CustomScripts

Edit New Page

robo-mo edited this page on Feb 5, 2017 · 51 revisions

Advanced AFWall+ users may wish to define a custom script to be executed by AFWall+ this section explains the most common questions and shows some examples.

Index

Pages 28

Find a Page...

Home
Advertisements blocking
Android kernel traffic
Apps leak private user data during boot
BusyBox
Contributors
CustomScripts
DNS
Error codes
FAQ
Google Play services and other special cases
HOWTO blocking WhatsApp
HOWTO Compile AFWall
HOWTO Compiling busybox
HOWTO Compiling iptables
Show 13 more pages...

Clone this wiki locally

https://github.com/ukanth/a

Clone in Desktop

Introduction

WARNING: This functionality should be used only by **experienced users that know what they are doing!** These examples may block your internet connection if not executed with proper care. So be careful when applying these settings, especially on remote device 'servers' with an ssh session! If you have any trouble with it an want to report an error first make sure you deleted/disabled all custom script first - we won't accept issue reports which are custom script related!

Once a custom script is defined, it will be automatically executed every time that the AFWall+ rules are applied (inclusive on startup/shutdown if the firewall is enabled).

To define a custom script, just choose `Set custom script` from the menu (right corner) on the main menu.

Important notes about IPv4 and IPv6 differences

```
IPv4 vs IPv6
IPv4 = 2^32
  4,290,000,000
  |   |   |   Hundreds
  |   |   Thousands
  |   Millions
  Billions

IPv6 = 2^128
340,000,000,000,000,000,000,000,000,000,000,000,000
  |   |   |   |   |   |   |   |   |   |   |   |   Hundreds
  |   |   |   |   |   |   |   |   |   |   |   Thousands
  |   |   |   |   |   |   |   |   |   |   Millions
  |   |   |   |   |   |   |   |   |   Billions
  |   |   |   |   |   |   |   |   Trillions
  |   |   |   |   |   |   |   Quadrillions
  |   |   |   |   |   |   Quintillions
  |   |   |   |   |   Sextillion
  |   |   |   |   Septillions
  |   |   |   Octillion
  |   |   Nonillion
  |   Decillion
  Undecillion
```

If you do not need IPv6 or your provider does not support it (native) you can just disable it via `net.ipv6.conf.all.disable_ipv6=1` in your build.prop or *better* in your sysctl.conf file which will be applied right after the boot.

IPTables only filters IPv4 traffic (RFC 1918). Rules setup in iptables will not touch ipv6 traffic so we need our ip6tables. **IPv6 does not include private network features such as NAT**. Because of the very large number of IPv6 addresses. However, `FC00::/7 (and FC20::/7)` prefix used to identify Local IPv6 unicast addresses. All IPv6 users should be able to obtain IPv6 address space for use at their discretion and without artificial barriers between their network and the Internet.

IPv6 uses ICMP a lot more than IPv4, and not letting ICMP packets ingoing can severely cripple your traffic because you won't receive error messages related to that traffic. This can cause long delays + timeouts. Allowing ICMPv6 traffic in usually doesn't hurt, so we can can add this to our firewall rules `ip6tables -A INPUT -p icmpv6 -j ACCEPT`. There are some guys out there which want really block ping6 (for unknown reasons), so here we are `ip6tables -A INPUT -p icmpv6 --icmpv6-type 128 -j DROP`.

The available ICMPv6 error codes are listed in RFC 4443, which specifies connection attempts blocked by a firewall rule.

An example IPv6 only script can be found here.

Important notes about 3G and WiFi

Some of the examples maybe not working on 2G/3G/4G because DHCP/DHCPv6 reasons. Normally the IP addresses are negotiated over PPP there. So it could be handled a bit different in your ROM but you can just use Wireshark/Burp and make a 3G connection to see what's going on, feel free to send us an email if something is incorrect.

DROP vs REJECT

DROP isn't more *secure* compared to REJECT, this is a wrong myth on the internet, see here for more details:

- RFC 1122 3.3.8
- DROP vs REJECT explanation

Loading scripts from files

- Please first go into `Settings` -> `Developer Options` and enable `USB Debugging` and change Root Access to `Apps and ADB`. (Necessary if you push/test your files via adb).

- Whenever you finish using adb, always remember to disable USB Debugging and restore Root Access to Apps only. While Android 4.2+ ROMs now prompt you to authorize an RSA key fingerprint before allowing a debugging connection (thus mitigating adb exploit tools that bypass screen lock and can install root apps), you still risk additional vulnerability surface by leaving debugging enabled.

