System Monitoring with Cfengine

Edition 2.2.9 for version 2.2.9

Mark Burgess
Faculty of Engineering, Oslo University College, Norway

This manual corresponds to CFENGINE Edition 2.2.9 for version 2.2.9 as last updated 11 September 2008.

1 Overview of concepts

Cfengine is a system for self-sufficient maintenance and repair of computer systems. Making cfengine as autonomous as possible, while at the same time guiding it by strict policies, is a goal of the cfengine project. To be autonomous, cfengine has to be able to observe the state of a computer system and make decisions about it. In other words, it has to be able to perform *self-monitoring*.

The idea that a configuration management system would perform monitoring breaks with the philosophy of most software for IT management. Traditionally, monitoring software has been completely independent of change management software – but here we are rolling them into one.

Let's try to explain. From an automation perspective, it makes no sense to separate monitoring from change management. A repair system needs to know when things are not working so that it can fix them. A monitoring system alone is useless unless it can schedule a repair. The process of maintenance is therefore much more efficient if these two complementary aspects of system management are integrated.

In this document we shall explain how to make the most of the features that cfengine offers you freely. In the future it will be possible to buy additional software that allows you to derive deep insights into system performance and behaviour from the cfengine software, but this document is limited to what you can achieve with only the free tools.

Since a lot of features have been added to cfengine lately, we'll assume that you are running versin 2.2.3 or later.

1.1 Monitoring features

Let's begin by thinking of the different ways in which cfengine can monitor the state of a host. Cfengine is no ordinary monitoring system. It does not simply generate warnings when certain values cross certain thresholds.

- Every policy rule that we code into cfengine leads to a check being performed on the policy rule. If there is a problem it might result in a warning, or in the problem being fixed automatically with not even a message.
- Some checks are implicit. For example, when ever cfengine scans through directories looking for files, it is on the look-out for different kinds of files that could be dangerous. It will warn about these, so in effect it is monitoring your filesystems.
- The daemon `cfenvd` collects observations silently in the background of your system and records the patterns of change that it sees taking place. It learns these using simple machine-learning techniques enabling cfengine to classify the state of your system in a number of different ways as it interacts with the network.
- Each time cfengine copies a file or executes a script, it measures the time this operation takes, and so it implicitly measures the overall performance of your system.

These are examples of the way in which cfengine collects information about the operation of each computer on which it runs. The question then is: how can we view this information and what is it good for?

1.2 Intrusion detection

A lot of companies perform monitoring first and foremost for security purposes. Cfengine does not care about the reason for monitoring, and it does not assume any special interpretation if it detects changes or events.

What is an intrusion or an attempted intrusion? This can be difficult to define. If someone tries to login at root once? If someone tries to login at root fifty times? Port scanning, SATAN or ISS scan? Someone trying a known security hole? These things are quite uncertain. The aim of an intrusion detection system is to detect events that can be plausibly connected to break-ins, hopefully while they are still in progress so that something can be done about them.

Intrusion detection is a special form of fault-diagnosis. Faults (in a security system) are events that are not supposed to happen, but the fact is that they do happen. As with all fault diagnosis systems, Intrusion Detection Systems (IDS) give the wrong answers from time to time. Because it is so difficult to define what intrusion actually means in a generic sense intrusion detection systems tend to err on the side of caution and report many false positives, i.e. false alarms.

One way of doing fault diagnosis is to compare a system to a working specification continuously. This is essentially what cfengine does with systems.

There are many approaches to intrusion detection. These go well beyond the scope of this document. Suffice it to say that cfengine is not meant to be an intrusion detection system specifically. One thing cfengine can detect however is *change*, and unexpected changes can sometimes be interpreted as tell-tale signs of something unauthorized happening. So there is scope for using cfengine as part of a host-based intrusion system. Cfengine does not, however, try to examine and diagnose network traffic.

1.3 Change Detection

Change monitoring is about detecting when stored data, or other measurable aspects of a computer system change. A change detection system is not normally concerned with the reason for a change, but if you are monitoring for change then we shall take it for granted somehow that you are expecting to find changes that you didn't plan for yourself.

1.3.1 Cryptographic checksums

The most important bulk of information on a computer is its filesystem data. Change detection for filesystems uses a technique made famous in the program Tripwire, which collects a "snapshot" of the system in the form of a database of file checksums (cryptographic hashes) and permissions and rechecked the system against this database at regular intervals. Tripwire examines files, and looks for change in their contents or their attributes. This is a very simple (even simplistic) view of change. If a legitimate change is made to the system, such a system responds to this as a potential threat. Databases must then be altered, or rebuilt.

1.3.2 Hashes or Digests

A cryptographic hash (also called a *digest*) is an algorithm that reads (digests) a file and computes a single number (the hash value) that is based on the contents. If so much as a

single bit in the file changes then the value of the hash will change. You can compute hash values manually, for example:

```
host$ openssl md5 cfengine-2.2.4a.tar.gz
MD5(cfengine-2.2.4a.tar.gz)= 6d2b31c4814354c65cbf780522ba6661
```

There are several kinds of hash function. The most common ones are MD5 and SHA1. Recently both of the algorithms that create these hashes have been superceded by the newer SHA2. Cfengine supports MD5 and SHA1 and it will support SHA2 as soon as the OpenSSL library supports an interface to the new algorithm.

1.3.3 Computing hashes

Cfengine has adopted part of the Tripwire model, but with a few provisoes. Tripwire assumes that all change is unauthorized and is bad. Cfengine cannot reasonably take this viewpoint. Cfengine expects systems to change dynamically, so it allows users to define a policy for what changes are considered to be okay.

Integrity checks on files whose contents are supposed to be static are a good way to detect tampering with the system, from whatever source. Running MD5 or SHA1 checksums of files regularly provides us with a way of determining even the smallest changes to file contents.

To use the checksum based change detection we first ask cfengine to collect MD5 hash data for specified files. Here is an excerpt from a cfengine configuration program that would check the /usr/local filesystem for file changes. Note that it excludes files such as log files that we therefore allow to change (log files are supposed to change):

```
files:

  /usr/local owner=root,bin,man
             mode=o-w          # check permissions separately
             r=inf
             checksum=best     # this switches on change detection
             action=warnall
             ignore=logs
             exclude=*.log

  # repeat for other files or directories
```

The first time we run this, cfengine collects data and treats all files as “unchanged”. It builds a database of the checksums. The next time the rule is checked, cfagent recomputes the checksums and compares the new values to the ‘reference’ values stored in the database. If no change has occurred, the two should match. If they differ, then the file as changed and a warning is issued.

```
cf:nexus: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
cf:nexus: SECURITY ALERT: Checksum (md5) for /etc/passwd changed!
cf:nexus: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

This message is designed to be visible. If you do not want the embracing rows of ‘!’ characters, then this control directive turns them off:

```
control:

 Exclamation  = ( off )
```

The next question to ask is: what happens if the change that was detected is actually okay (which is almost always the case). If you activate this option:

```
control:

 ChecksumUpdates = ( on )
```

Then, as soon as a change has been detected, the database is updated and the message will not be repeated. If this is set to `off`, which is the default, then warning messages will be printed each time the rule is checked.

New files are automatically detected, as they are not in the database. If you want to be notified when files are deleted, then set the option

```
control:

 ChecksumPurge = ( on )
```

1.3.4 file_hash_event_history

The file '`file_hash_event_history`' contains a separate text log of file changes.

