Proceedings of the Linux Symposium

Volume Two

July 20nd–23th, 2005
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Stephanie Donovan, *Linux Symposium*

Review Committee

Gerrit Huizenga, *IBM*
Matthew Wilcox, *HP*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Matt Domsch, *Dell*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

Flow-based network accounting with Linux

Harald Welte
netfilter core team / hmw-consulting.de / Astaro AG
laforge@netfilter.org

Abstract

Many networking scenarios require some form of network accounting that goes beyond some simple packet and byte counters as available from the 'ifconfig' output.

When people want to do network accouting, the past and current Linux kernel didn't provide them with any reasonable mechanism for doing so.

Network accounting can generally be done in a number of different ways. The traditional way is to capture all packets by some userspace program. Capturing can be done via a number of mechanisms such as `PF_PACKET` sockets, `mmap()`ed `PF_PACKET`, `ipt_ULOG`, or `ip_queue`. This userspace program then analyzes the packets and aggregates the result into per-flow data structures.

Whatever mechanism used, this scheme has a fundamental performance limitation, since all packets need to be copied and analyzed by a userspace process.

The author has implemented a different approach, by which the accounting information is stored in the in-kernel connection tracking table of the `ip_conntrack` stateful firewall state machine. On all firewalls, that state table has to be kept anyways—the additional overhead introduced by accounting is minimal.

Once a connection is evicted from the state table, its accounting relevant data is transferred to userspace to a special accounting daemon for further processing, aggregation and finally storage in the accounting log/database.

1 Network accounting

Network accounting generally describes the process of counting and potentially summarizing metadata of network traffic. The kind of metadata is largely dependant on the particular application, but usually includes data such as numbers of packets, numbers of bytes, source and destination ip address.

There are many reasons for doing accounting of networking traffic, among them

- transfer volume or bandwisth based billing
- monitoring of network utilization, bandwidth distribution and link usage
- research, such as distribution of traffic among protocols, average packet size, ...

2 Existing accounting solutions for Linux

There are a number of existing packages to do network accounting with Linux. The follow-

ing subsections intend to give a short overview about the most commonly used ones.

2.1 nacctd

`nacctd` also known as `net-acct` is probably the oldest known tool for network accounting under Linux (also works on other Unix-like operating systems). The author of this paper has used `nacctd` as an accounting tool as early as 1995. It was originally developed by Ulrich Callmeier, but apparently abandoned later on. The development seems to have continued in multiple branches, one of them being the netacct-mysql[1] branch, currently at version 0.79rc2.

Its principle of operation is to use an `AF_PACKET` socket via `libpcap` in order to capture copies of all packets on configurable network interfaces. It then does TCP/IP header parsing on each packet. Summary information such as port numbers, IP addresses, number of bytes are then stored in an internal table for aggregation of successive packets of the same flow. The table entries are evicted and stored in a human-readable ASCII file. Patches exist for sending information directly into SQL databases, or saving data in machine-readable data format.

As a pcap-based solution, it suffers from the performance penalty of copying every full packet to userspace. As a packet-based solution, it suffers from the penalty of having to interpret every single packet.

2.2 ipt_LOG based

The Linux packet filtering subsystem iptables offers a way to log policy violations via the kernel message ring buffer. This mechanism is called `ipt_LOG` (or `LOG target`). Such messages are then further processed by `klogd` and `syslogd`, which put them into one or multiple system log files.

As `ipt_LOG` was designed for logging policy violations and not for accounting, its overhead is significant. Every packet needs to be interpreted in-kernel, then printed in ASCII format to the kernel message ring buffer, then copied from klogd to syslogd, and again copied into a text file. Even worse, most syslog installations are configured to write kernel log messages synchronously to disk, avoiding the usual write buffering of the block I/O layer and disk subsystem.

To sum up and anlyze the data, often custom perl scripts are used. Those perl scripts have to parse the LOG lines, build up a table of flows, add the packet size fields and finally export the data in the desired format. Due to the inefficient storage format, performance is again wasted at analyzation time.

2.3 ipt_ULOG based (ulogd, ulog-acctd)

The iptables `ULOG target` is a more efficient version of the `LOG target` described above. Instead of copying ascii messages via the kernel ring buffer, it can be configured to only copies the header of each packet, and send those copies in large batches. A special userspace process, normally ulogd, receives those partial packet copies and does further interpretation.

`ulogd`[2] is intended for logging of security violations and thus resembles the functionality of LOG. it creates one logfile entry per packet. It

[1] `http://netacct-mysql.gabrovo.com`

[2] `http://gnumonks.org/projects/ulogd`

supports logging in many formats, such as SQL databases or PCAP format.

