

HOW SCYPE BYPASSES FIREWALLS

By Jurgen Schmidt

The Hole Trick

Peer-to-peer software applications are a network administrator's nightmare. In order to be able to exchange packets with their counterpart as directly as possible they use subtle tricks to punch holes in firewalls, which shouldn't actually be letting in packets from the outside world.

Increasingly, computers are positioned behind firewalls to protect systems from internet threats. Ideally, the firewall function will be performed by a router, which also translates the PC's local network address to the public IP address (Network Address Translation, or NAT). This means an attacker cannot directly address the PC from the outside - connections have to be established from the inside.

This is of course a problem when two computers behind NAT firewalls require to talk directly to each other - if, for example, their users want to call each other using Voice over IP (VoIP). The dilemma is clear - whichever party calls the other, the recipient's firewall will decline the apparent attack and will simply discard the data packets. The telephone call doesn't happen. Or at least that's what a network administrator would expect.

Punched

But anyone who has used the popular internet telephony software Skype knows that it works as smoothly behind a NAT firewall as it does if the PC is connected directly to the internet. The reason for this is that the inventors of Skype and similar software have come up with a solution.

Naturally every firewall must also let packets through into the local network - after all the user wants to view websites, read e-mails, etc. The firewall must therefore forward the relevant data packets from outside, to the workstation computer on the LAN. However it only does so, when it is convinced that a packet represents the response to an outgoing data packet. A NAT router therefore keeps tables of which internal computer has communicated with which external computer and which ports the two have used.

The trick used by VoIP software consists of persuading the firewall that a connection has been established, to which it should allocate subsequent incoming data packets. The fact that audio data for VoIP is sent using the connectionless UDP protocol acts to Skype's advantage. In contrast to TCP, which includes additional connection information in each packet, with UDP, a firewall sees only the addresses and ports of the source and destination systems. If, for an incoming UDP packet, these match an NAT table entry, it will pass the packet on to an internal computer with a clear conscience.

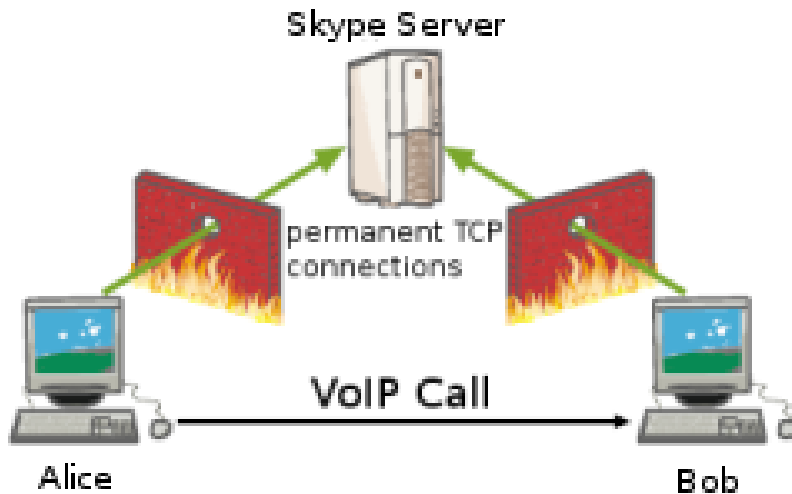
Switching

The switching server, with which both ends of a call are in constant contact, plays an important role when establishing a connection using Skype. This occurs via a TCP connection, which the clients themselves establish. The Skype server therefore always knows under what address a Skype user is currently available on the internet. Where possible the actual telephone connections do not run via the Skype server; rather, the clients exchange data directly.