```
Sun Sep  7 09:42:49 2008,/usr/local/sbin/cfMonitord
Sun Sep  7 09:42:49 2008,/usr/local/sbin/cfexecd
Sun Sep  7 09:42:49 2008,/usr/local/sbin/cfkey_1220624104_Fri_Sep__5_16_15_15_2008_.cfsaved
Sun Sep  7 09:42:49 2008,/usr/local/lib/libgd.so.2.0.0
Sun Sep  7 09:42:49 2008,/usr/local/lib/libpromises.la
Sun Sep  7 09:42:49 2008,/usr/local/lib/libcfengine.a
Sun Sep  7 09:42:49 2008,/usr/local/lib/libgd.la
Sun Sep  7 09:42:49 2008,/usr/local/lib/libgd.a
Sun Sep  7 09:42:49 2008,/usr/local/lib/libcfengine.la
Sun Sep  7 09:42:49 2008,/usr/local/lib/libpromises.a
```

1.3.5 Tamperproof data

Message digests are supposed to be unbreakable, tamperproof technologies, but of course everything can be broken by a sufficiently determined attacker. Suppose someone wanted to edit a file and alter the cfengine checksum database to cover their tracks. If they had broken into your system, this is potentially easy to do. How can we detect whether this has happened or not?

A simple solution to this problem is to use another checksum-based operation to copy the database to a completely different host. By using a copy operation based on a checksum value, we can also remotely detect a change in the checksum database itself.

Consider the following code:

```
# Neighbourhood watch

control:

   allpeers = ( SelectPartitionNeighbours(/path/cfrun.hosts,#,random,4) )

copy:

     /var/cfengine/checksum_digests.db

                     dest=/safekeep/chkdb_$(this)
                     type=checksum
                     server=$(allpeers)
                     inform=true          # warn of copy
                     backup=timestamp
                     define=tampering
```

```
alert:

 tampering::

      'Digest tampering detected on a peer'
```

It works by building a list of neighbours for each host. The function `SelectPartitionNeighbours` can be used for this. Using a file which contains a list of all hosts running cfengine (e.g. the '`cfrun.hosts`' file), we create a list of hosts to copy databases *from*. Each host in the network therefore takes on the responsibility to watch over its neighbours.

The copy rule attempts to copy the database to some file in a safekeeping directory. We label the destination file with `$(this)` which becomes the name of the server from which the file was collected. Finally, we backup any successful copies using a timestamp to retain a complete record of all changes on the remote host. Each time a change is detected, a copy will be kept of the old. The rule contains triggers to issue alerts and warnings too just to make sure the message will be heard.

In theory, all four neigbours should signal this change. If an attacker had detailed knowledge of the system, he or she might be able to subvert one or two of these before the change was detected, but it is unlikely that all four could be covered up. At any rate, this approach maximizes the chances of change detection.

Finally, in order to make this copy, you must, of course, grant access to the database in '`cfservd.conf`'.

```
# cfservd.conf

admit:

any::

  /var/cfengine/checksum_digests.db  mydomain.tld
```

Let us now consider what happens if an attacker changes a file an edits the checksum database. Each of the four hosts that has been designated a neighbour will attempt to update their own copy of the database. If the database has been tampered with, they will detect a change in the md5 checksums of the remote copy versus the original. The file will therefore be copied.

It is not a big problem that others have a copy of your checksum database. They cannot see the contents of your files from this. A potentially greater problem is that this configuration will unleash an avalanche of messages if a change is detected. This does make messages visible however.

1.4 Cfenvd, a learning agent

Cfengine employs sophisticated machine learning techniques to learn and compress information about the behaviour of each host into a small round-robin database. Unlike monitors based on RRD-tool, cfengine keeps data from several months' worth of measurements in

only a week's worth of space. It measures not only values but keeps statistical profiles of values too.[1]

Although '`cfenvd`' is not a compulsory part of cfengine, it is highly recommended that you run this daemon. It requires few resources and poses no vulnerability to the system. It will play an increasingly important role in future developments.

In cfengine 2.x, running '`cfenvd`' means that additional classes are automatically evaluated based on how the current state of the host compares to an average of all corresponding times of week that have occurred over the past 6-8 weeks. The analysis is accomplished by the '`cfenvd`' daemon, which continually updates a database of system averages and variances, which characterize "normal" behaviour. Every 2.5 minutes, the state of the system is examined and compared to the database values. Unlike a file change, numerical values are not just different, they have ordinality. The current state can be greater than or less than the norm.

Simply being greater than the norm in a particular measurement is not of itself very interesting. Random fluctuations in the patterns of behaviour mean that the values are changing a lot all the time, but these changes are not significant unless they become of the order of magnitude of a standard deviation above the mean. Cfengine takes this into account and classifies the current measured values on a scale of standard deviations about the currently applicable mean. For instance, it might set the following classes in '`cfagent`':

```
RootProcs_low_dev2
netbiosssn_in_low_dev2
smtp_out_high_anomalous
www_in_high_dev3
ftp_in_high_microanomaly
```

The first of these tells us that the number of root processes is two standard deviations below the average of past behaviour, which might be fortuitous, or might signify a problem, such as a crashed server. The WWW item tells us that the number of incoming connections is three standard deviations above average. The smtp item tells us that outgoing smtp connections are more than three standard deviations above average, perhaps signifying a mail flood. The setting of these classes is transparent to the user, but the additional information is only visible to the privileged owner of the cfengine work-directory, where the data are cached.

The term 'microanomaly' is used to describe two standard deviations above normal, when the delta of the change is less than the arbitray value of 5. This is a small number, and anomalies of these kinds are generally noise.

Any deviation from the mean value can be called an anomaly, but as we said above, anomalies do not necessarily have anything to do with security, and definitely do not need to have anything to do with system intrusions.

1.4.1 Interpreting anomalies

Simply specifying statistical number anomalies is not sufficient to provide well-honed anomaly characteristics. Cfengine tries to organize the information surrounding an anomaly first in terms of statistical significance and then only later in terms of event characteristics. There are too many events in which the numerical values exceed thresholds

[1] The '`cfenvd`' program serves two purposes: it is an anomaly detection engine and as a source of 'entropy' for generating random numbers, such as for encryption keys.

determined by an arbitrary policy. Other criteria are needed to pin down which anomalies are interesting and which are not. As a second level of policy filtering, cfengine provides a measure of the entropy of the source IP addresses of the measured data. A low entropy value means that most of the events came from only a few (or one) IP addresses. A high entropy value implies that the events were spread over many IP sources. These conditions are described by classes of the form:

```
entropy_www_in_high
entropy_smtp_in_low
```

Thus, for example, in the first case the class will be set if incoming traffic at the peak event of the last data sample was spread evenly over all the incoming addresses. Such an event indicates that the resource usage is not due to a single source (e.g. an attacker from a single location) but is evenly spread — perhaps just a coincidental anomaly. In the second case, the low entropy smtp traffic must come from one or two addresses and is more likely to be spam or an attack of some kind. These classes can be combined with the specific anomaly thresholds (see example below).

