Author: **Sam Protsenko** <semen.protsenko at linaro dot org>

Make Android use upstream PPP VPN code

Contents

Synopsis

This wiki page is intended to document next things:

- Setup configuration for LMG-882 ticket ("Upstream android PPP related drivers")
- Development status
- Debugging notes

Hereafter is described how to create "Android <-> PC" setup:

- BeagleBoard X15 will be acting as PPP client. It may be any other ARM board with Android and Ethernet on it.
- PC will be acting as PPP server

Software:

- The OS used for PC is Debian "stretch" (testing). But it should work flawlessly for Ubuntu as well.
- On BBB X15 I used Android "Nougat"

PC and X15 should be connected via Ethernet patch cord.

This article describes setup of **Legacy VPN** (which is needed for LMG-882 task), for both client (Android board) and server (PC) side.

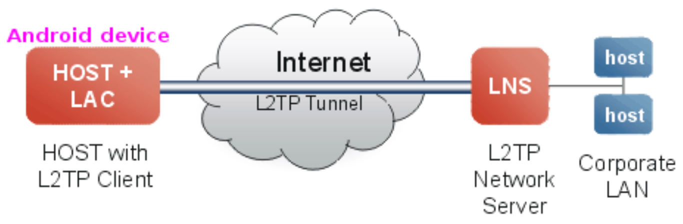
What is "Legacy VPN"?

Prior to Android 4.0, VPN support was entirely built into the platform and wasn’t extensible. Support for new VPN types could only be added as part of platform updates. To distinguish it from **application-based** implementations, built-in VPN support is referred to as **legacy VPN**.

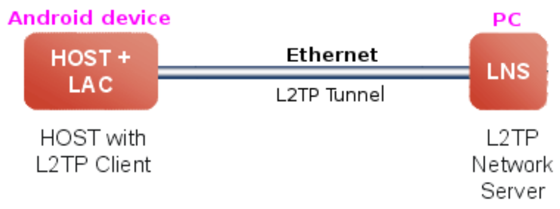
Early Android versions supported different VPN configurations based on PPTP and L2TP/IPsec, with support for “pure-IPSec” VPNs using IPSec Xauth added in version 4.0. In addition to new built-in VPN configurations, Android 4.0 also introduced **application-based VPNs** by supplying the base platform class `VpnService`, which applications could extend in order to implement a new VPN solution.

VPN use-cases

General VPN use-case for Android devices is next: Android device connected to Internet (via WiFi), and we are creating tunnel over Internet to some corporate network. This use-case is shown on picture below.



But in this article we are gonna use another setup (it's more convenient for development): Android device connected to developer's PC via Ethernet. This use-case is shown on picture below.



Busybox

We will need Busybox binary for sending packets via TCP using `netcat` tool (for testing VPN connection).

Build instructions:

- obtain Busybox from `git://git.busybox.net/busybox`
- use latest stable branch
- configure it for static linking (enable `CONFIG_STATIC` option)
- use "-linux" toolchain for building (because it's user-space tool and it uses libc)
  - be sure to use latest Linaro toolchain available (like this one); otherwise you will encounter "MTD_FILE_MODE_RAW definition is missing" error
- once built, it should be copied to `/system/bin` on Android board

Common configuration

Things described in this section are common for both PPTP and L2TP protocols.

Android

Android version I'm using: 7.0 ("Nougat")

See these instructions for AOSP build instructions (for X15 board).

Kernel

kernel version I'm using: 4.4

Kernel for Android should be built with next options enabled:

```
# Android kernel PPP implementation
CONFIG_PPPOPNS=y
CONFIG_PPPOLAC=y

# Upstream kernel PPP implementation
CONFIG_NET_IPGRE_DEMUX=y
CONFIG_PPP=y
CONFIG_PPP_MPPE=y
CONFIG_PPPOE=y
CONFIG_PPTP=y
CONFIG_L2TP=y
CONFIG_L2TP_DEBUGFS=y
CONFIG_L2TP_V3=y
CONFIG_L2TP_IP=y
CONFIG_L2TP_ETH=y
CONFIG_PPPOL2TP=y

# Crypto
CONFIG_CRYPTO=y
```

```
CONFIG_CRYPTO_AUTHENC=y
CONFIG_CRYPTO_MD5=y
CONFIG_CRYPTO_CBC=y
CONFIG_CRYPTO_SHA1=y
CONFIG_CRYPTO_SHA512=y
CONFIG_CRYPTO_RMD160=y
CONFIG_CRYPTO_ECHAINIV=y
CONFIG_CRYPTO_AES=y
CONFIG_CRYPTO_ANUBIS=y
CONFIG_CRYPTO_ARC4=y
CONFIG_CRYPTO_BLOWFISH=y
CONFIG_CRYPTO_BLOWFISH_COMMON=y
CONFIG_CRYPTO_CAMELLIA=y
CONFIG_CRYPTO_CAST_COMMON=y
CONFIG_CRYPTO_CAST5=y
CONFIG_CRYPTO_CAST6=y
CONFIG_CRYPTO_DES=y
CONFIG_CRYPTO_FCRYPT=y
CONFIG_CRYPTO_KHAZAD=y
CONFIG_CRYPTO_SALSA20=y
CONFIG_CRYPTO_CHACHA20=y
CONFIG_CRYPTO_SEED=y
CONFIG_CRYPTO_SERPENT=y
CONFIG_CRYPTO_TEA=y
CONFIG_CRYPTO_TWOFISH=y
CONFIG_CRYPTO_CMAC=y
CONFIG_CRYPTO_HMAC=y
CONFIG_CRYPTO_XCBC=y
CONFIG_CRYPTO_VMAC=y

# IPsec support
CONFIG_XFRM=y
CONFIG_XFRM_USER=y
CONFIG_NET_KEY=y

# Data compression (AH, ESP and IPComp)
CONFIG_INET_AH=y
CONFIG_INET_ESP=y
CONFIG_XFRM_IPCOMP=y
CONFIG_INET_IPCOMP=y

# Support IPsec in NetFilter
CONFIG_IP_NF_MATCH_AH=y
CONFIG_NETFILTER_XT_MATCH_ESP=y
CONFIG_NETFILTER_XT_MATCH_POLICY=y

# Transport, tunnel and BEET mode support
CONFIG_INET_XFRM_TUNNEL=y
CONFIG_INET_XFRM_MODE_TRANSPORT=y
CONFIG_INET_XFRM_MODE_TUNNEL=y
CONFIG_INET_XFRM_MODE_BEET=y

# IPv6 support
CONFIG_INET6_AH=y
CONFIG_INET6_ESP=y
CONFIG_INET6_IPCOMP=y
CONFIG_INET6_XFRM_TUNNEL=y
CONFIG_INET6_XFRM_MODE_TRANSPORT=y
CONFIG_INET6_XFRM_MODE_TUNNEL=y
CONFIG_INET6_XFRM_MODE_BEET=y
CONFIG_IP6_NF_MATCH_AH=y
```

These options enable Android implementation of PPP. Without these there will be some error (on attempt to start mtpd), related to missing PPPoX implementation in kernel.

Particularly we are interested in next two options (added in Android kernel):

Option	**Meaning**	**Details**
CONFIG_PPPOPNS	PNS (PPTP Network Server)	RFC2637
CONFIG_PPPOLAC	LAC (L2TP Access Concentrator)	RFC2661

In order to use mainline implementation of L2TP next options should be enabled in kernel:

- `CONFIG_L2TP`
- `CONFIG_L2TP_DEBUGFS`
- `CONFIG_L2TP_V3`
- `CONFIG_L2TP_IP`
- `CONFIG_L2TP_ETH`
- `CONFIG_PPPOL2TP`

Network details

BBB X15 doesn't have any WiFi chips on it, so I will use Ethernet connection for PPP. Because Android GUI doesn't have an option to use Ethernet, all configuration will be done from console exclusively.

