TrafficWatch Documentation

Peter Hawkins

1 Abstract

The TrafficWatch system is a set of tools and web pages which attempt to keep track of internet usage in a college environment. It is targeted at a Linux environment, using a combination of C, Python and shell scripting.

2 Basic Design

The purpose of the TrafficWatch system is to account for the internet usage of users in a university college / school environment. We make the following starting assumptions:

- Users are to be charged by data volume for their internet usage.
- Users access the internet from machines on a Local Area Network, and are to be charged for their data traffic that leaves the border of this network.

In order to account for usage, we firstly need to be able to count the data traffic that leaves the network, and more importantly, we need to be able to attribute this data traffic to the user that generated it. There are several different types of data traffic we will have to account for, each in a slightly different way:

- Direct traffic from student computers to the outside world. This includes protocols such as ICMP, Instant Messaging, SSH, Telnet, and many others.
- Squid proxy services, including HTTP, HTTPS, FTP services, all of which involve having a proxy server making requests on the behalf of other computers. These type of requests require special treatment in order to correctly apportion the traffic generated by the proxy server to each user.
- Email - here a SMTP server handles the network traffic on behalf of the network user. This is not currently dealt with by the TrafficWatch system although adding support would not be that difficult.

In order to account for direct traffic, we use a sniffer daemon - `ipcap`, which runs on a network router computer and produces a log of the flows of data over time, together with source IP and MAC address information. By keeping a registry of the MAC addresses in use by each user, we can count the direct network traffic used by each user's computer.

To account for proxy services, such as Squid, we parse the request log file produced by the daemon as it runs. By using devices such as proxy authentication, the log file contains sufficient information for a log analysis script to attribute each request in the log file to a user.

Hence the basic sources of information for the system are Squid's `access.log`, the `flow.log` of `ipcap` (part of the TrafficWatch system), and mail logs. In order to deal with these diverse information sources, the system converts each of these log files into a common intermediate format (described below), and imports them into an SQL database. From the SQL database we can produce reports on a user's usage (both to the user and for administrators), grant/deny access based upon usage, and so on. For this we use a set of web pages which produce reports, and a set of scripts that regularly update firewall rules, send emails and generally interact with the world.

3 Network architecture

As described in the previous section, there are several machines within the network on which the TrafficWatch system has to be installed. A single machine may play more than one of the following roles, but for flexibility this documentation assumes each role is played by different machines. Each of these machines will need the TrafficWatch package and its dependencies installed; some of these machines will need additional packages installed as well.

Database server TrafficWatch stores almost all of its data in a single PostgreSQL database. Hence a PostgreSQL server must be installed on a machine which we will denote as `dbserver`.

Gateway router In order to account for direct TCP/IP traffic, the `ipcap` sniffer daemon must be installed on a gateway router. The gateway router will also play the role of blocking internet access to users who have not registered their MAC addresses or who have exceeded their quota of internete traffic. We will assume the gateway router is called `gwrouter`.

Squid proxy server TrafficWatch needs to parse the Squid log files in order to account for web proxy traffic. We will assume the Squid proxy machine is called `proxy`.

Web server In order to provide feedback to users on their usage, TrafficWatch has a series of web pages that allow users and administrators to make queries about internet usage. We will assume the web server machine is named `wwwserver`.

4 Installation and configuration

The easiest way by far to install TrafficWatch is to install it as a Debian package. The installation procedure when installing using the Debian package is as follows.