```
host% cfagent -p -v

[snip]

Defined Classes = ( 128_39_89 128_39_89_232
2001_700_700_3_20f_1fff_fe92_2cd3 32_bit Day28 Hr20 Hr20_Q2 January
Min15 Min15_20 Monday Q2 SuSE Yr2008 addr_ any cfengine_2 cfengine_2_2
cfengine_2_2_3a1 compiled_on_linux_gnu diskfree_normal_microanomaly
entropy_cfengine_in_low entropy_cfengine_out_low entropy_dns_in_low
entropy_dns_out_low entropy_ftp_in_low entropy_ftp_out_low
entropy_icmp_in_low entropy_icmp_out_low entropy_irc_in_low
entropy_irc_out_low entropy_misc_out_low entropy_netbiosdgm_in_low
entropy_netbiosdgm_out_low entropy_netbiosns_in_low
entropy_netbiosns_out_low entropy_netbiosssn_in_low
entropy_netbiosssn_out_low entropy_nfsd_in_low entropy_nfsd_out_low
entropy_smtp_in_low entropy_smtp_out_low entropy_ssh_in_low
entropy_ssh_out_low entropy_tcpack_in_low entropy_tcpack_out_low
entropy_tcpfin_in_low entropy_tcpfin_out_low entropy_tcpsyn_in_low
entropy_tcpsyn_out_low entropy_udp_in_low entropy_udp_out_low
entropy_www_in_low entropy_www_out_low entropy_wwws_in_low
entropy_wwws_out_low fe80__20f_1fff_fe92_2cd3 hio_no i686 ipv4_128
ipv4_128_39 ipv4_128_39_89 ipv4_128_39_89_232 iu_hio_no linux
linux_2_6_22_13_0_3_default linux_i686
linux_i686_2_6_22_13_0_3_default
linux_i686_2_6_22_13_0_3_default__1_SMP_2007_11_19_15_02_58_UTC
lsb_compliant messages_high_anomaly messages_high_dev1
messages_high_dev2 net_iface_eth0 net_iface_lo no rootprocs_high_dev1
rootprocs_high_dev2 slogans slogans_iu_hio_no suse suse_10 suse_10_3
suse_n/a )

[snip]
```

1.4.2 Entropy and its interpretation

Entropy is a word that has entered the popular consciousness in different ways. In physics entropy represents that amount if energy in a system of fixed temperature that has become unavailable for turning into useful work. It is commonly associated with the idea of *disorder*. In computer science the term entropy is often used in association with cryptographic keys and passwords. Bad passwords have 'insufficient entropy', which we understand to mean

too little content in some sense. All of these interpretations are correct but they can be misleading.

Entropy is actually a measure of how spread-out a signal is. In the case of thermodynamics, it tells us how spread out heat is. An engine or refrigerator needs some parts of the system to be hot and some to be cold in order to drive work around the system. If the entropy is too high, the temperature is spread out and there is nothing to drive work. Disorder could mean that a beautiful compact crystal dissolves into solution and is spread out evenly. In a password, low entropy means that all of the symbols in the password are close together in the alphabet. A high entropy password would show significant variation.

Cfengine measures all kinds of different signals, from network connection numbers to temperature perhaps. It does not measure the entropy of these values specifically. However, for signals of network origin, it measures the entropy of the addresses from which the connections arrive.

If all connections come from a single address, the entropy is low (0%). If each connection comes from a different address the entropy is maximal (100%). This information can be useful when interpreting an anomaly. A sudden increase in web connections from a single location might be an attack (or a search-engine), while a sudden burst from many different sources could be a coincidence (or a distributed attack).

1.5 Cfagent collected data

Cfagent itself is able to collect data about the performance of a host during its normal operation. Many of these data can be extracted using the '`cfshow`' command.

1.5.1 Last seen database

The last-seen database is maintained automatically by cfagent and cfservd. Each time one of these components successfully connects to another host it records this information in its database. A plus sign is used if '`cfagent`' instigated contact to '`cfservd`' another host itself. A negative sign means that '`cfservd`' was contacted by a external '`cfagent`'.

```
slogans:~ # cfshow --last-seen
IP +           nexus.iu.hio.no    128.39.89.10   [Tue Aug 21 10:40] not seen for (3851.73) hrs, Av 0.03 +
IP +        eternity.iu.hio.no   128.39.89.233   [Tue Aug 21 10:40] not seen for (3851.72) hrs, Av 0.03 +
IP +        eternity.iu.hio.no   ...:feeb:5d08   [Mon Jan 28 21:20] not seen for (0.06) hrs, Av 0.08 +/-
IP +           nexus.iu.hio.no   ...:fe9b:dd4a   [Mon Jan 28 21:20] not seen for (0.06) hrs, Av 0.08 +/-
IP +            cube.iu.hio.no   ...:fe93:6723   [Mon Jan 28 21:20] not seen for (0.06) hrs, Av 0.08 +/-
```

1.5.2 Intermittency times

Intermittency times are recorded automatically by cfagent. They are are complementary to the last-seen times above. Essentially cfengine records the variability in last-seen times and calculates their entropy. This statistic only means something if cfagent regularly contacts a remote server. If contact is regular, then the entropy of the last-seen times will be low, as the connection time will always be the same. However, if a host connection is unreliable for whatever reason, the entropy will increase. Cfengine translates these computations into a percentage which is show in a `FriendStatus(0)` alert.

Here is a result after a series of power outages in a lab at Oslo University College.

```
cf:nexus: Host nexus2.iu.hio.no i.e. 2001:700:700:3 last hailed us  [Tue Jan 22 12:30] (overdue by 303 mi
cf:nexus: i.e. (5.60) hrs ago, Av 0.53  0.14 hrs
```

```
cf:nexus: FriendStatus reports the intermittency of PH427LINUX9.iu.hio.no above 50% (scaled entropy units
cf:nexus: FriendStatus reports the intermittency of PH427LINUX6.iu.hio.no above 50% (scaled entropy units
cf:nexus: FriendStatus reports the intermittency of 2001:700:700:4:20c:29ff:fea8:94ba above 50% (scaled e
cf:nexus: FriendStatus reports the intermittency of 2001:700:700:4:20c:29ff:fef4:b527 above 50% (scaled e
cf:nexus: FriendStatus reports the intermittency of 2001:700:700:4:21a:a0ff:fea3:3d53 above 50% (scaled e
cf:nexus: FriendStatus reports the intermittency of 2001:700:700:4:21a:a0ff:fea3:c00 above 50% (scaled en
cf:nexus: FriendStatus reports the intermittency of 2001:700:700:4:21a:a0ff:fea3:cb3d above 50% (scaled e
```

1.5.3 Performance

The performance database is maintained automatically by cfagent. It records the average time taken to complete certain jobs (generally copy jobs and scripts since other tasks happen so fast that they are not measurable).

