(ipta) IP Tables Log Analyzer

Documentation

by Ichimusai

This document is revision 0.0.1 and is documenting release 0.0.0 of the ipta tool.

IPTA is a tool used to create statistics and take care of the loggings from IP Tables in Linux. Tools are run from the command line to interprete syslogd files, create a database of events in mySQL and then other tools are used to perform extraction of statistics from this database.

It also has a “almost real-time” mode where logged packets are shown directly as they are written to the log file by syslog.

Documentation is delivered in Google Docs and PDF formats as needed. The Google Docs will always have the most current documentation. Anything marked in red is not yet implemented.

0 License

Credits for this software

All coding, documentation, testing of the original package and inventing the software in the first place: Anders "Ichimusai" Sikvall <ichi@ichimusai.org> http://ichimusai.org/

1 Introduction

I have searched the Internets for a while to find a good logging and reporting tool for IP tables but most tools were either too complex or not good enough. I am a firm believer in that security comes from being clear, simple and transparent and complexity is a very dangerous path. Therefore I decided to write my own version of a logging tool that can be used for anyone deploying a Linux server with IP Tables to get some kind of statistics for what is going on with their machine.

Now, this project is still in its initial phase, call it Alpha software if you like, so there may be lots of bugs and problems, missing functionality and other things. Some parts of this manual describes aims and goals rather than actually implemented features. I hope I will have marked those sections clearly enough.

1.1 Project aims

The aims of the project is to creat a collection of tools to be able to analyze logs from IP Tables, create reports on various issues, which types of packets are most commonly rejected, suspicious IP addresses in the world and many other things. It is a rather ambitious goal and my time is limited but I hope to spend some time on this and perhaps get a couple of other people involved in this.

We should rely as little as possible on external software. Not use obscure libraries that we are not able to distribute together with the source or the software and so on. Some things are unavoidable such as dependence on mySQL in this project wherefore those libraries needs to be installed but that should be easy to do.

Minimal configuration should be required to get up and running. A quick read through the manual and anyone really that knows their way around a terminal in Linux should be able to use this.

CGI friendly is also a goal, these tools should be friendly to call from a web server CGI such as Apache and yield nice results. This may mean we will implement reporting in HTML directly or perhaps XML that can be turned into HTML easily.

1.2 Todo

- Field length checking when importing into the database would be good.
- More analyzer modules
- Interactive command-driven analyzer not developed yet

2 Setting up

This log analyzer uses IP Tables only and is not compatible with other firewall projects as of now. In order to make it work properly you need to start logging dropped packets (perhaps also accepted packets) using the IP Tables log functions.

To prepare IP Tables for ipta you need to add a few rules. Exactly how you do this is up to you but normally this is done by adding a LOGDROP and LOGACCEPT rule that will cause IP Tables to log packets dropped and/or accepted through the syslog facility on the local machine. Although not necessary it may also be a good idea to create a LOGINVALID table and perhaps LOGABUSE that can be used.

2.1 Setting up the logger

I recommend to use a minimum of two log chains, the LOGDROP and the LOGACCEP chains should be the first to use. This means you can log interesting traffic (accepted) and distinguish it from dropped packets to make different types of analysis. However sometimes you will see packets that are dropped even on ports belonging to a service and this may be because you are dropping packets that have invalid states or other problems. In this case it makes sense to log those separately also, for example, using a LOGINVALID chain. You may also have some special abuse rules in the firewall and why not create a LOGABUSE for those.

Here is how they should be created:

```
iptables -N LOGACCEPT
iptables -A LOGACCEPT -j LOG --log-prefix “IPT: ACCEPT “ --log-level 7
iptables -A LOGACCEPT -j ACCEPT
```

Now, any rules you wish to log and accept, just use “-j LOGACCEPT” instead of the normal “-j ACCEPT”.

For the log and drop chain we do:

```
iptables -N LOGDROP
iptables -A LOGDROP -j LOG --log-prefix “IPT: DROP “ --log-level 7
iptables -A LOGDROP -j DROP
```

This will first log the packet with the prefix “IPT: DROP”, the IPT: is necessary for the ipta to recognize the log line, and the DROP part will be input in the database as the action for this packet.

All packets not matching any rules that are OK should normally be dropped. Therefore you can as the last part of your iptables setup add the line

```
iptables -A INPUT -j LOGDROP
```

Anything not matching a previous rule to be accepted or dropped earlier will then drop on this rule. That makes it rather safe.