Firstly, install the TrafficWatch Debian package on each of the machines described above. The relevant Debian package will be called something like `trafficwatch_x.y.z_all.deb`. The easiest way to install it is to add the following APT source to `/etc/apt/sources.list`:

```
# Packages for TrafficWatch and ipcap
deb http://www.hawkins.emu.id.au/software ./
```

If you are using Debian 3.0 (Woody) release of Debian, you will also need to add the following APT sources as well (these contain backports of packages from the upcoming 'Sarge' release of Debian):

```
# python 2.2, 2.3 and associated package backports
deb http://www.hawkins.emu.id.au/backports ./

# Postgresql 7.3 backport
deb http://people.debian.org/~elphick/debian woody main
```

Having done this, you can install the TrafficWatch package and all of its dependencies by running:

```
apt-get install trafficwatch
```

In addition to the dependencies of the package, you will need to install various other sets of packages depending on the role of a particular machine in an installation. If a given machine is going to perform more than one of the roles below, install the union of the relevant sets of packages suggested below.

Database server `postgresql`, `postgresql-client`

Web server `apache` and `libapache-mod-python` or `apache2` and `libapache2-mod-python`. If you are using Windows NT/Samba authentication, you will also need the `smbclient` package.

Gateway router `iptables`, and the `ipcap` package which is part of Trafficwatch. Since we also use the gateway router to display web pages for MAC address registration and to unauthenticated clients, you will also need to install the same set of packages as described for the web server.

In addition, on the firewall machine, you will need to compile and build the `mac_mask` kernel module from source (for which you will require the kernel headers for your kernel, and the iptables source tree corresponding to your installed version of the iptables command). This is a tiny iptables extension module which allows matching on masked bits of a MAC address and is used for efficiency reasons. To build this module, firstly download the file `http://www.hawkins.emu.id.au/software/mac_mask/iptables_mac_mask-0.0.1.tar.gz`, then perform the following commands:

```
# First, we need to install and build the iptables package from source.
# Replace x.y.z with the actual version number of the iptables package.
apt-get source iptables
cd iptables-x.y.z/upstream
tar jxvf iptables-x.y.z.tar.bz2
cd ../..

# Now, we need the kernel headers for the version of the kernel you have
# installed. Amend the package name below to match your installed kernel
apt-get install kernel-headers-2.4.xx

tar zxvf iptables_mac_mask-0.0.1.tar.gz
cd iptables_mac_mask
./configure \
    --with-kernel-headers=/usr/src/kernel-headers-2.4.xx/include \
    --with-iptables-headers=../iptables-x.y.z/upstream/iptables-x.y.z/include
make

# Then as root, run:
make install
```

The first thing you will need to do is configure Trafficwatch on each machine for your network environment. The main configuration files you will need to edit are `/etc/trafficwatch/config.py`, `/etc/cron.d/trafficwatch`, `/etc/ipcap.conf`, `/etc/apache/httpd.conf`, and `/etc/trafficwatch/apache.conf`.

4.1 `/etc/trafficwatch/config.py`

This configuration file holds most of the configuration data for the TrafficWatch system. The file is relatively well commented, and most of the changes should be fairly self-explanatory. Please note that this is in fact a python script, so you must write valid python in it! Do not change the structure of the configuration file, and do not comment out any of the value assignments.

Start by configuring the copy of this file on the database server `dbserver`. Firstly, set the `unconfigured` option to 0 to indicate that you have configured the system. You may wish to adjust the `database_name` and `database_connect_string` options to suit your installation of Postgresql, but the defaults should suffice for most people. Next, in the WebConfig section, you will need to configure the `support_url`, `escape_url`, `mac_registration_url` and `mac_check_url` options.

4.2 `/etc/cron.d/trafficwatch`

Certain aspects of the trafficwatch system, such as log imports, or firewall rule updates, run from `cron`. In this file you should enable only the items appropriate to the machine on which you are working (eg. only one machine should have the mailout jobs running).

On the `gwrouter` and `proxy` machines you will need to enable the cron jobs that perform log imports every 10-15 minutes. This is one of two places from which log imports are run - the other is immediately after log rotation, so you'll also need to modify your logrotate scripts for `squid` and `ipcap` (see below for details).

Also on the `gwrouter` machine you will need to enable the line which runs `twatch_firewall_regen` - this script creates and updates iptables rules that set a configurable mark on packets that come from authenticated users.