Next networks and addresses will be used in this setup:

```
Ethernet:
    192.168.0.1   - server
    192.168.0.100 - client

PPP:
    192.168.2.1   - VPN server (PC)
    192.168.2.100 - Android (BB X15)
```

Network configuration (PC)

Enable IP forwarding. Edit `/etc/sysctl.conf` and add/change next line:

```
net.ipv4.ip_forward=1
```

and do next to enable this setting for current session:

```
$ sudo service procps start
```

Ethernet configuration for server (in `/etc/network/interfaces`):

```
auto eth0
allow-hotplug eth0
iface eth0 inet static
    address 192.168.0.1
    netmask 255.255.255.0
```

NOTE: if you are using NetworkManager, you also may be needed to enable "managed" mode in NetworkManager config file.

Install PPP daemon:

```
$ sudo aptitude install ppp
```

Next CHAP authentication parameters will be used (corresponding lines will be added to `/etc/ppp/chap-secrets` later):

user	joe
password	test1234

Bring up Ethernet

On Android: bring up Ethernet

```
$ su
# ip addr add 192.168.0.100 dev eth0
# ip link set eth0 up
# ip route add 192.168.0.0/24 dev eth0
# ip route add default via 192.168.0.1 dev eth0
# ip rule add from all lookup main pref 99
```

On PC: if you don't see `eth0` in `ifconfig` output, bring up the Ethernet like this:

```
$ sudo service networking stop
$ sudo service networking start
```

Test Ethernet connection

On Android:

```
# ip addr show dev eth0 scope global

    2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default
qlen 1000
        link/ether a0:f6:fd:ad:d9:b8 brd ff:ff:ff:ff:ff:ff
        inet 192.168.0.100/32 scope global eth0
           valid_lft forever preferred_lft forever

# ip route show

    default via 192.168.0.1 dev eth01
    192.168.0.0/24 dev eth0  scope link

# ping 192.168.0.1

    // ping should work
```

Forward Internet (optional)

This section describes how to provide internet connection to Android from PC via Ethernet.

On PC: Configure IPtables to grant access for your board's subnet to public network (with Internet):

```
$ sudo iptables -A POSTROUTING -t nat -o wlan0 -s 192.168.0.0/24 -j MASQUERADE
```

where public interface is `wlan0` and `192.168.0.0/24` is Ethernet network.

On Android: Configure DNS:

```
# ndc network interface add 100 eth0
# ndc resolver setnetdns 100 localdomain 8.8.8.8 8.8.4.4
# ndc network default set 100
```

Now you can test internet connection in Android browser, or just by pinging some site:

```
# ping google.com
```

Protocols

One of these protocols may be used to create VPN connection:

- PPTP
- L2TP

L2TP is more secure, but PPTP is easier to setup. I need both of them for my task.

PPTP protocol

Server setup

In order to test transfer over PPP, install PPTPD on PC (a.k.a server):

```
$ sudo aptitude install pptpd
```

PPTPD configuration (in `/etc/pptpd.conf`):

```
localip 192.168.2.1
remoteip 192.168.2.100-105
```

Add credentials for authentication using CHAP. In `/etc/ppp/chap-secrets` add next line:

```
joe      pptpd   test1234        *
```

Disable MPPE on server. In `/etc/ppp/pptpd-options`, remove next line (or comment it out):

```
require-mppe-128
```

NOTE: if previous step wasn't done, the connection will be tear down on attempt to start mtpd with next messages:

- on Android, in output of `logcat -s mtpd:*` command:

```
I/mtpd    ( 3323): Remote server hung up
```

- on PC, in `/var/log/syslog`:

```
pppd[9428]: MPPE required but peer refused
```

Restart networking and pptpd services on server:

```
$ sudo service networking stop
$ sudo service networking start
$ sudo service pptpd restart
```

Bring up PPTP

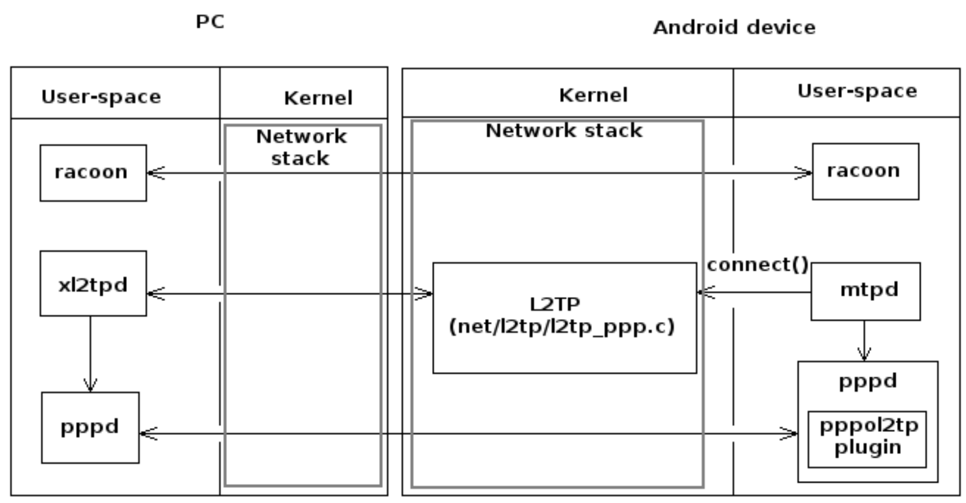
Now when server and client are all set to go, let's bring up PPP connection.

On Android: bring up PPP

```
# mtpd eth0 pptp 192.168.0.1 1723 linkname vpn name joe password test1234 refuse-eap
nodefaultroute usepeerdns idle 1800 mtu 1400 mru 1400 &
```

L2TP protocol

Approximate view of the big picture:



racoon configuration

Racoon is IPSec daemon (running on PC). There are two options how to use it:

- PSK (Pre-Shared Key)
- RSA Certificates

We will stick to PSK as it's easier to setup and use.

First install racoon:

```
$ sudo aptitude install racoon
```

(choose "direct" method, if asked by installer).

racoon version I'm using: 0.8.2

Edit `/etc/racoon/psk.txt` so it has only next one line:

```
myhomelan d41d8cd98f00b204e980
```

NOTE: be sure that there is no trailing spaces after PSK in racoon config file!

Restrict permissions to that file, to avoid security errors:

```
$ sudo chmod 0400 /etc/racoon/psk.txt
```

Parameters used are as follows:

IPSec identifier	myhomelan
IPSec PSK	d41d8cd98f00b204e980

Edit `/etc/racoon/racoon.conf` so it has next content:

```
log notify;
```

```
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote anonymous {
        exchange_mode aggressive;

        generate_policy on;
        nat_traversal on;

        dpd_delay 20;

        proposal {
                encryption_algorithm aes;
                hash_algorithm md5;
                authentication_method pre_shared_key;
                dh_group modp1024;
        }
}

sainfo anonymous {
        encryption_algorithm aes, 3des;
        authentication_algorithm hmac_sha256, hmac_md5;
        compression_algorithm deflate;
}
```

NOTE: pay attention to `exchange_mode`: it should be `aggressive`, since `main` doesn't work with Android for some reasons.

xl2tpd configuration

xl2tpd is L2TP daemon (running on PC).

