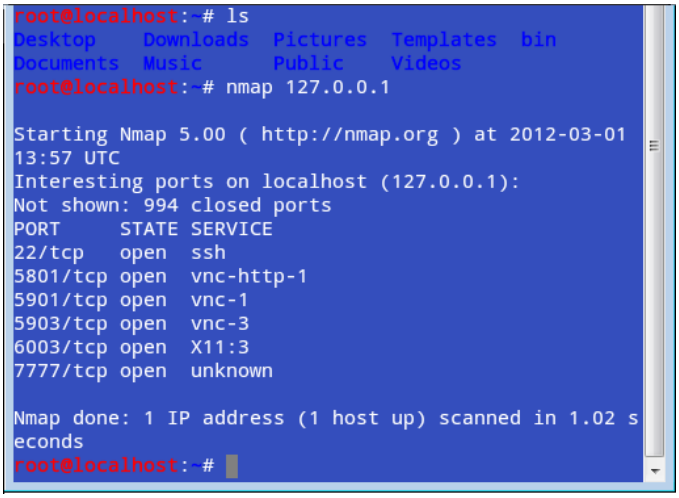
안드로이드 수상한 포트 확인 및 차단하기

UknowY
http://facebook.com/tinylament
2012. 3. 3

어느 날 안드로이드에서 백트랙을 부팅하면서 테스트를 해보려고 localhost를 **nmap**으로 스캔해 보았다.



그런데 아니 이게 무엇인가.

7777번 port가 unknown 서비스에 의해서 열려있었다.

데스크탑이나 노트북에서도 그렇듯 backdoor로 의심되는 포트가 있는 것이다.

아마 마켓에서 여러가지 어플들로 장난을 하다 딸려 온 악성 프로그램에서 열어 놓은 듯 하다.

먼저 구글에서 7777번에 대한 포트를 찾아 보았다.