```
slogans:~ # cfshow --performance
( 1.7713 mins  Thu Oct 18 10:50) Av  1.7713 +/-  0.0000 for Copy(cube:/var/cfengine > /cfengine/cube)
( 0.2428 mins  Mon Jan 28 21:20) Av  0.2437 +/-  0.0082 for Copy(cube:/var/cfengine/cfnerves > /cfengine/
( 0.0928 mins  Fri Aug 17 10:50) Av  0.0928 +/-  0.0000 for Copy(eternity:/var/cfengine > /cfengine/etern
( 0.1341 mins  Mon Jan 28 21:20) Av  0.1411 +/-  0.0154 for Copy(eternity:/var/cfengine/cfnerves > /cfeng
( 2.1452 mins  Thu Oct 18 10:52) Av  2.1452 +/-  0.0000 for Copy(localhost:/var/cfengine > /cfengine/loca
( 0.0006 mins  Mon Jan 28 21:20) Av  0.0009 +/-  0.0009 for Copy(localhost:/var/cfengine/cfnerves > /cfen
( 0.5779 mins  Thu Oct 18 10:50) Av  0.5779 +/-  0.0000 for Copy(nexus:/var/cfengine > /cfengine/nexus)
( 0.1920 mins  Mon Jan 28 21:20) Av  0.2365 +/-  0.1369 for Copy(nexus:/var/cfengine/cfnerves > /cfengine
( 0.4611 mins  Mon Oct 22 08:20) Av  0.4348 +/-  0.0249 for Exec(/usr/local/sbin/cfbrain)
( 0.0005 mins  Thu Oct 18 18:40) Av  0.0005 +/-  0.0003 for Exec(/usr/local/sbin/cfcore)
( 0.0053 mins  Mon Jan 28 21:20) Av  0.0058 +/-  0.0007 for Exec(/usr/sbin/ntpdate cube.iu.hio.no)
( 0.0160 mins  Mon Jan 28 21:20) Av  0.0336 +/-  0.0197 for Exec(/var/cfengine/bin/cfbrain)
( 0.0049 mins  Mon Jan 28 21:20) Av  0.0041 +/-  0.0008 for Exec(/var/cfengine/bin/cfenvgraph -o cfnerves
( 0.0066 mins  Mon Jan 28 21:01) Av  0.0075 +/-  0.0019 for Exec(/var/cfengine/bin/cfenvgraph -s -o cfner
```

1.5.4 Disk scans

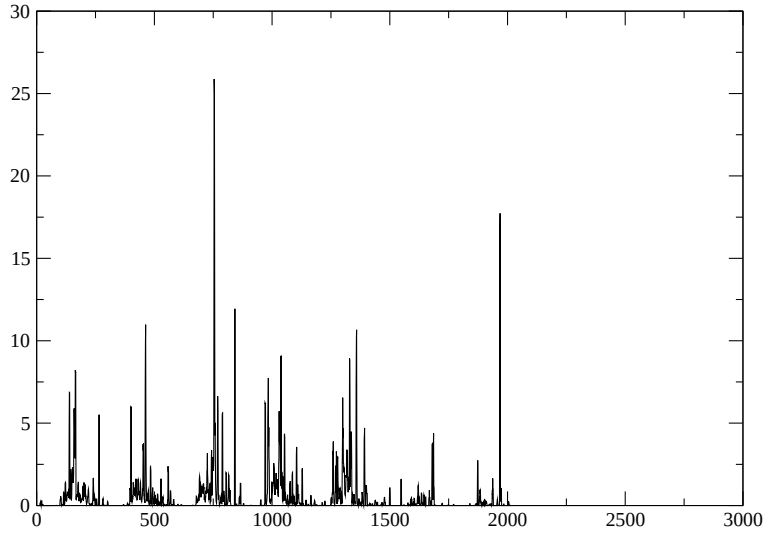
A measure that cfagent does not collect automatically is a scan of the arrival process of disk changes. This is a highly disk intensive scan so you should not perform this more than once per week. It is also debatable whether it is worse running more often than once per month. The distribution will not change signficantly and the information provided does not change very often.

```
disks:

  /filesystem
      scanarrivals=true
```

The arrival process determines a best-guess inter-arrival time distribution of files for a disk, which provides information about user behaviour and possible performance bottlenecks. The interpretation of this information is rather complicated and goes beyond the

scope of this document. If a scan has been made, '`cfenvgraph`' will output the scan data along with other graph data.



2 Detection of events

2.1 Anomaly Detection

There is no system available in the world today which can claim to detect and classify the functioning state of a computer system. Cfengine does not attempt to provide a "product" solution to this problem; rather it incorporates a framework, based on the current state of knowledge, for continuing research into this issue. In version 2.x of cfengine, an extra daemon '`cfenvd`' is used to collect statistical data about the recent history of each host (approximately the past two months), and classify it in a way that can be utilized by the cfengine agent.

The daemon may simply be started, with no arguments:

```
cfenvd
```

and it proceeds to collect data and work autonomously, without further supervision. The cf-environment daemon is meant be trivial to use. The current long-term data recorded by the daemon are: number of users, number of root processes, number of non-root processes, percentage disk full for root disk, number of incoming and outgoing sockets for netbiosns, netbiosdgm, netbiosssn, irc, cfengine, nfsd, smtp, www, ftp, ssh and telnet. These data have been studied previously, and their behaviour is relatively well understood. In future versions, it is expected to extend this repertoire, as more research is done.

The use of the daemon will not be reliable until about six to eight weeks after installing and running it, since a suitable training period is required to build up enough data for stable characterization. The daemon automatically adapts to the changing conditions, but has a built-in inertia which prevents anomalous signals from being given too much credence. Persistent changes will gradually change the 'normal state' of the host over an interval of a few weeks. Unlike some systems, cfengine's training period never ends. It regards normal behaviour as a relative concept, which has more to do with local stability than global constancy.

The final size of the database is approximately 2MB. Measurements are taken every five minutes (approximately). This interval is based on auto-correlation times measured for networked hosts in practice.

Cfenvd sets a number of classes in cfengine which describe the current state of the host in relation to its recent history. The classes describe whether a parameter is above or below its average value, and how far from the average the current value is, in units of the standard-deviation (see above). This information could be utilized to arrange for particularly resource-intensive maintenance to be delayed until the expected activity was low.

2.2 Starting with anomaly detection

Try importing the following file:

```
###
#
# BEGIN cf.environ
#
###
```

```
#
# Just a test for responses to measured anomalies
#

classes:

 anomaly_hosts = ( myhost1 myhost2 )


################################################

alerts:

  nfsd_in_high_dev2::

  "High NFS server access rate 2dev at $(host)/$(env_time)
current value $(value_nfsd_in) av $(average_nfsd_in) pm
$(stddev_nfsd_in)"

  ShowState(incoming.nfs)


 # ROOT PROCS

  anomaly_hosts.RootProcs_high_dev2::