Big scripts can be quite hard to edit in the `Set custom script` `screen, so it may be a good idea to put your script in a file, then load it from there.`

To do that, just use the "." (dot) shell command in the `Set custom script` dialog to load your script from an external file. e.g.:

```
. /path/to/script.sh  (e.g. . /data/local/myawesomescript.sh)
```

This will cause your script file to be loaded and executed every time the rules are applied. Make sure that this folder have the right permissions *0755* (Group/User *0*), if not AFWall+ can't read any script and you will get an error such "*Can't apply rules/script*". It's also very important if you are under Windows/Mac OS to *save yourscript.sh with Unix file encoding*! If you use Notepad++, go to *Edit - EOL Conversion* and hit *Convert to Unix format*. You can even have multiple scripts executed in sequence...

```
. /path/to/load-modules.sh (e.g. /data/local/script.sh)

# This is a comment or outlined rule/script
#. /path/to/myrules.sh

#. /path/to/myscript.sh
```

However, please note that this can create a serious security breach on your device, since the script will be always executed as root! You must place your script where other applications will not be able to modify it (the sdcard is NOT a good place!).

Warning If you mistype any of those files, things may break. Because the userinit.sh script blocks all network at boot, if you make a typo in the userinit.sh script (CM-User related), you will be unable to use the Internet at all!

Adding custom rules

If you want to add custom iptables rules, just use the **$IPTABLES** shell variable to call iptables.

The following iptables chains can be used to add custom rules:

> **afwall** - This is the main AFWall+ chain. All OUTPUT packets will pass through it. It is therefore the >perfect place if you want to add rules that apply to any interface.
>
> **afwall-3g** - This chain will only receive OUTPUT packets for the cellular network interface (no matter >if it is 2G, 3G, 4G, etc).
>
> **afwall-wifi** - This chain will only receive OUTPUT packets for the WiFi interface.
>
> **afwall-reject** - This chain should be used as a **target** when you want to reject and log a >packet. >When the logging is disabled, this is exactly the same as the built-in **REJECT** target

Please note that all those chains are guaranteed to be cleared before the custom script is executed, so you don't need to worry about rules cleanup on your script IF you are using those chains.

If you use any chain not listed above, then you need to manually purge it BEFORE adding your custom rules (otherwise the rules will be duplicated every time they are applied). *On this case, you will also need to manually purge you rules when the firewall is disabled, by defining a custom shutdown script.*

IMPORTANT: Never manually purge the OUTPUT chain - this will cause AFWall+ rules to be ignored. Use the 'afwall' chain instead.