Home » Ports Database » Port Details

Port 7777 Details

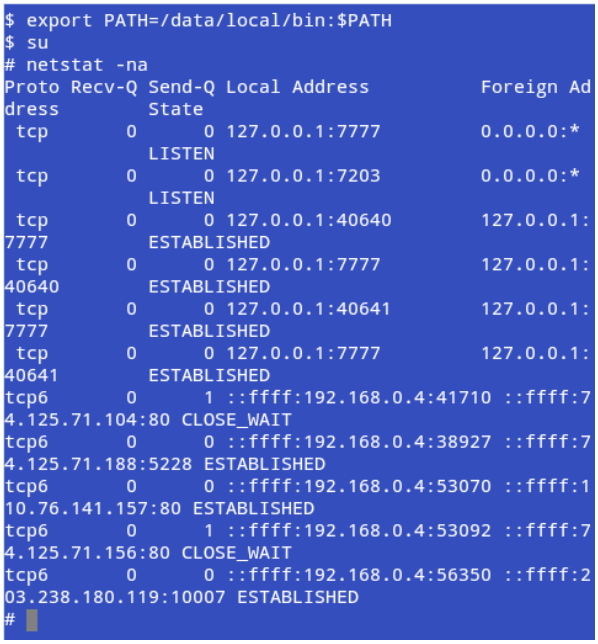
threat/application/port search: SEARCH

known port assignments

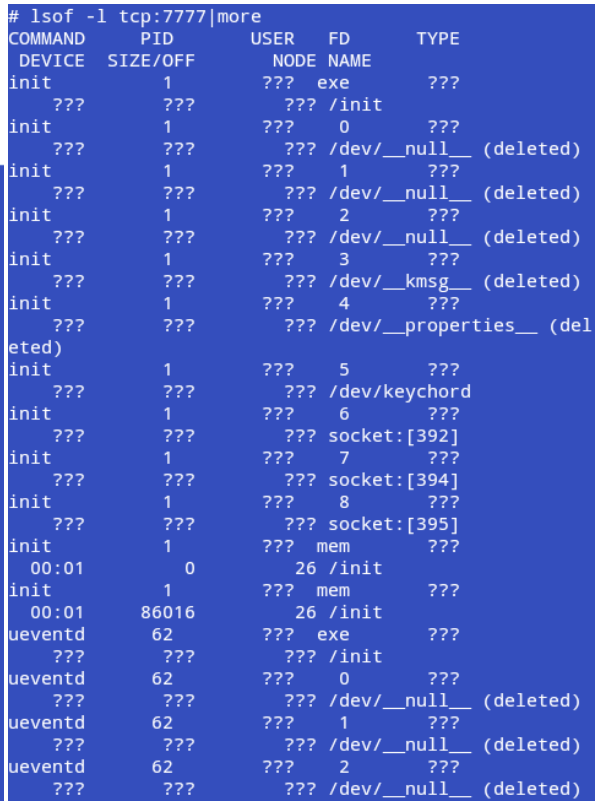
Port(s)	Protocol	Service	Details	Source
7777	tcp	trojans	Backdoor.Darkmoon (08.19.2005) - trojan that opens a backdoor on the compromised computer and has keylogging capabilities. Opens a backdoor and listens for remote commands on ports 6868/tcp and 7777/tcp. Port 777/tcp is also used by: iChat server file transfer proxy Oracle Cluster File System 2 GodMessage, The Thing trojans Windows backdoor program tini.exe Ultima Online Active Worlds (TCP/UDP)	*SG*
7777	udp	applications	Unreal Tournament 2004 Game port Terraria also uses port 7777 (TCP/UDP)	*SG*
7777	tcp		iChat server file transfer proxy (unofficial)	*Wikipedia*
7777	tcp		Default used by Windows backdoor program tini.exe (unofficial)	*Wikipedia*
7777	tcp	trojan	[trojan] God Message	*Trojans*
7777	tcp,udp	cbt	cbt	*IANA*
7777	tcp	FWTK-authsvr	FWTK-Gauntlet authentication server	*SANS*
7777	tcp	GodMessage	[trojan] God Message	*SANS*
7777	tcp	oracle-portal	Oracle 9i Portal - Apache HTTP (default)	*SANS*
7777	tcp	TheThing(modified)	[trojan] The Thing (modified)	*SANS*
7777	tcp	Tini	[trojan] Tini	*SANS*
3000,5670,7777,7000-7100	tcp	applications	Active Worlds	*Portforward*
6777,7777-7797,30660	tcp,udp	applications	BlackSite - Area 51	*Portforward*

[그림 : http://www.speedguide.net/port.php?port=7777]

iChat가 있지만 아이폰이 아니라 안드로이드이니 그건 아닐 듯 하고, 대부분이 **Trojan** 및 **backdoor**에서 주로 사용되는 포트라고 설명해준다. 역시 수상하지 않을 수 없으니 바로 분석에 들어가 보았다.



```
$ export PATH=/data/local/bin:$PATH
$ su
# netstat -na
Proto Recv-Q Send-Q Local Address          Foreign Ad
dress        State
 tcp       0      0 127.0.0.1:7777         0.0.0.0:*
             LISTEN
 tcp       0      0 127.0.0.1:7203         0.0.0.0:*
             LISTEN
 tcp       0      0 127.0.0.1:40640        127.0.0.1:
7777         ESTABLISHED
 tcp       0      0 127.0.0.1:7777         127.0.0.1:
40640        ESTABLISHED
 tcp       0      0 127.0.0.1:40641        127.0.0.1:
7777         ESTABLISHED
 tcp       0      0 127.0.0.1:7777         127.0.0.1:
40641        ESTABLISHED
tcp6       0      1 ::ffff:192.168.0.4:41710 ::ffff:7
4.125.71.104:80 CLOSE_WAIT
tcp6       0      0 ::ffff:192.168.0.4:38927 ::ffff:7
4.125.71.188:5228 ESTABLISHED
tcp6       0      0 ::ffff:192.168.0.4:53070 ::ffff:1
10.76.141.157:80 ESTABLISHED
tcp6       0      1 ::ffff:192.168.0.4:53092 ::ffff:7
4.125.71.156:80 CLOSE_WAIT
tcp6       0      0 ::ffff:192.168.0.4:56350 ::ffff:2
03.238.180.119:10007 ESTABLISHED
#
```



```
# lsof -l tcp:7777|more
COMMAND     PID        USER   FD      TYPE
 DEVICE  SIZE/OFF       NODE NAME
init          1         ???  exe       ???
    ???       ???         ??? /init
init          1         ???    0       ???
    ???       ???         ??? /dev/__null__ (deleted)
init          1         ???    1       ???
    ???       ???         ??? /dev/__null__ (deleted)
init          1         ???    2       ???
    ???       ???         ??? /dev/__null__ (deleted)
init          1         ???    3       ???
    ???       ???         ??? /dev/__kmsg__ (deleted)
init          1         ???    4       ???
    ???       ???         ??? /dev/__properties__ (del
eted)
init          1         ???    5       ???
    ???       ???         ??? /dev/keychord
init          1         ???    6       ???
    ???       ???         ??? socket:[392]
init          1         ???    7       ???
    ???       ???         ??? socket:[394]
init          1         ???    8       ???
    ???       ???         ??? socket:[395]
init          1         ???  mem       ???
  00:01         0          26 /init
init          1         ???  mem       ???
  00:01     86016          26 /init
ueventd      62         ???  exe       ???
    ???       ???         ??? /init
ueventd      62         ???    0       ???
    ???       ???         ??? /dev/__null__ (deleted)
ueventd      62         ???    1       ???
    ???       ???         ??? /dev/__null__ (deleted)
ueventd      62         ???    2       ???
    ???       ???         ??? /dev/__null__ (deleted)
```