   "RootProc anomaly high 2 dev on $(host)/$(env_time) current value
$(value_rootprocs) av $(average_rootprocs) pm $(stddev_rootprocs)"
  ShowState(procs)



 # USER PROCS

 anomaly_hosts.UserProcs_high_dev2::

   "UserProc anomaly high 2 dev on $(host)/$(env_time) current
value $(value_userprocs) av $(average_userprocs) pm $(stddev_userprocs)"
   ShowState(procs)


 anomaly_hosts.UserProcs_high_anomaly::

   "UserProc anomaly high 3 dev!! on $(host)/$(env_time)"
   ShowState(procs)


 # WWW IN

 # This happens too often
 # anomaly_hosts.www_in_high_dev2::
 #

 entropy_www_in_high.anomaly_hosts.www_in_high_anomaly::

   "HIGH ENTROPY Incoming www anomaly high anomaly dev!!
on $(host)/$(env_time) - current value $(value_www_in)
```

```
av $(average_www_in) pm $(stddev_www_in)"

   ShowState(incoming.www)

 entropy_www_in_low.anomaly_hosts.www_in_high_anomaly::

   "LOW ENTROPY Incoming www anomaly high anomaly dev!! on
$(host)/$(env_time) - current value $(value_www_in) av
$(average_www_in) pm $(stddev_www_in)"

   ShowState(incoming.www)


 # SMTP IN

 entropy_smtp_in_high.anomaly_hosts.smtp_in_high_dev2::

   "HIGH ENTROPY Incoming smtp anomaly high 2 dev on $(host)/$(env_time)"

 entropy_smtp_in_high.anomaly_hosts.smtp_in_high_anomaly::

   "HIGH ENTROPY Incoming smtp anomaly high anomaly !! on $(host)/$(env_time)"


 entropy_smtp_in_low.anomaly_hosts.smtp_in_high_dev1::

   "LOW ENTROPY Incoming smtp anomaly high 1 dev on $(host)/$(env_time)
 current value $(value_smtp_in) av  $(average_smtp_in) pm $(stddev_smtp_in)"

   ShowState(incoming.smtp)


 entropy_smtp_in_low.anomaly_hosts.smtp_in_high_dev2::

   "LOW ENTROPY Incoming smtp anomaly high 2 dev on $(host)/$(env_time)
current value $(value_smtp_in) av  $(average_smtp_in) pm $(stddev_smtp_in)"

   ShowState(incoming.smtp)


 entropy_smtp_in_low.anomaly_hosts.smtp_in_high_anomaly::

   "LOW ENTROPY Incoming smtp anomaly high anomaly !! on $(host)/$(env_time)
current value $(value_smtp_in) av  $(average_smtp_in) pm $(stddev_smtp_in)"

   ShowState(incoming.smtp)


 # SMTP OUT

 anomaly_hosts.smtp_out_high_dev2::

   "Outgoing smtp anomaly high 2 dev on $(host)/$(env_time) current value
$(value_smtp_out) av  $(average_smtp_out) pm $(stddev_smtp_out)"

   ShowState(outgoing.smtp)
```

```
 anomaly_hosts.smtp_out_high_anomaly::

   "Outgoing smtp anomaly high anomaly dev!! on $(host)/$(env_time)
current value $(value_smtp_out) av $(average_smtp_out) pm $(stddev_smtp_out)"

   ShowState(outgoing.smtp)


 # SAMBA

 anomaly_hosts.netbiosssn_in_high_dev2::

   "SAMBA access high 2 on $(host)/$(env_time) current value
$(value_netbiosssn_in) av $(average_netbiosssn_in) pm $(stddev_netbiosssn_in)"

   ShowState(incoming.netbiosssn)

#############################################################################

###
#
# END cf.environ
#
###
```

A sample of output generated by this file shows the current value of the quantity and a summary of the highest values during the last 40 minutes. Notice the low entropy anomaly, meaning a highly concentrated signal from a single source.

```
cf:cube: LOW ENTROPY Incoming www anomaly high anomaly dev!! on cube/Fri Feb 20 19:57:23 2004 - current v
cf:cube: ------------------------------------------------------------------------------
cf:cube: In the last 40 minutes, the peak state was:
cf:cube: ( 1) tcp        0      0 128.39.74.16:80         157.158.24.40:4049      TIME_WAIT
cf:cube: ( 2) tcp        0      0 128.39.74.16:80         157.158.24.40:3796      TIME_WAIT
cf:cube: ( 3) tcp        0      0 128.39.74.16:80         157.158.24.40:3544      TIME_WAIT
cf:cube: ( 4) tcp        0      0 128.39.74.16:80         157.158.24.40:4063      TIME_WAIT
cf:cube: ( 5) tcp        0      0 128.39.74.16:80         157.158.24.40:4035      TIME_WAIT
cf:cube: ( 6) tcp        0      0 128.39.74.16:80         157.158.24.40:3782      TIME_WAIT
cf:cube: ( 7) tcp        0      0 128.39.74.16:80         157.158.24.40:3530      TIME_WAIT
cf:cube: ( 8) tcp        0      0 128.39.74.16:80         157.158.24.40:3824      TIME_WAIT
cf:cube: ( 9) tcp        0      0 128.39.74.16:80         157.158.24.40:3572      TIME_WAIT
cf:cube: (10) tcp        0      0 128.39.74.16:80         157.158.24.40:4091      TIME_WAIT
cf:cube: (11) tcp        0      0 128.39.74.16:80         157.158.24.40:3839      TIME_WAIT
cf:cube: (12) tcp        0      0 128.39.74.16:80         157.158.24.40:3810      TIME_WAIT
cf:cube: (13) tcp        0      0 128.39.74.16:80         157.158.24.40:4077      TIME_WAIT
cf:cube: (14) tcp        0      0 128.39.74.16:80         157.158.24.40:3993      TIME_WAIT
cf:cube: (15) tcp        0      0 128.39.74.16:80         157.158.24.40:3740      TIME_WAIT
cf:cube: (16) tcp        0      0 128.39.74.16:80         157.158.24.40:3712      TIME_WAIT
cf:cube: (17) tcp        0      0 128.39.74.16:80         157.158.24.40:3979      TIME_WAIT
cf:cube: (18) tcp        0      0 128.39.74.16:80         157.158.24.40:3726      TIME_WAIT
cf:cube: (19) tcp        0      0 128.39.74.16:80         157.158.24.40:4021      TIME_WAIT
cf:cube: (20) tcp        0      0 128.39.74.16:80         157.158.24.40:3768      TIME_WAIT
cf:cube: (21) tcp        0      0 128.39.74.16:80         157.158.24.40:3516      TIME_WAIT
cf:cube: (22) tcp        0      0 128.39.74.16:80         80.203.17.11:11487      ESTABLISHED
cf:cube: (23) tcp        0      0 128.39.74.16:80         157.158.24.40:4007      TIME_WAIT
cf:cube: (24) tcp        0      0 128.39.74.16:80         157.158.24.40:3754      TIME_WAIT
cf:cube: (25) tcp        0      0 128.39.74.16:80         66.196.72.28:6545       TIME_WAIT
```