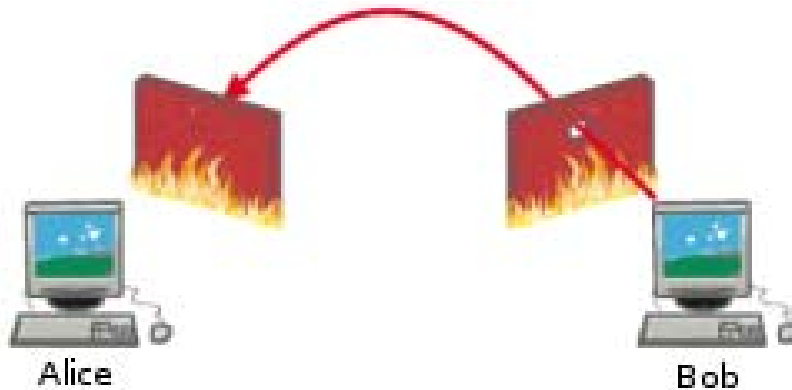
Let's assume that Alice wants to call her friend Bob. Her Skype client tells the Skype server that she wants to do so. The Skype server already knows a bit about Alice. From the incoming query it sees that Alice is currently registered at the IP address 1.1.1.1 and a quick test reveals that her audio data always comes from UDP port 1414. The Skype server passes this information on to Bob's Skype client, which, according to its database, is currently registered at the IP address 2.2.2.2 and which, by preference uses UDP port 2828.

HOW SCYPE BYPASSES FIREWALLS

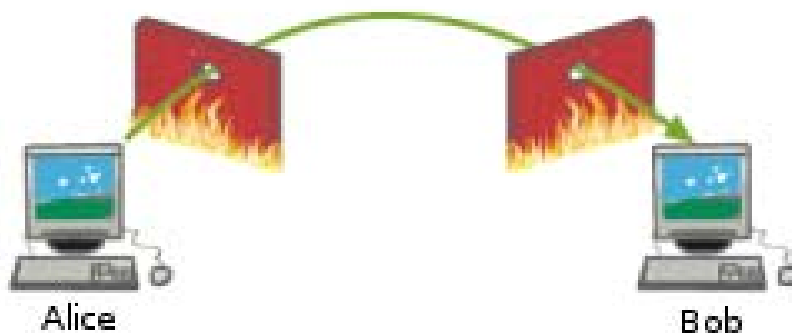
By Jurgen Schmidt



Bob's Skype program then punches a hole in its own network firewall: It sends a UDP packet to 1.1.1.1 port 1414. This is discarded by Alice's firewall, but Bob's firewall doesn't know that. It now thinks that anything which comes from 1.1.1.1 port 1414 and is addressed to Bob's IP address 2.2.2.2 and port 2828 is legitimate - it must be the response to the query which has just been sent.



Now the Skype server passes Bob's coordinates on to Alice, whose Skype application attempts to contact Bob at 2.2.2.2:2828. Bob's firewall sees the recognised sender address and passes the apparent response on to Bob's PC - and his Skype phone rings.



HOW SCYPE BYPASSES FIREWALLS

By Jurgen Schmidt

Doing the rounds

This description is of course somewhat simplified - the details depend on the specific properties of the firewalls used. But it corresponds in principle to our observations of the process of establishing a connection between two Skype clients, each of which was behind a Linux firewall. The firewalls were configured with NAT for a LAN and permitted outgoing UDP traffic.

Linux' NAT functions have the VoIP friendly property of, at least initially, not changing the ports of outgoing packets. The NAT router merely replaces the private, local IP address with its own address - the UDP source port selected by Skype is retained. Only when multiple clients on the local network use the same source port does the NAT router stick its oar in and reset the port to a previously unused value. This is because each set of two IP addresses and ports must be able to be unambiguously assigned to a connection between two computers at all times. The router will subsequently have to reconstruct the internal IP address of the original sender from the response packet's destination port.

Other NAT routers will try to assign ports in a specific range, for example ports from 30,000 onwards, and translate UDP port 1414, if possible, to 31414. This is, of course, no problem for Skype - the procedure described above continues to work in a similar manner without limitations.

It becomes a little more complicated if a firewall simply assigns ports in sequence, like Check Point's FireWall-1: the first connection is assigned 30001, the next 30002, etc. The Skype server knows that Bob is talking to it from port 31234, but the connection to Alice will run via a different port. But even here Skype is able to outwit the firewall. It simply runs through the ports above 31234 in sequence, hoping at some point to stumble on the right one. But if this doesn't work first go, Skype doesn't give up. Bob's Skype opens a new connection to the Skype server, the source port of which is then used for a further sequence of probes.