왼쪽 그림에서는 **Terminal Emulator** 어플을 이용해 **su**로 루트 권한을 얻고, **netstat**으로 연결된 네트워크 및 포트를 확인했다. (**busybox**를 설치해 리눅스 명령어들을 사용할 수 있게 해야 할 것이다.)

오른쪽 그림에서는 **lsof -i tcp:7777**을 이용해 포트 7777번이 열고 있는 파일을 확인하려 했으나, 필자가 설치한 busybox free에서는 losf의 옵션은 적용이 안 되는 듯 하다.

여기서 **netstat** 및 **lsof**에 대해 정리하자면,

netstat은 자신의 네트워크에서 연결되었거나 연결될 목록을 프로토콜과 함께 보여주는 명령어이다.

이 명령을 이용해 열린 포트들이 어떻게 사용되는지 확인할 수 있으며 이 명령어만 잘 활용해도 웬만한 해킹툴은 감지해 낼 수도 있다. 위의 그림처럼 굳이 **nmap**을 이용하지 않아도 7777번 포트가 열려서 활성화 된 것을 알 수 있었다.

다음은 리눅스에서 주로 쓰이는 netstat의 옵션이다.

netstat -a, -all	//모든 소켓을 다 보여준다.
netstat -r	//라우팅 테이블을 보여준다.
netstat -p	//소켓에 대응되는 PID와 프로그램 이름을 보여준다.

나머지는 **netstat -h**로 netstat의 옵션을 알 수 있다.

여기서 참고로 윈도우와 리눅스의 netstat 옵션은 다르니 주의하시길 바란다.

lsof는 프로세스가 열고 있는 파일을 확인하는 명령어이다. linux 시스템에서는 프로세스를 통하여 파일을 열 수 있는데, 여기서 파일이란 데이터를 담을 수 있는 파일 뿐 아니라, 통신을 위한 **socket, event poll 등 file descriptor, 라이브러리 파일, char device**까지 포함 된다.

lsof -i	//인터넷 소켓에 대한 접속 확인
lsof -i tcp:[포트]	//특정 포트로 접속한 리스트 확인
lsof [파일명]	//특정 파일에 접근하고 있는 프로세스 확인
lsof -u $UID	//특정 유저가 사용하는 프로세스 확인
lsof -p $PID	//지정한 프로세스가 열고 있는 파일 리스트 확인
lsof -c $CMD	//지정한 실행 프로그램이 접근하고 있는 파일 리스트 확인

lsof에 대한 자세한 내용은 losf 사용법에 대해 자세한 내용이 포스트 되어 있는 블로그 (http://blog.pages.kr/292)를 참고하시기 바란다.