Some examples

```
# Necessary at the beginning of each script!
OEM_SCRIPT_PATH=/system/bin/oem-iptables-init.sh (optional)
IP6TABLES=/system/bin/ip6tables
IPTABLES=/system/bin/iptables

# Now add your own rules...
```

```
# To change the names of the iptables to whatever you want
myownipv6tablesname=/system/bin/ip6tables
myownipv4tablesname=/system/bin/iptables

Remember that you now must use the selected names myownipv6tablesname for IP6TABLES and myow
```

```
# Interface configuration - replace with your interfaces (ifconfig -a)
# This is for advance users only which like o work with specific interfaces only!
WAN_IF="eth1"
LAN_IF="eth0"
DMZ_IF="eth2"
LAN_NET="2001:db8:1::/64"
DMZ_NET="2001:db8:2::/64"
```

```
# Flush/Purge all rules except OUTPUT
$IPTABLES -F INPUT
$IPTABLES -F FORWARD
$IPTABLES -t nat -F
$IPTABLES -t mangle -F
$IP6TABLES -F INPUT
$IP6TABLES -F FORWARD
$IP6TABLES -t nat -F
$IP6TABLES -t mangle -F
```

```
# Flush/Purge/resetting all rules
$IPTABLES -F
$IPTABLES -X
$IPTABLES -t nat -F
$IPTABLES -t nat -X
$IPTABLES -t mangle -F
$IPTABLES -t mangle -X
$IPTABLES -t raw -F
$IPTABLES -t raw -X
$IPTABLES -t security -F
$IPTABLES -t security -X
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
$IPTABLES -P OUTPUT ACCEPT

$IP6TABLES .....
.. your stuff belongs in here
```

```
# Allow loopback communication (necessary on IPv6)
$IP6TABLES -A INPUT -i lo -j ACCEPT
$IP6TABLES -A OUTPUT -o lo -j ACCEPT
```

```
# Stateful-Inspection
$IP6TABLES -A INPUT -m state –state ESTABLISHED,RELATED -j ACCEPT
$IP6TABLES -A FORWARD -m state –state ESTABLISHED,RELATED -j ACCEPT
$IP6TABLES -A OUTPUT -m state –state ESTABLISHED,RELATED -j ACCEPT
```

```
# Block Shellshock
$IPTABLES -A INPUT -m string --algo bm --hex-string '|28 29 20 7B|' -j DROP
$IP6TABLES -A INPUT -m string --algo bm --hex-string '|28 29 20 7B|' -j DROP
```

```
# Anti-Spoofing on loopback ::1 & the unique local address fe80::/1
$IP6TABLES -A INPUT ! -i lo -s ::1/128 -j DROP
$IP6TABLES -A INPUT -i $WAN_IF -s fe80::/1 -j DROP
$IP6TABLES -A FORWARD -s ::1/128 -j DROP
$IP6TABLES -A FORWARD -i $WAN_IF -s fe80::/1 -j DROP
```

```
# Allow SSH for eth0 only
$IPTABLES -A TCP -i eth0 -p tcp --dport ssh -j ACCEPT
```

```
# Prevent SYN attacks
$IPTABLES -I TCP -p tcp --match recent --update --seconds 60 --name TCP-PORTSCAN -j DROP
$IP6TABLES -I TCP -p tcp --match recent --update --seconds 60 --name TCP-PORTSCAN -j DROP
$IPTABLES -A INPUT -p tcp --match recent --set --name TCP-PORTSCAN -j DROP
$IP6TABLES -A INPUT -p tcp --match recent --set --name TCP-PORTSCAN -j DROP
```

```
#Prevent SMURF attacks
$IPTABLES -A INPUT -p icmp -m icmp --icmp-type address-mask-request -j DROP
$IPTABLES -A INPUT -p icmp -m icmp --icmp-type timestamp-request -j DROP
$IPTABLES -A INPUT -p icmp -m limit --limit 2/second --limit-burst 2 -j ACCEPT
$IP6TABLES -A INPUT -p icmpv6 -m limit --limit 2/second --limit-burst 2 -j ACCEPT
```

```
# Set iptables to masquerade
$IPTABLES -t nat -A POSTROUTING -j MASQUERADE
```

```
# Tunnel traffic (for native IPv6 connections only! -> 6to4: 2002::/16 & Teredo: 2001:0::/32
$IP6TABLES -A INPUT -s 2002::/16 -j DROP
$IP6TABLES -A INPUT -s 2001:0::/32 -j DROP
$IP6TABLES -A FORWARD -s 2002::/16 -j DROP
$IP6TABLES -A FORWARD -s 2001:0::/32 -j DROP
```

```
# Block all IPv6 in IPv4 communication (for native IPv6 connections only!)
# This must be done in our IPv4 tables!
$IPTABLES -A INPUT -p 41 -j DROP
$IPTABLES -A FORWARD -p 41 -j DROP
```

```
# Allow SSH/HTTPS/SNMP (Ports 22tcp/443tcp/161udp)
$IP6TABLES -A INPUT -i $LAN_IF -s $LAN_NET -p tcp -m multiport –dport 22,80,443 -j ACCEPT
```

```
# Syntax to block an IP address
$IPTABLES -A INPUT -s IP-ADDRESS -j DROP
# Example in AFWall+
$IPTABLES -A "afwall" -d 22.22.22.0/21 -j REJECT
```

```
# Force a specific NTP in this case  DE ntp0.fau.de (131.188.3.220), Location: University Er
$IPTABLES -t nat -A OUTPUT -p tcp --dport 123 -j DNAT --to-destination 131.188.3.222:123
$IPTABLES -t nat -A OUTPUT -p udp --dport 123 -j DNAT --to-destination 131.188.3.222:123
```

```
# If you just want to block access to one port from an ip 65.55.44.100 to port 25 then type
$IPTABLES -A INPUT -s 65.55.44.100 -p tcp --destination-port 25 -j DROP
```

```
# Always allow connections to 192.168.0.1, no matter the interface
$IPTABLES -A "afwall" --destination "192.168.0.1" -j RETURN
```

```
# Allow all connections to the local network (192.168.0.XXX) on Wi-fi
$IPTABLES -A "afwall-wifi" --destination "192.168.0.0/24" -j RETURN
```

```
# Block all connections in the TCP port 80 (http)
$IPTABLES -A "afwall" -p TCP --dport 80 -j "afwall-reject"
```

```
# Block all connections in the TCP port 22 (ssh)
$IPTABLES -A "afwall" -p TCP --dport 22 -j "afwall-reject"
```

```
# Block HTTP connections, but only on cellular interface
$IPTABLES -A "afwall-3g" -p TCP --destination-port 80 -j "afwall-reject"
```

```
# Allow all apps access to standard Orbot ports
$IPTABLES -A "afwall" -d 127.0.0.1 -p tcp --dport 9040 -j ACCEPT
$IPTABLES -A "afwall" -d 127.0.0.1 -p udp --dport 5400 -j ACCEPT
```

```
# Restore policies
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
```

```
# Drop all but outbound
$IPTABLES -P INPUT DROP
$IPTABLES -P FORWARD DROP
```