```
cf:cube: (26) tcp        0      0 128.39.74.16:80         157.158.24.40:3923      TIME_WAIT
cf:cube: (27) tcp        0      0 128.39.74.16:80         157.158.24.40:3670      TIME_WAIT
cf:cube: (28) tcp        0      0 128.39.74.16:80         80.202.77.107:1567      TIME_WAIT
cf:cube: (29) tcp        0      0 128.39.74.16:80         157.158.24.40:4189      TIME_WAIT
cf:cube: (30) tcp        0      0 128.39.74.16:80         157.158.24.40:3909      TIME_WAIT
cf:cube: (31) tcp        0      0 128.39.74.16:80         157.158.24.40:3656      TIME_WAIT
cf:cube: (32) tcp        0      0 128.39.74.16:80         157.158.24.40:3698      TIME_WAIT
cf:cube: (33) tcp        0      0 128.39.74.16:80         157.158.24.40:3965      TIME_WAIT
cf:cube: (34) tcp        0      0 128.39.74.16:80         80.202.77.107:1568      TIME_WAIT
cf:cube: (35) tcp        0      0 128.39.74.16:80         157.158.24.40:3937      TIME_WAIT
cf:cube: (36) tcp        0      0 128.39.74.16:80         157.158.24.40:3684      TIME_WAIT
cf:cube: (37) tcp        0      0 128.39.74.16:80         157.158.24.40:4203      TIME_WAIT
cf:cube: (38) tcp        0      0 128.39.74.16:80         157.158.24.40:3951      TIME_WAIT
cf:cube: (39) tcp        0      0 128.39.74.16:80         157.158.24.40:3600      TIME_WAIT
cf:cube: (40) tcp        0      0 128.39.74.16:80         157.158.24.40:4119      TIME_WAIT
cf:cube: (41) tcp        0      0 128.39.74.16:80         157.158.24.40:3867      TIME_WAIT
cf:cube: (42) tcp        0      0 128.39.74.16:80         157.158.24.40:3614      TIME_WAIT
cf:cube: (43) tcp        0      0 128.39.74.16:80         157.158.24.40:3586      TIME_WAIT
cf:cube: (44) tcp        0      0 128.39.74.16:80         157.158.24.40:4105      TIME_WAIT
cf:cube: (45) tcp        0      0 128.39.74.16:80         157.158.24.40:3853      TIME_WAIT
cf:cube: (46) tcp        0      0 128.39.74.16:80         157.158.24.40:4147      TIME_WAIT
cf:cube: (47) tcp        0      0 128.39.74.16:80         157.158.24.40:3895      TIME_WAIT
cf:cube: (48) tcp        0      0 128.39.74.16:80         157.158.24.40:3642      TIME_WAIT
cf:cube: (49) tcp        0      0 128.39.74.16:80         80.213.238.106:4318     FIN_WAIT2
cf:cube: (50) tcp        0      0 128.39.74.16:80         80.213.238.106:4319     TIME_WAIT
cf:cube: (51) tcp        0      0 128.39.74.16:80         157.158.24.40:4133      TIME_WAIT
cf:cube: (52) tcp        0      0 128.39.74.16:80         157.158.24.40:3881      TIME_WAIT
cf:cube: (53) tcp        0      0 128.39.74.16:80         157.158.24.40:3628      TIME_WAIT
 {
 DNS key: 157.158.24.40 = arm.iele.polsl.gliwice.pl (47/53)
 DNS key: 80.203.17.11 = 11.80-203-17.nextgentel.com (1/53)
 DNS key: 66.196.72.28 = j3118.inktomisearch.com (1/53)
 DNS key: 80.202.77.107 = 107.80-202-77.nextgentel.com (2/53)
 DNS key: 80.213.238.106 = ti100710a080-3690.bb.online.no (2/53)
 -
 Frequency: 157.158.24.40    |*********************************************  (47/53)
 Frequency: 80.203.17.11     |*  (1/53)
 Frequency: 66.196.72.28     |*  (1/53)
 Frequency: 80.202.77.107    |**  (2/53)
 Frequency: 80.213.238.106   |**  (2/53)
 }
 -
 Scaled entropy of addresses = 12.7 %
 (Entropy = 0 for single source, 100 for flatly distributed source)
 -
cf:cube: ------------------------------------------------------------------------------
cf:cube: State of incoming.www peaked at Fri Feb 20 19:57:23 2004
```

Another example of a high entropy smtp (possible distributed spam operation in progress):

```
cf:nexus: HIGH ENTROPY Incoming smtp anomaly high 2 dev on nexus/Sat Aug  6 14:29:58 2005
cf:nexus: ------------------------------------------------------------------------------
cf:nexus: In the last 40 minutes, the peak state was q = 25:
 {
 DNS key: 81.218.96.62 = bzq-218-96-62.red.bezeqint.net (1/25)
```