First install xl2tpd:

```
$ sudo aptitude install xl2tpd
```

xl2tpd version I'm using: 1.3.8

Edit `/etc/xl2tpd/xl2tpd.conf` so it has next content:

```
[global]
access control = no

[lns default]
ip range = 192.168.2.100-192.168.2.105
local ip = 192.168.2.1
require authentication = yes
require chap = yes
refuse pap = yes
length bit = yes
name = l2tpd
pppoptfile = /etc/ppp/xl2tpd-options
```

Edit `/etc/ppp/xl2tpd-options` so it has next content:

```
auth
nodefaultroute
proxyarp
require-chap
ms-dns 8.8.8.8
ms-dns 8.8.4.4
```

Add next line to `/etc/ppp/chap-secrets`:

```
joe           l2tpd   test1234                *
```

Bring up L2TP

On Android: start IPSec client (in background):

```
# racoon eth0 192.168.0.1 udppsk myhomelan d41d8cd98f00b204e980 1701 &
```

On Android: start L2TP client:

```
# mtpd eth0 l2tp 192.168.0.1 1701 "" linkname vpn name joe password test1234 refuse-eap
nodefaultroute usepeerdns idle 1800 mtu 1400 mru 1400 &
```

NOTE: we are passing empty param (`""`) instead of secret, because we don't have `auth` option enabled in xl2tp config file.

If you want to run mtpd with mainline L2TP implementation (under development, may not work properly!), use these instructions.

Test PPP

NOTE: if you can't bring up ppp0 interface (i.e. you can't see ppp0 in "ifconfig" output on PC) -- reflash all Android images, reboot once Android is booted, and try again. It helps somehow (haven't figured why though).

Once PPTP or L2TP is up, we can test if everything works as expected. In this test we will copy file over `ppp0` interface (via TCP protocol), using `netcat` tool.

On Android: test that PPP connection work

```
# ping -I ppp0 192.168.2.1
```

On Android: start receiving file via PPP (using `netcat` from busybox). You may also want to start **Wireshark** on PC (for `ppp0` interface) to sniff TCP packets for file transfer.

```
# busybox nc -l -p 1234
```

On PC: start transmitting file (`file.dat` should contain some text message).

```
$ cat file.dat | nc -w 3 192.168.2.100 1234
```

Now you should see file content in console on Android side. To kill `mtpd` on Android, figure out `mtpd` process ID:

```
# ps | grep mtpd
```

And kill it:

```
# kill <mtpd_pid>
```

Development status

In order to use upstream kernel implementation of L2TP, some changes were made to `mtpd` and `pppd` Android projects. Patches are on review on Android Gerrit (patchsets v2):

- https://android-review.googlesource.com/#/c/platform/external/ppp/+/330806
- https://android-review.googlesource.com/#/c/platform/external/ppp/+/330807
- https://android-review.googlesource.com/#/c/platform/external/ppp/+/330808

- https://android-review.googlesource.com/#/c/platform/external/ppp/+/330809
- https://android-review.googlesource.com/#/c/platform/external/ppp/+/566042
- https://android-review.googlesource.com/#/c/platform/external/ppp/+/566043
- https://android-review.googlesource.com/#/c/platform/external/mtpd/+/330856
- https://android-review.googlesource.com/#/c/platform/external/mtpd/+/330857

If you want to try it out, apply patches and rebuild patched projects. Then run L2TP VPN as usual. Upstream implementation (OL2TP) will be tried first; if it's not supported by your kernel, old OLAC implementation will be used (fall-back mechanism). Check your `logcat` output to see details.

Debugging and development hints

Next projects may need to be debugged:

1. **xl2tpd**: running on PC
2. **mtpd** (part of AFS): running on Android
3. **kernel** (L2TP/PPTP implementation): running on Android

Further described some debugging techniques allow us to debug those projects.

Also we can sniff packets via `eth0` interface using Wireshark.

Wireshark

Change racoon encoding algorithm

In case of L2TP, traffic is encoded by `racoon` server (IPSec), using one of next algorithms:

- DES
- 3DES
- AES

Previously we configured `racoon` to use AES encryption, as it's the most secure of algorithms listed above. Unfortunately, Wireshark is not able (yet) to decrypt AES encrypted packets. So in order to decrypt packets, we need to change `racoon` configuration to use either DES or 3DES algorithm. Let's stick to 3DES, as it's more secure.

Change `/etc/racoon/racoon.conf`, so that `encryption_algorithm` property is set to `3des` in all sections. Also, you can switch log level to `debug2` (most verbose debug), so some extra information can be obtained further from `/var/log/syslog`.

Modified config file should look like this:

```
log debug2;
path pre_shared_key "/etc/racoon/psk.txt";
path certificate "/etc/racoon/certs";

remote anonymous {
        exchange_mode aggressive;

        generate_policy on;
        nat_traversal on;

        dpd_delay 20;

        proposal {
                encryption_algorithm 3des;
                hash_algorithm md5;
                authentication_method pre_shared_key;
                dh_group modp1024;
        }
}

sainfo anonymous {
        encryption_algorithm 3des;
        authentication_algorithm hmac_sha1, hmac_md5;
        compression_algorithm deflate;
}
```

Now, restart `racoon` service:

```
# service racoon restart
```

Run sniffing

Start Wireshark and run sniffing on `eth0` protocol.

Start `racoon` and `mtpd` commands on Android device, as it's described above.

Once packets of interest are caught -- stop sniffing.

Figure out IPSec keys

Run next command (from [8]):

```
# ip xfrm state
```

It prints next params (for both "client -> server" and "server -> client" transmissions):

- SPI (Security Parameter Index)
- Encryption Algorithm (we are using 3DES)
- Encyption Key
- Authentication Algorithm (it's HMAC with SHA-1 as hash function, with MAC truncated to 96 bits)
- Authentication Key

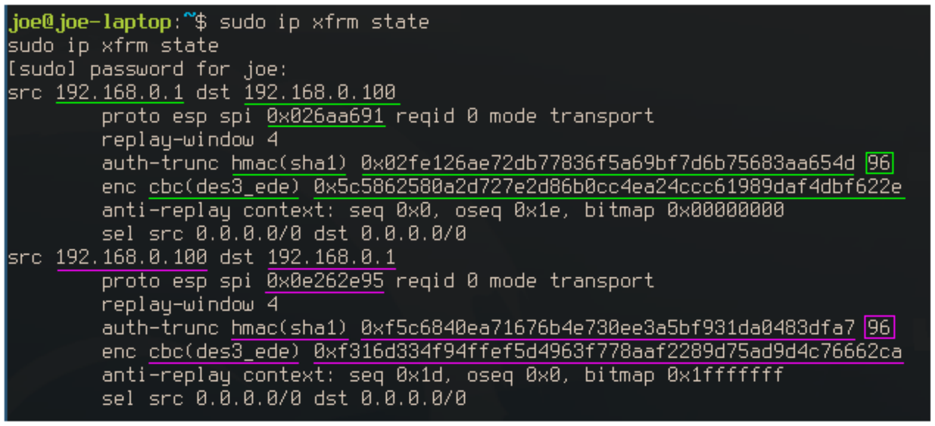
Decrypt ESP packets

We only want to decrypt ESP packets (as ISAKMP packets don't have any interesting information for us).