4.3 /etc/ipcap.conf

This file configures the ipcap daemon. For more information about ipcap, see the relevant section below.

4.4 /etc/apache/httpd.conf, /etc/trafficwatch/apache.conf

On the web server wwwserver which is going to provide the web page aspects of the TrafficWatch system, you will need to configure Apache for TrafficWatch. Add the following line to your httpd.conf:

```
Include /etc/trafficwatch/apache.conf
```

This will allow you to access the TrafficWatch user visible web pages via http://yourserver/trafficwatch/. If you use a different web server path (something other than /trafficwatch, then you will have to make the corresponding change in /etc/trafficwatch/config.py.

To display the other web pages that TrafficWatch offers, you will need to set up a basic web server root. You can do this by creating a directory with symlinks to images and stylesheet.css in /usr/share/trafficwatch/web/h and creating a link to the script you wish to publish in /usr/share/trafficwatch/web/python. Scripts that can be web visible are traffic.py, mac.py, admin.py, accessdenied.py. You will need to add an appropriate <Directory> directive (like those in /etc/trafficwatch/apache.conf), as well as .htaccess protection for any administrative pages.

4.5 /etc/logrotate.d/ipcap, /etc/logrotate.d/squid

As well as performing regular partial log file imports from cron jobs, TrafficWatch also needs to perform log imports on log rotation - so you'll need to make some modifications to the logrotate scripts for the ipcap and squid daemons on the machines where those daemons run. An example of the modified ipcap logrotate file is shown below. You can find logrotate files already modified for ipcap and squid in /usr/share/doc/trafficwatch/examples/.

```
/var/log/ipcap/flow.log {
    rotate 60
    daily
    compress
    delaycompress

# Begin addition
    prerotate
        lockfile-create --retry 20 /var/lib/trafficwatch/offsets_locks/ipcap-flo
w-log.offset
    endscript

    postrotate
        killall -WINCH ipcap
        /usr/share/trafficwatch/scripts/log_partial_import.sh -l -s 2 \
            /var/log/ipcap/flow.log.1 \
            /var/lib/trafficwatch/offsets_locks/ipcap-flow-log.offset
        lockfile-remove /var/lib/trafficwatch/offsets_locks/ipcap-flow-log.offse
t
    endscript

# End addition
}
```

Once you have configured all of these files (phew!), you should arrange to get some users into the database. You can import user data with the twatch_user_sync command, which is documented below. User data can either come from CSV files with the structure described below, or from an LDAP directory.

On the gwrouter machine, you'll need to set up firewall rules that block access to people who are unregistered or are over quota. Trafficwatch, through the twatch_firewall_regen script which is run as a cron job, creates iptables rules in the mangle table which set a configurable mark on packets that come from recognised users. If block_over_quota_users is set in config.py, then a different mark value will be set for those users who are over quota. You will need to set up your own iptables ruleset to act upon these mark values. For example (assuming eth0 is your internal network interface and eth1 is your external network interface, and TrafficWatch is configured to set a mark value of 1 on authenticated users):

```
# Allow all incoming traffic
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT

# Allow outgoing traffic only from authenticated users
iptables -A FORWARD -i eth0 -o eth1 -m mark --mark 1 -j ACCEPT
iptables -A FORWARD -i eth0 -j REJECT

# We can use the NAT features of iptables to replace any web pages that
# an unauthenticated user attempts to view with an "Access denied" page.
# This assumes that there is an apache process running on port 80 set up
# as described below.

# Don't NAT anything marked.
iptables -t nat -A PREROUTING -i eth0 -m mark --mark 1 -j ACCEPT

# NAT any web or web proxy traffic from unauthenticated users to an access
# denied page.
iptables -t nat -A PREROUTING -i eth0 --dport 80 -j REDIRECT --to-ports 80
iptables -t nat -A PREROUTING -i eth0 --dport 3128 -j REDIRECT --to-ports 80
iptables -t nat -A PREROUTING -i eth0 --dport 8080 -j REDIRECT --to-ports 80
```

