Tunneling protocol

In computer networks, a **tunneling protocol** is a communications protocol that allows for the movement of data from one network to another. It involves allowing private network communications to be sent across a public network (such as the Internet) through a process called encapsulation.

Because tunneling involves repackaging the traffic data into a different form, perhaps with encryption as standard, it can hide the nature of the traffic that is run through a tunnel.

The tunneling protocol works by using the data portion of a packet (the payload) to carry the packets that actually provide the service. Tunneling uses a layered protocol model such as those of the OSI or TCP/IP protocol suite, but usually violates the layering when using the payload to carry a service not normally provided by the network. Typically, the delivery protocol operates at an equal or higher level in the layered model than the payload protocol.

Contents

Uses

A tunneling protocol may, for example, allow a foreign protocol to run over a network that does not support that particular protocol, such as running IPv6 over IPv4.

Another important use is to provide services that are impractical or unsafe to be offered using only the underlying network services, such as providing a corporate network address to a remote user whose physical network address is not part of the corporate network.

Circumventing firewall policy

Users can also use tunneling to "sneak through" a firewall, using a protocol that the firewall would normally block, but "wrapped" inside a protocol that the firewall does not block, such as HTTP. If the firewall policy does not specifically exclude this kind of "wrapping", this trick can function to get around the intended firewall policy (or any set of interlocked firewall policies).

Another HTTP-based tunneling method uses the HTTP CONNECT method/command. A client issues the HTTP CONNECT command to a HTTP proxy. The proxy then makes a TCP connection to a particular server:port, and relays data between that server:port and the client connection.[1] Because this creates a security hole, CONNECT-capable HTTP proxies commonly restrict access to the CONNECT method. The proxy allows connections only to specific ports, such as 443 for HTTPS.[2]

Technical overview

As an example of network layer over network layer, Generic Routing Encapsulation (GRE), a protocol running over IP (IP protocol number 47), often serves to carry IP packets, with RFC 1918 private addresses, over the Internet using delivery packets with public IP addresses. In this case, the delivery and payload protocols are the same, but the payload addresses are incompatible with those of the delivery network.

It is also possible to establish a connection using the data link layer. The Layer 2 Tunneling Protocol (L2TP) allows the transmission of frames between two nodes. A tunnel is not encrypted by default: the TCP/IP protocol chosen determines the level of security.

SSH uses port 22 to enable data encryption of payloads being transmitted over a public network (such as the Internet) connection, thereby providing VPN functionality. IPsec has an end-to-end Transport Mode, but can also operate in a tunneling mode through a trusted security gateway.

To understand a particular protocol stack imposed by tunneling, network engineers must understand both the payload and delivery protocol sets.

Common tunneling protocols

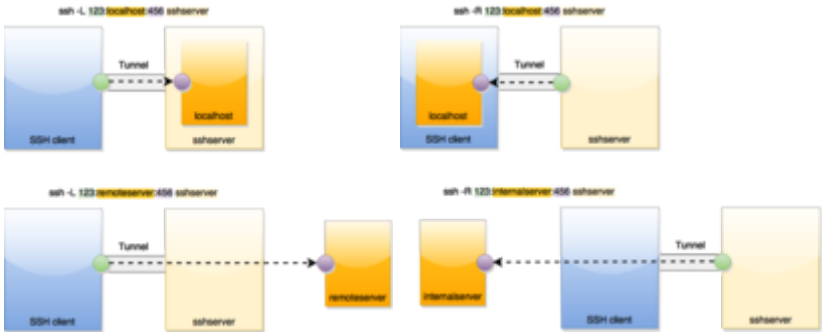
- IP in IP (Protocol 4): IP in IPv4/IPv6
- SIT/IPv6 (Protocol 41): IPv6 in IPv4/IPv6
- GRE (Protocol 47): Generic Routing Encapsulation
- OpenVPN (UDP port 1194)
- SSTP (TCP port 443): Secure Socket Tunneling Protocol
- IPSec (Protocol 50 and 51): Internet Protocol Security
- L2TP (Protocol 115): Layer 2 Tunneling Protocol
- VXLAN (UDP port 4789): Virtual Extensible Local Area Network.
- WireGuard

Secure Shell tunneling

A *Secure Shell (SSH) tunnel* consists of an encrypted tunnel created through an SSH protocol connection. Users may set up SSH tunnels to transfer unencrypted traffic over a network through an encrypted channel. It is a software-based approach to network security and the result is transparent encryption.[3]

For example, Microsoft Windows machines can share files using the Server Message Block (SMB) protocol, a non-encrypted protocol. If one were to mount a Microsoft Windows file-system remotely through the Internet, someone snooping on the connection could see transferred files. To mount the Windows file-system securely, one can establish a SSH tunnel that routes all SMB traffic to the remote fileserver through an encrypted channel. Even though the SMB protocol itself contains no encryption, the encrypted SSH channel through which it travels offers security.

Once an SSH connection has been established, the tunnel starts with SSH listening to a port on the remote or local host. Any connections to it are forwarded to the specified address and port originating from the opposing (remote or local, as previously) host.



Local and remote port forwarding with ssh executed on the blue computer.