Creating other chains for invalid packets or abuse packets follows the same rule. For invalid packets we can do the following:

```
iptables -N LOGINVALID
iptables -A LOGINVALID -j LOG --log-prefix “IPT: INVALID “ --log-level 7
iptables -A LOGINVALID -j DROP
```

2.2 A full iptables script

Normally a firewall configuration is not input manually but rather there is a script that is executed just after boot or sometimes by a cron job or similar. This script can take many forms but they generally do something like this:

1. Clear all previous rules
2. Set default policies
3. Set rules for accepting traffic to specific services
4. Deny all traffic not accepted

Below is an example iptables script, adapted to the ipta tool as well that you can use and expand upon yourself for your system. This should give you a general idea on how to set it up.

To fully explain all things with iptables is outside the scope of this manual but there are many tutorials and howto-docs on the net that will help you get started. The one below is rather well documented. It will also assume anything related to the loop-back interface (lo) is considered safe, if not, you can take that part out.

```
#!/bin/bash
# This is the IP Tables script used for ipta (IP Tables Analyzer) version x.x.x

# If your binary for iptables is located in a different directory
# change this definition
IPTABLES=”/sbin/iptables”

# Setting default policies
# if you want to log dropped packets the default policy should be ACCEPT so that
# you can send them to the log facility, otherwise they will be dropped silently
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P OUTPUT ACCEPT
$IPTABLES -P FORWARD DENY  # Change to ACCEPT if you use forwarding

# Flushing existing rules to make sure our iptables are empty
$IPTABLES -F

# Create the logging chains for our iptables
# What happens here is that we add two rules to each chain. One that sends
# the packet to be logged via syslog using a recognizable prefix as well as a
# single word telling what happened to the packet (DROP or ACCEPT) and then the
# packet is either dropped or accepted.
$IPTABLES -N LOGDROP # This rule may already exist
$IPTABLES -A LOGDROP -j LOG --log-prefic “IPT: DROP “ --log-level 7
$IPTABLES -A LOGDROP -j DROP

$IPTABLES -N LOGACCEPT
$IPTABLES -A LOGACCEPT -j LOG --log-prefix “IPT: ACCEPT “ --log-level 7
$IPTABLES -A LOGACCEPT -j ACCEPT

$IPTABLES -N LOGINVALID
$IPTABLES -A LOGINVALID -j LOG --log-prefix “IPT: INVALID “ --log-level 7
$IPTABLES -A LOGINVALID -j DROP
```

```
# Drop early all malformed packets
$IPTABLES -A INPUT -m state --state INVALID -j LOGINVALID

# Allow all traffic on local interfalce "lo" without logging
$IPTABLES -A INPUT -i lo -j ACCEPT
$IPTABLES -A OUTPUT -i lo -j ACCEPT

# Enable ICMP but log them
$IPTABLES -A INPUT -p icmp -j LOGACCEPT
$IPTABLES -A OUTPUT -p icmp -j LOGACCEPT

# Enable DNS server connections from local host, no logging here
# DNS is specified for both tcp and udp connections but almost always only
# udp is used so you may disable the tcp parts if you like
$IPTABLES -A INPUT -p tcp --sport domain -j ACCEPT
$IPTABLES -A INPUT -p udp --sport domain -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --dport domain -j ACCEPT
$IPTABLES -A OUTPUT -p udp --dport domain -j ACCEPT

# If you are using NTP for time synchronization you should have these
# enables, otherwise comment them out. I like to log them also.
$IPTABLES -A INPUT -p udp --sport 123 -m state --state ESTABLISHED -j LOGACCEPT
$IPTABLES -A OUTPUT -p udp --sport 123 -m state --state NEW,ESTABLISHED -j LOGACCEPT

# This enables incoming SSH connections to the local host
# we will log this activity. Logging the ESTABLISHED state usually renders a lot
# of packets in the log since that would be file transfers and so on, therefore we
# just log the new packets.
$IPTABLES -A INPUT -p tcp --dport 22 -m state --state NEW -j LOGACCEPT
$IPTABLES -A INPUT -p tcp --dport 22 -m state --state ESTABLISHED -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --sport 22 -m state --state ESTABLISHED -J ACCEPT

# Enable traffic to your web server on port 80, no logging
$IPTABLES -A INPUT -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
$IPTABLES -A OUTPUT -p tcp --sport 80 -m state --state ESTABLISHED -j ACCEPT

# Everything else that falls through here will be considered non-wanted traffic and
# will therefore be logged also, then dropped.
$IPTABLES -A INPUT -j LOGDROP
$IPTABLES -A OUTPUT -j LOGDROP
```

2.1.1 Logging loops

As with any complex system there is the potential to have unintended consequences depending on how you set it up. During development I found out that it is easy to get into a "logging loop" with the --follow mode engaged. The problem here is that if you log certain types of packets, such as DNS requests (port 53) or ssh (port 22) and you use the --rdns option or view the traffic over SSH the ipta may generate it's own traffic that in its turn creates even more traffic as it attempts to display the already generated traffic. I will show you two examples of what not to do.