If you want AFWall+to report failures on your rules, you must manually "exit" from the script on error. E.g.:

```
# Try to apply my custom rule, but report any failure (and abort)
$IPTABLES -A "afwall" --destination "192.168.0.1" -j RETURN || exit
```

```
# Try to apply another custom rule, but ignore any errors on it
$IPTABLES -A "afwall" -p TCP --destination-port 80 -j "afwall-reject"
```

```
# Connect Wifi-Tethered Clients to VPN
$IPTABLES -t filter -F FORWARD
$IPTABLES -t nat -F POSTROUTING
$IPTABLES -t filter -A FORWARD -j ACCEPT
$IPTABLES -t nat -A POSTROUTING -j MASQUERADE
```

```
# Force all traffic over VPN (on Wifi only) and drop them if there is no VPN connection anym
$IPTABLES -A INPUT  -i tun0 -j ACCEPT
$IPTABLES -A INPUT  -s 10.10.10.10 -j ACCEPT
$IPTABLES -A OUTPUT -i tun0 -j ACCEPT
$IPTABLES -A OUTPUT -d 10.10.10.10 -j ACCEPT
# For DNS whitelistening (if not already done)
$IPTABLES -A OUTPUT -p udp --dport 53 -j ACCEPT
$IPTABLES -A INPUT  -p udp --sport 53 -j ACCEPT
```

```
# Possible Tethering + OpenVPN issues on KitKat 4.x devices
# iptables --flush
push "dhcp-option DNS 208.67.222.222"
push "dhcp-option DNS 208.67.220.220"
# push "redirect-gateway autolocal def1" (server config)
$IPTABLES -t filter -F FORWARD
$IPTABLES -t nat -F POSTROUTING
$IPTABLES -t filter -I FORWARD -j ACCEPT
$IPTABLES -t nat -I POSTROUTING -j MASQUERADE

# DHCP with OpenVPN set to tun0 and Lan is set to 192.168.43.whatever
ip rule add from 192.168.43.0/24 lookup 61
ip route add default dev tun0 scope link table 61
ip route add 192.168.43.0/24 dev wlan0 scope link table 61
ip route add broadcast 255.255.255.255 dev wlan0 scope link table 61
```

```
# Force a specific DNS for mobile networks (in this case http://www.opendns.com/)
# First two lines delete current DNS settings
$IPTABLES -t nat -D OUTPUT -p tcp --dport 53 -j DNAT --to-destination 208.67.222.222:53 || t
$IPTABLES -t nat -D OUTPUT -p udp --dport 53 -j DNAT --to-destination 208.67.222.222:53 || t

# This two new lines set our new DNS to OpenDNS
$IPTABLES -t nat -I OUTPUT -p tcp --dport 53 -j DNAT --to-destination 208.67.222.222:53
$IPTABLES -t nat -I OUTPUT -p udp --dport 53 -j DNAT --to-destination 208.67.222.222:53

# For IPv6 we need to change it in a different way since there is no nat!
$IP6TABLES -A INPUT -i $LAN_IF -s $LAN_NET -p udp –dport 53 -j ACCEPT
$IP6TABLES -A FORWARD -i $LAN_IF -s $LAN_NET -p udp –dport 53 -j ACCEPT
$IP6TABLES -A OUTPUT -p udp –dport 53 -j ACCEPT
```

```
# Deny IPv6 only connections
$IP6TABLES -P INPUT DROP
$IP6TABLES -P FORWARD DROP
$IP6TABLES -P OUTPUT DROP
```

```
# Web and mail from the LAN
$IP6TABLES -A FORWARD -i $LAN_IF -s $LAN_NET -p tcp -m multiport –dport 25,80,443 -j ACCEPT
```

```
# Web and mail from the internet
$IP6TABLES -A FORWARD -i $WAN_IF -s 2000::/3 -d $DMZ_NET -p -m multiport tcp –dport 25,80,44
```

```
# ICMPv6 - Do not allow that IPv6 gets control over everything
# Type 1: Destination unreachable
# Type 2: Time Exceeded
# Type 3: Parameter problem
# -> Multicast Listener Discovery (130, 131, 132 INPUT and OUTPUT)
# -> Ping (128 + 129, INPUT, OUTPUT and FORWARDING)
# -> Neighbor Discovery (ICMPv6-Typen 135 + 136)
# -> Router Discovery (ICMPv6-Typen 133 + 134)
# -> Path MTU-Discovery (ICMPv6-Typ 2)
# If you want multicast connections you can use some of this shown rules as an example!
$IP6TABLES -A INPUT -p icmpv6 –icmpv6-type 1 -j ACCEPT
$IP6TABLES -A INPUT -p icmpv6 –icmpv6-type 2 -j ACCEPT
$IP6TABLES -A INPUT -p icmpv6 –icmpv6-type 3 -j ACCEPT
$IP6TABLES -A INPUT -p icmpv6 –icmpv6-type 4 -j ACCEPT
$IP6TABLES -A FORWARD -i $WAN_IF -p icmpv6 –icmpv6-type 1 -j ACCEPT
$IP6TABLES -A FORWARD -i $WAN_IF -p icmpv6 –icmpv6-type 2 -j ACCEPT
$IP6TABLES -A FORWARD -i $WAN_IF -p icmpv6 –icmpv6-type 3 -j ACCEPT
$IP6TABLES -A FORWARD -i $WAN_IF -p icmpv6 –icmpv6-type 4 -j ACCEPT

# Router & Neighbor Discovery in-/outgoing
$IP6TABLES -A INPUT -p icmpv6 –icmpv6-type 133 -j ACCEPT
$IP6TABLES -A INPUT -p icmpv6 –icmpv6-type 134 -j ACCEPT
$IP6TABLES -A INPUT -p icmpv6 –icmpv6-type 135 -j ACCEPT
$IP6TABLES -A INPUT -p icmpv6 –icmpv6-type 136 -j ACCEPT
$IP6TABLES -A OUTPUT -p icmpv6 –icmpv6-type 133 -j ACCEPT
$IP6TABLES -A OUTPUT -p icmpv6 –icmpv6-type 134 -j ACCEPT
$IP6TABLES -A OUTPUT -p icmpv6 –icmpv6-type 135 -j ACCEPT
$IP6TABLES -A OUTPUT -p icmpv6 –icmpv6-type 136 -j ACCEPT

# Ping-Request to the Firewall from the LAN and DMZ
$IP6TABLES -A INPUT ! -i $WAN_IF -p icmpv6 –icmpv6-type 128 -j ACCEPT

# Ping-Request from the Firewall, LAN and DMZ
$IP6TABLES -A OUTPUT -p icmpv6 –icmpv6-type 128 -j ACCEPT
$IP6TABLES -A FORWARD ! -i $WAN_IF -p icmpv6 –icmpv6-type 128 -j ACCEPT
```

```
# Allow full outgoing connection but no incoming stuff
$IP6TABLES -A INPUT -i $WAN_IF -m state --state ESTABLISHED,RELATED -j ACCEPT
$IP6TABLES -A OUTPUT -o $WAN_IF -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