Tunneling a TCP-*encapsulating* payload (such as PPP) over a TCP-based connection (such as SSH's port forwarding) is known as "TCP-over-TCP", and doing so can induce a dramatic loss in transmission performance (a problem known as "TCP meltdown"),[4][5] which is why virtual private network software may instead use a protocol simpler than TCP for the tunnel connection. However, this is often not a problem when using OpenSSH's port forwarding, because many use cases do not entail TCP-over-TCP tunneling; the meltdown is avoided because the OpenSSH client processes the local, client-side TCP connection in order to get to the actual payload that is being sent, and then sends that payload directly through the tunnel's own TCP connection to the server side, where the OpenSSH server similarly "unwraps" the payload in order to "wrap" it up again for routing to its final destination.[6] Naturally, this wrapping and unwrapping also occurs in the reverse direction of the bidirectional tunnel.

SSH tunnels provide a means to bypass firewalls that prohibit certain Internet services – so long as a site allows outgoing connections. For example, an organization may prohibit a user from accessing Internet web pages (port 80) directly without passing through the organization's proxy filter (which provides the organization with a means of monitoring and controlling what the user sees through the web). But users may not wish to have their web traffic monitored or blocked by the organization's proxy filter. If users can connect to an external SSH server, they can create an SSH tunnel to forward a given port on their local machine to port 80 on a remote web server. To access the remote web server, users would point their browser to the local port at http://localhost/

Some SSH clients support dynamic port forwarding that allows the user to create a SOCKS 4/5 proxy. In this case users can configure their applications to use their local SOCKS proxy server. This gives more flexibility than creating an SSH tunnel to a single port as previously described. SOCKS can free the user from the limitations of connecting only to a predefined remote port and server. If an application doesn't support SOCKS, a proxifier can be used to redirect the application to the local SOCKS proxy server. Some proxifiers, such as Proxycap, support SSH directly, thus avoiding the need for an SSH client.

In recent versions of OpenSSH it is even allowed to create layer 2 or layer 3 tunnels if both ends have enabled such tunneling capabilities. This creates `tun` (layer 3, default) or `tap` (layer 2) virtual interfaces on both ends of the connection. This allows normal network management and routing to be used, and when used on routers, the traffic for an entire subnetwork can be tunneled. A pair of `tap` virtual interfaces function like an Ethernet cable connecting both ends of the connection and can join kernel bridges.

See also

- HTTP tunnel
- ICMP tunnel
- NVGRE
- GPRS Tunnelling Protocol (GTP)
- Pseudo-wire

- Tunnel broker
- Virtual Extensible LAN (VXLAN)
- Virtual private network (VPN)
- Stunnel
- OSI model (Diagram)

References

1. "Upgrading to TLS Within HTTP/1.1" (http://www.ietf.org/rfc/rfc2817.txt). *RFC 2817*. 2000. Retrieved March 20, 2013.
2. "Vulnerability Note VU#150227: HTTP proxy default configurations allow arbitrary TCP connections" (http://www.kb.cert.org/vuls/id/150227). US-CERT. 2002-05-17. Retrieved 2007-05-10.
3. Barrett, Daniel J.; Barrett, Daniel J.; Silverman, Richard E.; Silverman, Richard (2001). *SSH, the Secure Shell: The Definitive Guide* (https://books.google.com/books?id=JFa5aLIIl6oC&q=secure+shell+tunneling+protocol&pg=PP11). "O'Reilly Media, Inc.". ISBN 978-0-596-00011-0.
4. Titz, Olaf (2001-04-23). "Why TCP Over TCP Is A Bad Idea" (http://sites.inka.de/bigred/devel/tcp-tcp.html). Retrieved 2015-10-17.
5. Honda, Osamu; Ohsaki, Hiroyuki; Imase, Makoto; Ishizuka, Mika; Murayama, Junichi (October 2005). "Understanding TCP over TCP: effects of TCP tunneling on end-to-end throughput and latency". In Atiquzzaman, Mohammed; Balandin, Sergey I (eds.). *Performance, Quality of Service, and Control of Next-Generation Communication and Sensor Networks III*. **6011**. Bibcode:2005SPIE.6011..138H (https://ui.adsabs.harvard.edu/abs/2005SPIE.6011..138H). CiteSeerX 10.1.1.78.5815 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.78.5815). doi:10.1117/12.630496 (https://doi.org/10.1117%2F12.630496). S2CID 8945952 (https://api.semanticscholar.org/CorpusID:8945952).
6. Kaminsky, Dan (2003-06-13). "Re: Extensions for long fat networks?" (https://marc.info/?l=openssh-unix-dev&m=105554033415532). *openssh-unix-dev@mindrot.org* (Mailing list). "the TCP forwarding code is pretty speedy as well. Just to pre-answer a question, ssh decapsulates and re-encapsulates TCP, so you don't have classic TCP-over-TCP issues."

External links

- PortFusion (http://portfusion.sourceforge.net) distributed reverse / forward, local forward proxy and tunneling solution for all TCP protocols
- SSH VPN tunnel, see the SSH-BASED VIRTUAL PRIVATE NETWORKS section (http://www.freebsd.org/cgi/man.cgi?query=ssh)
- BarbaTunnel Project - Free open source implementation of HTTP-Tunnel and UDP-Tunnel on Windows (https://github.com/BarbaTunnelCoder/BarbaTunnel)
- VpnHood Project - Free open source implementation of a VPN using socket redirection (https://github.com/vpnhood/VpnHood)