Logging of DNS

Logginf DNS requests may seen fine, and it is most of the time. However if you use the --follow option and the --rdns option, each line printed on the screen will do 2 DNS requests, right? One for source IP address, one for destination IP address. This means that for each line printed two reqests will be logged, which in their turn will mean two lines printed with four DNS requests which means eight lines printed with 16 DNS requests…

You get the picture. Avoid this. It may eat your disk also.

Logging of SSH

If you are accessing your system over the Internet with SSH and you are logging this and using the --follow option then you can get a similar result as with the DNS log/request cascade. Every time ipta will try to print something on the screen will trigger new SSH packets to be logged which in their turn should be printed on the screen and so on.

You should therefore only log NEW packets for SSH and accept ESTABLISHED or RELATED silently. That logs just one packet

2.3 Recommended actions

The standard log actions are ACCEPT, DENY or DROP where DENY is actually not very common, but here are a few suggestions to implement:

Action	**Description**
DROP	Packet was dropped without further ado, should be pretty safe to do with all packets that are not explicitly allowed.
ACCEPT	Packets that are accepted but you still wanted to log it for some reason. This is useful logging in conjunction with -m state --state NEW as we can then see who is knocking on the door. This action is required. The reason is that in many built-in statistics modules in the analyzer the system needs to be able to distinguish between wanted and unwanted traffic. It will therefore assume that anything that is ACCEPT is wanted and anything with any other label is not wanted traffic.
CBLK	CIDR blocked networks. In many cases it is pretty useful to implement a blocking regime in the iptables firewall where entire countries can be blocked. In order to log those packets dropped as a result of CIDR or country blocking this may be a good action name to use.
INVALID	All dropped invalid packets, since these packets are dropped because they are doing something that does not look right to iptables they are one of the more important parts to keep track of.

You can add as many actions as you like but keep the name short, less than 10 letters is recommended or it will be truncated. Generally you do not want to have too many actions.

The maximum length of this field is 10 chars. Do not exceed this as the field will be truncated when read into the database.

In the analyzer module the “ACCEPT” has a special status as that is excluded from some of the built-in statistics functions. This is useful when looking at nefarious packets and you wish to remove all OK packets, therefore you must tag all wanted (but still logged) traffic with the ACCEPT action since that means the statistics module may distinguish between different types of traffic.

2.4 Making syslog log to a separate file

In many systems the iptables are logged to the same file as any other messages. Although this is absolutely fine, it will take some extra time to process (and throw away) all lines in the log file that does not concern us. The ipta tool will always look for the “IPT:” in each line to log. If there is no “IPT:” prefix, the line is skipped. If there are lines that contains the “IPT:” marker and are not iptables syslog lines, then that will yield empty rows in the database and which will not show up in statistics anyway…

But you may want to set up syslog in a way that iptables are logged to a separate file. This may vary a little from system to system but usually there is a configuration directory where this is set up. In Ubuntu 14.04 the syslog is rsyslog and there is a configuration directory in /etc/rsyslog.d

In order to separate the iptables log from the rest of the syslog you can create a config file here. You can call the file what you want but 20-iptables.conf is a good bet will work pretty well. The contents of this file should be something like this:

```
# Syslog iptables to separate file
:msg, contains, “IPT: “ -/var/log/iptables.log
# Remove from other logs
& ~
```

And that’s that, save the file and restart syslog, this may be done with

```
sudo service rsyslog restart
```

If your system uses a different syslog or if there is something else going on you may have to check your system docs to find out. After you have done that, you probably want to logrotate this file so that is does not fill up your disk. You may look at something like 50 000 - 150 000 lines per day in this file!

2.5 Logrotate your log

Since the logfile has a potential of becoming big, I suggest rotating it on a daily basis and just keeping what you need. It is probably a good idea to zip the files older than today and yesterday also to save space on the disk.

To set this up you need to find your logrotate.d directory, it usually resides in /etc/logrotate.d/ and if you go there you will find the configuration files for logrotate. One of them is probably already made for syslog, in Ubuntu 14.04 it will be called rsyslog. Open and edit this file carefully with an editor of your choice.

```
/var/log/iptables.log
{
      rotate 30
      daily
      missingok
      notifempty
      compress
      delaycompress
      sharedscripts
      postrotate
```

```
            reload rsyslog >/dev/null 2>&1 || true
      endscript
}
```

This will rotate your iptables.log every day and keep them 30 days back. It will perform some housekeeping as well as restarting the rsyslog (you may need to change that to the syslog in your system) in order to keep sytems nice. It will also compress all files from day before yesterday and back while keeping the two latest days, today and yesterday, uncompressed.