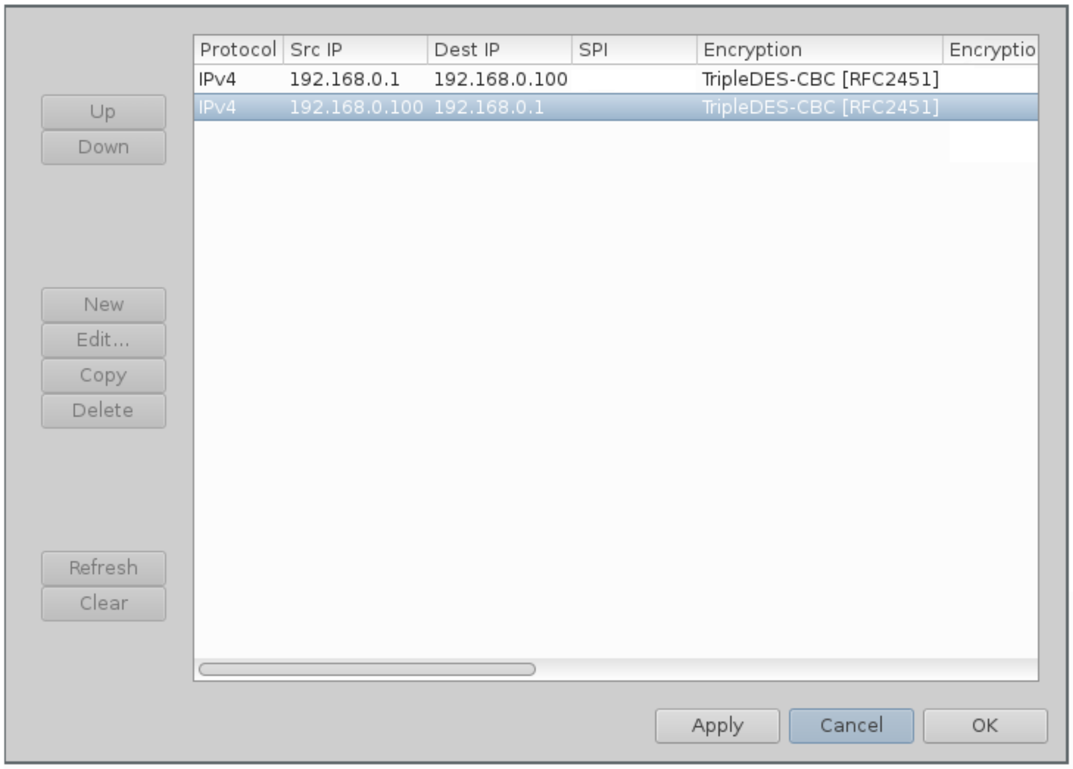
Enable ESP decryption in Wireshark:

```
Edit -> Preferences -> Protocols -> ESP -> Attempt to detect/decode encrypted ESP payloads
```

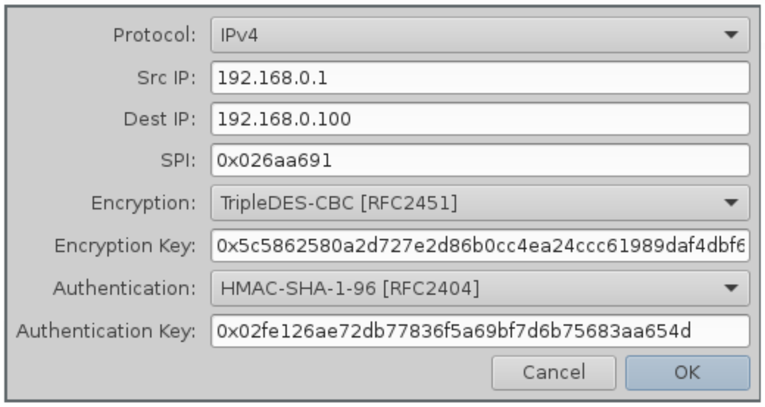
Now add the two ESP SAs (one for each direction), as shown on image below. Use information obtained from `ip xfrm state`.



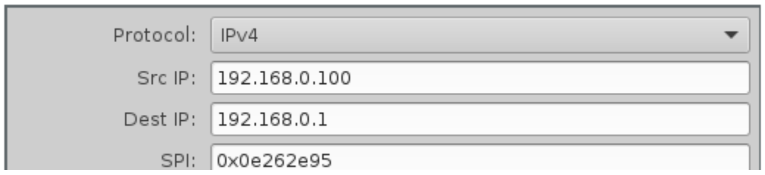
```
joe@joe-laptop:~$ sudo ip xfrm state
sudo ip xfrm state
[sudo] password for joe:
src 192.168.0.1 dst 192.168.0.100
        proto esp spi 0x026aa691 reqid 0 mode transport
        replay-window 4
        auth-trunc hmac(sha1) 0x02fe126ae72db77836f5a69bf7d6b75683aa654d 96
        enc cbc(des3_ede) 0x5c5862580a2d727e2d86b0cc4ea24ccc61989daf4dbf622e
        anti-replay context: seq 0x0, oseq 0x1e, bitmap 0x00000000
        sel src 0.0.0.0/0 dst 0.0.0.0/0
src 192.168.0.100 dst 192.168.0.1
        proto esp spi 0x0e262e95 reqid 0 mode transport
        replay-window 4
        auth-trunc hmac(sha1) 0xf5c6840ea71676b4e730ee3a5bf931da0483dfa7 96
        enc cbc(des3_ede) 0xf316d334f94ffef5d4963f778aaf2289d75ad9d4c76662ca
        anti-replay context: seq 0x1d, oseq 0x0, bitmap 0x1fffffff
        sel src 0.0.0.0/0 dst 0.0.0.0/0
```



Protocol	Src IP	Dest IP	SPI	Encryption	Encryptio
IPv4	192.168.0.1	192.168.0.100		TripleDES-CBC [RFC2451]	
IPv4	192.168.0.100	192.168.0.1		TripleDES-CBC [RFC2451]	