```
# Allow incoming ICMP ping
$IP6TABLES -A INPUT -i $WAN_IF -p ipv6-icmp -j ACCEPT
$IP6TABLES -A OUTPUT -o $WAN_IF -p ipv6-icmp -j ACCEPT
```

```
# Drop normal Multicast-addresses
$IPTABLES -A INPUT -s 224.0.0.0/4 -j DROP
$IPTABLES -A INPUT -d 224.0.0.0/4 -j DROP
$IPTABLES -A INPUT -s 240.0.0.0/5 -j DROP
$IPTABLES -A INPUT -d 240.0.0.0/5 -j DROP
$IPTABLES -A INPUT -s 0.0.0.0/8 -j DROP
$IPTABLES -A INPUT -d 0.0.0.0/8 -j DROP
$IPTABLES -A INPUT -d 239.255.255.0/24 -j DROP
$IPTABLES -A INPUT -d 255.255.255.255 -j DROP
```

DroidWall only examples

```
# Temporarily allow network adb access when you need it at port 5555 (for Android Studio 500
IP6TABLES=/system/bin/ip6tables
IPTABLES=/system/bin/iptables
ADB_UID=10004
SHELL_UID=2000
SAFE_NETWORK=10.23.69.0/24
```

```
# Allow adb (port 5555)
$IPTABLES -I INPUT-firewall -s $SAFE_NETWORK -p tcp --dport 5555 -j RETURN
$IPTABLES -I droidwall -m owner --uid-owner $ADB_UID -d $SAFE_NETWORK -m conntrack --ctstate
$IPTABLES -I droidwall -m owner --uid-owner $SHELL_UID -d $SAFE_NETWORK -m conntrack --ctsta

# Remove transproxy for adb output
$IPTABLES -t nat -I OUTPUT -d $SAFE_NETWORK -m conntrack --ctstate ESTABLISHED -p tcp --spor
```

```
# Allow the UDP activity of LinPhone to bypass Tor, to allow ZRTP-encrypted Voice and Video
IPTABLES=/system/bin/iptables
VOIP_UID1=`dumpsys package org.linphone | grep userId | cut -d= -f2 - | cut -d' ' -f1 -`
#VOIP_UID2=`dumpsys package org.lumicall.android | grep userId | cut -d= -f2 - | cut -d' ' -

# Allow VOIP client UDP return
$IPTABLES -I INPUT-firewall -m owner --uid-owner $VOIP_UID1 -m conntrack --ctstate RELATED,E
#$IPTABLES -I INPUT-firewall -m owner --uid-owner $VOIP_UID2 -m conntrack --ctstate RELATED,

# Remove transproxy for VOIP client UDP
$IPTABLES -I droidwall -m owner --uid-owner $VOIP_UID1 -p udp -j RETURN
#$IPTABLES -t nat -I OUTPUT -m owner --uid-owner $VOIP_UID2 -p udp -j RETURN

$IPTABLES -I droidwall -m owner --uid-owner $VOIP_UID2 -p udp -j RETURN
#$IPTABLES -t nat -I OUTPUT -m owner --uid-owner $VOIP_UID2 -p udp -j RETURN
```

```
# Shutdown script to run when AFWall+ is disabled. It will block everything.
# Droidwall
#chmod 0755 /data/data/com.googlecode.droidwall/app_bin/droidwall.sh

# Clear Rules
$IP6TABLES -t nat -F
$IP6TABLES -F
$IPTABLES -t nat -F
$IPTABLES -F INPUT-firewall
$IPTABLES -F OUTPUT-firewall

## Create Chains ##
$IPTABLES -N INPUT-firewall
$IPTABLES -D INPUT -j INPUT-firewall
$IPTABLES -I INPUT -j INPUT-firewall
$IPTABLES -N OUTPUT-firewall
$IPTABLES -D OUTPUT -j OUTPUT-firewall
$IPTABLES -I OUTPUT -j OUTPUT-firewall