Of course if you want to rotate on a weekly basis or something else, you can change this accordingly. Insert it into the rsyslog or create a new one called “ipta” or something like that, then save it. Check it out a couple of days later that it is actually rotating your logs.

3 IPTA usage and syntax

This tool is used to parse log files from syslogd regarding IP-tables data. Depending on how you set your iptables up to log you may log only denied, accepted or both types of packets. This is entirely up to you and you must configure the iptables on your machine to do some logging first in order to get some data for the ipta-parse to work with.

You need to have your MySQL setup for this tool to work. Also, remember that you may want to clear your existing tables when starting a new import, otherwise it will just be imported even if the rows already exists.

You can change the settings for the database in the ipta-system configuration file in your home folder. It should be called .ipta-conf and will be read upon start to find your database names, passwords etc.

3.1 Options

The settings can be overridden by using command line parameters:

`--license`	Display the software license. The license is a modified MIT license and gives you the right to do a lot with the software including modifying it and re-distribute it.
`-i, --import <file>`	This is a mode and must be followed by a path to the syslog file to import into the database.
`-c, --clear-db`	Clear database before importing, if not the default behaviour is to keep old data and append new imported data
`-d, --db-name <name>`	Use database <name>. If not given ipta expect the database “ipta” to be used
`-h, --db-host <host>`	Host to connect to for the MySQL database. Default is “localhost”.
`-u, --db-user <username>`	Username for the database connection. Default is “ipta”.
`-p, --db-pass <password>`	Password. This is unsecure on a multiuser system as it may show in process lists and logs, shell history etc. Use -pi or

	--db-pass-interactive and you will get prompted for password instead.
`-pi, --db-pass-i <password>`	Will prompt for password rather than accepting it at the command line. Safer but not useful for scripting.
`-t, --db-table <table>`	Use <table> instead of the default "ipta". This means you can use different tables in different sessions, keeping tables between sessions and so on.
`-lt, --list-tables`	List the tables in the ipta database. This is used when you need to find out what tables you have there
`-dt, --delete-table`	Remove any unwanted tables from the database. Use the --list-tables to find out what they are first, then remove any you do not want any more. To specifically say which table also use the --db-table option, otherwise the default one is used. To specify database use --db-name.
`-da, --delete-all-tables`	Delete all tables from the database, start fresh. Easier if you want to delete many tables rather than doing them one by one.
`-ct, --create-table`	Create a new table that can be used with ipta. This is necessary before using ipta the first time. If no other arguments are given default name, password, database and table name will be used. Use --db-name, --db-table to select a different database and a different table name.
`-r, --rdns`	This is a flag that it used together with the analyzer module. It affects the presentation of IP addresses. Default is to present numerical IP numbers. If the flag --rdns is supplied the analyzer module will attempt to reverse-DNS the IP number into a proper host name. This will sometimes yield good information such the network provider or other info.
`-a, --analyze`	Start the analyser using some predetermined queries to form a teori on what is currently going on. This will run and output various analysis such as most not accepted packets, which ports are most frequently involved, which IP addresses are responsible for most dropped packets and so on.
`-ai, --analyze-interactive`	This will throw you into the deep end of the pond where you may create your own SQL queries from the command line to dig deeper. Not for the faint hearted, but gives you enormous flexibility. Check out the Appendix I on SQL commands!
`-s, --save-db`	Save-db option will write out a .ipta file with the standard settings of your database so you do not need to put them on the command line all the time. This file will then be read by ipta everytime you use it. You may edit this file in an editor also.

`-l, --limit <limit>`	The standard limit for the number of lines that are presented in the anlyzer module is 10 lines per rule. If you want to change this you can use the command line argument here and change this to <limit> lines instead.
`-f, --follow <file>`	The follow option is a special mode which puts the ipta into a real-time display of what happens in the syslog file. Basically it reads the syslog file and outputs the contents in a nicely formatted way much easier to read than the original syslog. You may use --rdns option or --no-header if you wish. The --no-count can be used to remove the leading packet counter on each line.
`--no-header`	Remove the header printed every 20 lines when using the --follow mode.
`--no-count`	Remove the counter in front of each line in the follow mode.
`--no-lo`	Remove all packets that are either originating from or destined to the local interface “lo”. This works currently only in --follow mode
`--no-accept`	Remove all packets that has the action ACCEPT. This works currently only in --follow mode.

4 How to use the tool

The first thing you should do is to set up your database and have the tool wsave it for you as default. This is done by running the command:

```
ipta --setup-db
```

It will prompt you for the following information: The host to connect to (default is localhost), the database username and password, the database name and the table name to use for storing the data.