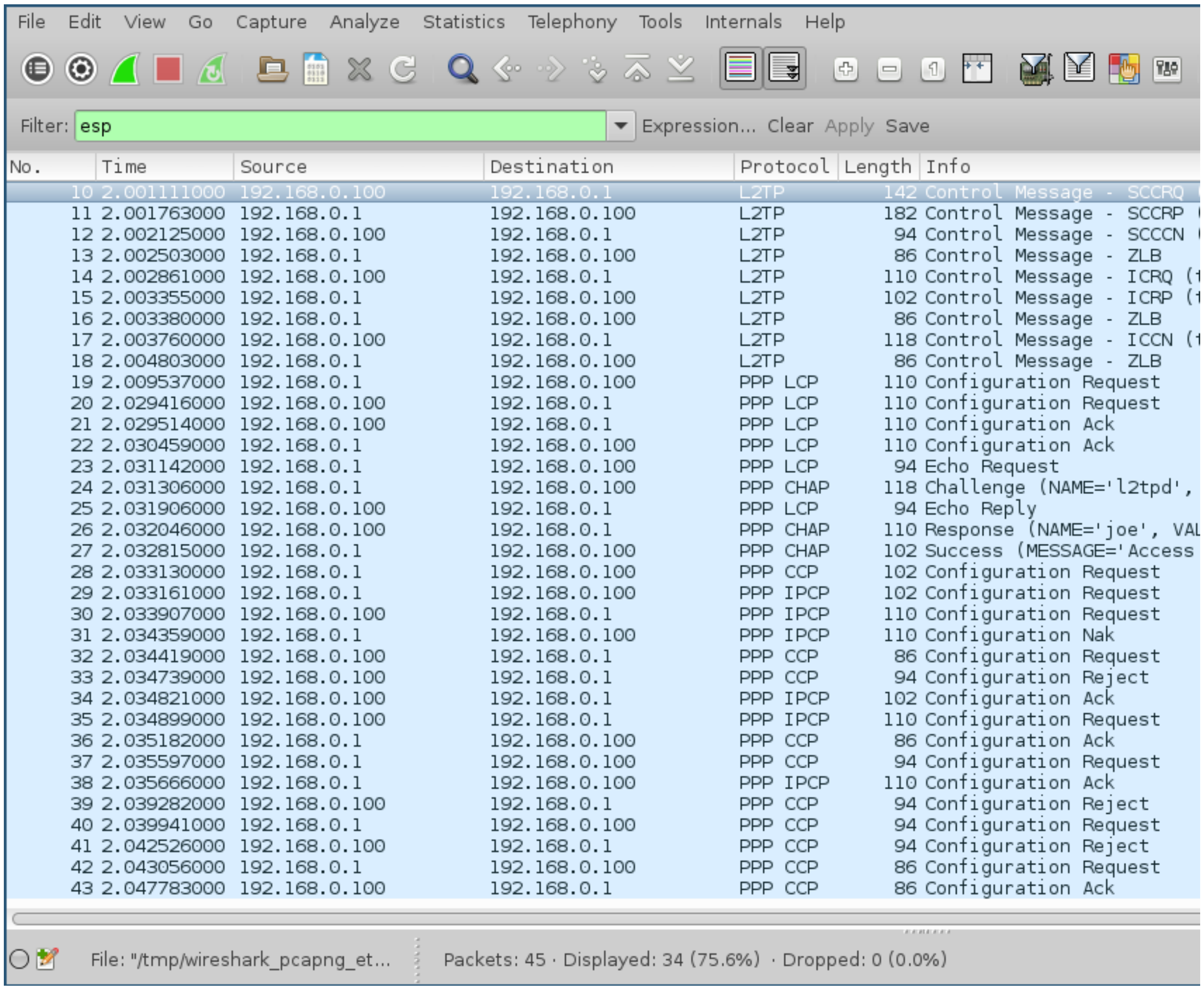


Protocol:	IPv4
Src IP:	192.168.0.1
Dest IP:	192.168.0.100
SPI:	0x026aa691
Encryption:	TripleDES-CBC [RFC2451]
Encryption Key:	0x5c5862580a2d727e2d86b0cc4ea24ccc61989daf4dbf6
Authentication:	HMAC-SHA-1-96 [RFC2404]
Authentication Key:	0x02fe126ae72db77836f5a69bf7d6b75683aa654d



Protocol:	IPv4
Src IP:	192.168.0.100
Dest IP:	192.168.0.1
SPI:	0x0e262e95

Apply changes and all ESP packets will become decrypted, changing their "Protocol" field to "L2TP", "PPP", etc, as it's shown on picture below.



Filter: esp

No.	Time	Source	Destination	Protocol	Length	Info
10	2.001111000	192.168.0.100	192.168.0.1	L2TP	142	Control Message - SCCRQ (
11	2.001763000	192.168.0.1	192.168.0.100	L2TP	182	Control Message - SCCRP (
12	2.002125000	192.168.0.100	192.168.0.1	L2TP	94	Control Message - SCCCN (
13	2.002503000	192.168.0.1	192.168.0.100	L2TP	86	Control Message - ZLB
14	2.002861000	192.168.0.100	192.168.0.1	L2TP	110	Control Message - ICRQ (t
15	2.003355000	192.168.0.1	192.168.0.100	L2TP	102	Control Message - ICRP (t
16	2.003380000	192.168.0.1	192.168.0.100	L2TP	86	Control Message - ZLB
17	2.003760000	192.168.0.100	192.168.0.1	L2TP	118	Control Message - ICCN (t
18	2.004803000	192.168.0.1	192.168.0.100	L2TP	86	Control Message - ZLB
19	2.009537000	192.168.0.1	192.168.0.100	PPP LCP	110	Configuration Request
20	2.029416000	192.168.0.100	192.168.0.1	PPP LCP	110	Configuration Request
21	2.029514000	192.168.0.100	192.168.0.1	PPP LCP	110	Configuration Ack
22	2.030459000	192.168.0.1	192.168.0.100	PPP LCP	110	Configuration Ack
23	2.031142000	192.168.0.1	192.168.0.100	PPP LCP	94	Echo Request
24	2.031306000	192.168.0.1	192.168.0.100	PPP CHAP	118	Challenge (NAME='l2tpd',
25	2.031906000	192.168.0.100	192.168.0.1	PPP LCP	94	Echo Reply
26	2.032046000	192.168.0.100	192.168.0.1	PPP CHAP	110	Response (NAME='joe', VAL
27	2.032815000	192.168.0.1	192.168.0.100	PPP CHAP	102	Success (MESSAGE='Access
28	2.033130000	192.168.0.1	192.168.0.100	PPP CCP	102	Configuration Request
29	2.033161000	192.168.0.1	192.168.0.100	PPP IPCP	102	Configuration Request
30	2.033907000	192.168.0.100	192.168.0.1	PPP IPCP	110	Configuration Request
31	2.034359000	192.168.0.1	192.168.0.100	PPP IPCP	110	Configuration Nak
32	2.034419000	192.168.0.100	192.168.0.1	PPP CCP	86	Configuration Request
33	2.034739000	192.168.0.100	192.168.0.1	PPP CCP	94	Configuration Reject
34	2.034821000	192.168.0.100	192.168.0.1	PPP IPCP	102	Configuration Ack
35	2.034899000	192.168.0.100	192.168.0.1	PPP IPCP	110	Configuration Request
36	2.035182000	192.168.0.1	192.168.0.100	PPP CCP	86	Configuration Ack
37	2.035597000	192.168.0.1	192.168.0.100	PPP CCP	94	Configuration Request
38	2.035666000	192.168.0.1	192.168.0.100	PPP IPCP	110	Configuration Ack
39	2.039282000	192.168.0.100	192.168.0.1	PPP CCP	94	Configuration Reject
40	2.039941000	192.168.0.1	192.168.0.100	PPP CCP	94	Configuration Request
41	2.042526000	192.168.0.100	192.168.0.1	PPP CCP	94	Configuration Reject
42	2.043056000	192.168.0.1	192.168.0.100	PPP CCP	86	Configuration Request
43	2.047783000	192.168.0.100	192.168.0.1	PPP CCP	86	Configuration Ack

File: "/tmp/wireshark_pcapng_et... Packets: 45 · Displayed: 34 (75.6%) · Dropped: 0 (0.0%)

See more details on [8].

Debugging x2ltpd

Enable verbose logs

Add next lines to `/etc/xl2tpd/xl2tpd.conf` file:

```
[global]
...
debug avp = yes
debug network = yes
debug state = yes
debug tunnel = yes

[lns default]
...
ppp debug = yes
```

It will make xl2tpd log messages more verbose. See `/var/log/syslog` file for xl2tpd logs.

Add debug code to xl2tpd

For further investigation one may need to change xl2tpd code (add some tracing messages, etc). We need to be sure that next requirements are met:

- The same version of xl2tpd is used
- Modified xl2tpd installed as Debian package
- Previous xl2tpd removed, but config files should be left (use `remove` command instead of `purge` one)

First of all, obtain xl2tpd sources:

```
$ apt-get source xl2tpd
```

Modify xl2tpd sources (add traces, etc).

