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 masquarading 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 over 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 are otherwise would be blocked by network operator. VPN solutions 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 the traffic through the tunnel towards the corporate network. This way they can get services, which are otherwise impossible to get from the outside world [1].

II. Background

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 we have considered the following setup. For hardware we have selected a micro isntance 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 select, 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 genral view of the architecture and in Figure 2 we show working prototype.

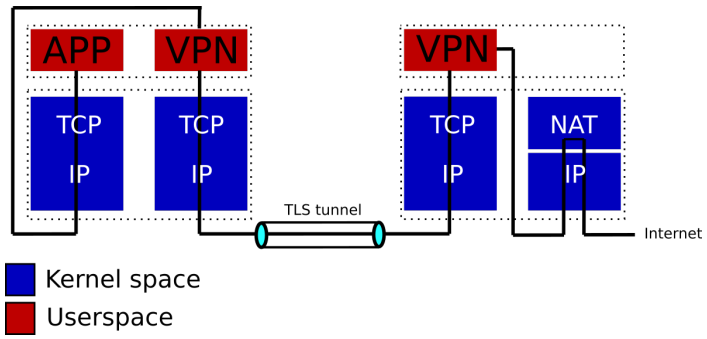

Fig. 1: Architecture of the prototype

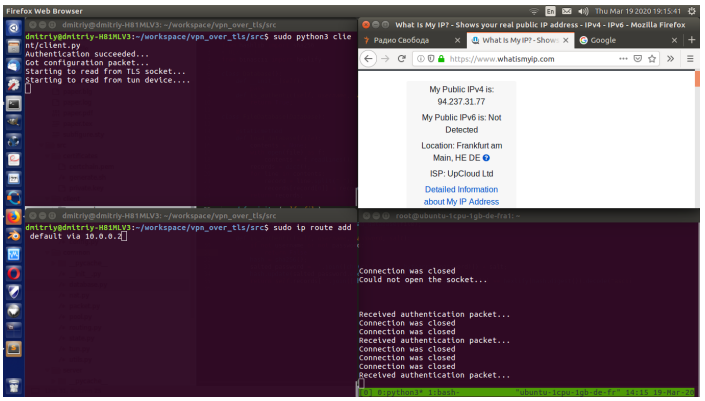

Fig. 2: Working prototype

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 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 the obtained throughputs 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.

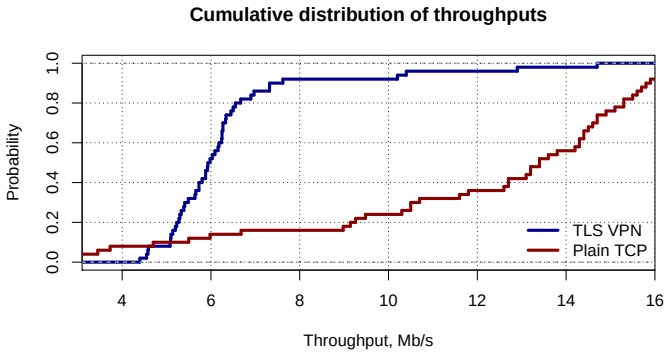

Fig. 3: Distribution of throughputs

In Figure 4 we show the distribution of round-trip times (RTT). Obviously, the distributions are similar, and RTT for plain ICMP packets just slightly smaller.

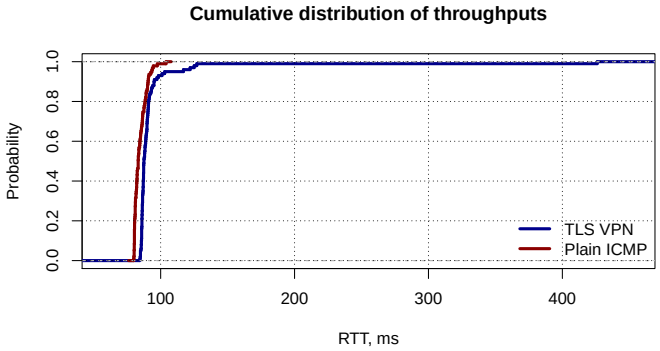

Fig. 4: Distribution of RTTs

V. Conclusions

References

[1] E. Tews, J. Wälde, and M. Weiner. Breaking DVB-CSA. In *Research in Cryptology*, pages 45–61, 2011.