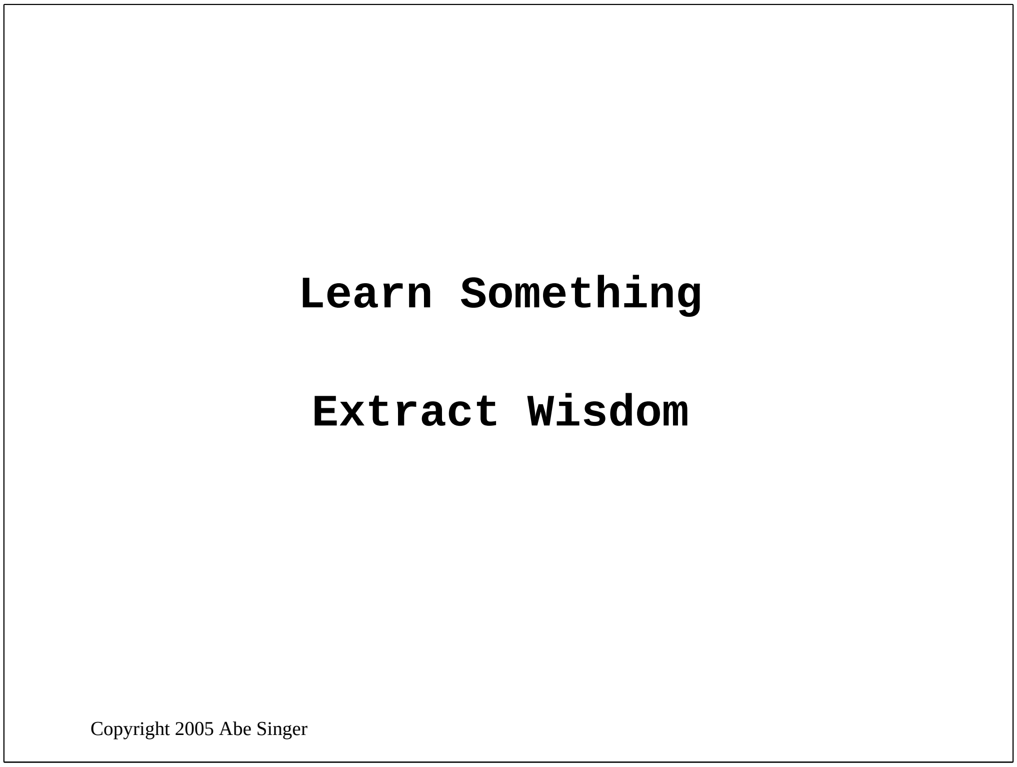
Building a Logging Infrastructure

Abe Singer



Learn Something

Extract Wisdom

What are we here for?

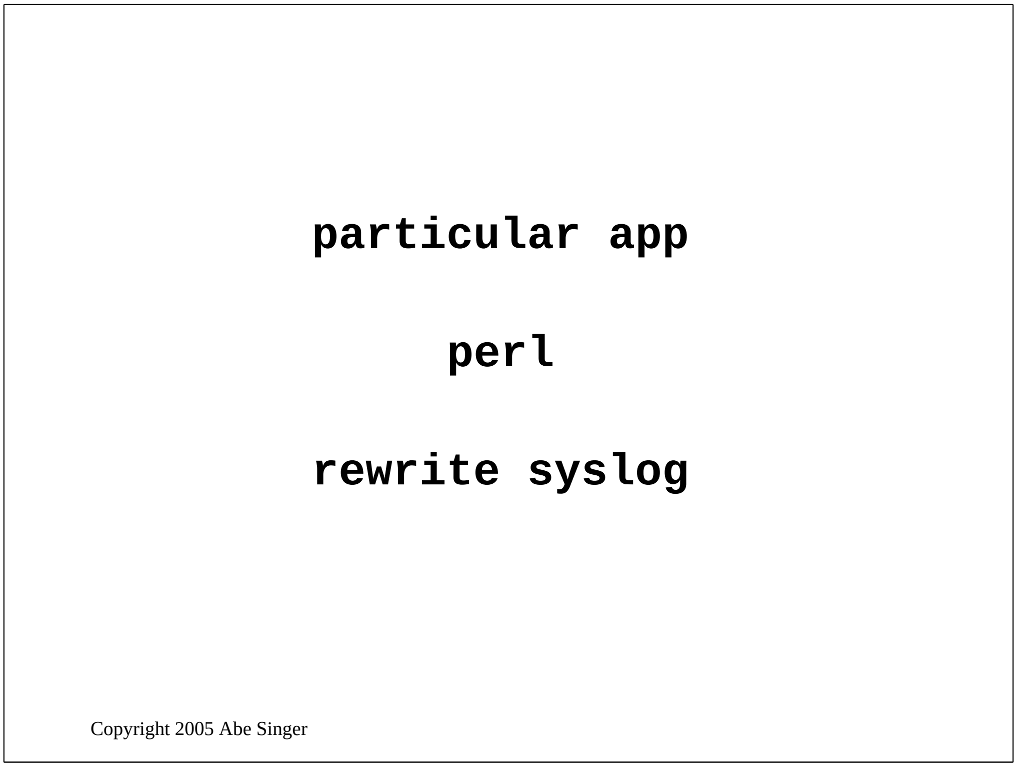
to learn about how to better generate, collect, store, and analyze logs

-----

Ultimate goal: to extract wisdom, to learn more about your network

This book describes how to build an infrastructure to collect, preserve, and extract useful information from your computer operating system and application logs. We will focus primarily on UNIX *syslog*, with some discussion of Windows logging and other sources of log data. Logfiles hold a wealth of information, from resource utilization diagnostics to problems with hardware and software, security problems, and forensic traces of intrusions. [examples are heavily weighted towards security issues, but we provide some examples of resource and diagnostic monitoring.[i]

[i] For a couple of reasons. First, it's what people most often ask about; second, logs are often underrated as a source of security information.



particular app

perl

rewrite syslog

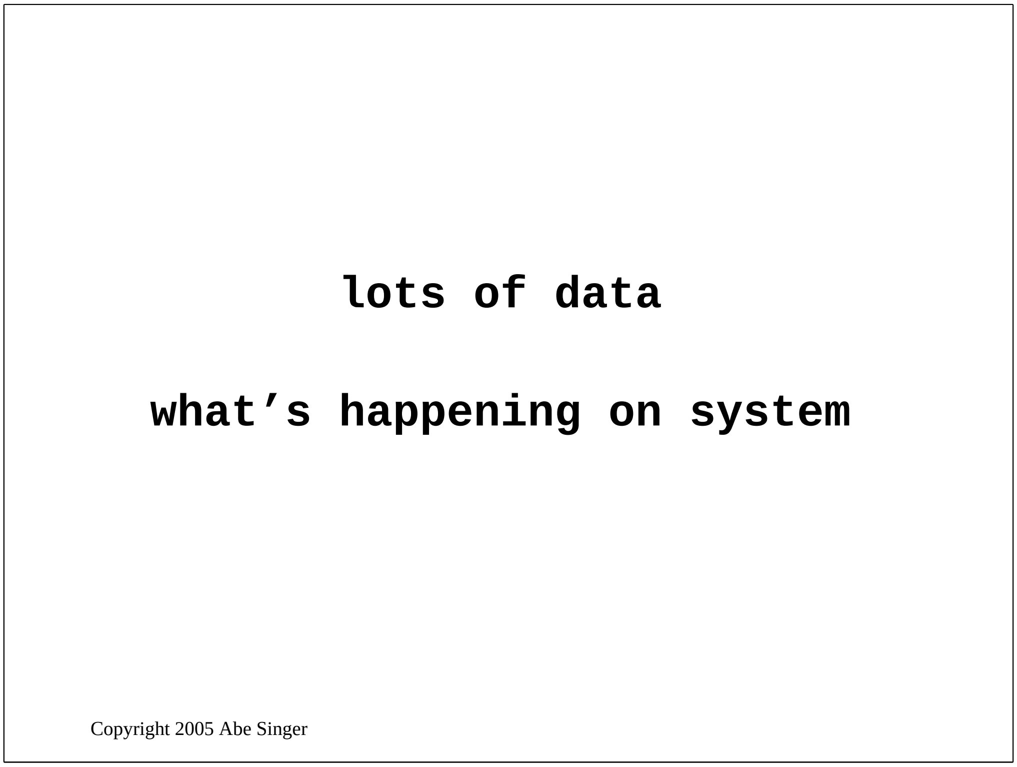
what we’re not here for

The goal of this presentation is *not* to teach you how to interpret log files from any particular system (how would we pick?),

-----

how to write Perl scripts, or how to rewrite *syslog*.

It’s to provide an overview of the sorts of things your logfiles can tell you – how an archetypal UNIX log system (*syslog*) works – how to consolidate your UNIX and Windows XP/2000 logging – and how to monitor your network for intrusion detection, forensic analysis and chaos reduction.



lots of data

what’s happening on system

why logs are useful

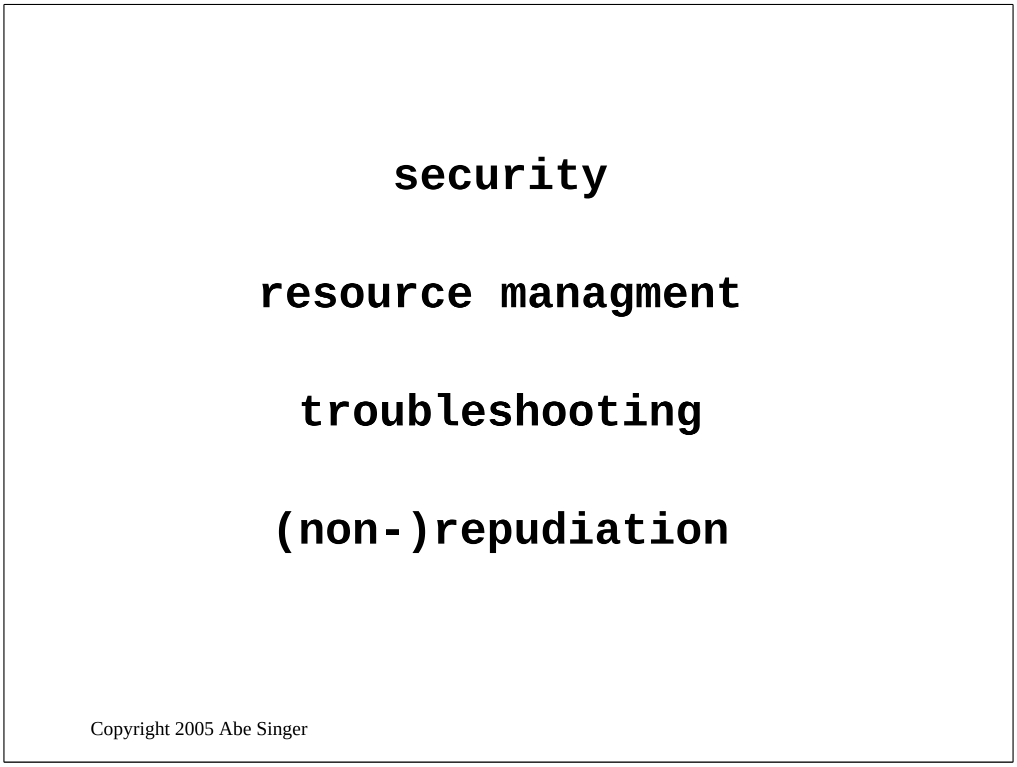
Logs have a wealth of data

Tell you what’s happening on the machine

-----

sometimes they even tell you *why*

Operating systems and the applications they host generate records for a lot of their activities – some indicate administrative activity, some record details of normal operations, some give you information on unusual events. Logs help *tell you what’s happening on your systems*. Sometimes the machines even tell you *why* they’re doing those things, which is really useful.



security

resource managment

troubleshooting

(non-)repudiation

logs are good for security

Can help detect intrusions, probes

can be used for policy enforcement

Logs are critical for detecting and responding to intrusions, attempted intrusions, and other not-so-nice behavior. Off-the-shelf operating systems and applications typically need some tuning to make them more effective at recording malicious activity,

With some simple configuration changes on your computers, you can detect port scans and other probes, letting you know when malicious attackers are looking for weakness in your defenses.

are good for resource management

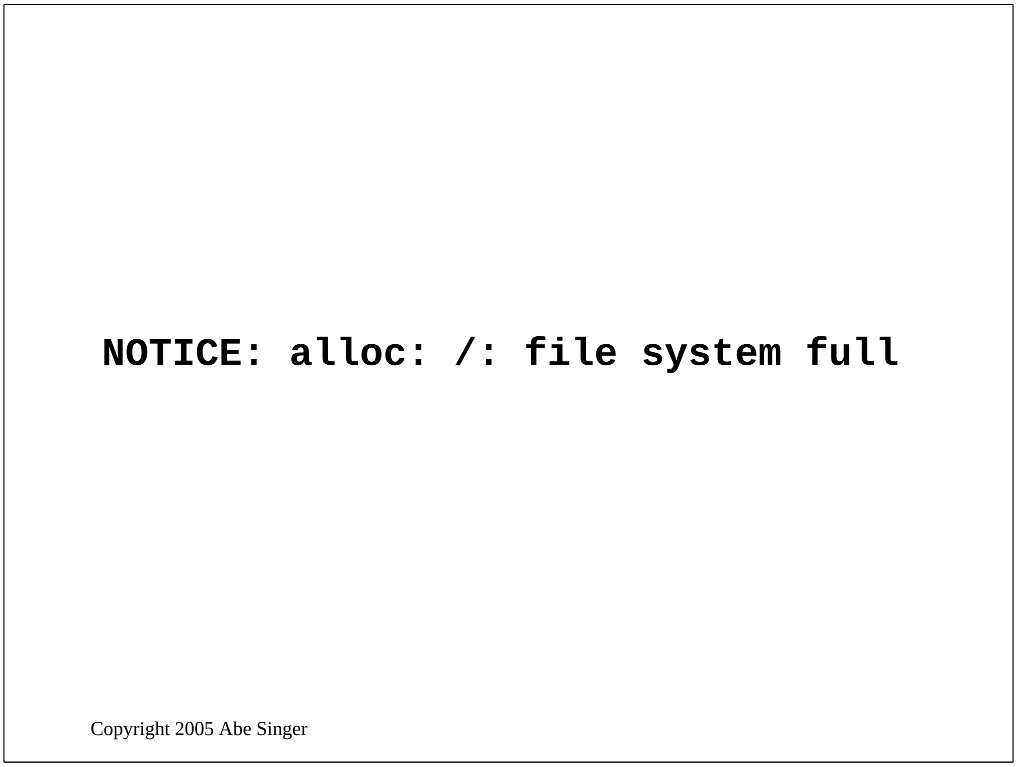
-----

can identify what’s running on your network

measure utilization

uptime, etc.

can discover failures

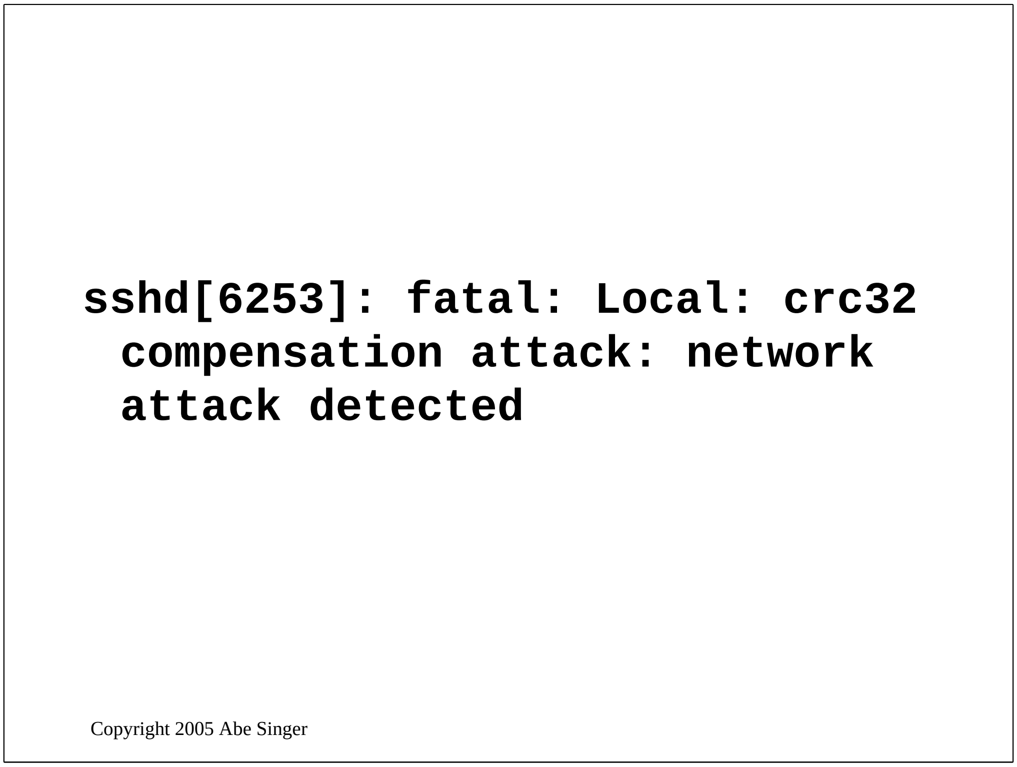


```
NOTICE: alloc: /: file system full
```