Remove old xl2tpd from your system and install build dependencies for xl2tpd:

```
$ sudo service xl2tpd stop
$ sudo aptitude remove xl2tpd
$ sudo aptitude build-dep xl2tpd
```

Build xl2tpd and create deb-package:

```
$ cd xl2tpd-1.3.6+dfsg/
$ sudo debuild -b -uc -us
$ cd ..
```

Install built package:

```
$ sudo dpkg -i xl2tpd_1.3.6+dfsg-3_amd64.deb
$ sudo aptitude hold xl2tpd
```

Finally, make sure that xl2tpd is running:

```
$ sudo service xl2tpd start
```

Now you should be able to catch your traces in `/var/log/syslog`:

```
$ sudo less /var/log/syslog
```

mtpd

Changes to mtpd code can be break down to next categories:

- adding new protocols (like PPTP and L2TP from mainline kernel)
- adding traces and other debugging stuff

Debugging mtpd

Added prints can be seen in `logcat`:

```
# logcat -s mtpd:*
```

Kernel (on Android side)

l2tp debugging

At this point we are only interested in modifying this file: `net/l2tp/l2tp_ppp.c`. The only stuff we should add there is just trace prints (to figure out tunnel id, session id, point where things go wrong, etc).

Added printings can be seen in kernel log (on Android device). Issue next command if you can't see your prints at once (i.e. your log level is not sufficient):

```
# dmesg
```

Kernel log size

If you can see only last part of kernel log, log buffer size needs to be increased. For this next option should be set as follows:

```
CONFIG_LOG_BUF_SHIFT=17
```

Development issues

"Can not find tunnel" issue: investigation

This picture shows comparison of next logs:

- xl2tpd logs for mainline kernel L2TP implementation (on left)
- xl2tpd logs for Android kernel L2TP implementation (on right)

We can see from picture above that `st->ourtid` and `tunnel` values should be the same (like it's happening for Android L2TP implementation). But those values are different when using code that I added to mtpd, which uses mainline kernel L2TP implementation. Because of this xl2tpd cannot find tunnel id in its list (see error in picture above). It's because I return `session_fd` in `create_pppox()` function here. In case when I return `tunnel_fd` instead of `session_fd` -- those values are the same, but it's unable to transmit any data (it's basically because mainline kernel L2TP implementation only allows to send management commands via tunnel, and data is intended to be send via session).

In original mtpd code (`l2tp.c`) there is only one socket being created (`pppox`). Further this socket FD is being passed to pppd process as `pppox` parameter. This commit (for Android pppd implementation) handles `pppox` parameter.

But for modified mtpd code (which is using mainline kernel L2TP implementation) we need to pass 3 parameters to pppd:

- `session_fd`
- `tunnel_id`
- `session_id`

So my suspicion is that we have to port pppol2tp plugin to Android pppd implementation. After that mtpd should be modified so that it runs pppol2tp plugin and passes required parameters for L2TP to it.

Byte order issues

pppol2tp plugin had to be ported, but it didn't fix the issue with erroneous tunnel IDs. It's basically because that issue was caused by wrong byte order in some variables in code.

So there were endianness issues preventing upstream L2TP implementation from properly working. "Wrong tunnel id" issue was caused exactly by wrong byte order. In fact, it can be seen by swapping bytes in picture from this issue:

```
correct tunnel id = 20061 = 0x4e5d
wrong tunnel id   = 23886 = 0x5d4e
```

Decrypting of packets in Wireshark helped to root cause the issue.

First of all, I noticed that in L2TP packets Wireshark shows correct tunnel ID, but in `logcat` I see wrong values. Here is the patches that fix that issue:

- patch #1
- patch #2

Of course, those fixes didn't help me with actual issue, but helped me to figure out what was happening there.

Interesting thing is that I haven't thought about possible byte order issue in tunnel ID value before actually saw code which builds this value from bytes arrived from network (see `mtpd` project, `l2tp_up.c` file, `get_attribute_raw()` function).

So next I checked all variables in `mtpd` which were involved in network transactions, figured which byte order is used for those values and checked those variables usage throughout the whole code. When doing that I noticed, that I'm passing `tunnel_id` and `session_id` in network byte order (big endian) to pppol2tp plugin, when starting pppd. Ok, this is understandable, as in original L2TP code in mtpd they are not passing those IDs at all, so I just haven't thought about this possible issue at the time. Here is the patch that fixes this issue:

- patch #3

But issue still existed, so I continued to check variables. Next thing I found was wrong filling of L2TP structures for further passing to `connect()` call (i.e. in network byte order instead of host byte order). Why I haven't noticed that and I haven't thought of such issue when coding -- it's just because Android implementation requires just the opposite byte order, so I basically just copied the code. Btw, neither documentation mentions which byte order should be used for each field. So the last patch which finally fixes last issue is here:

- patch #4

With all patches listed I managed to run `mtpd` with mainline implementation of L2TP (like it's described here, just be sure to configure network and run `racoon` first: link 1, link 2) and successfully run test (copying file from PC to Android device via `ppp0` interface).

pppol2tp.so is 32-bit instead of 64-bit

As part of transition my patches from Android-N to Android-O, I changed next lines in `external/ppp/pppd/Android.mk`:

```
-LOCAL_MODULE_PATH := $(TARGET_OUT_SHARED_LIBRARIES)/pppd/$(VER)
-LOCAL_UNSTRIPPED_PATH := $(TARGET_OUT_SHARED_LIBRARIES_UNSTRIPPED)/pppd/$(VER)
+LOCAL_MODULE_RELATIVE_PATH := pppd/$(VER)
```

So now I have two versions of `pppol2tp.so` plugin built, 32-bit and 64-bit one:

```
/system/lib/pppd/2.4.7/pppol2tp.so
/system/lib64/pppd/2.4.7/pppol2tp.so
```

When trying test `racoon` + `mtpd`, next error messages appeared in `logcat`:

```
I mtpd    : Starting pppd (tunnel_fd = 8, session_fd = 9)
I mtpd    : Pppd started (pid = 2926)
pppd: dlopen failed: "/system/lib/pppd/2.4.7/pppol2tp.so" is 32-bit instead of 64-bit
pppd: Couldn't load plugin pppol2tp.so
E pppd    : dlopen failed: "/system/lib/pppd/2.4.7/pppol2tp.so" is 32-bit instead of 64-bit
E pppd    : Couldn't load plugin pppol2tp.so
I mtpd    : Received signal 17
I mtpd    : Pppd is terminated (status = 2)
```

```
D mtpd    : Sending STOPCCN
I mtpd    : Mtpd is terminated (status = 34)
```

It was fixed with next patch (in `external/ppp/`):

Toggle line numbers

```
--- a/pppd/pathnames.h
+++ b/pppd/pathnames.h
@@ -55,11 +55,17 @@
 #endif
 #endif /* __STDC__ */
 
+#if defined(__LP64__)
+#define LIB_DIR                "lib64"
+#else
+#define LIB_DIR                "lib"
+#endif
+
 #ifdef PLUGIN
 #ifdef __STDC__
-#define _PATH_PLUGIN   DESTDIR "/lib/pppd/" VERSION
+#define _PATH_PLUGIN   DESTDIR "/" LIB_DIR "/pppd/" VERSION
 #else /* __STDC__ */
-#define _PATH_PLUGIN   "/usr/lib/pppd"
+#define _PATH_PLUGIN   "/usr/" LIB_DIR "/pppd"
 #endif /* __STDC__ */
 
 #endif /* PLUGIN */
```

pppol2tp.so is not building on fresh Android build

Since PPPoL2TP plugin was added to `external/ppp/pppd project`, it has to be built, because `pppd` uses `pppol2tp.so` for L2TP protocol. So let's add `pppol2tp` plugin to `LOCAL_REQUIRED_MODULES` in `pppd` executable definition (`Android.mk` file):

Toggle line numbers

```
 LOCAL_MODULE:= pppd
+LOCAL_REQUIRED_MODULES := pppol2tp
 include $(BUILD_EXECUTABLE)
```

Another way to do that would be adding it to `LOCAL_SHARED_LIBRARIES` variable of `pppd` executable definition in `external/ppp/pppd/Android.mk`, but it would lead to linking `pppd` binary against `pppol2tp.so`, which we don't want, as `pppol2tp.so` is just a plugin and should be loaded via `dlopen()` only when it's needed.