`ulog-acctd`[3] is a hybrid between `ulogd` and `nacctd`. It replaces the `nacctd` libpcap/PF_PACKET based capture with the more efficient ULOG mechanism.

Compared to `ipt_LOG`, `ipt_ULOG` reduces the amount of copied data and required kernel/userspace context switches and thus improves performance. However, the whole mechanism is still intended for logging of security violations. Use for accounting is out of its design.

2.4 iptables based (ipac-ng)

Every packet filtering rule in the Linux packet filter (`iptables`, or even its predecessor `ipchains`) has two counters: number of packets and number of bytes matching this particular rule.

By carefully placing rules with no target (so-called *fallthrough*) rules in the packetfilter ruleset, one can implement an accounting setup, i.e., one rule per customer.

A number of tools exist to parse the iptables command output and summarized the counters. The most commonly used package is `ipac-ng`[4]. It supports advanced features such as storing accounting data in SQL databases.

The approach works quite efficiently for small installations (i.e., small number of accounting rules). Therefore, the accounting granularity can only be very low. One counter for each single port number at any given ip address is certainly not applicable.

[3] `http://alioth.debian.org/projects/pkg-ulog-acctd/`

[4] `http://sourceforge.net/projects/ipac-ng/`

2.5 ipt_ACCOUNT (iptaccount)

`ipt_ACCOUNT`[5] is a special-purpose iptables target developed by Intra2net AG and available from the netfilter project patch-o-matic-ng repository. It requires kernel patching and is not included in the mainline kernel.

`ipt_ACCOUNT` keeps byte counters per IP address in a given subnet, up to a '/8' network. Those counters can be read via a special `iptaccount` commandline tool.

Being limited to local network segments up to '/8' size, and only having per-ip granularity are two limiteations that defeat `ipt_ACCOUNT` as a generich accounting mechainism. It's highly-optimized, but also special-purpose.

2.6 ntop (including PF_RING)

`ntop`[6] is a network traffic probe to show network usage. It uses `libpcap` to capture the packets, and then aggregates flows in userspace. On a fundamental level it's therefore similar to what `nacctd` does.

From the ntop project, there's also `nProbe`, a network traffic probe that exports flow based information in Cisco NETFLOW v5/v9 format. It also contains support for the upcoming IETF IPFIX[7] format.

To increase performance of the probe, the author (Luca Deri) has implemented `PF_RING`[8], a new zero-copy mmap()ed implementation for

[5] `http://www.intra2net.com/opensource/ipt_account/`

[6] `http://www.ntop.org/ntop.html`

[7] IP Flow Information Export `http://www.ietf.org/html.charters/ipfix-charter.html`

[8] `http://www.ntop.org/PF_RING.html`

packet capture. There is a libpcap compatibility layer on top, so any pcap-using application can benefit from `PF_RING`.

`PF_RING` is a major performance improvement, please look at the documentation and the paper published by Luca Deri.

However, `ntop` / `nProbe` / `PF_RING` are all packet-based accounting solutions. Every packet needs to be analyzed by some userspace process—even if there is no copying involved. Due to `PF_RING` optimiziation, it is probably as efficient as this approach can get.

3 New ip_conntrack based accounting

The fundamental idea is to (ab)use the connection tracking subsystem of the Linux 2.4.x / 2.6.x kernel for accounting purposes. There are several reasons why this is a good fit:

- It already keeps per-connection state information. Extending this information to contain a set of counters is easy.
- Lots of routers/firewalls are already running it, and therefore paying its performance penalty for security reasons. Bumping a couple of counters will introduce very little additional penalty.
- There was already an (out-of-tree) system to dump connection tracking information to userspace, called ctnetlink.

So given that a particular machine was already running `ip_conntrack`, adding flow based acconting to it comes almost for free. I do not advocate the use of `ip_conntrack` merely for accounting, since that would be again a waste of performance.

3.1 ip_conntrack_acct

`ip_conntrack_acct` is how the in-kernel `ip_conntrack` counters are called. There is a set of four counters: numbers of packets and bytes for original and reply direction of a given connection.

If you configure a recent (>= 2.6.9) kernel, it will prompt you for `CONFIG_IP_NF_CT_ACCT`. By enabling this configuration option, the per-connection counters will be added, and the accounting code will be compiled in.

However, there is still no efficient means of reading out those counters. They can be accessed via *cat /proc/net/ip_conntrack*, but that's not a real solution. The kernel iterates over all connections and ASCII-formats the data. Also, it is a polling-based mechanism. If the polling interval is too short, connections might get evicted from the state table before their final counters are being read. If the interval is too small, performance will suffer.