Some useful log messages

------

this is the classic example: file system full

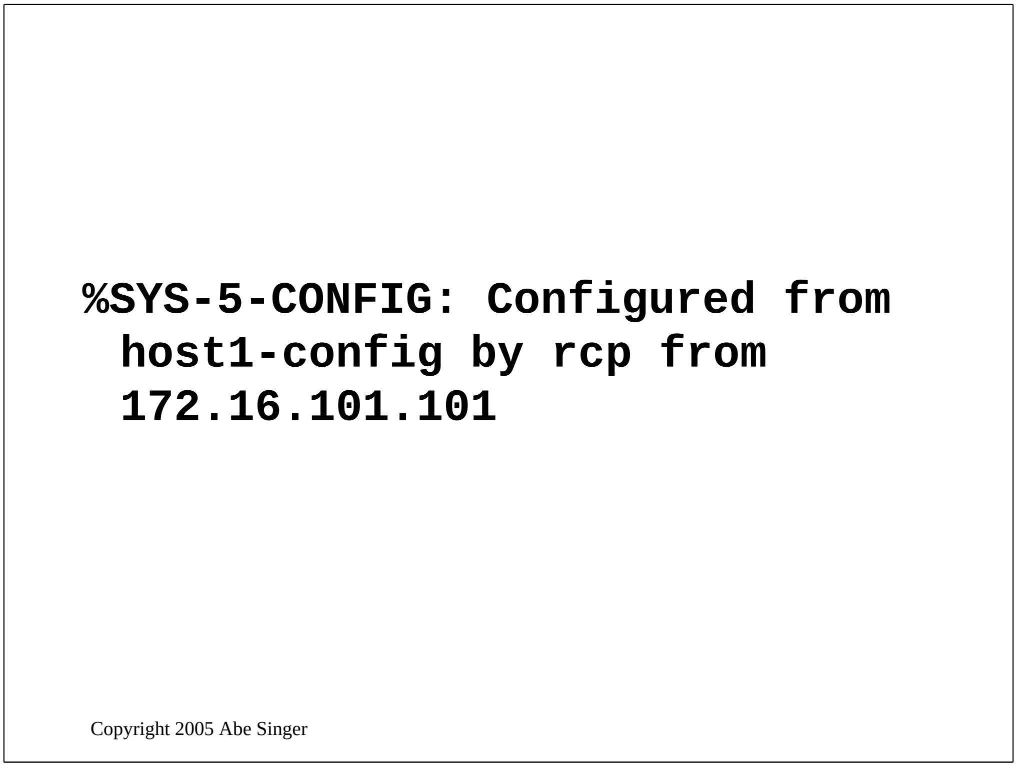


```
sshd[6253]: fatal: Local: crc32
  compensation attack: network
  attack detected
```

an example of a failed attack attempt

-----

this was used in the matrix

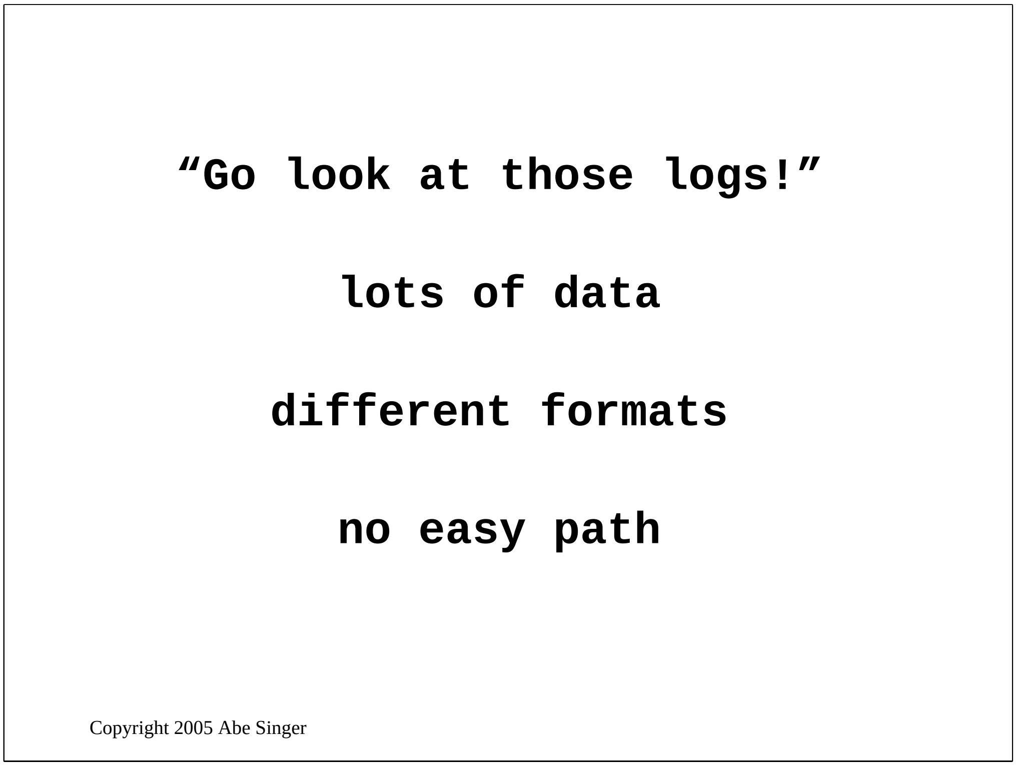


```
%SYS-5-CONFIG: Configured from
 host1-config by rcp from
 172.16.101.101
```

and an example of an IOS system being reconfigured

-----

which might be something you’d want to know about if it wasn’t supposed to be reconfigured



“Go look at those logs!”

lots of data

different formats

no easy path

The log problem:

Many system administrators have been told to “go figure out those logs

-----

. It’s a daunting task – there’s an awful lot of data, little of which seems to be useful or pertinent, at least at first glance.

If the sysadmin keeps going at all, she tends to build a monitoring system based on that relatively random data that’s showed up since the beginning,

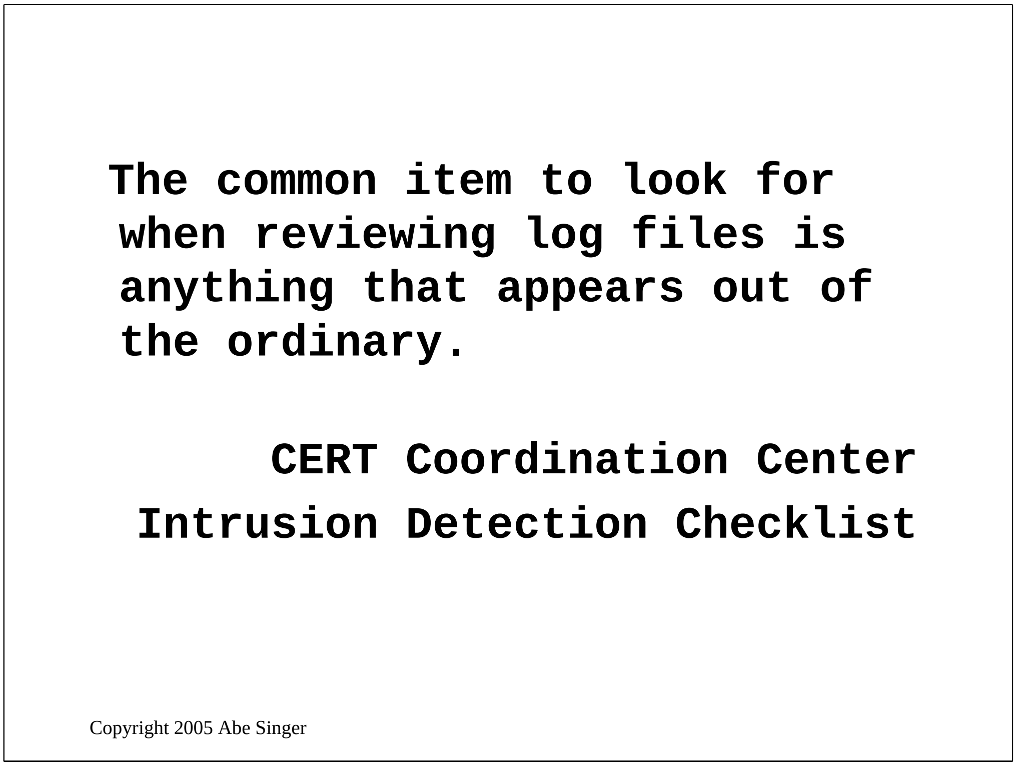
and the data is differing formats

---

some difficult to work with

and there’s no easy tried-and-true plug-it-in way of getting results

but we’re going to try and make it easier



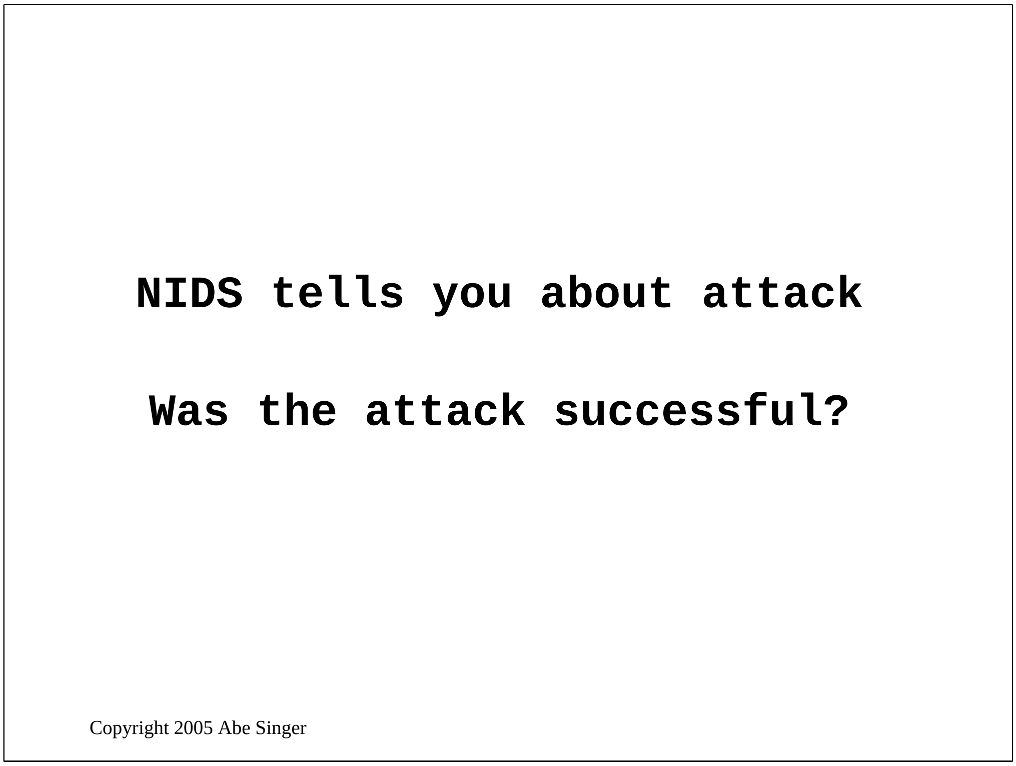
The common item to look for when reviewing log files is anything that appears out of the ordinary.

CERT Coordination Center
Intrusion Detection Checklist

CERT has some sage advice:

Great advice from CERT, if you know how to tell what's "out of the ordinary." First of all, there is a lot of data to wade through.   on most OSes and applications, we observe that administrative and security-relevant events form a small fraction of the total volume of log data, often less than 5%.

it's actually really hard to know what's "ordinary"



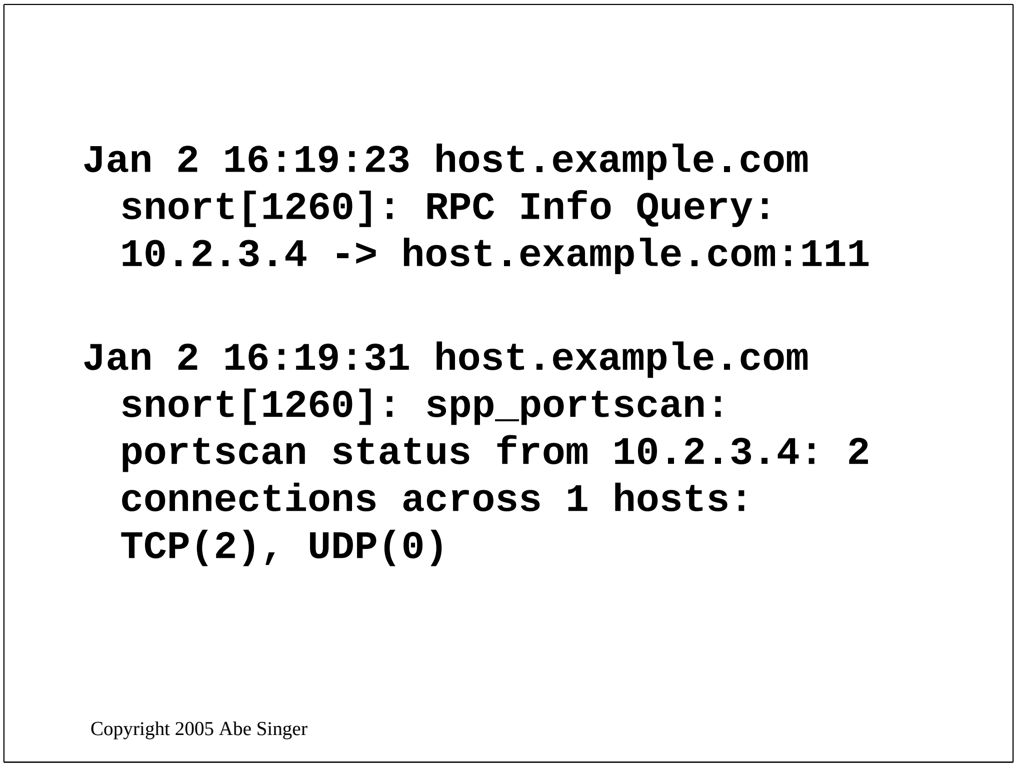
NIDS tells you about attack

Was the attack successful?

why NIDS is not enough

"I don't need no stinkin' logs, my <*security widget of the month*> will tell me when something important is happening.” And it’s true enough that most general-purpose computer systems aren’t very good at detecting and recording potentially malicious behavior without help. However, while security-specific devices like network intrusion detection systems and firewalls make it easier to notice malicious activity, they don’t eliminate the need for collecting, analyzing and archiving host-based logs.

. Having the IDS doesn’t mean that you can discard your host-level logs. It just lets you know particularly good times to go look at them.