Then, we configure our apache installation on port 80 of gwrouter as follows (this configuration file is /usr/share/doc/trafficwatch/examples/router_httpd.conf in the default installation):

```
# We assume that mod_rewrite, mod_proxy and mod_python have all been
# enabled.
<VirtualHost _default_:80>
    RewriteEngine on

    # Allow local trafficwatch requests
    RewriteCond %{HTTP_HOST} ^gwrouter\.example\.org$ [NC]
    RewriteRule ^.* - [L]

    # Pass through local requests
    RewriteCond %{HTTP_HOST}  ^(.*)\.example\.org$ [NC]
    RewriteRule ^/(.*)        http://%{HTTP_HOST}/$1  [P]

    # Rewrite everything else to point to trafficwatch
    RewriteRule ^.* http://gwrouter.example.org/trafficwatch/accessdenied.py/index  [L,R]

    # Proxy configuration
    ProxyRequests off
    NoProxy *
    ProxyDomain .example.org
    NoCache *

    Alias /trafficwatch/ /usr/share/trafficwatch/web/ht-docs/
    Alias /trafficwatch/accessdenied.py /usr/share/trafficwatch/web/python/accessdenied.py
    Alias /trafficwatch/mac.py /usr/share/trafficwatch/web/python/mac.py
</VirtualHost>
```

```
<Directory /usr/share/trafficwatch/web/python>
        AddHandler python-program .py
        PythonHandler mod_python.publisher
        PythonPath "sys.path + ['/usr/share/trafficwatch/web/python']"
</Directory>
```

If you wish to import any additional log files, as produced by `ipcap` or `squid`, you should place them in `/var/lib/trafficwatch/logs/normal` or `/var/lib/trafficwatch/logs/squid` (respectively), from where the hourly import cron job will import them. Any log files that cannot be imported for whatever reason will be moved into the corresponding directory with `-failed` on the end. Exactly how you get the log files into those directories will depend very much on your particular configuration. If the relevant daemon is running locally, a simple `cp` after the daily log rotation should be sufficient. Otherwise, methods as using `scp` together with ssh public key authentication should suffice.

5 Components

5.1 `ipcap` - Internet flow accounting daemon

`ipcap` produces accounting information for IP flows over a network by sniffing packets from a network interface using libpcap. This daemon must be run on a machine that sees all the packets for which we are attempting to account, and hence must be run on a Linux/*BSD router (recommended) or on a passive sniffer machine connected to a switch SPAN port or similar.

It is recommended that this daemon be run on the router itself because this makes it somewhat more difficult for local users to fake packets to other users from places that are deemed expensive and apparently causing victims to run up large bills.

Note that since origin fields in IPv4 packets are untrustworthy, it is possible for users external to the college network to fake packet origin information and attempt the same trick. No particularly good solution to this problem is known, other than effective firewalling.

In theory at least, MAC addresses are untrustworthy. Although in an ideal world, MAC addresses would be irrevocable linked to a piece of network hardware, there are several potential sources of problems. Firstly, there is a chance that two network cards could have the same physical address. This will cause network connectivity problems anyway and the only real solution is to pull one of the network cards off the network. Secondly, MAC addresses can be changed/faked in several ways. Some ethernet cards have the MAC address set in firmware in such a way that it can be easily altered. Alternatively, the ethernet card can be placed in promiscuous mode and outgoing packets sent with a forged MAC address. This should be kept in mind when using MAC addresses for authentication and accounting purposes.

Full documentation on how to configure the `ipcap` daemon is provided in its configuration file, typically `/etc/ipcap/ipcap.conf`.

`ipcap` produces two log files - `flow.log` and `mac.log`. These are the Data and Authentication logfiles as described below. The authentication logfile effectively provides a mapping between MAC address and IP address over time. `mac.log` is considered untrustworthy and so it is not currently used.