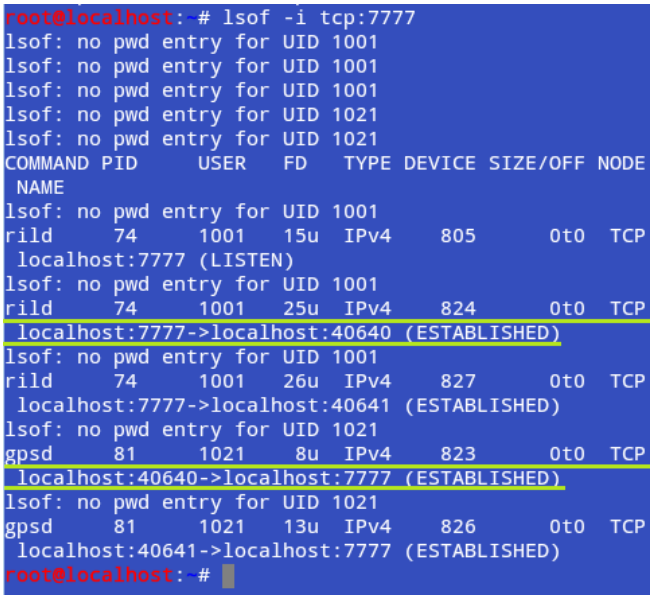
다시 돌아와서 필자의 busybox free 버전으로는 명령어나 옵션들이 제대로 안 먹히니 설치된 **backtrack5**를 이용하여 분석해보자 하였다.

```
# su
# cd sdcard/bt5
# sh bt
mkdir failed for /data/local/mnt, File exists
net.ipv4.ip_forward = 1
Ubuntu is configured with SSH and VNC servers that ca
n be accessed from the IP:
eth0: ip 192.168.0.4 mask 255.255.255.0 flags [up bro
adcast running multicast]

 * Starting OpenBSD Secure Shell server sshd  [ OK ]
Do you want to start a VNC server? [y/n] n
VNC not started; if you change your mind invoke 'star
tvnc' from prompt.

root@localhost:~# ps
  PID TTY          TIME CMD
  920 pts/0    00:00:00 sh
 1223 pts/0    00:00:00 sh
 1225 pts/0    00:00:00 sh
 1242 pts/0    00:00:00 bash
 1273 pts/0    00:00:00 ps
root@localhost:~#
```

사실 우분투나 다른 리눅스를 이용해도 되고, 좋은 **busybox**를 이용해 필요한 명령어와 옵션들을 다 사용할 수 있다면 굳이 백트랙을 이용할 필요는 없다.
(backtrack5를 안드로이드에서 부팅하기 원한다면,
http://forum.xda-developers.com/showthread.php?t=1079898 나
http://www.youtube.com/watch?v=QnqQpDnoCg4 를 참고하시길 바란다.)



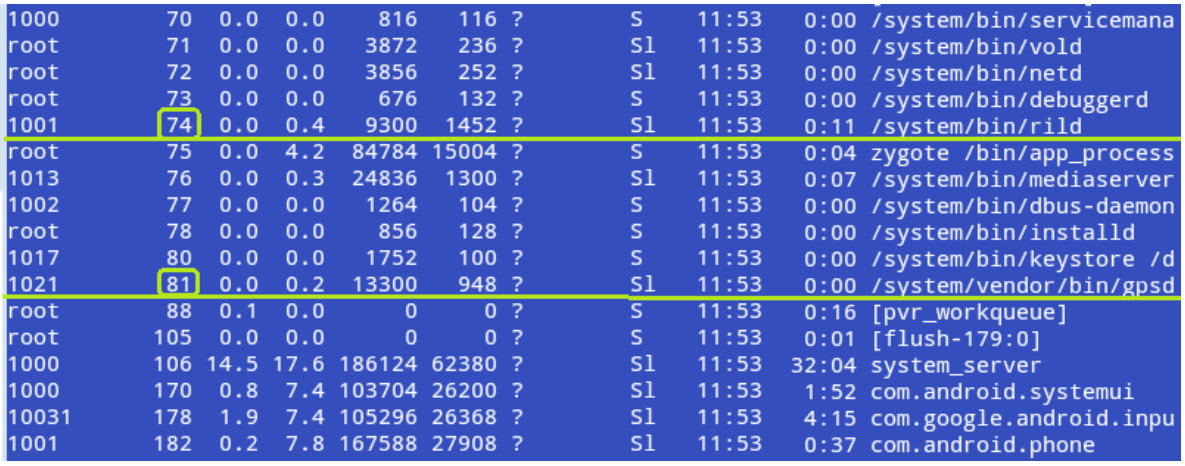
바로 **lsof -i tcp:7777** 및 **netstat -p**로 확인해 보니 port 7777에서 쓰이는 PID는 **rild**와 **gpsd**에서 각각 **74**와 **81**임을 알아냈다. 그런데 다시 보니 localhost:7777->localhost:40640 으로 localhost 단에서만 움직이는 로컬 프로세스였다.
그렇다면 백도어는 아닌 것 같은데,
왜 nmap에서 unknown으로 처리하고, 다른 사람의 안드로이드 폰에서는 왜 7777포트가 열려있지 않은가 계속 의심스러워 그래도 분석하기로 하였다.