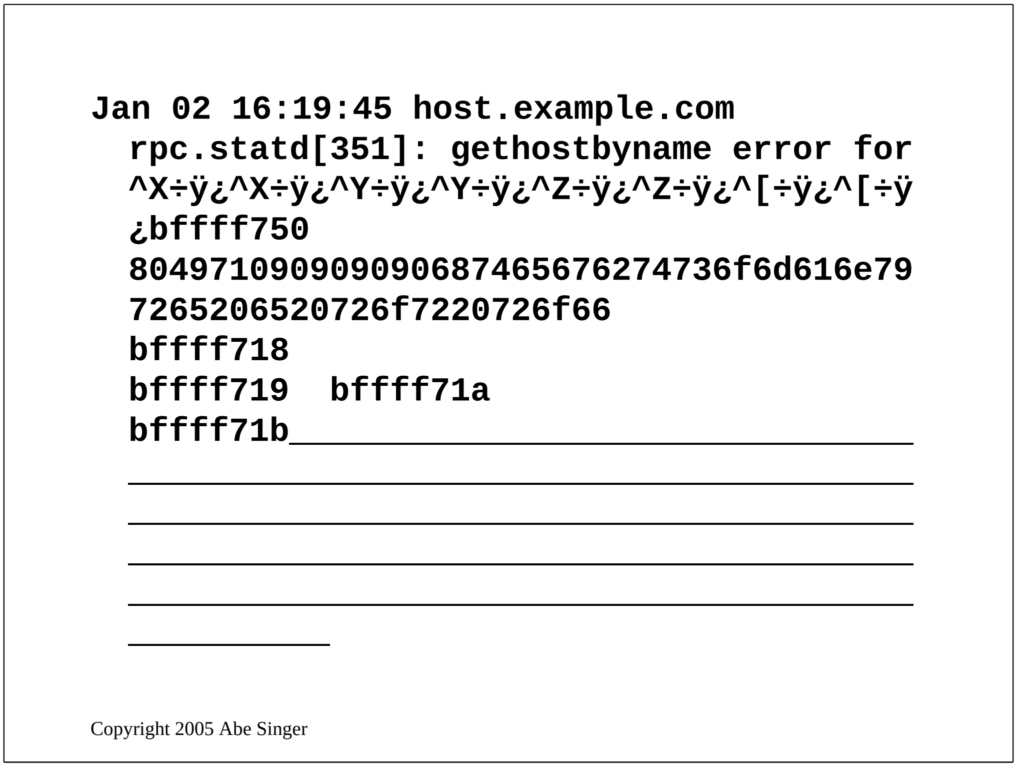
```
Jan 2 16:19:23 host.example.com
  snort[1260]: RPC Info Query:
  10.2.3.4 -> host.example.com:111

Jan 2 16:19:31 host.example.com
  snort[1260]: spp_portscan:
  portscan status from 10.2.3.4: 2
  connections across 1 hosts:
  TCP(2), UDP(0)
```

for example, here’s an message from snort:

-----

it shows a portscan and connectoins to portmapper.

but doesn’t tell you what actually happened

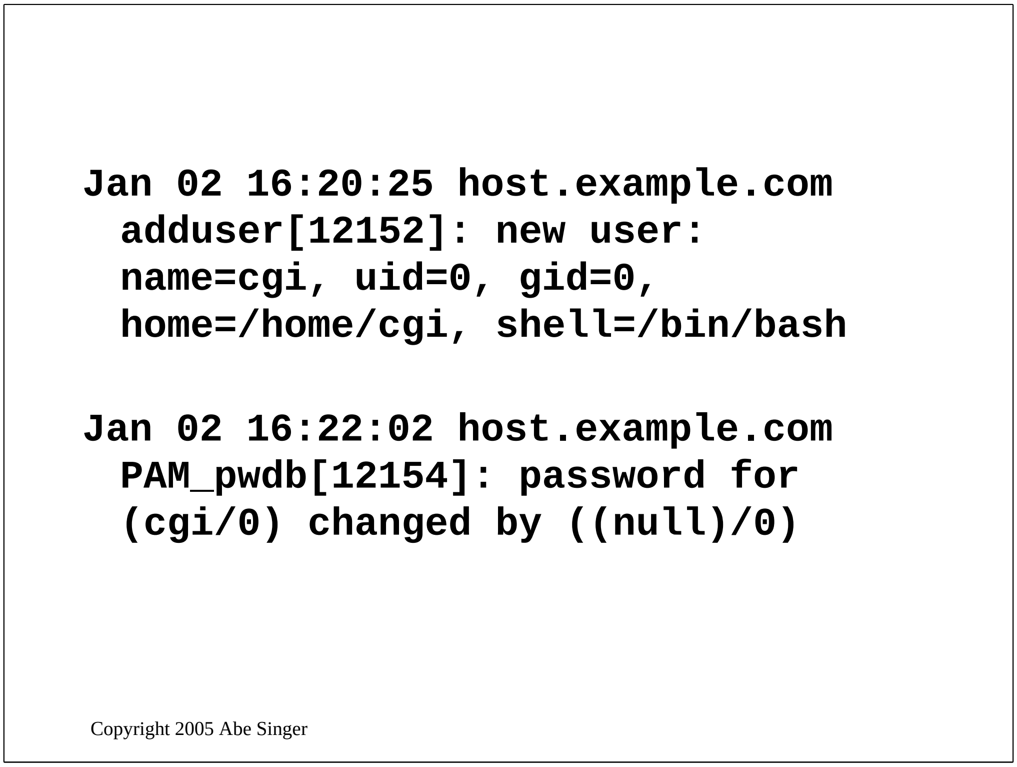


```
Jan 02 16:19:45 host.example.com
  rpc.statd[351]: gethostbyname error for
  ^X÷ÿ¿^X÷ÿ¿^Y÷ÿ¿^Y÷ÿ¿^Z÷ÿ¿^Z÷ÿ¿^[÷ÿ¿^[÷ÿ
  ¿bffff750
  804971090909090687465676274736f6d616e79
  7265206520726f7220726f66
  bffff718
  bffff719  bffff71a
  bffff71b_______________________________
  _______________________________________
  _______________________________________
  _______________________________________
  _______________________________________
  __________
```

------

looks like someone attempted a buffer overflow

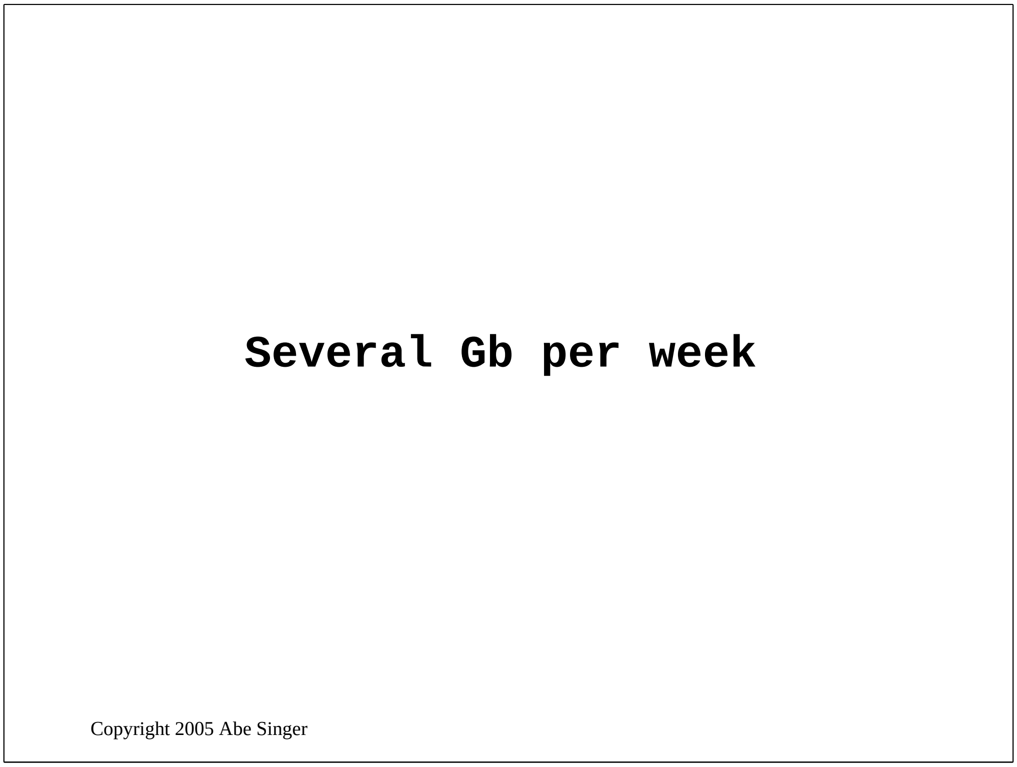
So at this point, the sysadmin of the victim machine knows that his host machine was scanned for a vulnerable service, and from this *syslog* message she knows that the attacker executed a buffer overflow attack. But things get tricky here. For the vast majority of applications on UNIX and Windows, any log message that's created by an application undergoing a buffer overflow attack is a sign that *the attack failed*. If the attack is successful, it usually interrupts the normal workflow of the victim application *before* the log message is written out.



```
Jan 02 16:20:25 host.example.com
  adduser[12152]: new user:
  name=cgi, uid=0, gid=0,
  home=/home/cgi, shell=/bin/bash

Jan 02 16:22:02 host.example.com
  PAM_pwdb[12154]: password for
  (cgi/0) changed by ((null)/0)
```

These lines – showing the creation of a new user named cgi and the associated password creation – reveal that the perpetrator of the buffer overflow in the previous example “got r00t.” As user names go, cgi is a choice that the attacker probably hoped would be unremarkable amongst the other UNIX user names, which are frequently associated with processes and applications rather than people. But cgi is different -- it has UID=0, indicating that the account has superuser privileges, unlimited access to the host machine.

As if that wasn’t bad enough, the password for cgi was changed by a UID 0 user with a null login. That’s a really bad sign. In the vast majority of situations, UID 0 users are associated with specific user accounts, because that’s how you record users doing system administration. In this particular case, the attacker was dropped into a shell with UID 0 privileges after successfully executing the buffer overflow. This particular line of log data is the only thing that definitively records an activity executed within that shell.



Several Gb per week

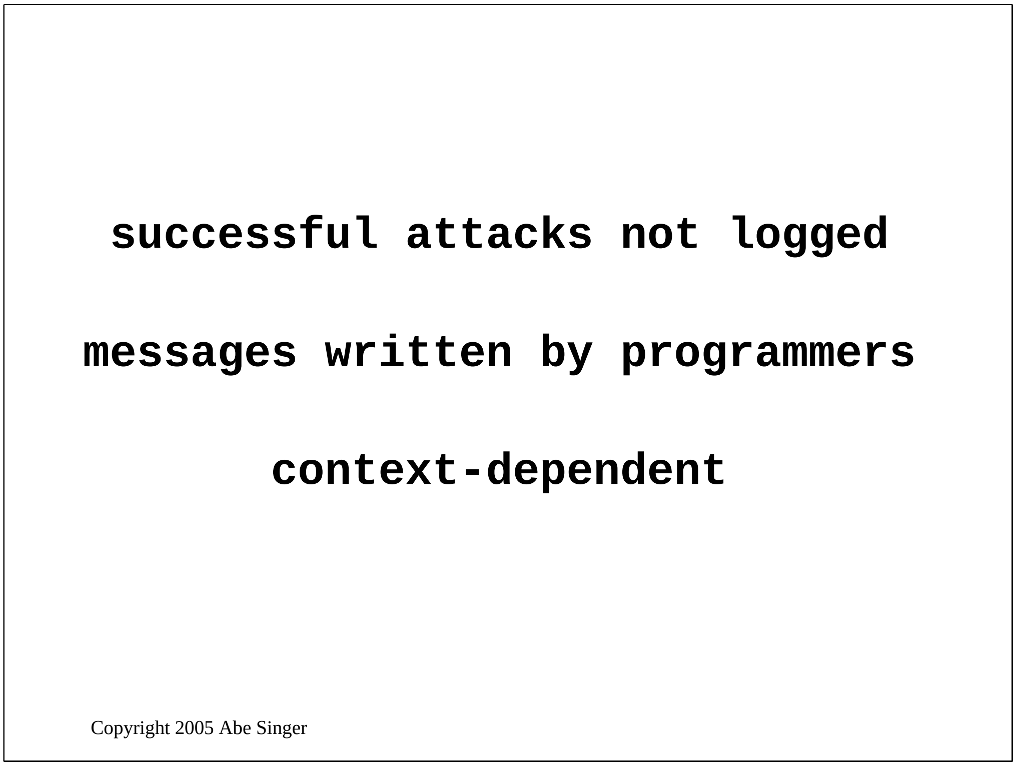
So I said a *lot* of data, how much are we talking about?

SDSC logs millions of lines per day, several Gb per week

-----

not counting web logs

Some places log several Gb per *day*



successful attacks not logged

messages written by programmers

context-dependent

And logs can suck. For instances,

Successful attacks are often not logged

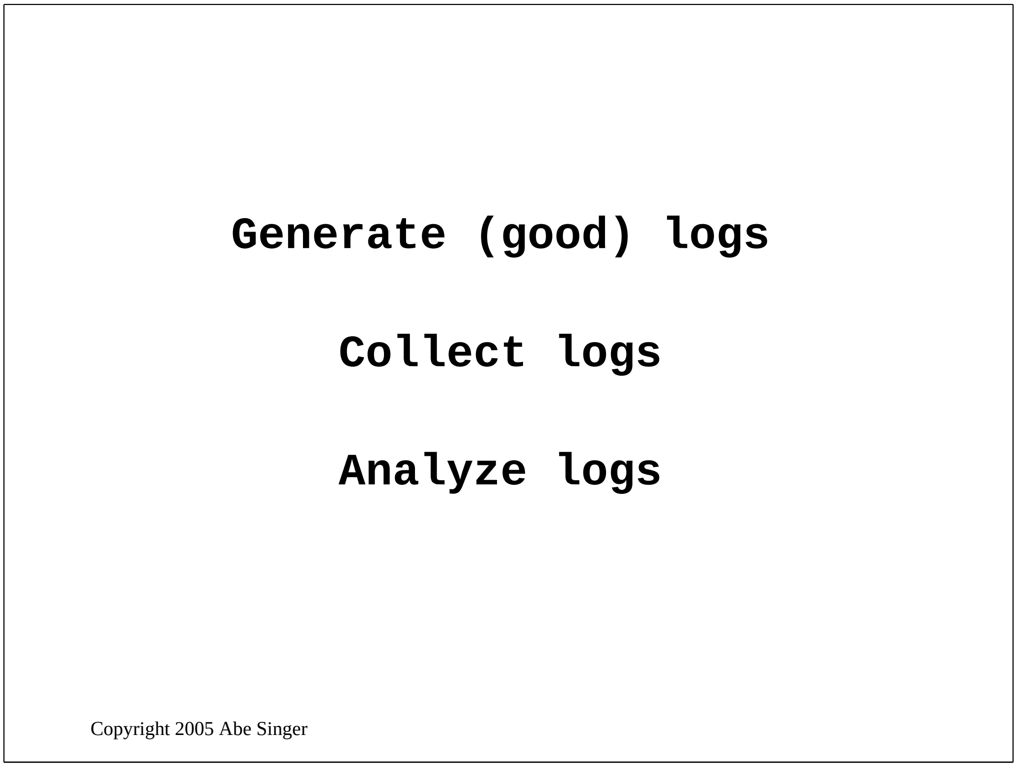
Log messages vary in quality, and not designed for machine parsing. Logs are writen by programmers, who often don’t

think about how they’re gonna be used

What’s “interesting” is very dependent on your environment

-----

what’s interesting to one person may be totally boring to another.



Generate (good) logs

Collect logs

Analyze logs

what does it take?

basic strategy:

we’re going to talk about understading logs
basic logging
indentifying sources of log information and

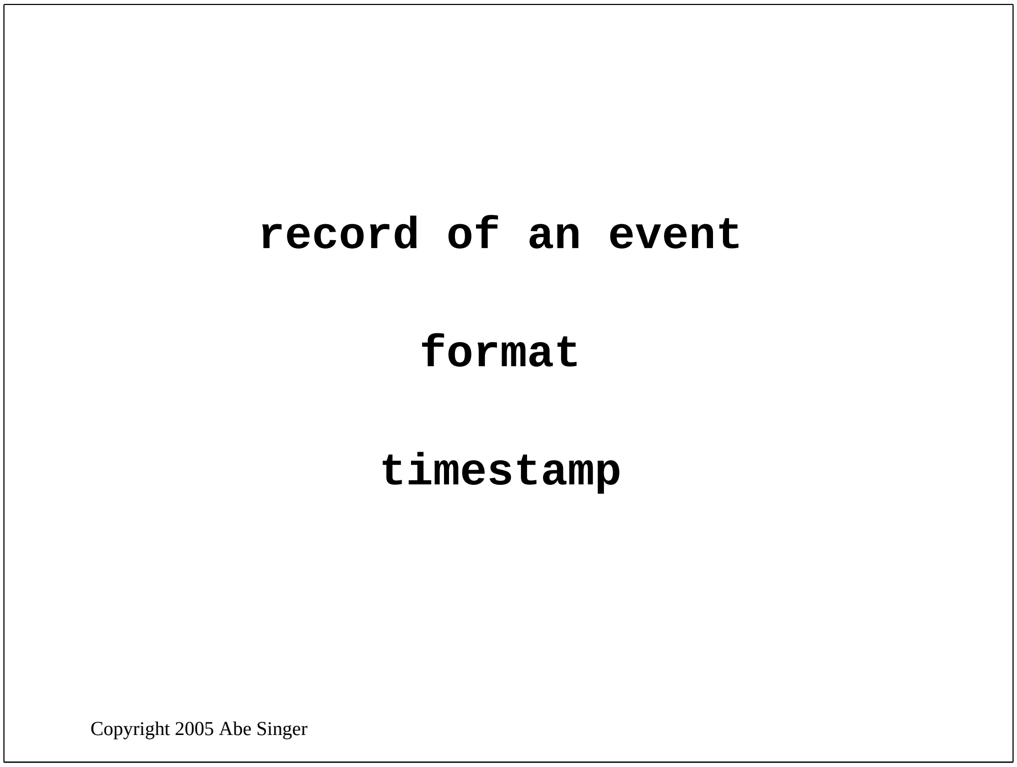
, and look for ways to improve their quality

we’ll look at centralized logging infrastructure
ot collect everything together in one place

-----
and talk about archiving and preserving it

analyze the collected data

What’s in a Log?



record of an event

format

timestamp

Understanding logs

what is a log?

a record of an event in time.