```
 DNS key: 85.152.128.208 = cm-85-152-128-208.telecable.es (1/25)
 DNS key: 82.40.141.102 = 82-40-141-102.cable.ubr04.uddi.blueyonder.co.uk (1/25)
 DNS key: 68.189.49.33 = 68-189-49-33.dhcp.rdng.ca.charter.com (1/25)
 DNS key: 61.47.218.167 = 61.47.218.167 (1/25)
 DNS key: 211.170.184.229 = 211.170.184.229 (2/25)
 DNS key: 64.65.134.186 = static-64-65-134-186.dsl.pdx.eschelon.com (1/25)
 DNS key: 61.9.82.105 = 61.9.82.105.mozcom.net (1/25)
 DNS key: 219.234.19.166 = 219.234.19.166 (1/25)
 DNS key: 68.60.199.206 = pcp07641774pcs.calhun01.ga.comcast.net (1/25)
 DNS key: 82.216.163.149 = ip-149.net-82-216-163.suresnes3.rev.numericable.fr (1/25)
 DNS key: 84.9.43.170 = host-84-9-43-170.bulldogdsl.com (1/25)
 DNS key: 84.59.55.171 = dsl-084-059-055-171.arcor-ip.net (1/25)
 DNS key: 200.104.102.141 = pc-141-102-104-200.cm.vtr.net (1/25)
 DNS key: 85.152.225.126 = cm-85-152-225-126.telecable.es (1/25)
 DNS key: 218.81.136.17 = 218.81.136.17 (1/25)
 DNS key: 83.84.225.251 = 5354E1FB.cable.casema.nl (1/25)
 DNS key: 60.198.145.92 = 60-198-145-92.static.tfn.net.tw (1/25)
 DNS key: 61.53.185.218 = 61.53.185.218 (1/25)
 DNS key: 81.53.86.235 = ANantes-154-1-63-235.w81-53.abo.wanadoo.fr (1/25)
 DNS key: 58.142.251.14 = 58.142.251.14 (1/25)
 DNS key: 220.234.173.3 = 220.234.173.3 (1/25)
 DNS key: 219.251.118.195 = 219.251.118.195 (1/25)
 DNS key: 68.44.158.165 = pcp04364785pcs.glstrt01.nj.comcast.net (1/25)
 -
 Frequency: 211.170.184.229  |**        (2/25)
 Frequency: 68.44.158.165    |*         (1/25)
 Frequency: 219.251.118.195  |*         (1/25)
 Frequency: 220.234.173.3    |*         (1/25)
 Frequency: 58.142.251.14    |*         (1/25)
 Frequency: 81.53.86.235     |*         (1/25)
 Frequency: 61.53.185.218    |*         (1/25)
 Frequency: 60.198.145.92    |*         (1/25)
 Frequency: 83.84.225.251    |*         (1/25)
 Frequency: 218.81.136.17    |*         (1/25)
 Frequency: 85.152.225.126   |*         (1/25)
 Frequency: 200.104.102.141  |*         (1/25)
 Frequency: 84.59.55.171     |*         (1/25)
 Frequency: 84.9.43.170      |*         (1/25)
 Frequency: 82.216.163.149   |*         (1/25)
 Frequency: 68.60.199.206    |*         (1/25)
 Frequency: 219.234.19.166   |*         (1/25)
 Frequency: 61.9.82.105      |*         (1/25)
 Frequency: 64.65.134.186    |*         (1/25)
 Frequency: 61.47.218.167    |*         (1/25)
 Frequency: 68.189.49.33     |*         (1/25)
 Frequency: 82.40.141.102    |*         (1/25)
 Frequency: 85.152.128.208   |*         (1/25)
 Frequency: 81.218.96.62     |*         (1/25)
 }
 -
 Scaled entropy of addresses = 4.0 %
 (Entropy = 0 for single source, 100 for flatly distributed source)
 -
cf:nexus: ------------------------------------------------------------------------------
cf:nexus: State of incoming.smtp peaked at Sat Aug  6 14:29:58 2005
cf:nexus: HIGH ENTROPY Incoming smtp anomaly high anomaly !! on nexus/Sat Aug  6 14:29:58 2005
```

2.3 cfenvgraph

The data revealed in the alerts above conceal the basic patterns that underlie the detection of unusual signals. However, this information can be extremely valuable to an analyst. It shows that patterns of usage of a system over weeks. It shows stable patterns that can emerge in the use of particular resources and it shows us when there are no discernable patterns of usage.

Knowing the patterns of usage for a system can be important when deciding whether a host can cope with the load placed upon it. When is a client likely to receive good service, or slow service. The data can have implications for Service Level Agreements (SLA).

The data underlying the learned patterns of behaviour can be shown using the additional tools provided. The '`cfenvgraph`' command can use used to dump a graph of averages for visual inspection of the normal state database. The format of the file is

```
t,y_1,y_2,y_3...
```

which can be viewed using '`gnuplot`' or '`xgmr`' or other graphical plotting program. This would allow the policy-maker to see what is likely to be a good time for such work (say 06:00 hours), and then use this time for the job, unless an anomalous load is detected.

The cfenvgraph command is used to extract data from the database used by the cfenvd environment daemon.

```
cfenvgraph -f filename.db [-r -T -t -s -e]
```

The command normally generates two files with format

```
t, y_1, y_2, y_3, y_4...
```

in a sub-directory of the current directory '`cfenvgraphs-snapshot`' (or '`cfenvgraphs-`'*TIMESTAMP* if '`-T`' is used).

The files are called

```
cfenv-average
cfenv-stddev
```

and contain, respectively, the weighted average values of all the recorded data and the square-root of the weighted variances with respect to the averages. Data are weighted in such a way that older values are gradually deprecated, becoming irrelevant after about two months.

Normally the vertical scale of each graph is scaled so that each line has a maximum value of 1 and a minimum value of 0, this allows all the lines to be seen in maximum detail. However, this makes it difficult to see the absolute values of the lines. With the '`-n`' option, no scaling is performed and true values are plotted.

The complete data span a one-week period, and the daily rhythm of the system may normally be viewed as a number of peaks, one per day.

The options are:

'`--help (-h)`'
: List command options

'`--file (-f)`'
: Specifiy file to plot.

'`--titles (-t)`'
: If the '`-t`' option is given, comments are generated at the start of the file which describe the columns. These are in a format understood by '`vvgraph`' as title/label data.

‘--timestamps (-T)’
: If the ‘-T’ option is given, the output filenames are time-stamped with the current time, in order to give a unique name.

‘--resolution (-r)’
: If the ‘-r’ option is given then high resolution data are generated (five minute resolution), otherwise data are averaged over periods of one hour to generate simpler and smoother graphs.

‘--separate (-s)’
: If the ‘-s’ option is given, cfenvgraph generates separate files for each metric, in the format

```
t,y,dy
```

where dy is the height of a vertical error-bar. This set of graphs combines the average with the standard-deviation. (Note that the error bars show the standard-deviation, and not the standard error of the mean i.e. stddev/sqrt(N)); the latter has no obvious meaning here. If ‘-e’ is specified, then error bars are omitted.

‘--no-error-bars (-e)’
: No error bars are plotted.

‘--no-scaling (-n)’
: The graphs are not scaled, so that (min,max) is mapped onto the interval (0,1).

‘--erasehistory (-E)’
: Wipes out the average and variance of the named observation categories from the learning database.

Note that the values printed for sockets always look higher than they should for highly active services. This is because even those sockets which are in CLOSE_WAIT are counted. This is the correct way to determine a normal state based on the recent past. It is a local averaging performed by the kernel. If one counts only those connections which are currently active, one gets a distorted view of activity with a 5-minute sample rate. To measure more often than this would place unacceptably high load on the system.

Graphs may be viewed in ‘vvgraph’, ‘xmgr’ (used in the pictures above) or ‘gnuplot’, or other graphical viewer. These graphs are not meant for continuous viewing. The data are averages, not time-series.

For example, with gnuplot

```
host$ cfenvgraph -s
host$ gnuplot
gnuplot> plot "www-in.cfenv" with errorbars
gnuplot> plot "www-in.cfenv" with lines
```

The new version of xmgr is called xmgrace. It can be invoked as follows:

```
host$ xmgrace -nxy cfenv-averages
host$ xmgrace  rootprocs.cfenv
host$ xmgrace -settype xydy  rootprocs.cfenv
host$ xmgrace -settype xydy  rootprocs.cfenv -hardcopy -hdevice JPEG
```

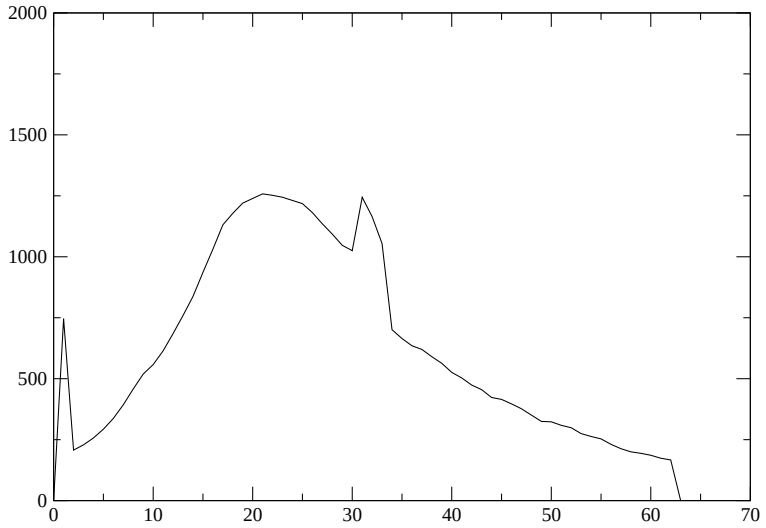
If you see the error "Strings are not allowed", it might be because some "nan" values have come into the text file.

2.4 Fluctuation profiles

Any model of fluctuating values is based on the idea that the changing signal has a basic separation of signal and noise. The variability of the signal is generally characterized by a probability distribution which often peaks about the mean value. Some tools and many papers assume that the distribution of fluctuations is Gaussian. This is almost never the case in real computer systems.

To see what the distribution of fluctuations about the mean looks like, you can plot the distibution files.