여기서 먼저 **ps** 명령어에 대해 정리하자면,
ps는 현재 실행되는 프로세스의 상태를 보여주는 명령어이다.
ps만 쳤을 때는 **PID**(프로세스 아이디), **TTY**(프로세스와 연결된 터미널 포트), **TIME**(프로세스에서 사용한 CPU 시간), **CMD**(명령어)의 값들이 나온다.
다음은 옵션이다.

ps -a	//모든 사용자의 프로세스를 출력한다.
ps -u	//자세한 정보를 출력한다.
ps -x	//제어터미널이 없는 프로세스까지 모두 출력한다.
ps -j	//작업 중심의 형태로 출력한다.
ps -l	//자세한 형태의 정보를 출력한다.

더 다양한 옵션은 **ps -?**를 이용해 매뉴얼을 확인해보면 알 수 있다.

그래서 **ps -aux** 명령어로 현재 실행되고 있는 모든 프로세스와 그에 대해 자세한 정보를 출력해 보았다.



```
1000        70  0.0  0.0      816      116 ?        S    11:53   0:00 /system/bin/servicemana
root        71  0.0  0.0     3872      236 ?        Sl   11:53   0:00 /system/bin/vold
root        72  0.0  0.0     3856      252 ?        Sl   11:53   0:00 /system/bin/netd
root        73  0.0  0.0      676      132 ?        S    11:53   0:00 /system/bin/debuggerd
1001        74  0.0  0.4     9300     1452 ?        Sl   11:53   0:11 /system/bin/rild
root        75  0.0  4.2    84784    15004 ?        S    11:53   0:04 zygote /bin/app_process
1013        76  0.0  0.3    24836     1300 ?        Sl   11:53   0:07 /system/bin/mediaserver
1002        77  0.0  0.0     1264      104 ?        S    11:53   0:00 /system/bin/dbus-daemon
root        78  0.0  0.0      856      128 ?        S    11:53   0:00 /system/bin/installd
1017        80  0.0  0.0     1752      100 ?        S    11:53   0:00 /system/bin/keystore /d
1021        81  0.0  0.2    13300      948 ?        Sl   11:53   0:00 /system/vendor/bin/gpsd
root        88  0.1  0.0        0        0 ?        S    11:53   0:16 [pvr_workqueue]
root       105  0.0  0.0        0        0 ?        S    11:53   0:01 [flush-179:0]
1000       106 14.5 17.6   186124    62380 ?        Sl   11:53  32:04 system_server
1000       170  0.8  7.4   103704    26200 ?        Sl   11:53   1:52 com.android.systemui
10031      178  1.9  7.4   105296    26368 ?        Sl   11:53   4:15 com.google.android.inpu
1001       182  0.2  7.8   167588    27908 ?        Sl   11:53   0:37 com.android.phone
```

흠.. 이 부분에서 살짝 당황하긴 했다.
역시 rild와 gpsd로서 괜히 로컬 시스템을 가지고 호들갑 떠는 것일지도 모르겠다.
그래도 혹시 모를 수 있고, 만약 백도어나 해킹툴 등이 실제로 수상한 포트를 열어놨을 때 그에 대한 대응을 다루고자 했던 것이 이 포스트의 목적이니 이것을 계속 악성 프로그램이라 가정하고 계속 진행을 하였다;;;

일단 port 7777에서 들어오고 나가는 패킷이 있나 **tcpdump** 명령어를 이용해 검사해 보았다.
여기서 **tcpdump**는 조건식을 만족하는 데이터 패킷들의 헤더를 캡쳐하고 모니터링할 수 있는 프로그램을 말한다. 보통 백도어나 해킹툴이라면 나의 정보가 다른 곳으로 패킷을 이용해 오고 갈 것이기에 tcpdump를 이용해 수상한 패킷들을 점검하곤 한다.
tcpdump는 별도의 옵션을 주지 않으면 eth0의 트래픽을 전부 모니터링 한다. 다음은 주요 옵션이다.