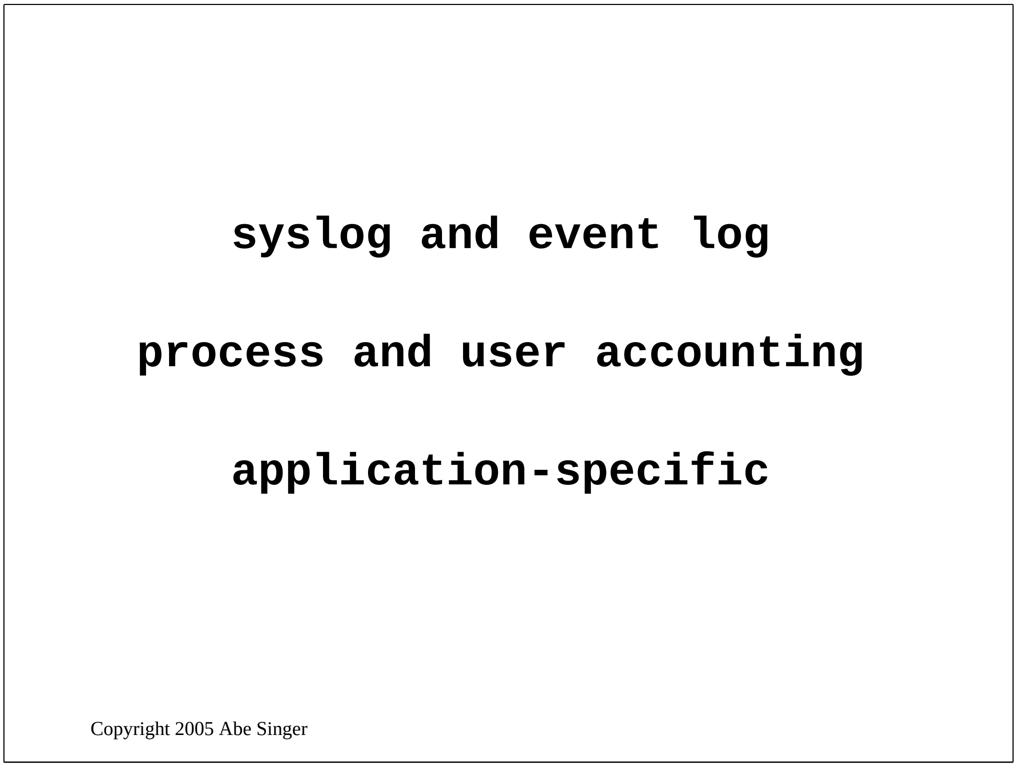
-----

logs have some comon properties. They all have some sort of format., be it text or binary

-----

key elements are a timestamp (useless without one) and some description of what happened

-----



syslog and event log

process and user accounting

application-specific

some different types of logs

syslog and the event log

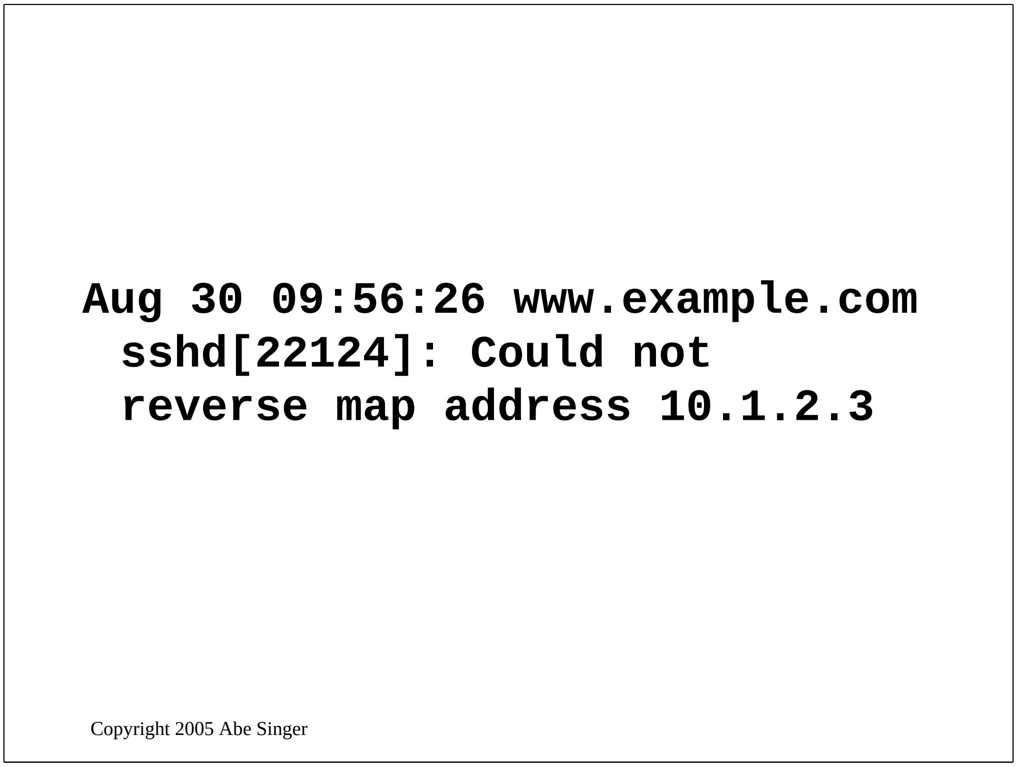
we’re going to talk mostly about syslog

-----

pacct, wtmp, ptmp

-----

application specific: web, IDS, FW, VPN, etc.



```
Aug 30 09:56:26 www.example.com
  sshd[22124]: Could not
  reverse map address 10.1.2.3
```

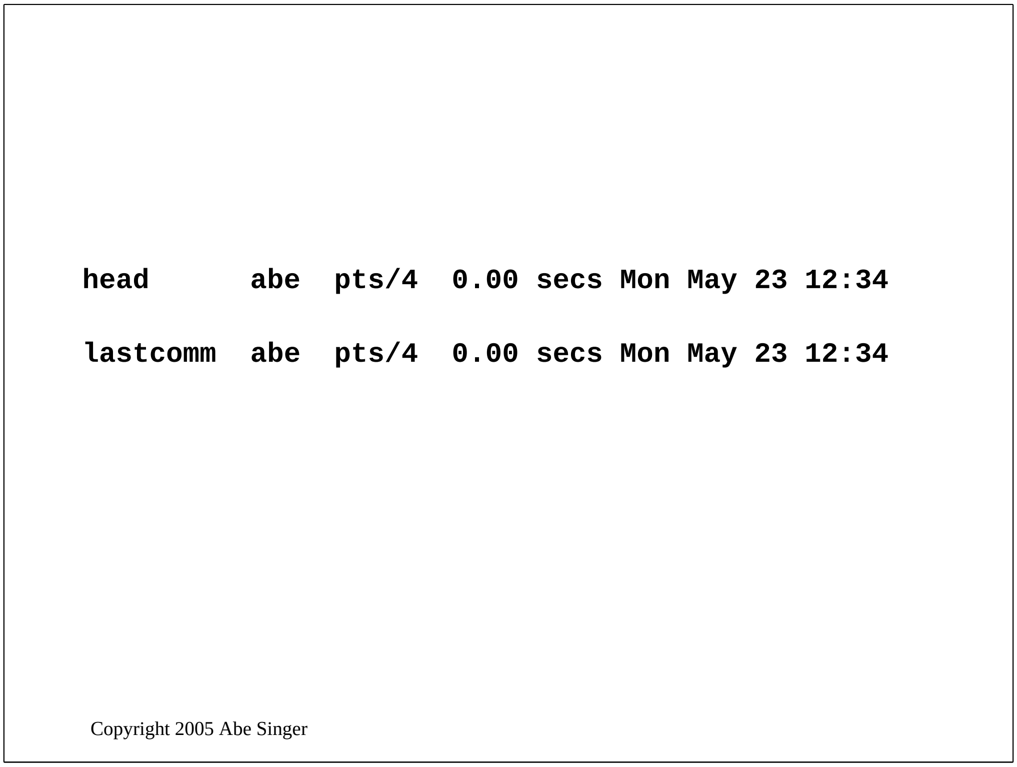
log formats: syslog

-----

has timestamp (note no year or time zone)

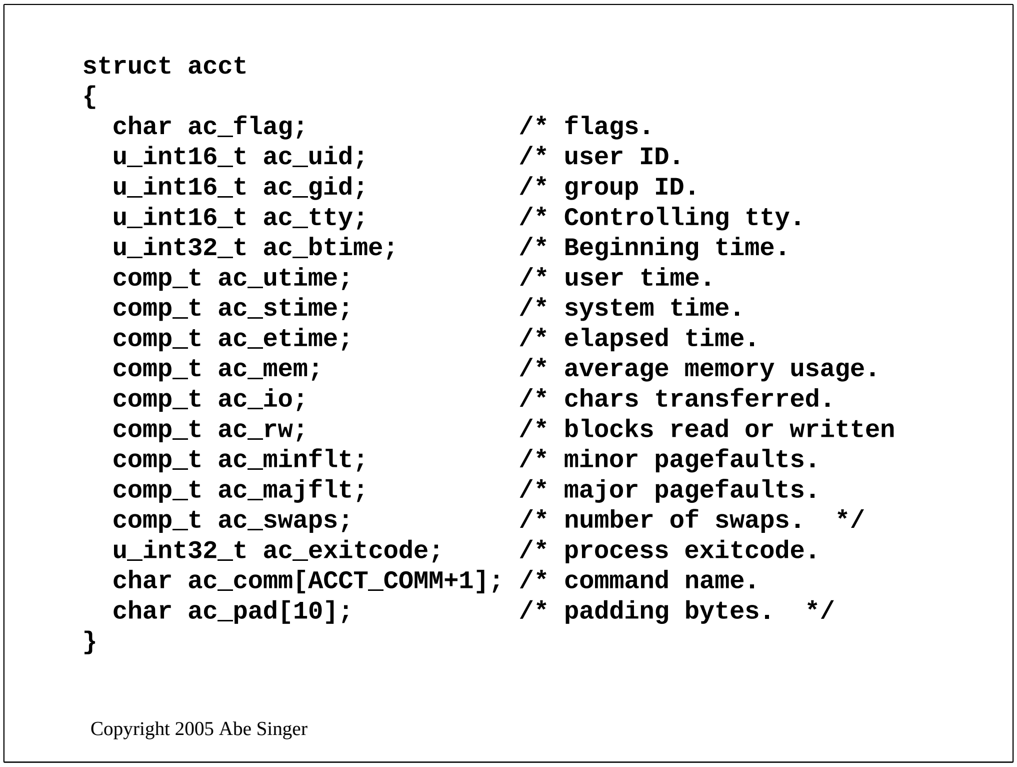
hostname, service name, and PID

format varies between different systems, but basically like this



```
head      abe  pts/4  0.00 secs Mon May 23 12:34

lastcomm  abe  pts/4  0.00 secs Mon May 23 12:34
```

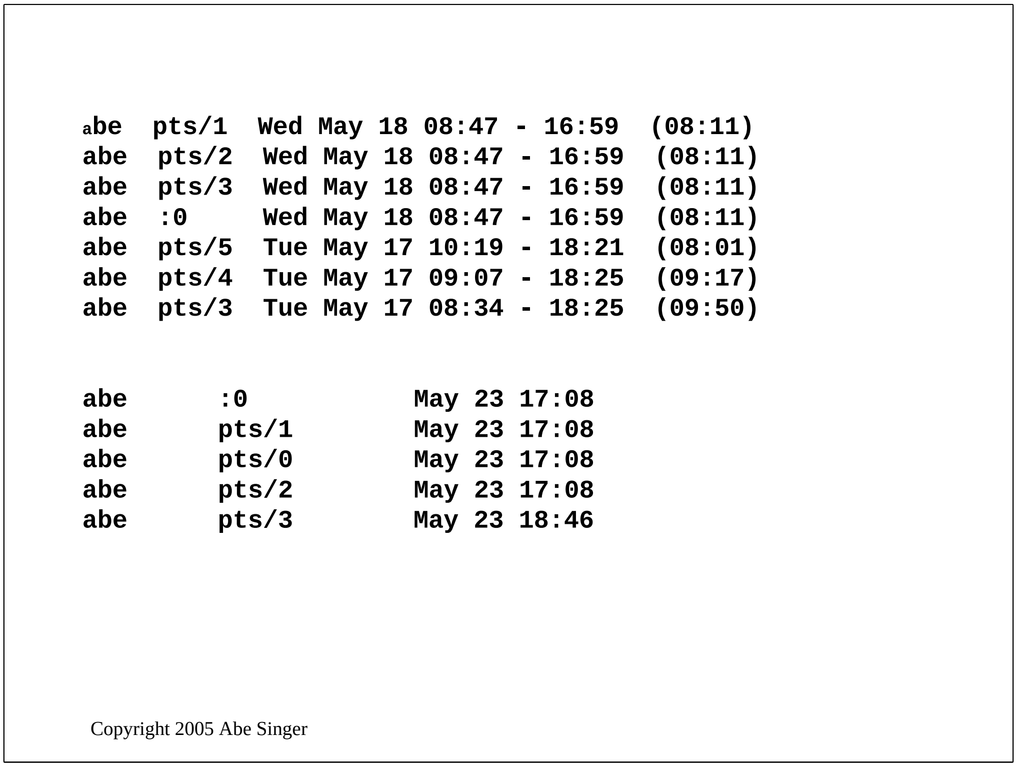
process accounting logs



```
struct acct
{
  char ac_flag;              /* flags.
  u_int16_t ac_uid;          /* user ID.
  u_int16_t ac_gid;          /* group ID.
  u_int16_t ac_tty;          /* Controlling tty.
  u_int32_t ac_btime;        /* Beginning time.
  comp_t ac_utime;           /* user time.
  comp_t ac_stime;           /* system time.
  comp_t ac_etime;           /* elapsed time.
  comp_t ac_mem;             /* average memory usage.
  comp_t ac_io;              /* chars transferred.
  comp_t ac_rw;              /* blocks read or written
  comp_t ac_minflt;          /* minor pagefaults.
  comp_t ac_majflt;          /* major pagefaults.
  comp_t ac_swaps;           /* number of swaps.  */
  u_int32_t ac_exitcode;     /* process exitcode.
  char ac_comm[ACCT_COMM+1]; /* command name.
  char ac_pad[10];           /* padding bytes.  */
}
```