It is recommended that libpcap version 0.6 or later is used on Linux machines and the 'Linux Socket Filter' kernel configuration option be turned on, since this allows `ipcap` to use in-kernel packet filters which will reduce CPU usage somewhat.

5.2 `twatch_import` - Importing flow data

`twatch_import` imports TrafficWatch Data logfile and Squid logfiles into the SQL database, optionally using a supplied Authentication logfile. The process of importing the logfiles includes both classification of flows into Traffic Classes (based on the remote host address), and aggegration of data within each class. `twatch_import` is a wrapper around the `db_import.py` and `squid.py` scripts.

For each line of the input log file, `twatch_import` attempts to classify the remote IP address either based on host name or on a routing data file provided by Melbourne University ITS. The exact details of how this is done are configurable (see `trafficconf.py`). The flow is then linked to a user, as directly as possible. Each of the following methods will be attempted in order:

- If there is a username in the input logfile, it will be used. If the username is not known to the TrafficWatch database, then a non-fatal error message '`Unknown user <user>`' will be produced.
- If there is a MAC address in the input logfile, `db_import.py` will attempt to link it back to a user directly by looking up the list of registered MAC addresses in the database.
- If there is only an IP address, see if we can find either a username or a MAC address in the supplied authentication logfile. If we find a MAC address, look up the MAC address in the registered MAC address list.
- If we still can't account for the logline, then we spit it back out to `stdout` in the hope that someone else will deal with it.

Squid log files are processed by the internal script `squid.py`, which is documented below. Note that squid log files must be in the Squid native format, not the alternate httpd-style log format.

In the case of squid log files, a "first person to request a page from the proxy pays" policy is used. The 'remote IP' address that TrafficWatch uses is always that of the peer given in the proxy log file. If no peer was used (implying that the page was not fetched or could not be fetched for whatever reason) then TrafficWatch assumes that that request did not involve fetching any data from a remote server, and hence should not be charged.

5.2.1 Usage

`twatch_import` [`-a <authlogfile>`] [`-s <sourceidnum>`] [`-d`] `<logfile1> <logfile2>...`

Command line options:

`-a <authentication logfile>` Specify location of authentication logfile to use (overrides location specified in the TrafficWatch configuration file).

`-s <sourceid value>` Specify the TrafficWatch sourceid value to attach to this data (see the configuration file for the meaning of sourceid values). Default is 1.

`-d` Specify that the logfiles given as arguments to this command are squid logfiles, not TrafficWatch format logfiles.

5.3 Adminstrative scripts

A brief description of each of the administrative scripts is given below. For full information on the arguments, etc. that each accepts, see the associated manual page.

5.3.1 `twatch_setup`

`twatch_setup` is a script that sets up TrafficWatch database from information contained in the TrafficWatch configuration file. This script will wipe all existing information from the database, and hence will refuse to run unless "confirm" is given as its first argument.

5.3.2 `twatch_config_sync`

`twatch_config_sync` is a script that updates certain aspects of the TrafficWatch database to match those in the configuration file `config.py` (namely those to do with user types and traffic classes).

5.3.3 `twatch_user_sync`

`twatch_sync` is a script that adds/updates/deletes the TrafficWatch user list in the TrafficWatch database. It accepts a single argument, which is the data source to use. Further options depend on the source (and are documented in the manual page). Currently valid sources are LDAP and CSV (Comma-Seperated-Values).

5.3.4 `twatch_firewall_flush`

`twatch_firewall_flush` is a script that deletes the TrafficWatch iptables mangle table rules.

5.3.5 `twatch_firewall_regen`

`twatch_firewall_regen` is a script that (re)generates a set of firewall rules that set a configurable mark value of all the packets which originate from the MAC addresses of authenticated users. This is done by adding a 'twatch' chain to INPUT chain of the iptables mangle table. MAC addresses are split into a configurable number of buckets, and a configurable mark set on packets originating from known MAC addresses.