tcpdump -i [인터페이스]	//특정 이더넷 인터페이스로 송수신되는 패킷 헤더를 캡쳐한다.
tcpdump port [포트]	//특정 포트에 대한 트래픽을 모두 캡쳐한다.
tcpdump host [IP]	//특정 IP에 대한 트래픽을 모두 캡쳐한다.
tcpdump -v	//좀 더 자세하게 캡쳐한다.
tcpdump -w [파일명]	//특정 파일에 캡쳐된 패킷을 저장한다.
tcpdump -r [파일명]	//저장된 파일 내용을 확인한다.

http://openmaniak.com/tcpdump.php에서 좀 더 쉽고 예제까지 들어서 tcpdump에 대해 설명해 준다.
참고로 안드로이드 어플 중에 **shark**라는 어플이 있어 그 어플로도 패킷 모니터링을 쉽게 할 수 있다.

그렇게 **tcpdump -v port 7777 -w test1.txt**를 해보았다.

```
root@localhost:~/Desktop# tcpdump -v port 7777 -w tes
t2.txt
tcpdump: WARNING: can't create rx ring on packet sock
et 7: 12-Cannot allocate memory
tcpdump: listening on eth0, link-type EN10MB (Etherne
t), capture size 65535 bytes
^C0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@localhost:~/Desktop#
```

그러나 인터넷을 하고, 지도 어플로 gps를 이용하고 체크인을 하고, 전화와 문자를 해보아도 아무 패킷도 잡히지 않았다.
묘하게 계속 수상하기에 이 부분에서 상당히 많은 시간을 쏟아 부었다.

신나게 삽질을 했으나
아무것도 건지지는 못 하였다.

```
root@localhost:~# iptables -A INPUT -p tcp --dport 77
77 -j DROP
root@localhost:~# netstat -p
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign A
ddress         State       PID/Program name
tcp        0  29154 192.168.0.4:5901        192.168.0
.2:54408       ESTABLISHED 1107/androidvncserv
tcp6       0      0 192.168.0.4%1:48395     110.76.14
0.153:www      ESTABLISHED 370/com.kakao.talk
tcp6       0      0 192.168.0.4%1:47779     203.238.1
80.119:10007   ESTABLISHED 370/com.kakao.talk
tcp6       1      0 192.168.0.4%1:38846     206.155.8
3.115:www      CLOSE_WAIT  1562/com.metago.ast
tcp6       0      0 192.168.0.4%1:55480     hx-in-f18
8.1e100.n:5228 ESTABLISHED 219/com.google.proc
tcp6       1      0 192.168.0.4%1:50258     hx-in-f15
6.1e100.ne:www CLOSE_WAIT  1562/com.metago.ast
```

```
root@localhost:~# nmap 127.0.0.1

Starting Nmap 5.00 ( http://nmap.org ) at 2012-03-02
07:08 UTC
Interesting ports on localhost (127.0.0.1):
Not shown: 996 closed ports
PORT     STATE    SERVICE
22/tcp   open     ssh
5801/tcp open     vnc-http-1
5901/tcp open     vnc-1
7777/tcp filtered unknown

Nmap done: 1 IP address (1 host up) scanned in 2.17 s
econds
root@localhost:~#
```

그러다 결국 **iptables** 명령어를 이용하여 필터링을 시켜버렸다.
포트가 열려서 악용될 서비스로 남겨두는 것 보다는 그냥 막아버리는게 상책이다.
오른쪽은 7777번 포트를 필터링시켜서 막아버린 것을 nmap에서도 표시해주는 화면이다.

여기서 패킷필터링 툴인 **iptables**에 대해서 정리해보자.
패킷필터링은 지나가는 패킷의 헤더를 보고 그 전체 패킷의 운명을 결정하는 것을 말한다.
리눅스에서 방화벽 역할을 한다고 보면 되겠다.

iptables 에는 기본 INPUT chain, FORWARD chain, OUTPUT chain 이 있다.