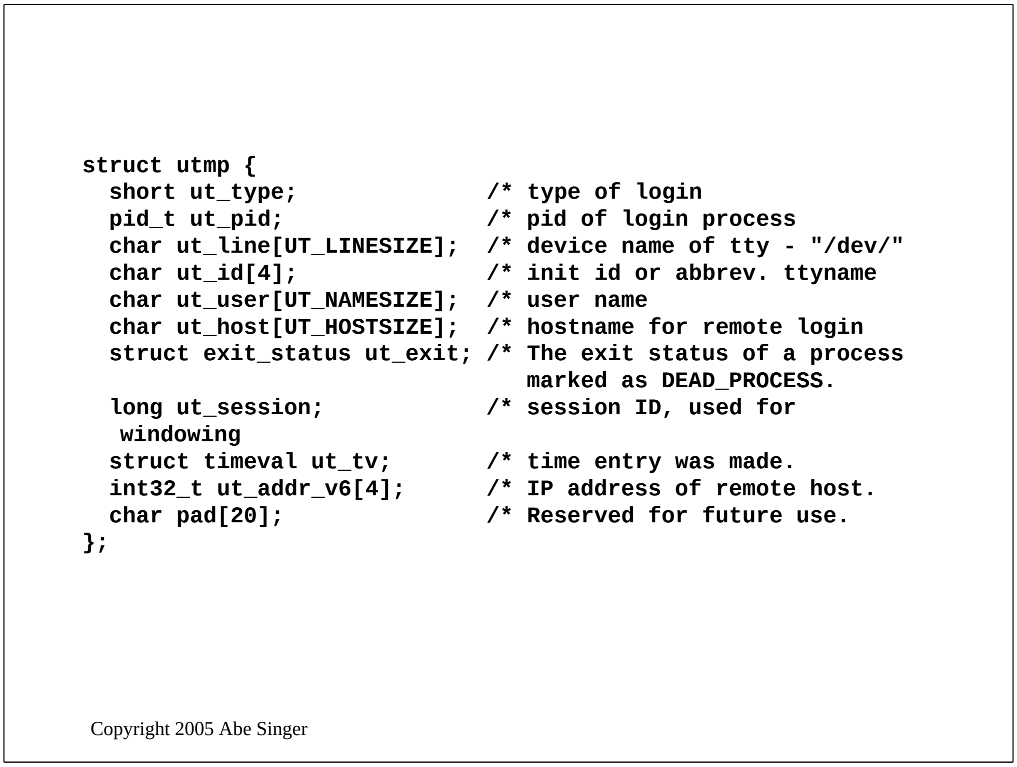
-----

here’s all the data that’s actually stored, notice that there’s a lot more than lastcomm tells you



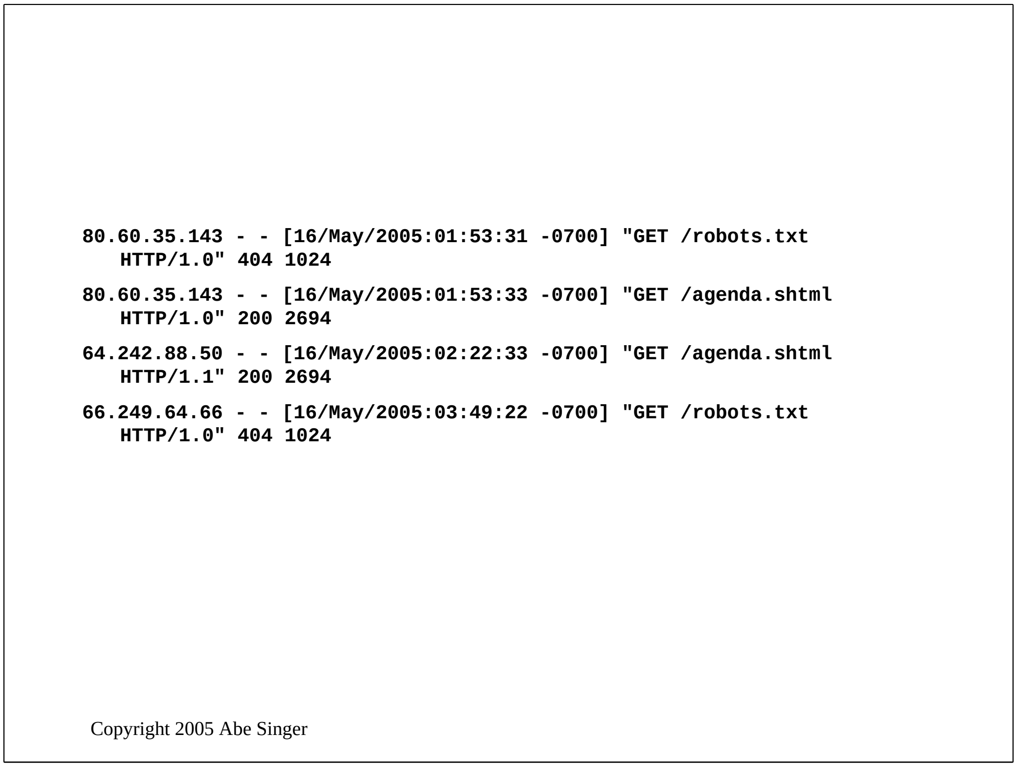
```
abe  pts/1  Wed May 18 08:47 - 16:59  (08:11)
abe  pts/2  Wed May 18 08:47 - 16:59  (08:11)
abe  pts/3  Wed May 18 08:47 - 16:59  (08:11)
abe  :0     Wed May 18 08:47 - 16:59  (08:11)
abe  pts/5  Tue May 17 10:19 - 18:21  (08:01)
abe  pts/4  Tue May 17 09:07 - 18:25  (09:17)
abe  pts/3  Tue May 17 08:34 - 18:25  (09:50)


abe      :0           May 23 17:08
abe      pts/1        May 23 17:08
abe      pts/0        May 23 17:08
abe      pts/2        May 23 17:08
abe      pts/3        May 23 18:46
```

examples of the output of last and who



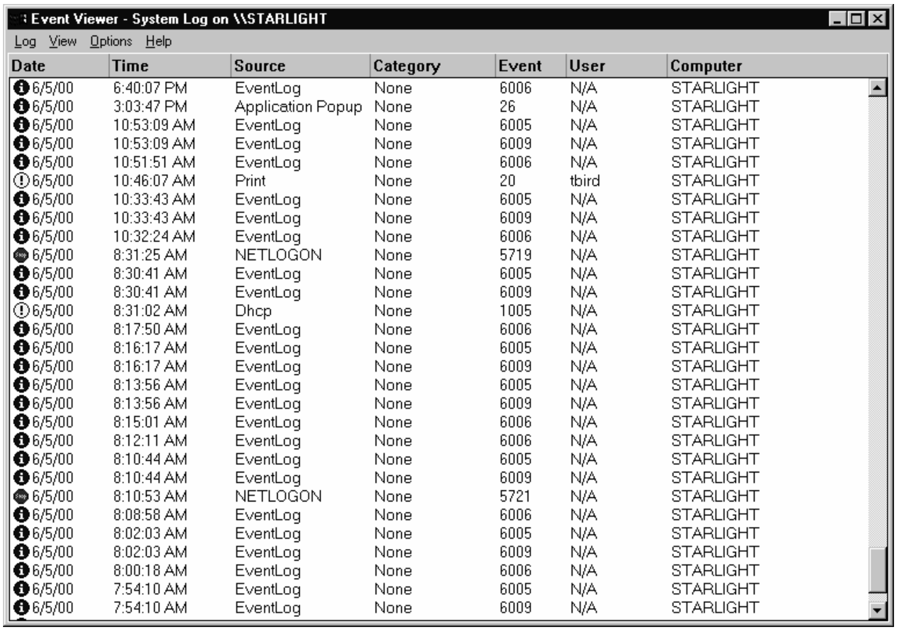
```
struct utmp {
  short ut_type;              /* type of login
  pid_t ut_pid;               /* pid of login process
  char ut_line[UT_LINESIZE];  /* device name of tty - "/dev/"
  char ut_id[4];              /* init id or abbrev. ttyname
  char ut_user[UT_NAMESIZE];  /* user name
  char ut_host[UT_HOSTSIZE];  /* hostname for remote login
  struct exit_status ut_exit; /* The exit status of a process
                                 marked as DEAD_PROCESS.
  long ut_session;            /* session ID, used for
   windowing
  struct timeval ut_tv;       /* time entry was made.
  int32_t ut_addr_v6[4];      /* IP address of remote host.
  char pad[20];               /* Reserved for future use.
};
```

the wtmp record structure



```
80.60.35.143 - - [16/May/2005:01:53:31 -0700] "GET /robots.txt
   HTTP/1.0" 404 1024
80.60.35.143 - - [16/May/2005:01:53:33 -0700] "GET /agenda.shtml
   HTTP/1.0" 200 2694
64.242.88.50 - - [16/May/2005:02:22:33 -0700] "GET /agenda.shtml
   HTTP/1.1" 200 2694
66.249.64.66 - - [16/May/2005:03:49:22 -0700] "GET /robots.txt
   HTTP/1.0" 404 1024
```

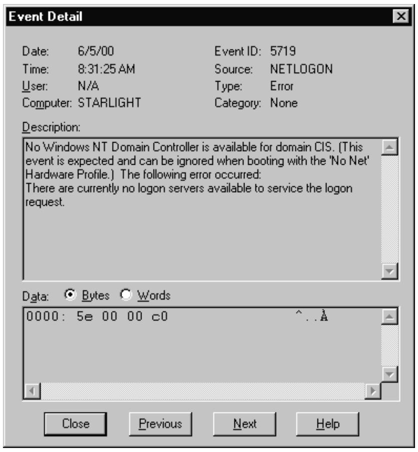
logs from an apache web server, notice that this has a year
and a regular format
although it can still be difficult to parse



Date	Time	Source	Category	Event	User	Computer
ℹ 6/5/00	6:40:07 PM	EventLog	None	6006	N/A	STARLIGHT
ℹ 6/5/00	3:03:47 PM	Application Popup	None	26	N/A	STARLIGHT
ℹ 6/5/00	10:53:09 AM	EventLog	None	6005	N/A	STARLIGHT
ℹ 6/5/00	10:53:09 AM	EventLog	None	6009	N/A	STARLIGHT
ℹ 6/5/00	10:51:51 AM	EventLog	None	6006	N/A	STARLIGHT
⚠ 6/5/00	10:46:07 AM	Print	None	20	tbird	STARLIGHT
ℹ 6/5/00	10:33:43 AM	EventLog	None	6005	N/A	STARLIGHT
ℹ 6/5/00	10:33:43 AM	EventLog	None	6009	N/A	STARLIGHT
ℹ 6/5/00	10:32:24 AM	EventLog	None	6006	N/A	STARLIGHT
⛔ 6/5/00	8:31:25 AM	NETLOGON	None	5719	N/A	STARLIGHT
ℹ 6/5/00	8:30:41 AM	EventLog	None	6005	N/A	STARLIGHT
ℹ 6/5/00	8:30:41 AM	EventLog	None	6009	N/A	STARLIGHT
⚠ 6/5/00	8:31:02 AM	Dhcp	None	1005	N/A	STARLIGHT
ℹ 6/5/00	8:17:50 AM	EventLog	None	6006	N/A	STARLIGHT
ℹ 6/5/00	8:16:17 AM	EventLog	None	6005	N/A	STARLIGHT
ℹ 6/5/00	8:16:17 AM	EventLog	None	6009	N/A	STARLIGHT
ℹ 6/5/00	8:13:56 AM	EventLog	None	6005	N/A	STARLIGHT
ℹ 6/5/00	8:13:56 AM	EventLog	None	6009	N/A	STARLIGHT
ℹ 6/5/00	8:15:01 AM	EventLog	None	6006	N/A	STARLIGHT
ℹ 6/5/00	8:12:11 AM	EventLog	None	6006	N/A	STARLIGHT
ℹ 6/5/00	8:10:44 AM	EventLog	None	6005	N/A	STARLIGHT
ℹ 6/5/00	8:10:44 AM	EventLog	None	6009	N/A	STARLIGHT
⛔ 6/5/00	8:10:53 AM	NETLOGON	None	5721	N/A	STARLIGHT
ℹ 6/5/00	8:08:58 AM	EventLog	None	6006	N/A	STARLIGHT
ℹ 6/5/00	8:02:03 AM	EventLog	None	6005	N/A	STARLIGHT
ℹ 6/5/00	8:02:03 AM	EventLog	None	6009	N/A	STARLIGHT
ℹ 6/5/00	8:00:18 AM	EventLog	None	6006	N/A	STARLIGHT
ℹ 6/5/00	7:54:10 AM	EventLog	None	6005	N/A	STARLIGHT
ℹ 6/5/00	7:54:10 AM	EventLog	None	6009	N/A	STARLIGHT

The windows event log main page

Severity of an event is indicated by the icon on the far left hand side of the window. A blue icon indicates diagnostic information that requires no action. Yellow indicates a warning that may require attention, but does not severely impact the ability to use the system. Red indicates an error that will probably make the system or service unavailable.



Date: 6/5/00
Time: 8:31:25 AM
User: N/A
Computer: STARLIGHT

Event ID: 5719
Source: NETLOGON
Type: Error
Category: None

Description:

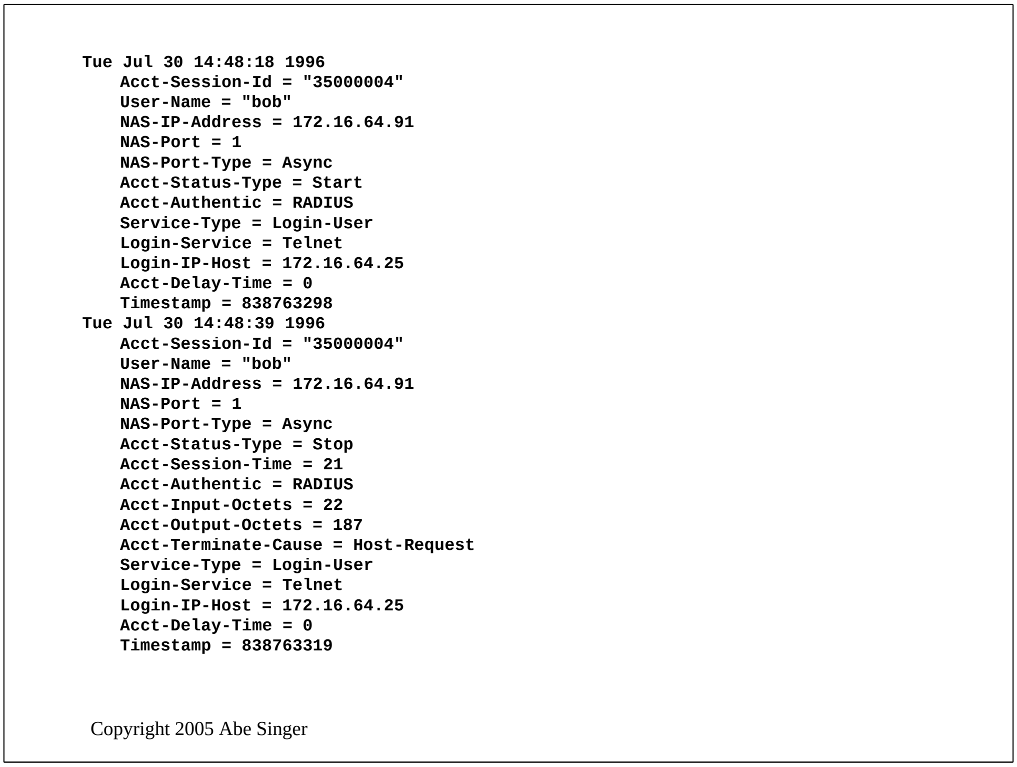
No Windows NT Domain Controller is available for domain CIS. (This event is expected and can be ignored when booting with the 'No Net' Hardware Profile.) The following error occurred:
There are currently no logon servers available to service the logon request.

Data: ● Bytes ○ Words

```
0000: 5e 00 00 c0                              ^..À
```

event detail

To retrieve more information about a particular event, select the item of interest, then under the “View” toolbar item, select “Details.” This translates the numerical error to a human-readable message.