This requires that the custom iptables target `mac_mask` be installed against your current linux kernel and iptables userspace tools.

The idea here is that you then add your own rules that act on the basis of whether the mark is set on a packet on not. eg.

```
iptables -A FORWARD -i eth0 -m mark --mark 1 -j ACCEPT
iptables -A FORWARD -i eth0 -j DROP
```

5.3.6 `twatch_backup`

`twatch_backup` produces a dump of the TrafficWatch PostgreSQL database to standard output.

5.4 The web pages

5.4.1 `mac.py`

`mac.py` is the web page users use to register their MAC addresses with the TrafficWatch system. It has one major method - `index` - which does all the work.

5.4.2 `accessdenied.py`

`accessdenied.py` has one method - `index`. What it says is pretty self explanatory... =)

5.4.3 `traffic.py`

`traffic.py` is the main interface seen by users of the system. It has several methods, which collectively produce the user visible portion of the system. The entry point of the page is the `login` method. The code to this page is rather ugly and could do with a rewrite, but it is functional.

5.4.4 `admin.py`

`admin.py` is the administrative interface to the system. From here, adminstrators can produce reports on internet usage, change the rates at which Traffic Classes are charged, administer quotas, and view usage reports on users. The entry point of the page is the `index` method. It is assumed that this page will be protected by a `.htaccess` file or similar.

5.5 Miscellaneous internal scripts

All of these scripts that are meant to be user visible are documented above (they have the prefix `twatch_*`).

`twatch_upgrade.py` Upgrade a database from an older release of TrafficWatch to the current database schema. Make sure you back up your database using before running this script.

`setup.py` Initialise the TrafficWatch database, and set up the database schema. All existing data in the database will be wiped, and the database will be recreated.

`ldap_sync.py` Synchronise the TrafficWatch user list with an LDAP server (as specified in the configuration file). Note synchronisation will never remove a user from the database - users will be added or updated, but never deleted. Hence a complete purge of the database should probably occur annually.

Users who exist in the database but who no longer exist in LDAP will be marked as not being current, which will prevent mail, etc. from being sent to them.

`csv_sync.py` Synchronise the TrafficWatch user list with a set of CSV files. Users can be added, updated, or deleted. For more details, see the `twatch_sync` manual page.

`db_import.py` Import a TrafficWatch Data logfile into the database.

Command line options of note are `-s` and `-a`. `-s` specifies a source ID number that will be attached to each piece of data added to the database. This is useful since it allows us to trace back a database item to the correct log file, as well as allowing us to scale different traffic sources differently. `-a` specifies the authentication log file we should use (although this can also be specified in the configuration file).

Typical usage might be:
```
zcat flow.log.1.gz | db_import.py -s 2 -a mac.log.1 >> unaccounted.log
```
For a squid log file, we might run:
```
squid.py < access.log.1 2>> unresolved.squid.log | db_import.py -s 1 -a mac.log.combined
>> unaccounted.log
```

Here `mac.log.combined` is a combination of the past two days' authentication log files as some added insurance that no authentication data will be missed due to the interval between rotation of the squid logfile and the ipcap logfiles.

`ip2fqdn.py` This script should not be used.

`squid.py` `squid.py` converts a Squid `access.log` logfile to a TrafficWatch Data Logfile. The wrapper script `twatch_import` may be more convenient to use as it provides nicer semantics.

Translation of squid log files into TrafficWatch data logfiles is mostly a text processing task - the only real work that is done is to resolve any hostnames into IP addresses. For example, a typical squid `access.log` line might look like this:
```
1014179024.094 10860 203.28.241.52 TCP_MISS/200 21344 GET
http://mirror.bom.gov.au/radar/IDR023.gif?  -
DIRECT/mirror.bom.gov.au image/gif
```
The output of `squid.py` would be:
```
1014179024 - - 203.28.241.52 203.63.53.11 21344 0
```

Note the major changes - the timestamp is truncated; no MAC address is used (although ident/proxy auth usernames would show up if they were present in the Squid log line); mirror.bom.gov.au is resolved into 203.63.53.11; sent data is recorded as 0 bytes.