------>INPUT------> Linux Box ------>OUTPUT--------->
--------- ↕ ---------- ↕
--------- └--------- FORWARD ----------┘
이것은 체인들의 모식도로 Linux box를 도착지로 삼는 모든 패킷은 INPUT Chain을 통과하게 되며,

Linux box에서 생성되어 외부로 보내지는 모든 패킷은 OUTPUT Chain을 통과하게 된다.
Forward chain은 도착지가 우리의 Linux box가 아닌 패킷이 통과하게 되는 체인을 말한다.

iptables -A INPUT -j DROP 의 형식으로 명령어를 입력하는데 이처럼 그대로 명령어를 친다면, 자신의 Linux box로 오는 모든 패킷을 거부한다. 즉 모든 통신이 끊어진다.
이 옵션에서 -A는 룰을 추가한다는 의미이고 INPUT은 패킷이 들어오는 체인을 뜻한다.
그리고 -j는 패킷의 운명을 결정 짓는데 그 운명이 DROP, 패킷을 버리라는 의미이다.
즉, INPUT 체인으로 들어오는 패킷을 모두 버리라는 룰을 추가하는 명령이다.

프로토콜제어 옵션은 -p이고, 그 인자는 tcp, udp, icmp 등이 될 수 있다.
앞에서 사용한 명령어는 포트 7777번으로 오고 가는 모든 패킷을 버리라고 한 것인데,
iptables -A INPUT -p tcp --dport 7777 -j DROP
iptables -A INPUT -p tcp --sport 7777 -j DROP
으로 적용시킬 수 있다.
(--dport는 도착지 포트를 의미하고, --sport는 출발지 포트를 의미한다.)

http://theeye.pe.kr/25이나 http://wiki.kldp.org/wiki.php/iptables를 참고하면 iptables에 대해 더욱 상세하고 풍부한 내용을 얻을 수 있다.

```
root@localhost:~# kill -9 74
root@localhost:~# netstat -p
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign A
ddress         State       PID/Program name
tcp        0      0 localhost:7777          localhost
:60008         TIME_WAIT   -
tcp        0      0 localhost:40640         localhost
:7777          TIME_WAIT   -
tcp        0      0 localhost:40641         localhost
:7777          TIME_WAIT   -
tcp        0      0 localhost:7777          localhost
:60007         TIME_WAIT   -
tcp6       1      0 192.168.0.4%1:46242     hx-in-f10
4.1e100.:https CLOSE_WAIT  1411/com.google.and
tcp6       0      0 192.168.0.4%1:48395     110.76.14
0.153:www      ESTABLISHED 370/com.kakao.talk
```

그러고 나서 확인사살로 **kill** 명령어로 **81**번 프로세스와 **74**번 프로세스를 **-9** 옵션으로 강제적으로 죽여버렸다.
만약 백도어나 해킹툴의 pid를 알아냈다면 **kill -9** 명령어를 이용해 그 프로세스를 강제 종료한다면 눈에 띄는 멀웨어들은 그렇게 임시대처할 수 있을 것이다.

그렇게 포트 7777에 관련된 74번, 81번 프로세스를 죽이고 나니..

전화 통화가 전혀 되지 않았다;;;;

아아
그는 '좋은' 로컬이였던 것일까..

그래도 만약 이것이 실제로 멀웨어로서 백도어와 트로이목마 역할을 한다면 이런 방법으로 추적을 하고 차단하여 임시방편으로 막아볼 수 있다고 말할 수 있겠다. 사실 이 방법은 리눅스에서 사용하던 방법을 그대로 안드로이드에 적용시킨 것이다.
그리고 더 나아가서 ps -aux에 보여진 위치를 계속 추적하고 삭제한다면 눈에 띄는 멀웨어들은 그렇게 방지할 수 있지 않을까 생각한다.

하필이면 그 포트만 unknown으로 처리되었고(이것만 서비스로 스캔 되는 것 자체가 이상했다.), 다른 안드로이드 폰에서는 전혀 그런 포트가 열려있지 않은데다 아무리 찾아봐도 android port 7777 관련 내용은 쉽게 나오질 않았었다.
아무래도 의심은 거둘 수가 없어 다음 번에는 공장 초기화를 해보고 비교분석을 해볼까 한다.

ps.
rild은 Vender RIL을 초기화하며, 안드로이드 telephony 서비스들로부터의 모든 통신을 프로세스하며, solicited 명령어를 통해 Vender RIL로 콜을 넘긴다고 한다. 즉, 통화에 관련된 많은 부분을 차지하는 듯 하다. (관련 블로그 설명: http://skyswim42.egloos.com/3380117)
gpsd는 GPS를 net에서 쓸 수 있는 프로세스이다. 그러나 이 부분은 kill을 한 뒤에도 gps가 인터넷에서 잘 돌아갔다는 점에서는 의아하다.