To counter this problem, a combination of conntrack notifiers and ctnetlink is being used.

3.2 conntrack notifiers

Conntrack notifiers use the core kernel notifier infrastructure (`struct notifier_block`) to notify other parts of the kernel about connection tracking events. Such events include creation, deletion and modification of connection tracking entries.

The `conntrack notifiers` can help us overcome the polling architecture. If we'd only listen to *conntrack delete* events, we would always get the byte and packet counters at the end of a connection.

However, the events are in-kernel events and therefore not directly suitable for an accounting application to be run in userspace.

3.3 ctnetlink

`ctnetlink` (short form for conntrack netlink) is a mechanism for passing connection tracking state information between kernel and userspace, originally developed by Jay Schulist and Harald Welte. As the name implies, it uses Linux `AF_NETLINK` sockets as its underlying communication facility.

The focus of `ctnetlink` is to selectively read or dump entries from the connection tracking table to userspace. It also allows userspace processes to delete and create conntrack entries as well as *conntrack expectations*.

The initial nature of `ctnetlink` is therefore again polling-based. An userspace process sends a request for certain information, the kernel responds with the requested information.

By combining `conntrack notifiers` with `ctnetlink`, it is possible to register a notifier handler that in turn sends `ctnetlink` event messages down the `AF_NETLINK` socket.

A userspace process can now listen for such *DELETE* event messages at the socket, and put the counters into its accounting storage.

There are still some shortcomings inherent to that *DELETE* event scheme: We only know the amount of traffic after the connection is over. If a connection lasts for a long time (let's say days, weeks), then it is impossible to use this form of accounting for any kind of quota-based billing, where the user would be informed (or disconnected, traffic shaped, whatever) when he exceeds his quota. Also, the conntrack entry does not contain information about when the connection started—only the timestamp of the end-of-connection is known.

To overcome limitation number one, the accounting process can use a combined event and polling scheme. The granularity of accounting can therefore be configured by the polling interval, and a compromise between performance and accuracy can be made.

To overcome the second limitation, the accounting process can also listen for *NEW* event messages. By correlating the *NEW* and *DELETE* messages of a connection, accounting datasets containign start and end of connection can be built.

3.4 ulogd2

As described earlier in this paper, `ulogd` is a userspace packet filter logging daemon that is already used for packet-based accounting, even if it isn't the best fit.

`ulogd2`, also developed by the author of this paper, takes logging beyond per-packet based information, but also includes support for per-connection or per-flow based data.

Instead of supporting only `ipt_ULOG` input, a number of interpreter and output plugins, `ulogd2` supports a concept called *plugin stacks*. Multiple stacks can exist within one deamon. Any such stack consists out of plugins. A plugin can be a source, sink or filter.

Sources acquire per-packet or per-connection data from `ipt_ULOG` or `ip_contnrack_acct`.

Filters allow the user to filter or aggregate information. Filtering is requird, since there is no way to filter the ctnetlink event messages within the kernel. Either the functionality is enabled or not. Multiple connections can be aggregated to a larger, encompassing flow. Packets could be aggregated to flows (like `nacctd`), and flows can be aggregated to even larger flows.

Sink plugins store the resulting data to some form of non-volatile storage, such as SQL databases, binary or ascii files. Another sink is a NETFLOW or IPFIX sink, exporting information in industy-standard format for flow based accounting.

3.5 Status of implementation

`ip_conntrack_acct` is already in the kernel since 2.6.9.

`ctnetlink` and the `conntrack event notifiers` are considered stable and will be submitted for mainline inclusion soon. Both are available from the patch-o-matic-ng repository of the netfilter project.

At the time of writing of this paper, `ulogd2` development was not yet finished. However, the ctnetlink event messages can already be dumped by the use of the "conntrack" userspace program, available from the netfilter project.

The "conntrack" prorgram can listen to the netlink event socket and dump the information in human-readable form (one ASCII line per ctnetlink message) to stdout. Custom accounting solutions can read this information from stdin, parse and process it according to their needs.

4 Summary

Despite the large number of available accounting tools, the author is confident that inventing yet another one is worthwhile.

Many existing implementations suffer from performance issues by design. Most of them are very special-purpose. nProbe/ntop together with `PF_RING` are probably the most universal and efficient solution for any accounting problem.

Still, the new `ip_conntrack_acct`, `ctnetlink` based mechanism described in this paper has a clear performance advantage if you want to do acconting on your Linux-based stateful packetfilter—which is a common case. The firewall is suposed to be at the edge of your network, exactly where you usually do accounting of ingress and/or egress traffic.