Nevertheless, in very active networks Alice may not find the correct, open port. The same also applies for a particular type of firewall, which assigns every new connection to a random source port. The Skype server is then unable to tell Alice where to look for a suitable hole in Bob's firewall.

However, even then, Skype doesn't give up. In such cases a Skype server is then used as a relay. It accepts incoming connections from both Alice and Bob and relays the packets onwards. This solution is always possible, as long as the firewall permits outgoing UDP traffic. It involves, however, an additional load on the infrastructure, because all audio data has to run through Skype's servers. The extended packet transmission times can also result in an unpleasant delay.

Source	Destination	Protocol	Info
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38906
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38907
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38893
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38894
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38895
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38896
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38897
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38898
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38899
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38900
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38901
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38892
82.176.176.212	82.82.93.34	TCP	39093 > 46757 [PSH, ACK] Seq=1263 Ack=1243 Win=161
82.82.93.34	82.41.204.47	TCP	51472 > 49803 [PSH, ACK] Seq=55 Ack=3137 Win=5687
82.82.93.34	82.176.176.212	TCP	46757 > 39093 [ACK] Seq=1257 Ack=1338 Win=8656 Len
193.99.15.1	82.82.93.34	UDP	Source port: 38901 Destination port: 35416
82.82.93.34	193.99.15.1	UDP	Source port: 35415 Destination port: 38901
193.99.15.1	82.82.93.34	UDP	Source port: 38901 Destination port: 35416

HOW SCYPE BYPASSES FIREWALLS

By Jurgen Schmidt

Use of the procedure described above is not limited to Skype and is known as "UDP hole punching". Other network services such as the Hamachi gaming VPN application, which relies on peer-to-peer communication between computers behind firewalls, use similar procedures. A more developed form has even made it to the rank of a standard - RFC 3489[3] "Simple Traversal of UDP through NAT" (STUN) describes a protocol which with two STUN clients can get around the restrictions of NAT with the help of a STUN server in many cases. The draft Traversal Using Relay NAT (TURN[4]) protocol describes a possible standard for relay servers.

DIY Hole Punching

With a few small utilities, you can try out UDP hole punching for yourself. The tools required, hping2 and netcat, can be found in most Linux distributions. Local is a computer behind a Linux firewall (local-fw) with a stateful firewall which only permits outgoing (UDP) connections. For simplicity, in our test the test computer remote was connected directly to the internet with no firewall.

Firstly start a UDP listener on UDP port 14141 on the local/1 console behind the firewall:

```
local/1# nc -u -l -p 14141
```

An external computer "remote" then attempts to contact it.

```
remote# echo "hello" | nc -p 53 -u local-fw 14141
```

However, as expected nothing is received on local/1 and, thanks to the firewall, nothing is returned to remote. Now on a second console, local/2, hping2, our universal tool for generating IP packets, punches a hole in the firewall:

```
local/2# hping2 -c 1 -2 -s 14141 -p 53 remote
```

As long as remote is behaving itself, it will send back a "port unreachable" response via ICMP - however this is of no consequence. On the second attempt

```
remote# echo "hello" | nc -p 53 -u local-fw 14141
```

the netcat listener on console local/1 then coughs up a "hello" - the UDP packet from outside has passed through the firewall and arrived at the computer behind it.

Network administrators who do not appreciate this sort of hole in their firewall and are worried about abuse, are left with only one option - they have to block outgoing UDP traffic, or limit it to essential individual cases. UDP is not required for normal internet communication anyway - the web, e-mail and suchlike all use TCP. Streaming protocols may, however, encounter problems, as they often use UDP because of the reduced overhead.

Astonishingly, hole punching also works with TCP. After an outgoing SYN packet the firewall / NAT router will forward incoming packets with suitable IP addresses and ports to the LAN even if they fail to confirm, or confirm the wrong sequence number (ACK). Linux firewalls at least, clearly fail to evaluate this information consistently. Establishing a TCP connection in this way is, however, not quite so simple, because Alice does not have the sequence number sent in Bob's first packet. The packet containing this information was discarded by her firewall.