Note that hostnames that do not resolve at the time of log processing are printed out to `stderr`. You may wish to store these for later processing (in the case of temporary nameserver failure or similar).

Typical usage is:
```
squid.py < access.log.1 2>> unresolved.log > squid.flow.log
```

A command line argument to note is `--ident`. By adding `--ident 0` or `--ident 1` to the `squid.py` command line the script will either ignore or use ident/proxy auth data in the squid log (respectively).

`data_dump_to_routing_txt.pl` Convert a Melbourne Uni ITS `data_dump` file to a `routing.txt` file as used by TrafficWatch. A `data_dump` file can be obtained from `http://www-networks.its.unimelb.edu.au/itu/unimelb/` at the time of writing; `route-update.sh` shows how this script should be used. Please note that the `data_dump` contains perl code that will be executed, albeit in a restricted compartment. Hence, this script must NOT be run as root.

`web_purge_sessions.py` Purge old web sessions from the database. The exact expiry time is set in the configuration file. This script should be run in a cron job on the database server every 15-30 minutes.

`firewall_flush.sh` Purge iptables rules created by TrafficWatch

`firewall_genrules.py` Generate a set of iptables commands which create rules which mark packets passing through a firewall based upon whether their source MAC address is in the list provided. The rules generated live in the iptables mangle table, and set a configurable mark on each packet if it has come from an authenticated source. A small custom iptables module (mac_mask) is needed to allow efficient rulesets to be generated.

`firewall_maclist.py` Generate a list of MAC addresses that TrafficWatch considers to be allowed to access the internet. This may or may not exclude users who are over quota, depending on the policy set in the configuration file.

`mail_audit_report.py` Report a list of quota modifications made within the last 24 hours to an email address of your choice. This may be useful for auditing purposes

`mail_over_quota.py` Send a message to users who are either over quota or who have exceeded their self-configured limit. This should be run in a daily cron job after the log importation process is complete.

`mail_usage_report.py` Send a monthly usage report to all users of the TrafficWatch system. This should be run in a cron job once a month.

`squid_acl.sh` Produce a list of networks for which we think it is cheaper for squid to communicate with directly, rather than use the parent proxy cache. At the moment, this is everything up to and including Australian Domestic. Not strictly part of the TrafficWatch system, but a useful tool nonetheless.

`log_import.sh` Import all of the logs in `/var/lib/trafficwatch/logs`, as documented above. This script is designed to be run regularly as a cron job.

6 File Formats

There are two types of logfiles created and used by different components of the Trafficwatch system.

6.1 Traffic Data Logfile

This logfile is produced by `ipcap` and `squid.py` in order to record a flow of data. Each flow is a single line of text, with fields delimited by spaces. Any field that is for whatever reason left blank should contain a single `-` character, although it is only valid for certain fields to be left unfilled in this manner.

```
<timestamp> <username> <local MAC address> <local IP address> <remote IP address> <bytes
received> <bytes sent>
```

timestamp The number of seconds since the unix epoch, expressed as an integer

username The username of the user responsible for this flow (if known), or '`-`' otherwise

local MAC address The MAC address of the computer that is on the 'local' end of this flow if known, or '`-`' otherwise. The definition of precisely which endpoint of a flow is 'local' is configurable. The local endpoint is the computer for whose usage we are attempting to account.

local IP address The IP address of the local endpoint of a flow.

remote IP address The IP address of the remote endpoint of a flow. The remote endpoint is the end that is not local.

bytes received Number of bytes received by the local endpoint from the remote endpoint.

bytes sent Number of bytes sent by the local endpoint to the remote endpoint (can often be 0 if unknown).