Also we could add it to `PRODUCT_PACKAGES` variables in `build/target/product/*.mk` files, but it's better to specify it as a dependency of `pppd`, which it actually is, and not tinker with such a generic files like `build/...`.

pppol2tp.so can't be loaded by Bionic loader

When PPPoL2TP plugin resides in `/system/lib[64]/pppd/2.4.7/`, it can't be loaded with `dlopen()` from `pppd` binary, as this path is not in permitted path for default linker namespace. Next errors can be observed in `logcat` output:

```
E linker  : library "/system/lib64/pppd/2.4.7/pppol2tp.so" ("/system/lib64/pppd/2.4.7
/pppol2tp.so") needed or dlopened by "/system/bin/pppd" is not accessible for the namespace:
[name="(default)", ld_library_paths="", default_library_paths="/system/lib64",
permitted_paths="/system/lib64/drm:/system/lib64/hw:/system/framework:/system/app:/system/priv-
app:/vendor/app:/vendor/framework:/oem/app:/data:/mnt/expand"]

E pppd    : dlopen failed: library "/system/lib64/pppd/2.4.7/pppol2tp.so" needed or dlopened by
"/system/bin/pppd" is not accessible for the namespace "(default)"

E pppd    : Couldn't load plugin pppol2tp.so
```

To fix this, let's install the `pppol2tp.so` in root of `/system/lib[64]/` without `pppd/2.4.7/` sub-path, and make `pppd` to look there for plugins too:

Toggle line numbers

```
--- a/pppd/Android.mk
+++ b/pppd/Android.mk

 include $(CLEAR_VARS)
 LOCAL_MODULE := pppol2tp
-VER := $(shell awk -F '"' '/VERSION/ { print $$2; }' $(LOCAL_PATH)/patchlevel.h)
-LOCAL_MODULE_RELATIVE_PATH := pppd/$(VER)

--- a/pppd/pathnames.h
+++ b/pppd/pathnames.h

 #ifdef PLUGIN
 #ifdef __STDC__
-#define _PATH_PLUGIN   DESTDIR "/" LIB_DIR "/pppd/" VERSION
+#define _PATH_PLUGIN   DESTDIR "/" LIB_DIR
 #else /* __STDC__ */
-#define _PATH_PLUGIN   "/usr/" LIB_DIR "/pppd"
+#define _PATH_PLUGIN   "/usr/" LIB_DIR
 #endif /* __STDC__ */
```

See `/etc/ld.config.txt` file on the board for Bionic loader configuration. In Android sources, there are three variants of `ld.config.txt` files (in `system/core/rootdir/etc`):

1. `ld.config.legacy.txt`: for non-Treble-ized devices (`PRODUCT_TREBLE_LINKER_NAMESPACES` is false)
2. `ld.config.txt`: for Treble-ized devices (`PRODUCT_TREBLE_LINKER_NAMESPACES` is true) but without strict linker namespace restriction (`BOARD_VNDK_RUNTIME_DISABLE` is true)
3. `ld.config.txt.in`: for Treble-ized devices with strict linker namespace restriction

Only 3rd option can cause the described problem. The lib can't be loaded because:

- the default namespace is configured as 'isolated' and
- `system/${LIB}/pppd` is not in the permitted paths of the default namespace

Possible solutions:

1. Add `/system/${LIB}/pppd` to `namespace.default.permitted.paths` or
2. Move your lib under `/system/lib64` (or under other paths in `permitted.paths`)

I've chosen 2nd one, as it doesn't require changing any linker files, and it's gonna work in all three cases mentioned above.

Issues running L2TP+IPSec on Android-N

When trying to run racoon+mtpd on Android-N, I faced with a bunch of issues. All those issues are listed below. Solutions are provided as well.

"Network is unreachable" issue

After configuring the Ethernet connection, I tried to ping my laptop from Android:

```
# ping 192.168.0.1
```

which gave me this error: `connect: Network is unreachable`. The only way to overcome that was running `ping -I eth0 192.168.0.1`, but that of course won't cut it (as we need to rely on eth0 as default interface for further PPP work).

Turns out, all `ip route` commands we ran, they were changing only main routing table (`ip route show` is actually `ip route show table main`). And main routing table has lowest priority (look at `ip rule show` command output, it's 23000 priority or even missing there). Setting the higher priority for main routing table solves the problem:

```
# ip rule add from all lookup main pref 99
```

After that we can check it:

```
# ip rule show
```

will show us something like this:

```
0:      from all lookup local
99:     from all lookup main
10000:  from all fwmark 0xc0000/0xd0000 lookup legacy_system
10500:  from all oif eth0 uidrange 0-0 lookup eth0
13000:  from all fwmark 0x10063/0x1ffff lookup local_network
13000:  from all fwmark 0x10064/0x1ffff lookup eth0
14000:  from all oif eth0 lookup eth0
15000:  from all fwmark 0x0/0x10000 lookup legacy_system
16000:  from all fwmark 0x0/0x10000 lookup legacy_network
17000:  from all fwmark 0x0/0x10000 lookup local_network
19000:  from all fwmark 0x64/0x1ffff lookup eth0
22000:  from all fwmark 0x0/0xffff lookup eth0
32000:  from all unreachable
```

where you can see main routing table with highest priority. Now ping command works just fine.

Above fix should also allow non-root user to use eth0 interface.

PSK permissions issue

racoon refuses to start, and reason is found in `/var/log/daemon.log`:

```
ERROR: /etc/racoon/psk.txt has weak file permission
```

FIX: The issue is fixed by changing mode of mentioned file:

```
$ sudo chmod 0400 /etc/racoon/psk.txt
```

"authtype mismatched" issue

`logcat` shows next line, when trying to run `racoon`:

```
W racoon  : authtype mismatched: my:hmac-sha256 peer:hmac-sha
```

FIX: In `/etc/racoon/racoon.conf`: change this line:

```
authentication_algorithm hmac_sha1, hmac_md5;
```

to this:

```
authentication_algorithm hmac_sha256, hmac_md5;
```

Supplemental: it turned out, that this issue is not critical, in the end. VPN will work with `hmac_sha1`, too. But let's use `hmac_sha256` just to avoid that warning.

Also, some other guys are experiencing issues with SHA2 implementation in Android: [9].

But it seems like my configuration (described at this wiki page) is not prone to that issue.

"pfkey failed" racoon issue

When running `racoon` + `mtpd`, I observe next issues in `logcat`:

```
pfkey UPDATE failed: No such file or directory
pfkey ADD failed: No such file or directory
```

The cause most probably (related to what I found on internet) is that `AUTHENC` module is not loaded in kernel. But I've got `CONFIG_CRYPTO_AUTHENC=y` in my `.config` file.

When I look into `/proc/crypto`, I don't see `authenc` module in list (but I see it in my PC's `/proc/crypto`). So it's not loaded somehow. `/proc/crypto` has entries like that: `kmod="crypto-ctr(aes)"`, where:

- `aes` is cipher
- `ctr` is template

See this for details.