It will then connect to MySQL on the host you provided, it will assume the user is already present and the database is also there, it will then attempt to create an empty table with the right set of columns, the definition of this table can be found in this document.

It will also write your credentials to the file .ipta in your home directory. This file should not be readable for anyone but you since it contains passwords. You can override anything in this file by specifying command line parameters to this effect but it will always retreive the default information from this file.

If the .ipta file do not exist, then the tool will assume the following information as default and use that if you do not override it with any command line arguments: hostname = “localhost”, database user = “ipta”, database password = “ipta”, database to use = “ipta” and table to store information in will be “logs”.

4.1 Import the logs

Importing the logs are easy providing they are in the right format. When everything is done correctly you should have syslog files that contains lines that looks something like this:

```
Sep 21 23:49:24 zathras kernel: [109970.165828] IPT: CBLK IN=eth0 OUT=
MAC=02:00:bc:7e:5d:a0:00:13:5f:21:29:40:08:00 SRC=178.154.243.111 DST=188.126.93.160
LEN=60 TOS=0x00 PREC=0x00 TTL=51 ID=13144 DF PROTO=TCP SPT=47908 DPT=80 WINDOW=14100
RES=0x00 SYN URGP=0
```

The above is interpreted so that the first fields are used to build a time stamp, then from the marker “IPT:” (iptables) it will start looking for the actual data. The first field is the action field which you control from your iptables settings, in this case the CBLK means the packet was dropped due to CIDR router blocking (country blocking). The next fields are the input or output interfaces (etho) the MAC address of the ethernet device delivering the IP packet, source IP address, destination ip address, length of packet, some other fields less important in this case but then we get the protocol type (TCP), source and destination ports.

From this a row is inserted in the MySQL database that will log the most important features of the packet, including the action field.

To import all lines from a syslog file use the command

```
ipta --import /var/log/iptables.log
```

If you have previous data in the database you may wish to clear that before you import the new data, especially if it is the same file you are importing, otherwise there will be duplicate entries in the database depending on being imported twice. To clear the database prior to importing new data use:

```
ipta --clear-db --import /var/log/iptables.log
```

This will clear all data in the database and then import everything from the file /var/log/iptables.log.

If you want you can tell ipta to use a different host, a different username, password and table name for the database using several switches.

```
ipta --db-host localhost --db-user anders --db-pass 8rI44ckEt7
  --db-table 141010_ipta --clear --import /var/log/iptables.log
```

This will now use the provided credentials. Be careful with specifying the password on the command line like this since it may show up in task lists such as from ‘ps’ or ‘top’ and therefore is not secure in a multiuser environment. In that case you shouls use the interactive version of asking for password with --sb-pass-i which will read the password interactively instead of from the command line.

Don’t forget also that if you set it up using the command

```
ipta --setup-db
```

Then unless you override with the command line the ipta tool will read its credentials from the file in your home dir: `~/.ipta` as specified.

During the import you will see a progress counter that tells you the number of lines and the time it has been working. For a large log this may take a few minutes. If you are running on something like a Raspberry Pi you can expect rather poor disk I/O at times so be patient. On a fullblown modern PC system it should be reasonably fast actually.

Once the import is done you can now start working with the analyzer module. This module has a short version that will just dump a number of pre-determined statistical runs on the data which is useful for quickly assessing if there is something going on that needs your attention, the other part is more interesting it is the interactive module which you can type commands to see what is going on and drill down to the data or even write your own SQL statements.

More on that in the analyzer.

4.2 Statistics Analyzer

There are two analyzers to chose from and this chapter will describe the quick built-in that will run queries pre-defined, or built-in if you like. To invoke this all you need to do is actually type

```
ipta --analyze
```

This will dump a bunch of information in your screen so you may want to pipe the output to a program like "less" in order to be able to view it back and forth. If you like you can also pipe the result to a text file and then analyze it, store it for later for comparison and so on.

If you wish to translage the IP addresses into hostnames you can also use the flag --rdns on the command line to tell the anaylzer module to attempt reverse-DNS the IP addresses into proper host names. This can sometimes give surprisingly good information about a host as it may tell you the network provider etc.

Analyzer requires you to have some data in the database to work on. You should import a couple of days worth of logs at least and then let it crunch on these. You may want to import all logs from the last week or even month (that may take some time though) and see what the long term trends are. Only you will set the limit.

```
ipta -- rdns --analyze > file.txt
ipta --analyze | less
```

So the output from the quick auto-analyzer focuses on the highest counts of suspicious packets in a number of ways to look at them.