## DROP TRAFFIC ##
$IP6TABLES -A INPUT -j LOG --log-prefix "Denied IPv6 input: "
$IP6TABLES -A INPUT -j DROP
$IP6TABLES -A OUTPUT -j LOG --log-prefix "Denied IPv6 output: "
$IP6TABLES -A OUTPUT -j DROP
$IPTABLES -A INPUT-firewall -j LOG --log-prefix "Denied IPv4 input: "
$IPTABLES -A INPUT-firewall -j DROP
$IPTABLES -A OUTPUT-firewall -j LOG --log-prefix "Denied IPv4 output: "
$IPTABLES -A OUTPUT-firewall -j DROP
```

```
# Allow stock Browser (com.android.browser)
IP6TABLES=/system/bin/ip6tables
IPTABLES=/system/bin/iptables
BROWSER_UID=`dumpsys package com.android.browser | grep userId | cut -d= -f2 - | cut -d' ' -

# Allow DNS input
$IPTABLES -I INPUT -m conntrack --ctstate RELATED,ESTABLISHED -p udp --sport 53 -j ACCEPT

# Allow already ESTABLISHED browser input
$IPTABLES -I INPUT -m conntrack --ctstate ESTABLISHED -m owner --uid-owner $BROWSER_UID -j A
```

```
# Allow root DNS output
$IPTABLES -I droidwall-wifi -m owner --uid-owner 0 -p udp --dport 53 -j ACCEPT

# Remove transproxy for root DNS
$IPTABLES -t nat -I OUTPUT -m owner --uid-owner 0 -p udp -m multiport --port 53 -j ACCEPT

# Remove transproxy for browser
$IPTABLES -t nat -I OUTPUT -m owner --uid-owner $BROWSER_UID -j ACCEPT
```

```
# Allow totify (org.torproject.android)
IP6TABLES=/system/bin/ip6tables
IPTABLES=/system/bin/iptables
ORBOT_UID=`dumpsys package org.torproject.android | grep userId | cut -d= -f2 - | cut -d' '

# Fix for https://code.google.com/p/droidwall/issues/detail?id=260
chmod 755 /data/data/com.googlecode.droidwall/app_bin/droidwall.sh

## Block all IPv6 and logs denied input ##
$IP6TABLES -t nat -F || true
$IP6TABLES -F
$IP6TABLES -A INPUT -j LOG --log-prefix "Denied IPv6 input: "
$IP6TABLES -A INPUT -j DROP
$IP6TABLES -A OUTPUT -j LOG --log-prefix "Denied IPv6 output: "
$IP6TABLES -A OUTPUT -j DROP

## INPUT ##
# Clear previous input firewall rules
# Re-create INPUT-firewall if it doesn't exist
$IPTABLES -N INPUT-firewall || true
$IPTABLES -F INPUT-firewall || true
$IPTABLES -D INPUT -j INPUT-firewall || true
$IPTABLES -I INPUT -j INPUT-firewall || true

# Create INPUT firewall. Only allow orbot input and local orbot ports
$IPTABLES -A INPUT-firewall -m conntrack --ctstate ESTABLISHED -m tcp -p tcp -m owner --uid-
$IPTABLES -A INPUT-firewall -d 127.0.0.1 -m tcp -p tcp --dport 8118 -j DROP || exit
$IPTABLES -A INPUT-firewall -i lo -j RETURN # Transproxy output comes from lo
$IPTABLES -A INPUT-firewall -d 127.0.0.1 -m udp -p udp --dport 5400 -j RETURN || exit
$IPTABLES -A INPUT-firewall -j LOG --log-prefix "INPUT DROPPED: " --log-uid || exit
$IPTABLES -A INPUT-firewall -j DROP || exit

## OUTPUT ##
# Clear previous output firewall rules
$IPTABLES -D OUTPUT -j OUTPUT-firewall || true

## Kill droidwall lan rule. It's not currently used, but if it ever gets
# activated it allows all DNS. So kill it as future-proofing
$IPTABLES -F droidwall-wifi-lan || true
$IPTABLES -D droidwall-wifi-lan || true

## Transproxy+Tor limits ##
# Note: We deliberately omit --syn because it causes leaks if used
$IPTABLES -t nat -F
$IPTABLES -t nat -A OUTPUT ! -d 127.0.0.1 -m owner ! --uid-owner $ORBOT_UID -p tcp -j REDIRE
$IPTABLES -t nat -A OUTPUT -p udp -m owner ! --uid-owner $ORBOT_UID -m udp --dport 53 -j RED

# Allow root's DNS proxy and kernel to do their transproxy thang
# FIXME: On newer kernels, this may require 1-9999999 instead of 0 (because
# the kernel is flagged as UID 0 instead of blank UID)
$IPTABLES -A droidwall -d 127.0.0.1/32 -p udp -m owner --uid-owner 0 -m udp --dport 5400 -j