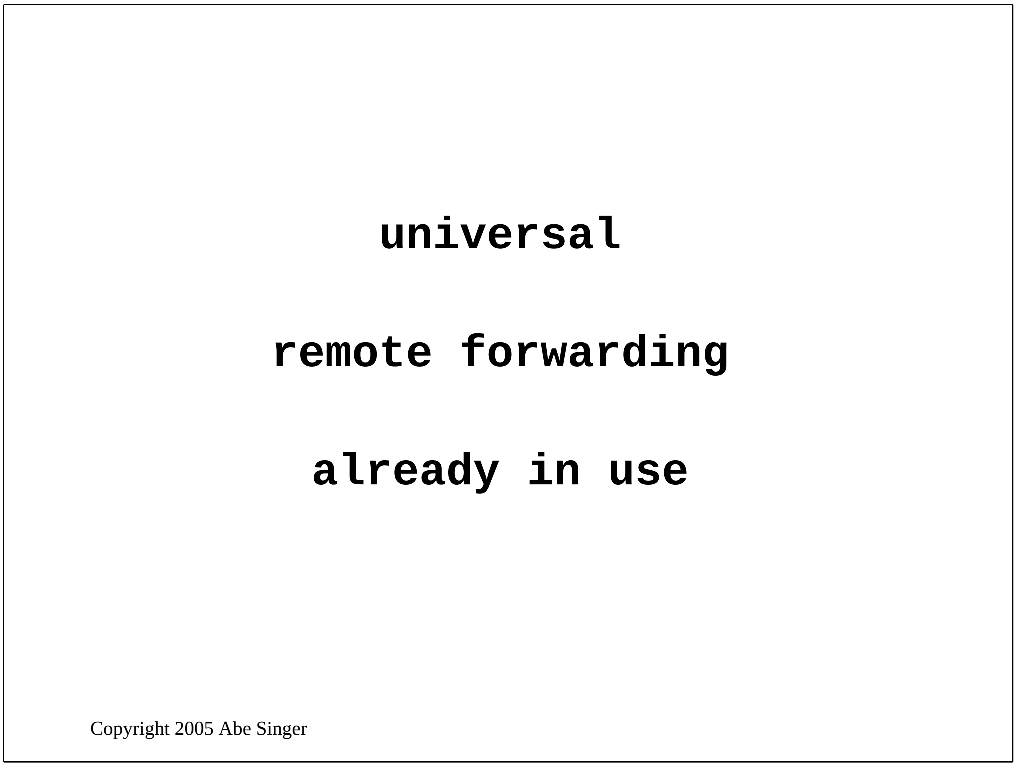


```
Tue Jul 30 14:48:18 1996
    Acct-Session-Id = "35000004"
    User-Name = "bob"
    NAS-IP-Address = 172.16.64.91
    NAS-Port = 1
    NAS-Port-Type = Async
    Acct-Status-Type = Start
    Acct-Authentic = RADIUS
    Service-Type = Login-User
    Login-Service = Telnet
    Login-IP-Host = 172.16.64.25
    Acct-Delay-Time = 0
    Timestamp = 838763298
Tue Jul 30 14:48:39 1996
    Acct-Session-Id = "35000004"
    User-Name = "bob"
    NAS-IP-Address = 172.16.64.91
    NAS-Port = 1
    NAS-Port-Type = Async
    Acct-Status-Type = Stop
    Acct-Session-Time = 21
    Acct-Authentic = RADIUS
    Acct-Input-Octets = 22
    Acct-Output-Octets = 187
    Acct-Terminate-Cause = Host-Request
    Service-Type = Login-User
    Login-Service = Telnet
    Login-IP-Host = 172.16.64.25
    Acct-Delay-Time = 0
    Timestamp = 838763319
```

RADIUS accounting records from a Livingston PortMaster:

notice that they take up multiple lines, a bit more work to parse and deal with

Basic Logging



universal

remote forwarding

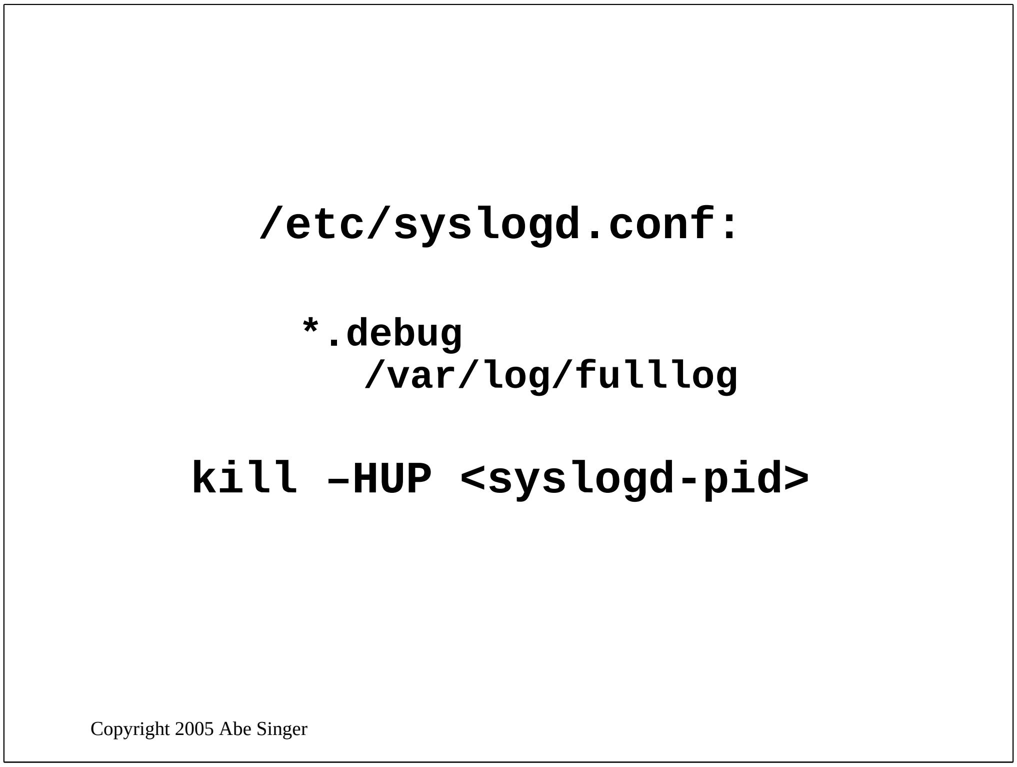
already in use

so why syslog? why do we spend so much time with something so crappy?

well, it’s universal

– comes installed with all unix systems
various devices supports it (like cisco routers)

and, lots of apps
-----
already use it



```
/etc/syslogd.conf:

  *.debug
     /var/log/fulllog

kill –HUP <syslogd-pid>
```

getting basic logging working

-----

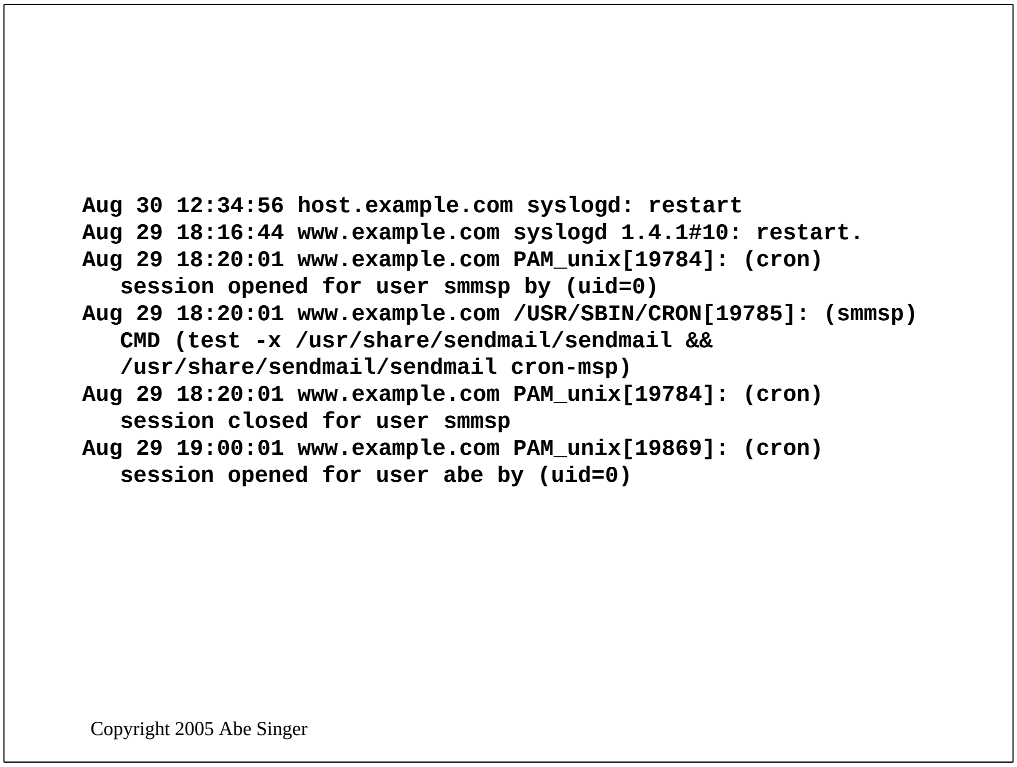
edit syslogd.conf, add this line:

-----

turns all all logging

----

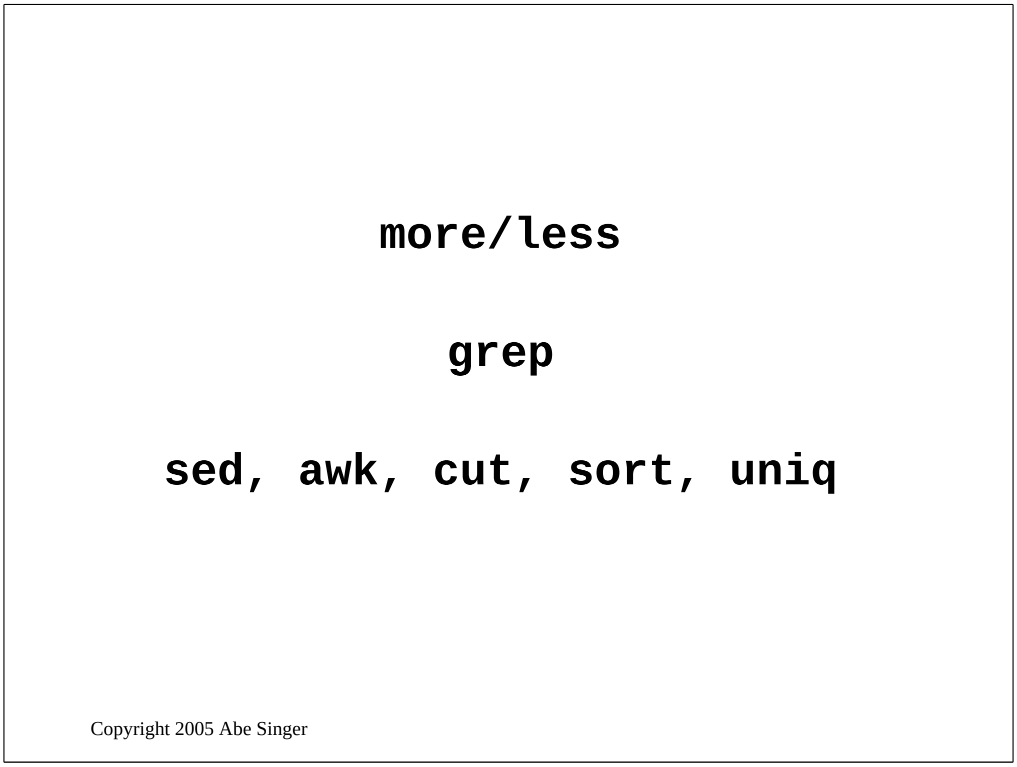
then send a HUP to the syslog daemon



```
Aug 30 12:34:56 host.example.com syslogd: restart
Aug 29 18:16:44 www.example.com syslogd 1.4.1#10: restart.
Aug 29 18:20:01 www.example.com PAM_unix[19784]: (cron)
   session opened for user smmsp by (uid=0)
Aug 29 18:20:01 www.example.com /USR/SBIN/CRON[19785]: (smmsp)
   CMD (test -x /usr/share/sendmail/sendmail &&
   /usr/share/sendmail/sendmail cron-msp)
Aug 29 18:20:01 www.example.com PAM_unix[19784]: (cron)
   session closed for user smmsp
Aug 29 19:00:01 www.example.com PAM_unix[19869]: (cron)
   session opened for user abe by (uid=0)
```

Here’s an example of what you see in the file

notice the first message is syslog restarting



more/less

grep

sed, awk, cut, sort, uniq

basic tools for working with logs.

start with more (or less)

-----

grep is incredibly useful.

-----

use it with –i

but it can take a long time on large files

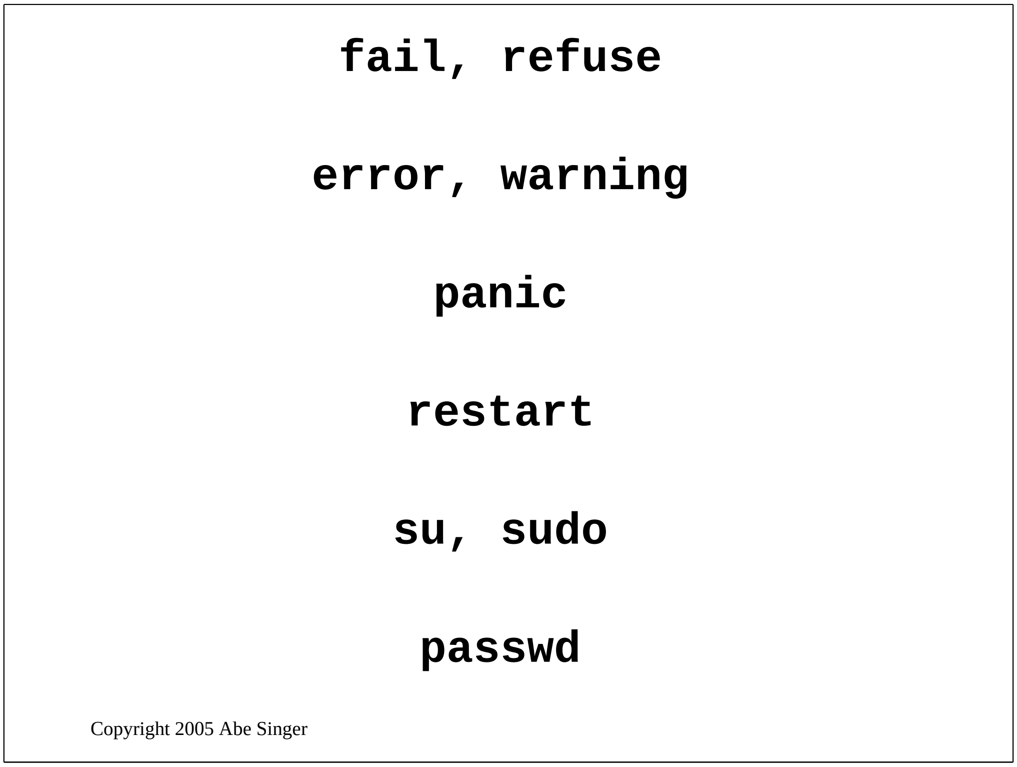
-----

sed and awk are perfectly useful

as is cut, sort, uniq.

And then there’s perl...

be very careful about using “vi” to look at your logs.



fail, refuse

error, warning

panic

restart

su, sudo

passwd

stuff to look for

-----

failures and things refused

-----

error, warning messages

-----

panic

-----

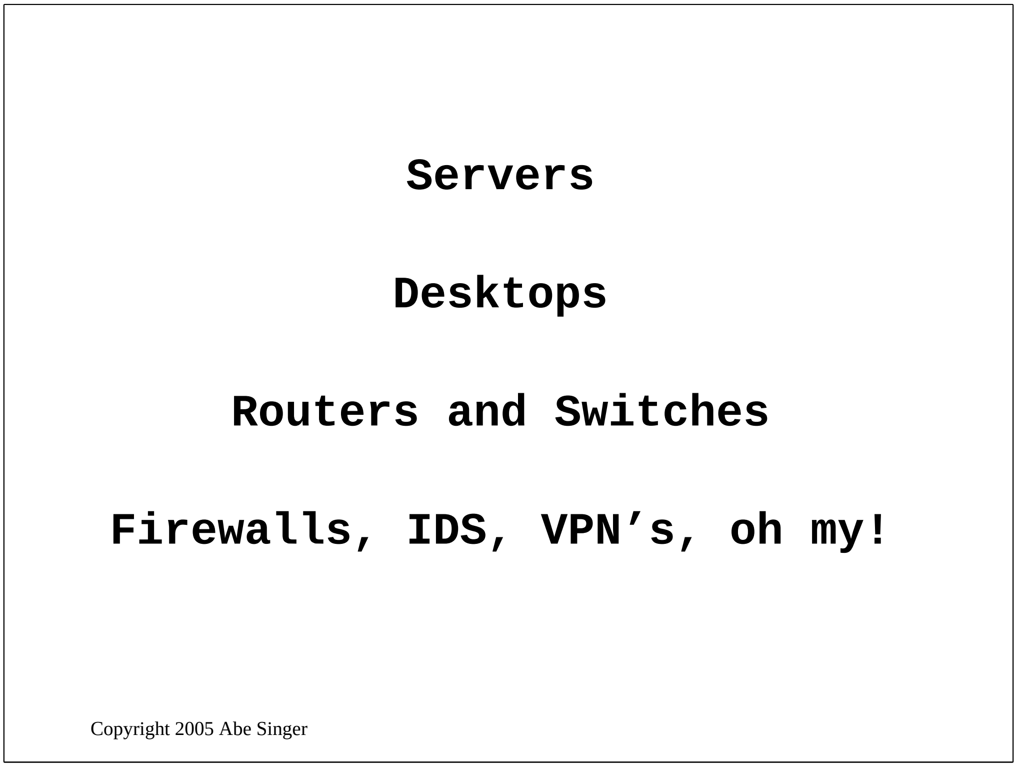
things restarting

-----

privileged access

-----

password



Servers

Desktops

Routers and Switches

Firewalls, IDS, VPN’s, oh my!

identifying source of log informaiton.