```
host$ xmgrace loadavg.distr
```



2.5 cfbrain

An additional tool for purchase will be available within the next year to extract the maximum use from the data learned by cfengine on different hosts. Even distributed systems like cfengine can benefit from exchanging and comparing data, just as the brain can add a whole new level of cognition to the collation of data from autonomously firing neurons.

Cfbrain is a project that will allow you to see beyond the simple traces and anomaly messages that a system generates often spuriously. It will go beyond the current state of the art and provide analyses that will provide significant information about the state of a data centre.

2.6 Anomaly Class Summary

When cfengine detects an anomaly, it classified the current statistical state of the system into a number of classes.

Cfengine classifies anomalies by whether the currently measured state of the system is higher or lower than the average for the current time of week. The amount of deviation is based on an estimate of the 'standard deviation'. The precise definition of the average and standard deviations is complex, and is discussed in the paper "M. Burgess, Probabilistic anomaly detection in distributed computer networks", (submitted to Science of Computer Programming, and available on the web).

The list of measured attributes is currently fixed to the following:

The first part of the string is from the list:

```
Users
RootProcs
UserProcs
DiskFree
LoadAvg
```

Socket counts of network services distinguish between incoming and outgoing sockets (to a service or from a client).

`netbiosns`
: Registers traffic to/from port 137.

`netbiosdgm`
: Registers traffic to/from port 138.

`netbiosssn`
: Registers traffic to/from port 139.

`irc` Registers traffic to/from port 194.

`cfengine` Registers traffic to/from port 5308.

`nfsd` Registers traffic to/from port 2049.

`smtp` Registers traffic to/from port 25.

`www` Registers traffic to/from port 80.

`ftp` Registers traffic to/from port 21.

`ssh` Registers traffic to/from port 22.

`wwws` Registers traffic to/from port 443.

If you have tcpdump program installed in a standard location, then `cfenvd -T` collects data about the network flows to your host.

`icmp` Traffic belonging to the ICMP protocol (ping etc).

`dns` Traffic to port 53, the Domain Name Service (usually a special case of UDP).

`udp` Miscellaneous UDP traffic that is not related to DNS.

`tcpsyn` Registers TCP packets with SYN flag set.

`tcpack` Registers TCP packets with ACK flag set.

`tcpfin` Registers TCP packers with FIN flag set.

`misc` Registers all other packets, not covered above.

When it has accurate knowledge of statistics, '`cfenvd`' classifies the current state into 3 levels:

`normal` means that the current level is less than one standard deviation above normal.

`dev1` means that the current level is at least one standard deviation about the average.

`dev2` means that the current level is at least two standard deviations about the average.

`anomaly` means that the current level is more than 3 standard deviations above average.

Each of these charaxterizations assumes that there are good data available. The '`cfenvd`' evaluates its data and decides whether or not the data are too noisy to be really useful. If the data are too noisy but the level *appears* to be more than two standard deviations above aaverage, then the category `microanomaly` is used.

Here are some example classes:

```
UserProcs_high_dev2
UserProcs_low_dev1
www_in_high_anomaly
smtp_out_high_dev2
```

2.6.1 Cfenvd Class list

Base classes:

```
users
rootprocs
otherprocs
diskfree
loadavg
netbiosns_in
netbiosns_out
netbiosdgm_in
netbiosdgm_out
netbiosssn_in
netbiosssn_out
irc_in
irc_out
cfengine_in
cfengine_out
nfsd_in
nfsd_out
smtp_in
smtp_out
www_in
www_out
ftp_in
ftp_out
ssh_in
ssh_out
wwws_in
wwws_out
icmp_in
icmp_out
udp_in
udp_out
dns_in
dns_out
tcpsyn_in
tcpsyn_out
tcpack_in
tcpack_out
tcpfin_in
tcpfin_out
tcpmisc_in
tcpmisc_out
```

Suffixes:

```
_high_microanomaly
_low_microanomaly

_high_dev1
_low_dev1

_high_dev2
_low_dev2

_high_anomaly
_low_anomaly

_high_ldt
_low_ldt
```

2.7 Variables

Cfenvd sets variables which cache the values that were valid at the time of the anomaly's occurrance. These are of the same form as above.

```
value_rootprocs
average_rootprocs
stddev_rootprocs

value_nsfd_in
average_nfsd_in
stddev_nfsd_in
```

The Leap Detection Test buffer is called

```
ldtbuf_users
ldtbuf_otherprocs
```

etc.

2.7.1 Entropy

For network related data, cfengine evaluates the entropy in the currently measured sample of measurements, with respect to the different IP addresses of the sources. You can use these to predicate the appearance of an anomaly, e.g.

```
entropy_www_in_high
entropy_smtp_in_low
```

For example, if you only want to know when a huge amount of SMTP traffic arrives from a single IP source, you would label your anomaly response:

```
entropy_smtp_in_low.smtp_in_high_anomaly::
```

since the entropy is low when the majority of traffic comes from only a small number of IP addresses (e.g. one). The entropy is maximal when activity comes equally from several different sources.

2.8 Log utilities

How shall we respond to an anomalous event? Alerts can be channelled directly to syslog:

```
SysLog(LOG_ERR,"Test syslog message")
```

Software that processes logs can thus be interfaced with via the syslog interface.

2.9 Persistent alerts

DEFCON 1

Another application for alerts is to pass signals from one cfengine to another by persistent, shared memory. For example, suppose a short-lived anomaly event triggers a class that relates to a security alert. The event class might be too short-lived to be followed up by cfagent in full. One could thus set a long term class that would trigger up several follow-up checks. A persistent class could also be used to exclude an operation for an interval of time.

Persistent class memory can be added through a system alert functions to give timer behaviour. For example, consider setting a class that acts like a non-resettable timer. It is defined for exactly 10 minutes before expiring.

```
SetState("preserved_class",10,Preserve)
```

Or to set a class that acts as a resettable timer. It is defined for 60 minutes unless the SetState call is called again to extend its lifetime.

```
SetState(non_preserved_class,60,Reset)
```

Existing persistent classes can be deleted with:

```
UnsetState(myclass)
```

Concept Index

Table of Contents