A sample logline that might be produced by `squid.py` from an (unauthenticated) squid logfile is:

```
123347562 - - 203.28.241.35 128.250.6.182 1234 0
```

If the squid logfile had username information (eg. squid was configured to use proxy authentication or to make RFC1413 IDENT lookups), then we might see:

```
123347562 phawkins - 203.28.241.35 128.250.6.182 1234 0
```

Note that in both cases since we do not know the MAC address of the local connection endpoint, the field is replaced by a '`-`'.

A line that might be produced by ipcap could be:

```
1014161056 - 0:50:bf:16:dc:a7 203.28.241.52 203.30.98.194 1718 0
```

6.2 Traffic Authentication Logfile

This type of logfile is only produced automatically by the `ipcap` daemon by default, although there is no reason why other sources of authentication information could not also be used to produce this log file.

The format of the log file is very simple. It consists of a series of lines in chronological order, each of which link a MAC address or an IP address to a particular user, or a MAC address to an IP address at a given time. Each such matching is considered to continue to apply until a later log line is found that contradicts that information (eg. the same MAC address is matched to a different user or IP, or the same IP address is matched to a different user). Authentication information is never considered to 'expire' by itself, although in reality all such records will have a time-to-live no greater than the log rotation/importation period for authentication logs.

The format is as follows:

```
<timestamp> <username> <mac address> <IP address>
```

timestamp The number of seconds since the unix epoch, expressed as an integer

username The username of the user responsible for this flow (if known), or '`-`' otherwise

MAC address A MAC address of the computer that is 'local' and for which we are attempting to account.

local IP address The IP address of a local computer.

A typical log line produced by `ipcap` in its `mac.log` authentication log file would be:

```
1014156218 - 0:50:bf:16:dc:a7 203.28.241.52
```

A login server might produce lines indicating login and logout events (respectively) like:

```
1014156218 phawkins - 203.28.241.52
1014156219 - - 203.28.241.52
```

An IP address could be statically matched to a user by creating a log line like:

```
0 phawkins - 203.28.241.52
```

7 Files

Configuration files usually live in `/etc`. User-visible scripts usually live in `/usr/sbin`. TrafficWatch's internal scripts and data usually live in `/usr/share/trafficwatch/` and `/var/lib/trafficwatch`.

`config.py` This is the configuration file for the TrafficWatch system. Most things you will want to tweak live here.

8 Dependencies

Packages that are required for the TrafficWatch system are listed below. The names in brackets after each represent the corresponding Debian dependency.

- Linux kernel v2.4 on a firewall machine, plus small extra iptables module included in TrafficWatch - mac_mask
- PostgreSQL v7.x or later (postgresql, postgresql-clients)
- Python 2.2 or later (python)
- Python packages (python-egenix-mxdatetime, python-egenix-mxtools, python-psycopg, python-numeric)
- Python LDAP package, if LDAP support needed (python-ldap)
- ADNS and python-adns module (libadns1, libadns1-dev, python-adns)
- Biggles plotting package and libplot (python-biggles, libplot)
- Apache 1 or 2 with mod_python (apache and libapache-mod-python, or apache2 and libapache2-mod-python)

- Perl
- libpcap ($\geq$ v0.6 strongly recommended on linux)
- logrotate
- cron
- Various utilities (ssh, wget, rsync, awk, sh, gzip, etc.)

9 Definitions

Traffic Class A group of internet hosts all charged at one rate. Typical classes might include the VRN, AARNET, Australia, and International

Megabyte (MB) In this software, megabyte means 10^6 bytes, and gigabyte (GB) means 10^9 bytes. This is in accordance with the current IEEE standards on computer units. This may matter when setting cost rates.

10 Bugs

Some of the scripts could do with more documentation.

The quota/accounting logic is very ugly and should be redone.

The web pages are on the whole extremely ugly code.

There is a known issue with potentially invalid authentication information being produced in the MAC logfile of ipcap. We do not use this logfile ourselves so the problem has not been extensively debugged.

The rest is silence.