You can look at it two ways, one is, what devices you have, or what services.

So what you got?

servers

-----

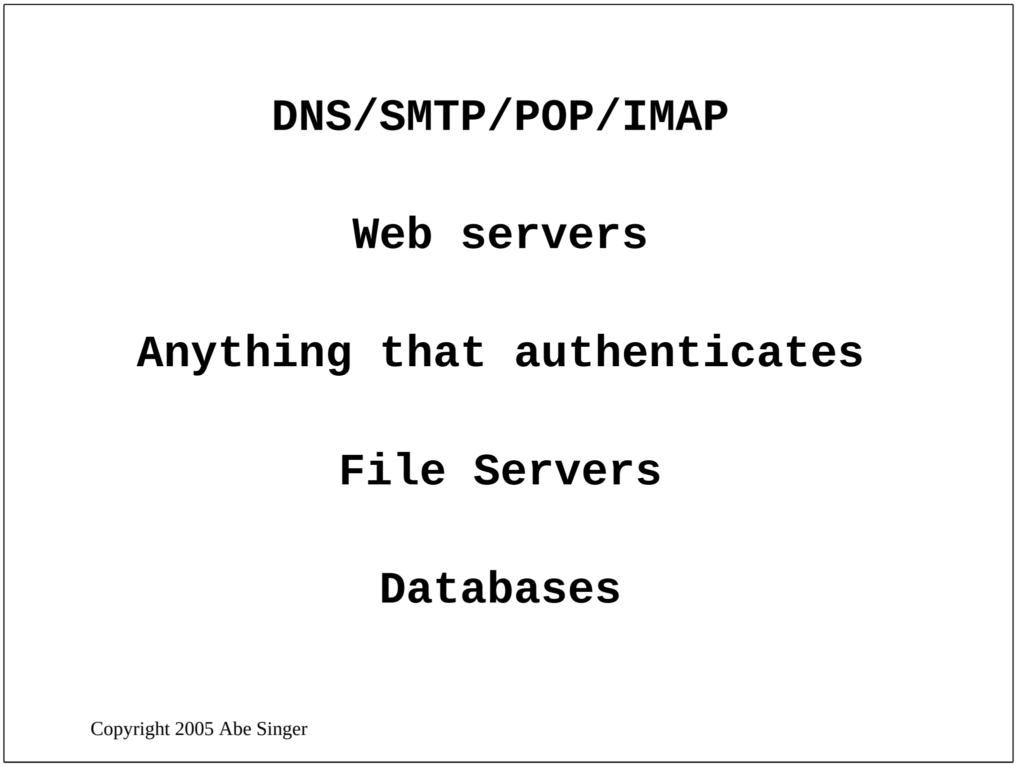
desktops

-----

routers and switches

-----

and other stuff



DNS/SMTP/POP/IMAP

Web servers

Anything that authenticates

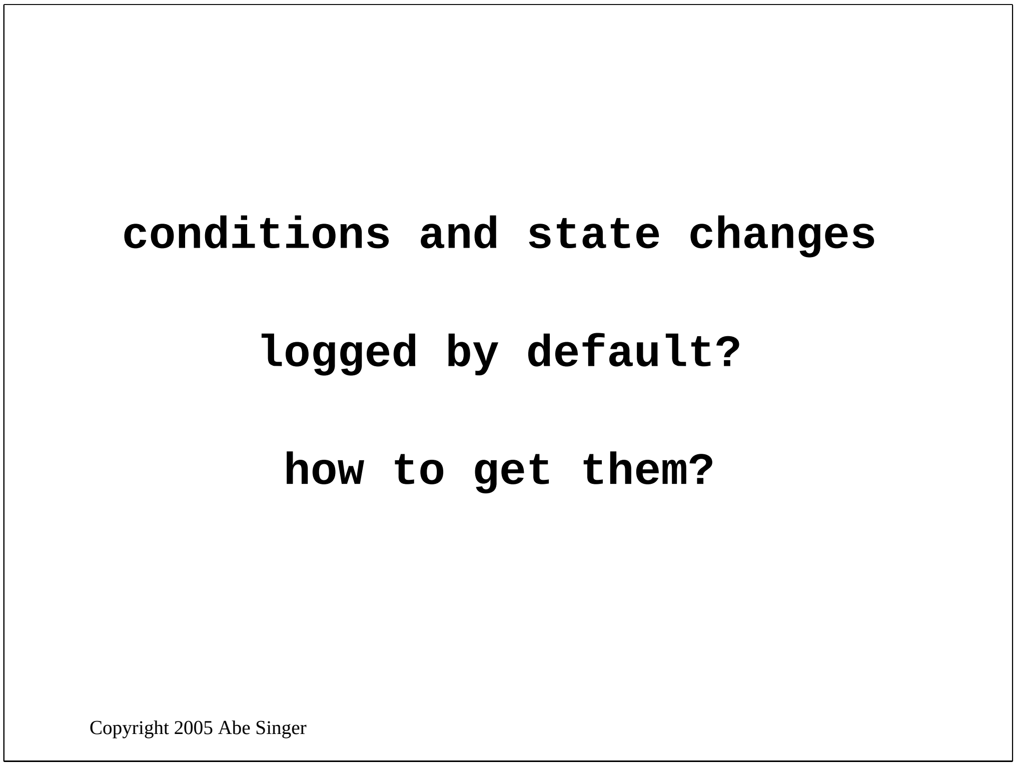
File Servers

Databases

and what sort of services might be interesting

there’s core infrastructure services...
-----

of course web servers
-----
anything with authentication
----
file servers are important, often the core
-----
databases for them database folk



conditions and state changes

logged by default?

how to get them?

and what do you want to know about them?

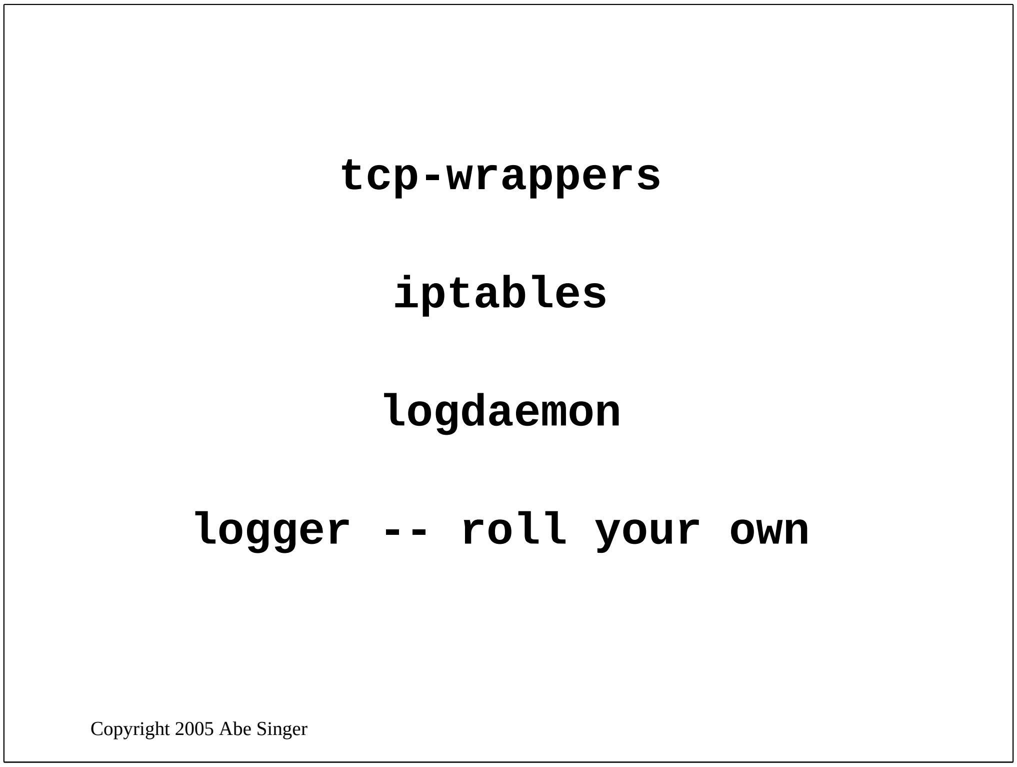
What conditions or “state changes” indicate malicious activity, component failure, or significant admin activity?

Do default logging mechanisms detect and record them?

If not, can we make them easier to detect?

-----

check configurations of applicaitons. some allow you to set logging levels and types



tcp-wrappers

iptables

logdaemon

logger -- roll your own

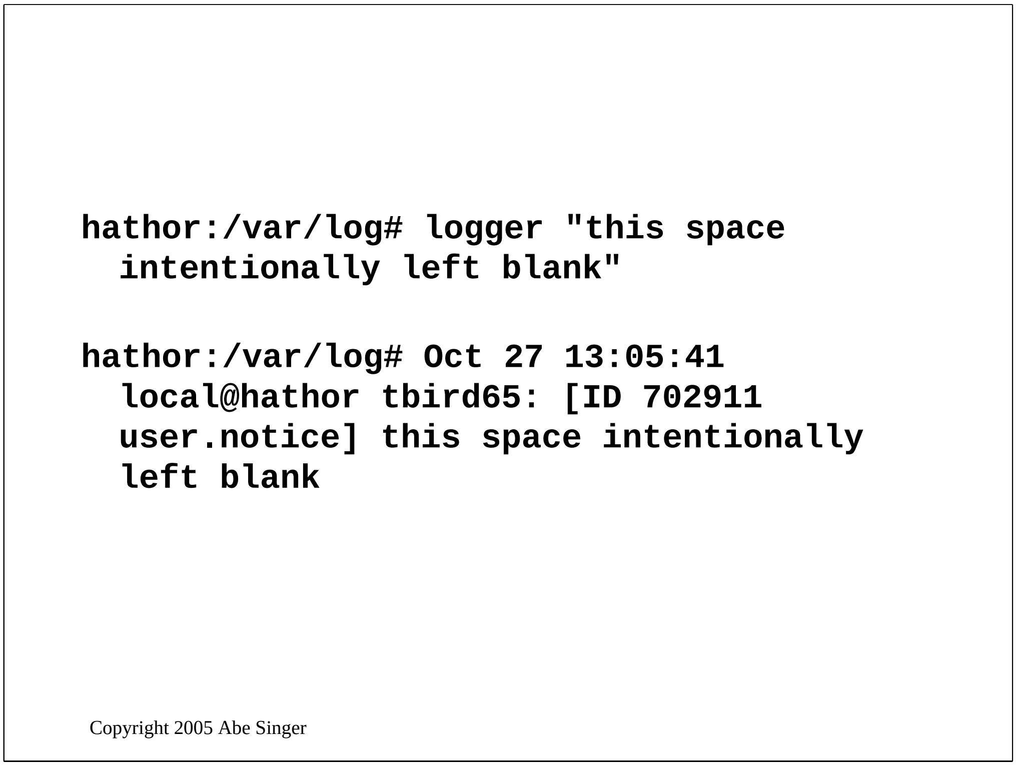
some ways you can get better logs

tcp-wrap things, even those that are open to the world
-----
useful for detecting probes on unused ports

use iptables to log “interesting” interesting traffic
-----
but be careful what you ask for!

the logdaemon suite of tools
-----
 replaces b0rken r-commands with ones
that log the remote host
(porcupine.org)

-----
and then there’s logger,



```
hathor:/var/log# logger "this space
  intentionally left blank"

hathor:/var/log# Oct 27 13:05:41
  local@hathor tbird65: [ID 702911
  user.notice] this space intentionally
  left blank
```

logger

UNIX command line utility writes arbitrary messages to *syslog*

As we’ll see in the configuration section, the `logger` utility makes it possible to get any data that can be expressed as text into the local system’s logging stream.

Centralized Logging

archive

correlate

preserve

Centralizing Your Logs

Why?

easier to archive

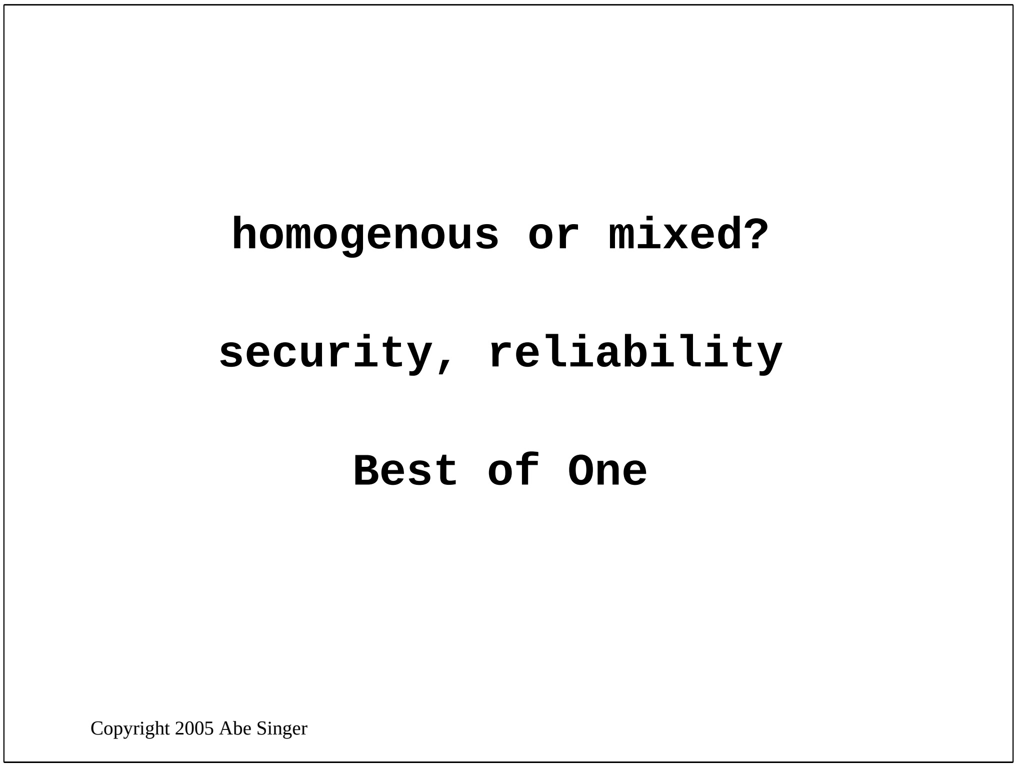
-----

easier to correlate

-----

log preservation if host is attacked

Mixed



homogenous or mixed?

security, reliability

Best of One

what kind of environment do you have?

Homogeneous or mixed?

Homogeneous unix: lucky you

    Built in mechanisms

Mixed environment:

syslog may not be a good choice

    security

    reliability

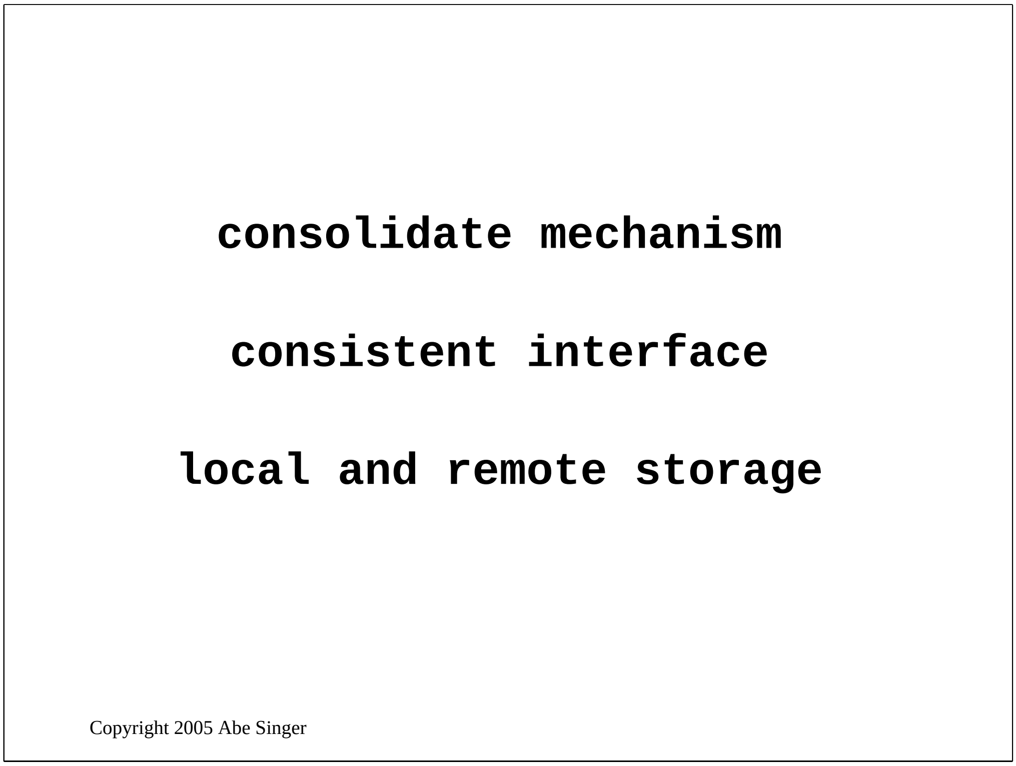
    -----

syslog may be the only choice

    most supported logging mechanism

So it's clearly the best choice!

