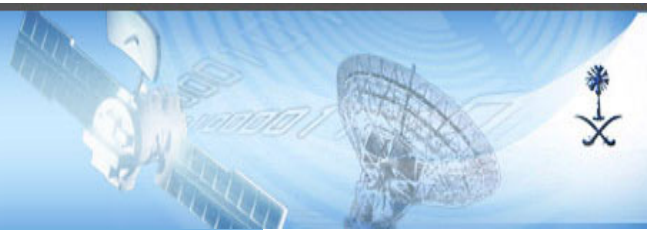
هيئة الاتصالات وتقنية المعلومات
Communications and Information Technology Commission



CITC IPv6 Tunnel Broker

Arun Natarajan S,
Senior System Administrator
CITC Internet operations center
arun.c@citc.gov.sa
+966 56 2804268

Target audience and why

- CITC work to promote IPv6 deployments in the Kingdom
- Still, some providers only give IPv4 service
- End hosts are usually IPv6 capable, but network infrastructure is not necessarily configured (last mile)
- CITC want to promote it as a free national service

Objectives

- Creating an open source tunnel server that would encourage IPv6 deployments
- Building local knowledge and experience
- In addition to CITC having a tunnel broker, any DSP/ISP can deploy the tunnel broker for minimal cost using CITC IPv6 tunnel broker
- Encourage end users and engineers to experiment with IPv6

Tunnel broker background and history

- Developed by CITC in cooperation with NXme consultancy services
- Assignment: ‘Figure out how to provide IPv6 to anyone who wants it. It must provide user authentication and be free; in other words open source’
- The project began in July/August 2010
- Please see http://www.ipv6.sa/tunnelbroker/ for more information on downloading, installation etc.

Evaluation – thinking cap

- How does the rest of the world do it?
- Many solutions:
  - TIC, TSP, AYIYA, protocol 41, 6to4, 6over4, GRE, Teredo
- Many potential customers also are behind NAT
- We could not find any free server software, so we decided to write our own

Design criteria...

- Fast, possibly scalable (distributed)
- User authentication required
- Compatible, usable anywhere (well… at least in many hosts)
- Also compatible with existing licensing in surrounding operating system

...turn to design requirements

- Fast and scalable:
  - meet or surpass $VENDOR's advertised 50k users
  - multiple tunnel servers with easy auth/mgmt
- Authentication:
  - need to define who can use the service
- Compatible:
  - NAT traversal, so protocol 41 or similar is not possible
  - clients available preferably for Windows, Linux, BSD and OSX
  - address potential licensing issues
  - smartphones and tablets ?

Our pick: TSP

- TSP is defined in RFC5572
- TSP defines a tunnel broker and one or more tunnel servers, that may or may not be in same physical host
- TSP is a signaling protocol: it is used for authentication, negotiation of tunnel parameters and tunneled IPv6 address space and such
- TSP uses XML [W3C.REC-xml-2004] basic messaging over TCP or UDP

Our pick: TSP

- Tunnels established by TSP are static tunnels, which are more secure than automated tunnels [RFC3964]; no third party relay required
- No dependency on the underlying IPv4 address
- Discovery of IPv4 NAT in the path
- Both use UDP/3653, difference in first nibble (0xf)

Design decisions...

- Use Linux
- Do tunneled packet processing in kernel space
  - want to avoid context switching
  - possibly simplify things
- Use Python for TSP server
  - common, easy, had necessary bells and whistles (even SASL)
- User database to solve mutual exclusion problem between concurrent clients trying to acquire prefix

...turn into

- kernel module 'utun' and TSP server 'ddtb'
- kernel module is very simple
  - decapsulation just removes the UDP/IPv4 header
  - encapsulation allocates new skb, then memcpy()s headers+payload
  - UDP checksum is optional, no calculation made
- Have all dynamic configuration in the database so a web management interface is easy to implement
- GPLv2 licensing for kernel module, 'GPLv2 or any later' for rest

Questions

- Where do I get it?
  - See http://www.ipv6.sa/tunnelbroker
- Where do I get client?
  - Use any TSP client.
- Where do I get connectivity?
  - CITC provides access to anyone living in Saudi Arabia.
- Reliability?
  - No long-term testing with multiple concurrent clients has been possible (we don't have such user mass that would generate much traffic)

Questions

- What prerequisites are there?
  - linux kernel with accept_local sysctl (>= 2.6.32)
  - Python (install tested on 2.6 and 2.7)
  - C compiler, make

- How do I install it?
  - make && make install
  - follow instructions of make output on how to create client user accounts and admin user accounts (included are command-line tools and web UI)

Credits

- Developers:
  - Mikko Rantanen
  - Kari Mettinen
- Contributors:
  - Dr. Ibraheem Al Furaih
  - Sami Al Daham
  - Arun Natarajan S
  - Bilal Al Sabbagh
  - Pekka Korolainen
  - Ilkka Tuohela
  - Paul Pietkiewicz

Questions?

Thank you.