The analyzer will show a maximum of 10 lines for each type of analysis that it is built in. It will show the packet counts for each type of packet and it will sort them in a descending fashtion.

Over time it is expected that you identify harmless traffic and update the iptables settings so that you drop traffikc you know is harmless silently, thus fine tuning the ipta reports. All you need to do is to send these packets directly to the chain DROP or ACCEPT rather than any of the logging chains. That will remove them from future statistics.

4.2.1 Denied traffic grouped by IP, destination port, action and protocol

```
Count   Source IP        SPort   Dest IP          DPort   Proto    Action
------   ---------------   -----   ---------------   -----   ------   ----------
   420   213.136.38.8       6919   188.126.93.160      80   TCP      INVALID
   348   188.126.93.76       138   188.126.93.255     138   UDP      DROP
   348   188.126.93.11       138   188.126.93.255     138   UDP      DROP
   336   178.154.243.111   40163   188.126.93.160      80   TCP      CBLK
   210   193.201.224.42    49910   188.126.93.160      80   TCP      CBLK
   192   193.201.224.80    54556   188.126.93.160      80   TCP      CBLK
   144   193.201.224.100   60649   188.126.93.160      80   TCP      CBLK
   136   83.172.66.74          0   188.126.93.160       0   41       DROP
   132   193.201.224.152   64186   188.126.93.160      80   TCP      CBLK
   108   193.201.224.132   53431   188.126.93.160      80   TCP      CBLK
```

This is an example output from an actual system. The first line shows 420 packets dropped because they were invalid, the destination port 138 shows there is some windows traffic on the LAN and if you notice you can see that the destination addres for these packages is 188.126.93.255 which is actually the network broadcast address. So they are not really aimed at us, they are just stray packets that flies around.

Then we have a few lines of dropped packets that where dropped because of CBLK which is our CIDR blocker that blocks certain countries. All packets dropped from these lines are coded with the action CBLK as described earlier in this document.

We also have dropped some 136 packets that was using a strange protocol number 41. This protocol turns out to be encapsulated IPv6 traffic. This may not be bad traffic but we don’t serve IPv6 here so they are dropped.

4.2.2 Showing ICMP traffic statistics

```
 Count   Source IP        SPort   Dest IP          DPort   Proto    Action
------   ---------------   -----   ---------------   -----   ------   ----------
    12   203.178.148.19        0   188.126.93.160       0   ICMP     CBLK
     8   188.126.93.160        0   198.20.99.130        0   ICMP     ACCEPT
     3   129.82.138.44         0   188.126.93.160       0   ICMP     ACCEPT
     3   128.9.168.98          0   188.126.93.160       0   ICMP     ACCEPT
     2   198.20.99.130         0   188.126.93.160       0   ICMP     ACCEPT
```

This part shows ICMP statistics. ICMP is generally not a very dangerous protocol and even though some people say you should block it, don’t do it unless you have specific reason for it, it is a necessary protocol to find out if a server or a network is down, if there is a different MTU somewhere so your packets fragment and also to “ping” a server to se if it is alive.

Unless you have a specific reason to be afraid of this protocol, don't block it.

4.2.3 Most denied ports

```
 Count   DPort   Proto    Action
------   -----   ------   ----------
  1586      80    TCP      CBLK
   837      80    TCP      INVALID
   696     138    UDP      DROP
   210      21    TCP      DROP
   136       0    41       DROP
    65      22    TCP      DROP
    46      23    TCP      DROP
    34      25    TCP      DROP
    33      53    UDP      DROP
    33   10142    UDP      DROP
```

This is a summary statistic on the most commonly blocked and dropped traffic no matter what the origin is. This is useful for checking trends and which ports are being “knocked at” mostly.

Above there is no surprise really, port 80 and 138 we already know, port 22 is the port used for standard SSH, port 23 is for Telnet (please disable Telnet if you are running a public server on the Internet). Port 53 is for DNS and apparently people are trying to get to us through sending spoofed DNS packets.

Ports 25 and 10142 as well as some other mor esoteric ports have all to do with SMTP over various transport mechanisms. People are looking for open relays probably.

4.2.4 Most invalid packets comes from

```
 Count   Source IP         SPort   Dest IP           DPort   Proto
------   ---------------   -----   ---------------   -----   ------
   420   213.136.38.8       6919   188.126.93.160       80   TCP
    98   37.59.15.42       50434   188.126.93.160       80   TCP
    49   62.210.78.197     59606   188.126.93.160       80   TCP
    46   72.50.82.165      47649   188.126.93.160       80   TCP
    42   107.150.52.42     64557   188.126.93.160       80   TCP
    18   199.30.24.108     23111   188.126.93.160       80   TCP
    14   195.154.184.126   56160   188.126.93.160       80   TCP
    14   87.241.85.2       49817   188.126.93.160       80   TCP
    14   85.188.73.66      54191   188.126.93.160       80   TCP
```

This part splits up the INVALID packets on who is transmitting them, source and destination port and protocol type. If you are gettin glots of invalid packets it may be a good idea to run a wireshark or something similar to understand why they are seen as invalid and what is the reason that you are seeing them.