`AUTHENC` module is trying to be loaded when starting `mtpd`:

```
### cryptomgr_probe(): cra_name        = echainiv(authenc(hmac(sha256),cbc(aes)))
  XXX: goto out
  XXX: out: echainiv(authenc(hmac(sha256),cbc(aes)))
### cryptomgr_probe(): cra_name        = echainiv(authenc(hmac(sha256),cbc(aes)))
  XXX: goto out
  XXX: out: echainiv(authenc(hmac(sha256),cbc(aes)))
```

Which means that this code was executed (from `cryptomgr_probe()` function):

```
        tmpl = crypto_lookup_template(param->template);

        if (!tmpl) {
                pr_err("  XXX: goto out");
                goto out;
        }
```

When we look at `echainiv(authenc(hmac(sha256),cbc(aes)))`, there are several components in play. Checking each of them in kernel configuration reveals that `CONFIG_CRYPTO_ECHAINIV=m`. Let's make it `=y`.

FIX: After enabling `CONFIG_CRYPTO_ECHAINV`, everything works. Another way to root cause that option would be to enable all crypto options in kernel, and once it works, bisect which one was missing (or just leave all crypto options enabled).

"Received signal 15" mtpd issue

`racoon` now works fine. But when trying to start `mtpd`, I'm observing next messages in `logcat` (Android side):

```
D mtpd    : Timeout -> Sending SCCRQ
I mtpd    : Received signal 15
D mtpd    : Sending STOPCCN
I mtpd    : Mtpd is terminated (status = 5)
```

On PC side I see next output in `/var/log/daemon.log`:

```
xl2tpd[3221]: check_control: Received out of order control packet on tunnel -1 (got 1, expected 0)
xl2tpd[3221]: handle_packet: bad control packet!
xl2tpd[3221]: network_thread: bad packet
```

```
xl2tpd[3221]: build_fdset: closing down tunnel 47774
```

or this one:

```
xl2tpd[9998]: result_code_avp: avp is incorrect size.  8 < 10
xl2tpd[9998]: handle_avps: Bad exit status handling attribute 1 (Result Code) on mandatory packet.
xl2tpd[9998]: handle_packet: bad AVP handling!
xl2tpd[9998]: network_thread: bad packet
xl2tpd[9998]: build_fdset: closing down tunnel 40027
```

FIX: Issue was fixed by just re-flashing Android images. Don't know what was wrong, but now it goes further.

Observations:

- usually it's enough to re-flash just `userdata.img`
- it was noticed that this problem occurs after running `reboot` command; it probably breaks something on userdata partition; so it's probably better to use hardware RESET button instead

"Received signal 15" mtpd issue (#2)

After moving to Android-O kernel, this issue appeared again. Sometimes it stops with signal 15:

```
D mtpd    : Timeout -> Sending SCCRQ
I mtpd    : Received signal 15
D mtpd    : Sending STOPCCN
I mtpd    : Mtpd is terminated (status = 5)
```

And sometimes it's just printing "Timeout -> Sending SCCRQ" messages constantly in `logcat`.

Turns out that HMAC SHA-256 truncation was changed from 96 bit to 128 bit in Android-O kernel (e.g. `android-4.4-o` branch), as required by RFC4868. This commit changes it:

ANDROID: make PF_KEY SHA256 use RFC-compliant truncation

But my server's `racoon` still uses 96-bit truncation, because it relies on mainline kernel, which is using 96-bit truncation.

FIX: There are two possible solutions to this issue:

1. Change server IPSec tool from `racoon` to another one (such as `libreswan`), which implements this SHA256 truncation in userspace (and able to choose 128 bit truncation)

2. Revert mentioned patch

For now I decided to go with revert, as configuring `libreswan` may take a lot of time, and hacking `racoon` won't help either (as it relies on kernel). In future I'll still need to provide working Road Warrior `libreswan` configuration, because we'll need simple and reliable way to test my patches.

It also should be noted that Android-N kernel uses 96-bit truncation, thus server must use it too.

Debugging the truncation

This issue can be seen from `ip xfrm state` command output.

On PC:

```
$ sudo ip xfrm state
src 192.168.0.1 dst 192.168.0.100
        ...
        auth-trunc hmac(sha256) ... 96
        ...
src 192.168.0.100 dst 192.168.0.1
        ...
        auth-trunc hmac(sha256) ... 96
```

On Android:

```
# ip xfrm state
src 192.168.0.100 dst 192.168.0.1
        ...
        auth-trunc hmac(sha256) ... 128
        ...
src 192.168.0.1 dst 192.168.0.100
        ...
        auth-trunc hmac(sha256) ... 128
        ...
```

"avp is incorrect size. 8 < 10" issue

After re-flashing Android images, I was faced with another issue. From `logcat`:

```
I mtpd    : Mtpd is terminated (status = 6)
```

Looking at `/var/log/daemon`:

```
xl2tpd[23168]: result_code_avp: avp is incorrect size.  8 < 10
```

Trying to figure out what's going on, I enabled debug logging in `/etc/xl2tpd/xl2tpd.conf`:

```
debug avp = yes
debug network = yes
debug state = yes
debug tunnel = yes
...
ppp debug = yes
```

Restarted xl2tpd service, started mtpd again. Looking at `/var/log/syslog` (as it was last modified file in `/var/log/`):

```
pppd[24195]: In file /etc/ppp/xl2tpd-options: unrecognized option 'lock'
```

So, it seems like `lock` option is deprecated now.

FIX: Removing `lock` option from `/etc/ppp/xl2tpd-options` fixes the problem, and now `ppp0` interface comes up, as seen in `ifconfig` output (on both sides, Android board and PC).

References

[1] http://www.marthijnvandenheuvel.com/2012/05/26/how-to-set-up-a-pptp-vpn-server-on-ubuntu/

[2] https://wiki.debian.org/HowTo/AndroidVPNServer

[3] http://www.brokenbitstream.com/mobile-and-vpns

[4] http://alvinalexander.com/java/jwarehouse/android/packages/VpnServices/src/com/android/server/vpn/VpnDaemons.java.shtml

[5] http://wiki.nikoforge.org/L2TP/IPSec_VPN_Setup_on_Centos_6_%2864-bit%29_for_use_with_Android_ICS_and_iOS_5_Clients

[6] "Android Security Internals" book: chapter 9 -> "Vpn Support"

[7] http://miqloproxy.com/what-is-l2tp/

[8] https://ask.wireshark.org/questions/12019/how-can-i-decrypt-ikev1-andor-esp-packets

[9] https://code.google.com/p/android/issues/detail?id=196939

Links from Grygorii Strashko

PPPoLAC links

- http://blog.csdn.net/linweig/article/details/6127270

- http://changyihsin.github.io/blog/2013/06/06/android-vpn-howto/

- https://books.google.com.ua/books?id=y11NBQAAQBAJ&pg=PA231&lpg=PA231&dq=android+pppolac&source=bl&ots=nUUCzOqX-y&sig=_pD6UcH1uzeS07X9tidTjbM1hGE&hl=en&sa=X&ei=_d8PVbuHCKnmyQO_lIJo&ved=0CC8Q6AEwBA#v=onepage&q=android%20pppolac&f=false

- http://www.techrepublic.com/blog/smartphones/connect-to-a-pptp-vpn-from-your-android-phone/

- RFC3931

- RFC2637

- PPTP on wikipedia

- http://pptpclient.sourceforge.net/#tryit

CTS tests links

- https://android.googlesource.com/platform/external/ppp/+/android-cts-4.1_r2

- https://source.android.com/compatibility/cts/index.html

- https://android.googlesource.com/platform/cts/+/33b838b

LMG/Kernel/PPP (last edited 2017-12-12 21:24:32 by semen.protsenko@linaro.org)