#$IPTABLES -A droidwall -s 127.0.0.1 -p tcp -m owner ! --uid-owner 0-9999999 -m tcp -m multi
$IPTABLES -A droidwall -s 127.0.0.1 -p tcp -m owner ! --uid-owner 0-9999999 -m tcp --sport 9
$IPTABLES -A droidwall -s 127.0.0.1 -p tcp -m owner ! --uid-owner 0-9999999 -m tcp --dport 9

#$IPTABLES -A droidwall -d 127.0.0.1 -p tcp -m owner ! --uid-owner 0-9999999 -m tcp -m multi
$IPTABLES -A droidwall -d 127.0.0.1 -p tcp -m owner ! --uid-owner 0-9999999 -m tcp --sport 9
```

```
$IPTABLES -A droidwall -d 127.0.0.1 -p tcp -m owner ! --uid-owner 0-9999999 -m tcp --dport 9

#$IPTABLES -A droidwall -s 127.0.0.1 -p udp -m owner ! --uid-owner 0-9999999 -m udp -m multi
$IPTABLES -A droidwall -s 127.0.0.1 -p udp -m owner ! --uid-owner 0-9999999 -m udp --sport 5
$IPTABLES -A droidwall -s 127.0.0.1 -p udp -m owner ! --uid-owner 0-9999999 -m udp --dport 5

#$IPTABLES -A droidwall -d 127.0.0.1 -p udp -m owner ! --uid-owner 0-9999999 -m udp -m multi
$IPTABLES -A droidwall -d 127.0.0.1 -p udp -m owner ! --uid-owner 0-9999999 -m udp --sport 5
$IPTABLES -A droidwall -d 127.0.0.1 -p udp -m owner ! --uid-owner 0-9999999 -m udp --dport 5

# Allow all from orbot
$IPTABLES -A droidwall -m owner --uid-owner $ORBOT_UID -j RETURN || exit

# We also want to block remaining UDP. All app UDP should be already transproxied
$IPTABLES -A droidwall -p udp ! --dport 5400 -j LOG --log-prefix "Denied UDP: " --log-uid ||
$IPTABLES -A droidwall -p udp ! --dport 5400 -j DROP || exit

## Transproxy state leak fixes for
# https://lists.torproject.org/pipermail/tor-talk/2014-March/032503.html
$IPTABLES -A droidwall -m conntrack --ctstate INVALID -j LOG --log-prefix "Transproxy ctstat
$IPTABLES -A droidwall -m conntrack --ctstate INVALID -j DROP
$IPTABLES -A droidwall -m state --state INVALID -j LOG --log-prefix "Transproxy state leak b
$IPTABLES -A droidwall -m state --state INVALID -j DROP

# XXX: These are probably overkill and uncessary.
# They remain for defense in depth/debugging.
$IPTABLES -A droidwall ! -o lo ! -d 127.0.0.1 ! -s 127.0.0.1 -p tcp -m tcp --tcp-flags ACK,F
$IPTABLES -A droidwall ! -o lo ! -d 127.0.0.1 ! -s 127.0.0.1 -p tcp -m tcp --tcp-flags ACK,F
$IPTABLES -A droidwall ! -o lo ! -d 127.0.0.1 ! -s 127.0.0.1 -p tcp -m tcp --tcp-flags ACK,F
$IPTABLES -A droidwall ! -o lo ! -d 127.0.0.1 ! -s 127.0.0.1 -p tcp -m tcp --tcp-flags ACK,F

## REJECT FIXUPS
# Rewrite the reject rule to drop. ICMP is just logspam and ends up going to
# the wrong ports anyways
$IPTABLES -F droidwall-reject
$IPTABLES -A droidwall-reject -j LOG --log-prefix "[DROIDWALL] " --log-uid || exit
$IPTABLES -A droidwall-reject -j DROP || exit
```

```
# Try to apply my custom rule, but report any failure (and abort)
$IPTABLES -A -firewall --destination "192.168.0.1" -j RETURN || exit
```

```
# CM 11 M5 boot fix (userinit.sh)
# It disables "Google Captive Portal Detection", which involves connection attempts to Googl
#!/system/bin/sh

IP6TABLES=/system/bin/ip6tables
IPTABLES=/system/bin/iptables

## Block all traffic at boot ##
$IP6TABLES -F
$IP6TABLES -t nat -F
$IP6TABLES -A INPUT -j LOG --log-prefix "Denied bootup IPv6 input: "
$IP6TABLES -A INPUT -j DROP
$IP6TABLES -A OUTPUT -j LOG --log-prefix "Denied bootup IPv6 output: "
$IP6TABLES -A OUTPUT -j DROP

$IPTABLES -N INPUT-afwall
$IPTABLES -A INPUT-afwall -j LOG --log-prefix "Denied bootup IPv4 input: "
$IPTABLES -A INPUT-afwall -j DROP
$IPTABLES -I INPUT -j INPUT-afwall

$IPTABLES -N OUTPUT-afwall
$IPTABLES -A OUTPUT-afwall -j LOG --log-prefix "Denied bootup IPv4 output: "
$IPTABLES -A OUTPUT-afwall -j DROP
$IPTABLES -I OUTPUT -j OUTPUT-afwall
```

The Droidwalls only scripts are all downloadable via git clone or as raw files.

Test above settings please do follow:

```
adb shell su -c 'cp /PATH/userinit.sh /data/local/userinit.sh'
```

To stop the DNS resolve of check the capital portal and HTTP request to, clients3.google.com on every connection to any Wifi Access Point (see also here):

```
adb shell su -c "settings put global captive_portal_server 127.0.0.1"
adb shell su -c "settings put global captive_portal_detection_enabled 0"
```

To stop the resolve/request for 2.android.pool.ntp.org on every boot (even with network time disabled!):

```
su -c "settings put global ntp_server 127.0.0.1"
```

How do I view blocked IP address?