4.2.5 Limit your results

The standard limit to any request done by the analyzer module is limited to 10 rows in the answer. If you want to change this to some other number of lines you should use the --limit option on the command line. The limit option takes exactly one argument which should be a numeric argument in the range of 1 - 1000 which is the maximum number of lines returned in any query that ipta will make.

ipta may return a fewer number of lines than what you specify, that depends on the actual matches in the database as this can vary from time to time. The limit is just a maximum number of returned lines.

4.3 Follow mode

Before you continue you should read the part on log loops in the iptables setup section because that could save you from some unintended consequences with this mode as the printouts or the software itself could potentially trigger new packets to be logged by syslog continuously.

The follow mode is a special mode where the ipta tool start at the end of the syslog file and output a nicely formatted version of the data as soon as new lines are written to it. The information is then displayed in a clear to understand and easy to read format (as the iptables syslog lines aren't that pretty).

This is very useful for two things. One is that you can debug your syslogs so that you can see the ipta tool is doing the right thing, that the fields are picked up in the right order and so on, properly interpreted. The other is that you can in real time follow what is going on with your server which of course gives piece of mind and a nice good easy feeling,

To initiate you run this

```
ipta --follow /var/log/iptables.log
```

Of course your file name may vary depending on how you set the system up. Could even be the syslog or messages although I recommend logging iptables to a separate file.

Follow mode can not be combined with --import or --analyze or any other mode. It will stay in effect until the user breaks by pressing CTRL-C or logout from the terminal. Therefore also any options dealing with the database will not actually do anything, although they will be accepted as options.

4.3.1 Options useful in follow mode

There are several options available in the follow mode. These options include options to show reverse DNS names for any logged IP address as well as filtering out traffic on the local interface or traffic that is logged but allowed.

--rdns

You may also use the --rdns in order to attempt making reverse DNS lookups on the IP addresses involved. The names may be truncated to fit the fields. A truncated name has an asterisk (*) in the first character position indicating it has been truncated to fit the field.

In the example below you can find several such cases. As we truncate from the start, country codes and domain names should remain visible in the table anyways.

--no-header

You may also use the --no-header option to remove the headers printed every 20 lines. This can be useful if you wish to pipe the resulting lines into another file for further processing using grep, awk, sed or other such tools.

--no-count

Each line is also preceeded by a counter that shows you in succession the number of packets that ipta has picked up and processed. You can remove this counter by issuing a --no-count option if you do not want it. The counter is useful but may be in the way if you wish to post-process the data later on.

--no-accept

Any packets with the “Action” ACCEPT will not be displayed, this filter lets you focus on only the packets that are dropped, invalid or cblocked if you set it up this way. It would probably be the sane thing to do in most cases.

Other options, such as database name and similar are accepted if you input them on the line but they won’t actually do anything. The reason is that once ipta enters follow mode it will stay until you break it with a CTRL-C or close the terminal.

4.3.2 Example

```
Count     Source                     Port  Destination                Port  Proto       Action
--------  ------------------------  -----  ------------------------  -----  ----------  ----------
   37161  zathras.ichimusai.org     45200  zathras.ichimusai.org        80  TCP         ACCEPT
   37162  zathras.ichimusai.org     45201  zathras.ichimusai.org        80  TCP         ACCEPT
   37163  *-12-129.unifiedlayer.com    80  *-12-129.unifiedlayer.com 65501  TCP         INVALID
   37164  zathras.ichimusai.org       123  zathras.ichimusai.org       123  UDP         ACCEPT
   37165  zathras.ichimusai.org       123  zathras.ichimusai.org       123  UDP         ACCEPT
   37166  188-126-93-11.vps.moln.is   138  188-126-93-11.vps.moln.is   138  UDP         DROP
   37167  188-126-93-11.vps.moln.is   138  188-126-93-11.vps.moln.is   138  UDP         DROP
   37168  *.pools.sprint-mobile.net   138  *.pools.sprint-mobile.net   138  UDP         DROP
   37169  *.pools.sprint-mobile.net   138  *.pools.sprint-mobile.net   138  UDP         DROP
   37170  *5.clients.your-server.de 53332  *5.clients.your-server.de    80  TCP         ACCEPT
   37171  zathras.ichimusai.org       123  zathras.ichimusai.org       123  UDP         ACCEPT
```

The above shows an example of running the --follow mode with --rdns turned on akin to the following command:

```
ipta --follow /var/log/iptables.log --rdns --no-accept --no-lo
```

If you are using the --rdns option and it is not able to look up the names it will instead show the ip address as expected. The --no-accept option means that the follow mode will show only those packets that are not ACCEPTed by the iptables log chain. It looks at the “action” field and determines this. The last option --no-lo means that no packets either destined to or emanating from the local loopback interface “lo” will be shown.

