic will look like a normal HTTPS traffic; (ii) it is rather complex task for a network operators to scan through all IP addresses and find those, which are used to send VPN traffic.

We made the software available for download on a github page. We hope that this project can make the Internet more democratic.

References

[1] Host identity protocol. https://en.wikipedia.org/wiki/Host_Identity_Protocol.

[2] Internet key exchange. https://en.wikipedia.org/wiki/Internet_Key_Exchange.

[3] Ipsec. https://en.wikipedia.org/wiki/IPsec.

[4] Openvpn. https://en.wikipedia.org/wiki/Secure_Shell.

[5] Secure shell. https://en.wikipedia.org/wiki/Secure_Shell.

[6] Tempered networks simplifies secure network connectivity and microsegmentation. https://www.networkworld.com/article/3405853/tempered-networks-simplifies-secure-network-connectivity-and-microsegmentation.html.