```
iptables -L -v
```

or

```
iptables -L INPUT -v
```

or

```
iptables -L INPUT -v -n
```

How do I block subnet?

If you like to block a subnet like (11.00.11.00/11) use the following syntax to block 11.00.11.00/11 on eth1 public interface:

```
iptables -i eth1 -A INPUT -s 10.0.0.0/8 -j DROP
```

Block incoming request from IP

The following command will drop any packet coming from the IP address 1.2.3.4:

```
iptables -I INPUT -s {IP-HERE} -j DROP
iptables -I INPUT -s 1.2.3.4 -j DROP
```

You can also specify an interface such as eth1 via which a packet was received:

```
iptables -I INPUT -i {INTERFACE-NAME-HERE} -s {IP-HERE} -j DROP
iptables -I INPUT -i eth1 -s 1.2.3.4 -j DROP
```

Block outgoing request from LAN

Block outgoing request from LAN IP 192.168.1.200? Here is the solution:

```
iptables -A OUTPUT -s 192.168.1.200 -j DROP
```

Orbot transparent proxy

Please ensure that you only use ONE !! iptables binary (orbot comes with it's one, so you have three to handle: system, AFWall's and Orbots!). To not szffering from iptables related problems, it's recommend to only use the system ones or AFWall+ to not override Orbot rules.

```
IPTABLES=/system/bin/iptables

# Example config
# lo = 127.0.0.1
# UID 10001
# Orbot Port (default) 9040
# Allow DNS (Port 53)

$IPTABLES -t filter -A OUTPUT -m owner --uid-owner 10001 -o lo -j ACCEPT
$IPTABLES -t nat -A OUTPUT -p tcp ! -o lo ! -d 127.0.0.1 ! -s 127.0.0.1 -m owner --uid-owner
$IPTABLES -t nat -A OUTPUT -p udp -m owner --uid-owner 10001 -m udp --dport 53 -j REDIRECT -
$IPTABLES -t filter -A OUTPUT -m owner --uid-owner 10001 ! -o lo ! -d 127.0.0.1 ! -s 127.0.0
```

After Orbot is shutdown, please re-flush all rules from AFWall+, if not you may get common connection problems.

How can I use a whitelist or blacklist?

If you want to log whitelisted or denied traffic in a .txt file(for debugging reasons) you can add this into the startup script.

```
#!/bin/bash

WHITELIST=/whitelist.txt
BLACKLIST=/blacklist.txt

# Clear all existent rules
echo 'Clearing all rules'
iptables -F


# Whitelist
for x in `grep -v ^# $WHITELIST | awk '{print $1}'`; do
        echo "Allowing $x..."
        $IPTABLES -A INPUT -t filter -s $x -j ACCEPT
done


# Blacklist
for x in `grep -v ^# $BLACKLIST | awk '{print $1}'`; do
        echo "Denied $x..."
        $IPTABLES -A INPUT -t filter -s $x -j DROP
done
```

Block traffic from secondary users

AFWall+ doesn't support multi-users on some Android versions, to workaround this you can base rules on the following example:

```
$IPTABLES -A OUTPUT -m owner --uid-owner $APPLICATION_UID  -j REJECT
```

This example will REJECT application outgoing traffic on each connectivity without logging, if you want to apply the rule to a specific connectivity, replace OUTPUT by the corresponding chain (e.g. `afwall-wifi`).

If you wish to log blocked packets, you should redirect to `afwall-reject` instead of dropping/rejecting. This solution is tedious but allow a tighter control.

Another solution would be to parse iptables rules and copy all rules using --uid-owner and recreate them adding a number after the first digit, for example 10234 would become 100234 or 110234 depending on the Android account ID you want to affect, downside of this solution is the rule list will become very big if you have too many users or too many applications.

External Links

- Simple Iptables Script Generator | Mista.nu (On some devices $IPT not working, try to replace with $IPTABLES or update your kernel!)
- 25 Most Frequently Used Linux IPTables Rules Examples | thegeekstuff.com
- Collection of basic Linux Firewall iptables rules | linuxconfig.org
- AFWall+/Droidwall permissions vulnerability | torproject.org - Original Issue 260 @ Droidwall
- [DNSleaktest | DNSleaktest.com] (https://www.dnsleaktest.com/)
- To fix the possible VPN leakage problem, take a look over here
- Test your IPv6 connectivity | test-ipv6.com