Most of the time you will not be interested in the loopback interface packets or those that are accepted but still logged so these options makes it easy to concentrate on these packet types. For full statistics you should consider using the --analyze mode which will show you summaries of different packets from different hosts and protocols etc.

Appendix I - Database design

The database is a rather simple database consisting of the tabe *logs* which contains the parsed log entries from the iptables syslogd files importet using the parser tool.

Logs

The table logs contains the following columns briefly described here:

timestamp	TIMESTAMP	Contains the time stamp from the syslogd row that was processed into this database
id	INT AUTO_INCREMENT	This is just an id in order to be able to select certain rows in the database. It may also be the primary key.
if_in	VARCHAR(10)	This is the interface name if the packet entered the iptables. Usually this will be something like eth0 or similar.
if_out	VARCHAR(10)	This has value if the interface is an exit interface for the packet, usually this will be something like eth0 or similar.
src_ip	INT UNSIGNED	Contains the source IP (sender) of the packet stored as an unsigned int. Use the function INET_ATON to convert to int unsigned and INET_NTOA() to convert back to IP address with dot notation when using this field.
src_prt	INT UNSIGNED	The source port of the packet. This ranges from 0-65535.
dst_ip	INT UNSIGNED	Contains the destination IP (receiver) of the packet stored as an unsigned int. Use the functions INET_ATON to convert to int unsigned and INET_NTOA to convert back to IP dot notation when using this field.
dst_prt	INT UNSIGNED	The destination port of the packet. This ranges from 0-65535.
proto	VARCHAR(10)	Describes the protocol as a string. This can be TCP, UDP or ICMP as an example of valid protocols.
action	VARCHAR(10)	This describes the action on the packet taken by the iptables system. This can be DROP, ACCEPT or DENY
mac	VARCHAR(40)	MAC address long format if applicable

DNS

The “dns” table is used to hold cached information on name/ip relationships that the get-host-by-address function provides but because a large number of DNS requests is slow we can cache this in the database instead and therefore have a shortcut.

The --rdns flag will engage this behaviour, and this is what happens: When ipta encounters an ip address it will look in this table to find the name to replace it with. If no name can be found it will ask the DNS subsystem to provide one, if it gets a name in response, that name is inserted in the database, if no name is provided the IP address is provided as the name field.

time	TIMESTAMP	The time this entry was inserted. This is used to check how old the entry is and if it should be used or renewed. This is updated on each update of the table.
ip	INT UNSIGNED	This is the IP address of the host that we are looking for, it is stored as an unsigned int so use the functions INET_ATOI() and INET_NTOA() to convert to and from this field. This field is also PRIMARY KEY.
name	VARCHAR(256)	Actual name. This field can be up to 255 characters plus a NULL termination.

Appendix II - Configuration file format

The configuration file is created by using the mode --setup-db which will do three things:

1. Ask you to input the data needed, which is:
    a. Hostname for database server
    b. Username for database
    c. Password for database
    d. Database name for database
    e. Default table name
2. Create a configuration file in your home dir ~/.ipta which will contain this information. The file contains password in plain text so check that it is only readable by you. The ipta sets mode 600 for this file.
3. Connect to the database server and attempt to create the database as well as the default table.

The created configuration file will be a file with key=value store of the various parameters. The following keywors will be used:

```
db_host=<hostname>
db_user=<username>
db_pass=<password>
db_name=<database name>
db_table=<default table>
```

That's it. Any line starting with a # will be regarded as a comment in the file although the automatic configuration may or may not write comments.

Appendix III - SQL Statements

This section details some of the more useful SQL statements to have a loot at the logs once they are processed into the MySQL database.

Top 15 addresses you are dropping packets from

```
SELECT COUNT(*), INET_NTOA(src_ip), src_prt, INET_NTOA(dst_ip), dst_prt, proto, action
FROM logs WHERE action='DROP' GROUP BY src_ip ORDER BY COUNT(*) DESC LIMIT 15;
```

The INET_NTOA() function converts the IP address to dotted format (182.54.33.123) because it is stored as an unsigned int in the database for efficiency. The reverse function used is INET_ATON() that take an IP address in dotted format and converts it to an unsigned int.