Implementation

- Each time a client authenticates, create interface of type 'utun'
- Add v4/v6 addresses to this interface and put it into 'UP' state
- Add route to client's IPv6 block through this interface
- Now we have routes and an interface for the client (traffic from client goes to correct interface)

Implementation

- How to identify and forward incoming packets to right interface?
- Mark them using '-j MARK --set-mark <nnn>' to fwmark the packets
- This is done in 'mangle' table PREROUTING chain

Implementation

- For mark <nnn>, add 'ip rule' to set routing table <yyy> for these packets
- Add host route to client interface IP address into route table <yyy>
- Flush route cache
- Result: tunneled traffic goes to correct interface (utun kernel module instance)

Performance

- When sending 100Mbps /dev/zero from client1 via tunnel broker to client2 using socat, top shows:

```
Cpu(s):  0.1%us,  0.1%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.6%si,  0.0%st
```

- vmstat

```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
0  0      0 1379512 133232 302768    0    0     5    15  230   50  0  0 100  0
0  0      0 1379512 133236 302768    0    0     0    54 10336  107  0  1 99  0
0  0      0 1379512 133236 302768    0    0     0     0 10280   75  0  0 100  0
0  0      0 1379512 133236 302768    0    0     0     0 10335   74  0  0 100  0
```

- Commands used:

```
client1:~# socat /dev/zero TCP6-CONNECT:[2001:67c:130:e010::2]:55666
client2:~# socat TCP6-LISTEN:55666,bind=[2001:67c:130:e010::2] pipe:/dev/null
```

- RX/TX byte counters from interface show about 10.5 mebibytes per second throughput (10.5 * 2^20 bytes per second)

Performance

- The python TSP server does 250 authentications per minute when two separate hosts run simultaneously, just logging in. CPU usage is 8%..10% for the python process doing TSP.
- TSP server has Intel Xeon E5410 @ 2.33GHz.
- Ping to tunnel interface (gateway) is received and replied to in about 16-20µs (ProLiant DL360 G5, NetXtreme II BCM5708)
- Regular IPv4 ping tends to have slightly better time of about 12-16µs (IPv4 on eth0)
- Timing seen from tcpdump packet capture timestamps
- Server runs Ubuntu 11.04

Problems

- Conflicting requirements:
  - tested clients expected broker and server in same IP address and UDP port
  - running a daemon doing bind() would steal all packets to userspace
- Classifying and routing incoming packets:
  - TSP signaling traffic (specific bit pattern)

    vs.
  - TSP tunneled traffic
  - both are from all clients to same IPv4/UDP/3653

Problem, 1.1

- We run broker and server in the same machine..
- ..but we can't run them on same IP address..
- ..because TSP daemon would get the tunneled traffic because of bind()..
- ..before the actual tunnel interface..
- ..and we don't want the traffic to go to user space.

Problem, 1.2

- We must run broker and server on different IP addresses..
- ..but tested clients needed broker and server IP to be same, for some reason
- Further, TSP server must bind() to real IP address in a real interface (not our utun-interface)

Solution, 1

- TSP server is run on IP address #1 bound to physical interface
- all tunnel interfaces share a different IP address #2 (no MAC address or ARP capability on interface)
- clients only send traffic to IP address #2: iptables is used to distinguish TSP signaling traffic and RAWDNAT it
- nearby router has ARP entry for shared 'virtual' tunnel interface IP address
  - (you might get away with gARPing)
- now we have broker (TSP) and server (tunnel interface) on two different IPs, clients can talk to broker and authenticate etc.

Solution, 2

- We can route traffic based on client IP address and port to different interface (policy routing based on source)
- For each client, create interface with special name based on client IP address and source port.
  - Client IP 172.16.2.42, port 58022
  - Interface name: ac10022a_58022
- Mark all packets coming from this combination (‘fwmark’)
- Create separate router table, point tunnel server IP to this interface name (‘via device’)
- Create ‘ip rule’ to ‘lookup’ for this ‘fwmark’

Problem 2

- Encapsulated packet comes in and is going to correct IP/ port..
- ..but there are 42 clients and 42 interfaces with same IP address!
- How to distinguish?

Solution, 2.1

- We know client source IP address and port
- We can route traffic based on that to different interface (policy routing based on source)

Solution, 2.2

- For each client, create interface with special name
  - client IP 172.16.2.42, port 58022
  - interface name: ac10022a_58022
  - (interface name length limit 16 characters)
- This is client source IP address and source port
- Mark all packets coming from this combination ('fwmark')
- Create separate route table, point tunnel server IP to this interface name ('via device')
- Create 'ip rule' to 'lookup' for this 'fwmark'

Sample debug output

- 2012-04-03 14:02:54,998 ddtb[11842] DEBUG: Running cmd:
- ip link add name ac10022a_58022 type utun
- ip addr add 192.0.2.69 dev ac10022a_58022
- ip -f inet6 addr add 2001:67c:130:e00e::1 dev ac10022a_58022
- ip link set dev ac10022a_58022 up
- ip route del local 192.0.2.69 dev ac10022a_58022 scope host
- ip route add 2001:67c:130:e00e::/64 dev ac10022a_58022
- iptables -t mangle -A PREROUTING -s 172.16.2.42 -d 192.0.2.69 -p udp -m multiport --sports 58022 -m multiport --dports 3653 -j MARK --set-mark 113
- iptables -t mangle -A INPUT -s 172.16.2.42 -d 192.0.2.69 -p udp -m multiport --sports 58022 -m multiport --dports 3653 -j MARK --set-mark 113
- iptables -t mangle -A OUTPUT -s 172.16.2.42 -d 192.0.2.69 -p udp -m multiport --sports 58022 -m multiport --dports 3653 -j MARK --set-mark 113

Sample debug output

- 2012-04-03 14:02:54,566 session[11842] DEBUG: Customer session 172.16.2.42:58022: init
- 2012-04-03 14:02:54,580 session[11842] DEBUG: md5: authentication successful for login "<removed>"
- 2012-04-03 14:02:55,068 ddtb[11842] DEBUG: Running cmd: ip rule add fwmark 113 table 1113
- … ip route add 192.0.2.69 dev ac10022a_58022 table 1113
- … ip route flush cache
- 2012-04-03 14:02:55,240 ddtb[8834] DEBUG: Child with PID 11842 exited with status code 0.
- 2012-04-03 14:02:55,240 session[8344] DEBUG: Closing customer session: 172.16.2.42:58022