In that infamous "Best of One" category...



consolidate mechanism

consistent interface

local and remote storage

The goal here isn’t to convince everyone on the planet to use *syslog*. But it’s been around a while, it is extremely flexible, and most of its features (and problems!) are common to other logging mechanisms. It’s a good model for how logging on all sorts of devices is performed.

Consolidated audit mechanism for UNIX kernel and application messages

-----

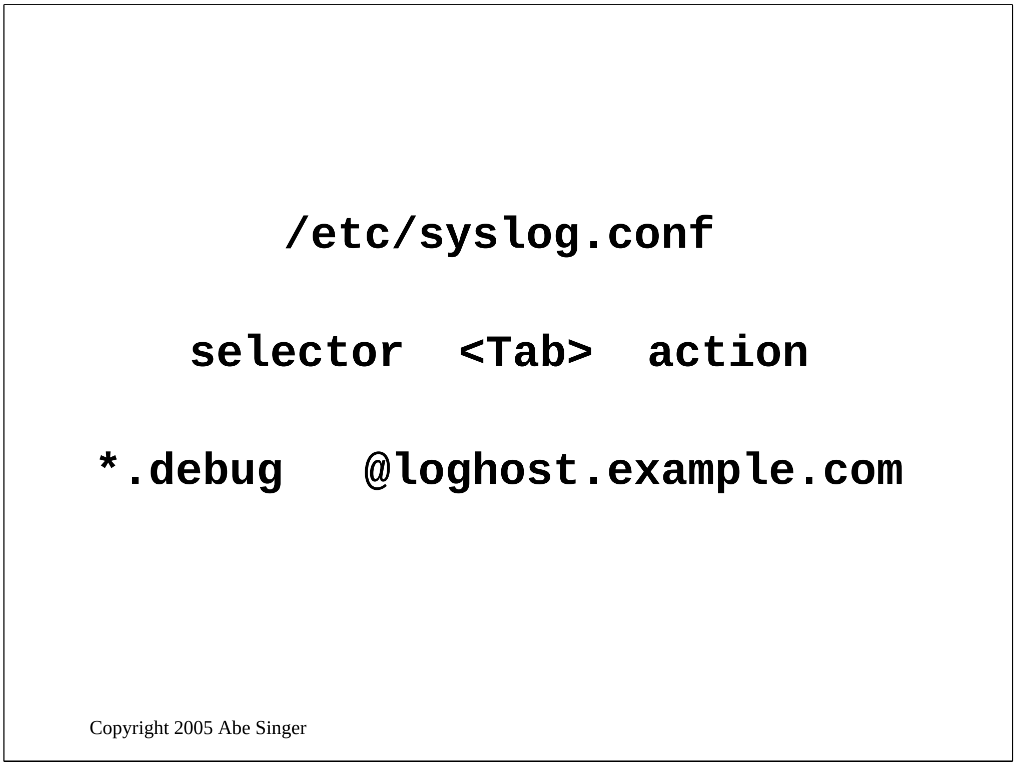
Gives application and OS developers a consistent interface for reporting significant events

-----

Allows local or remote storage of messages

-----

which is really important for a centralized architecture



/etc/syslog.conf

selector <Tab> action

***.debug   @loghost.example.com**

/etc/syslog.conf controls how much data is recorded, and what becomes of it

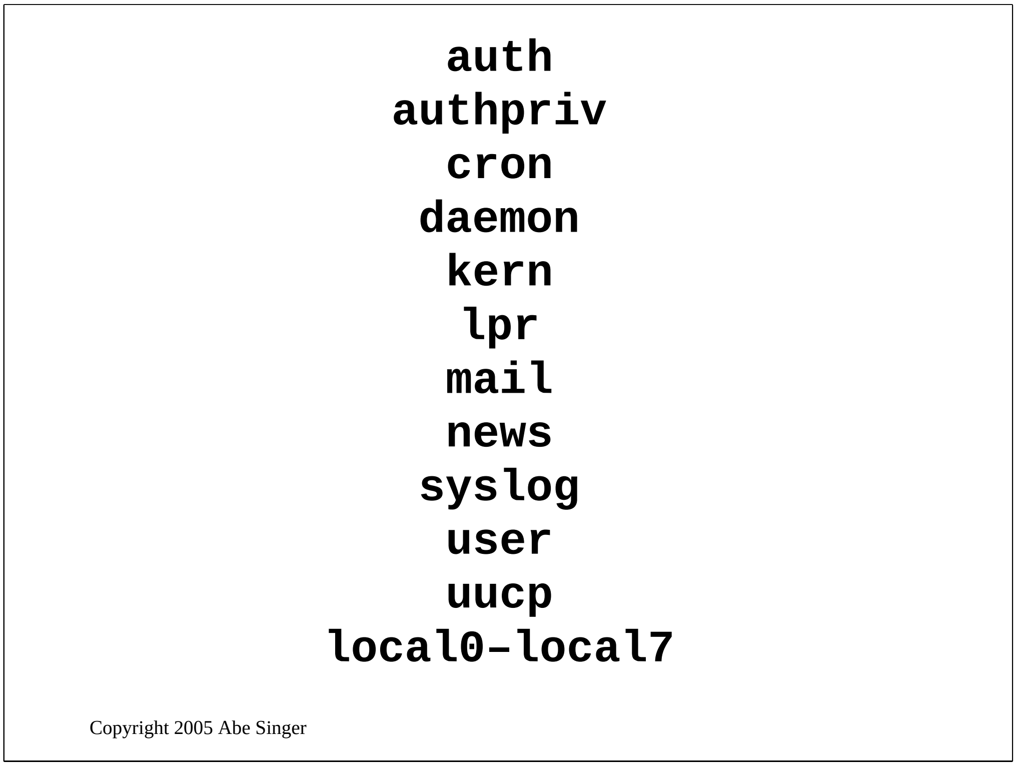
syslog.conf format:

`selector  <Tab>  action`

selectors indicate what's sending the message, and what criticality the message has

A sample */etc/syslog.conf* file from a Linux system reporting to a central loghost:

```
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.* /dev/console
*.*                    @172.18.1.34

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.debug;mail.none;authpriv.none       /var/log/messages

# The authpriv file has restricted access.
authpriv.*             /var/log/secure

# Log all the mail messages in one place.
mail.*                 /var/log/maillog
```

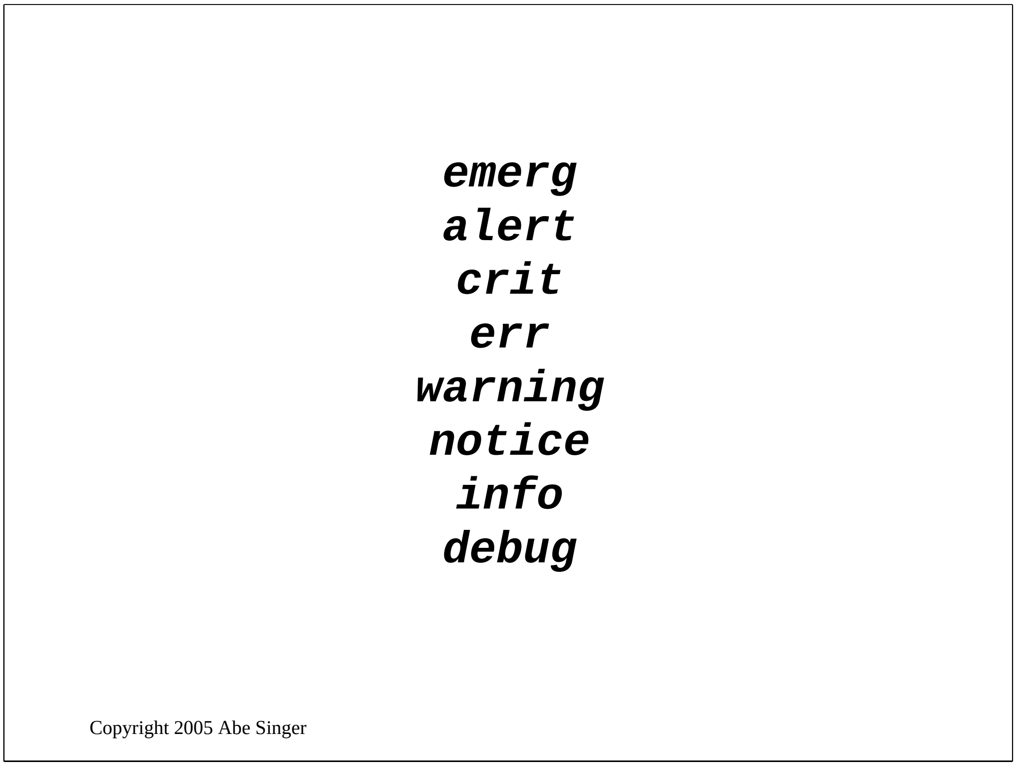


auth
authpriv
cron
daemon
kern
lpr
mail
news
syslog
user
uucp
local0–local7

facility – the application or system component that generates a log message

syslog standard facilities:

```
kern – kernel
user – application or user processes (default if
facility not specified)
mail/news/UUCP/cron – electronic
mail/NNTP/UUCP/cron subsystems
daemon – system daemons
auth – authentication and authorization related
commands
lpr – line printer spooling subsystem
mark – inserts timestamp into log data at regular
intervals
local0-7 – 8 facilities for customized auditing
syslog – internal messages generated by syslog
itself
authpriv – non-system authorization messages
* -- all facilities except “mark”
```



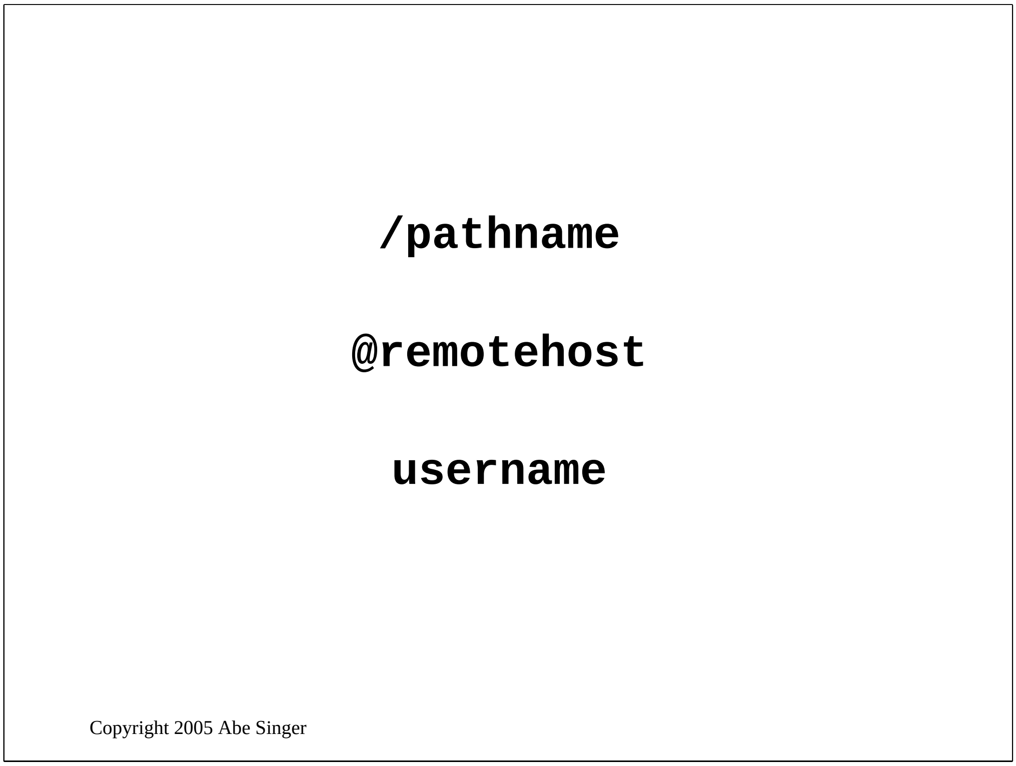
emerg
alert
crit
err
warning
notice
info
debug

levels are chosen by the programmer

syslog levels are nominally defined as:

`emerg` – system is or will be unusable if situation is not resolved
`alert` – immediate action required
`crit` – critical situations
`warning` – recoverable errors
`notice` – unusual situation that merits investigation; a significant event that is typically part of normal day-to-day operation
`info` – informational messages
`debug` – verbose data for debugging

Levels are sometimes expressed numerically, with 0 representing `emerg` (the most severe and least frequent) and 7 representing `debug` (the least severe and most verbose).



/pathname

@remotehost

username

action – what’s done with a message once it’s received from a facility

actions usually represent destinations – message is written to a local file, a syslog daemon on another system, the system console, or a user console

An action may be specified as

•a file on the local computer (for instance, */var/log/messages*)

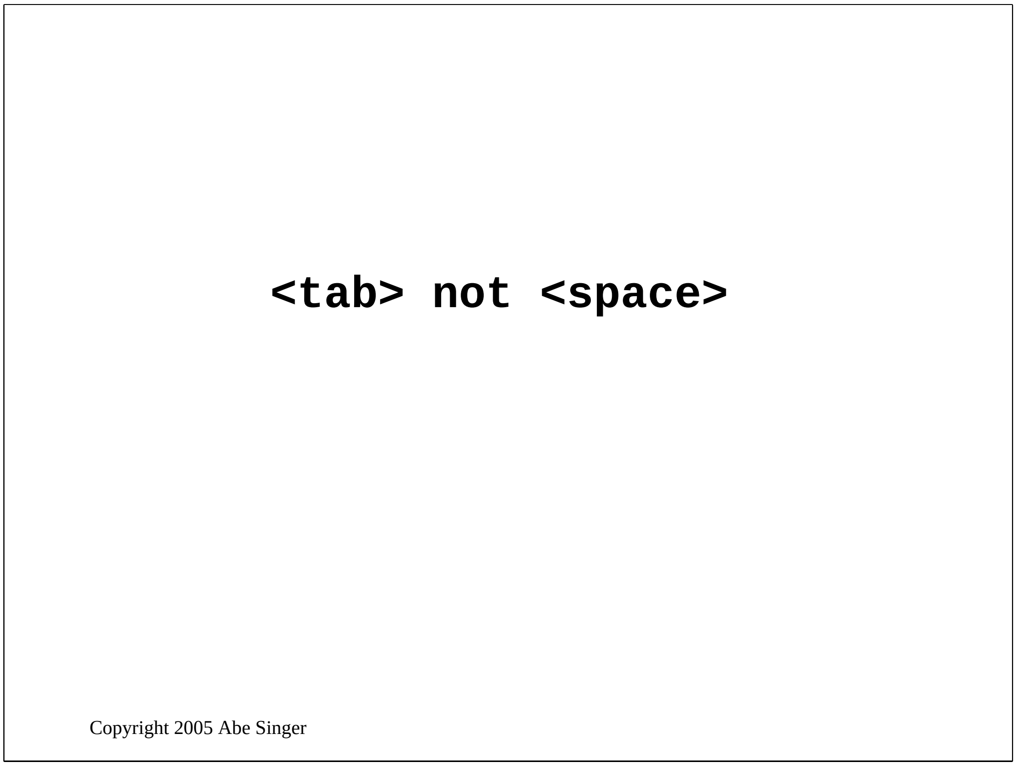
```
# Log anything (except mail) of level info or
higher.

# Don't log private authentication messages!

*.debug;mail.none;authpriv.none
            /var/log/messages
```

•the IP address or hostname of a remote system running *syslog* (designated by an @ and the hostname or IP)

```
# Log all system messages to the remote loghost.

*.*     @172.18.1.34
```

•a user’s login ID (only works if the user is logged in)

```
# Send emergency messages to tbird and root

*.emerg     tbird,root
```



`<tab> not <space>`

An oddity to remember

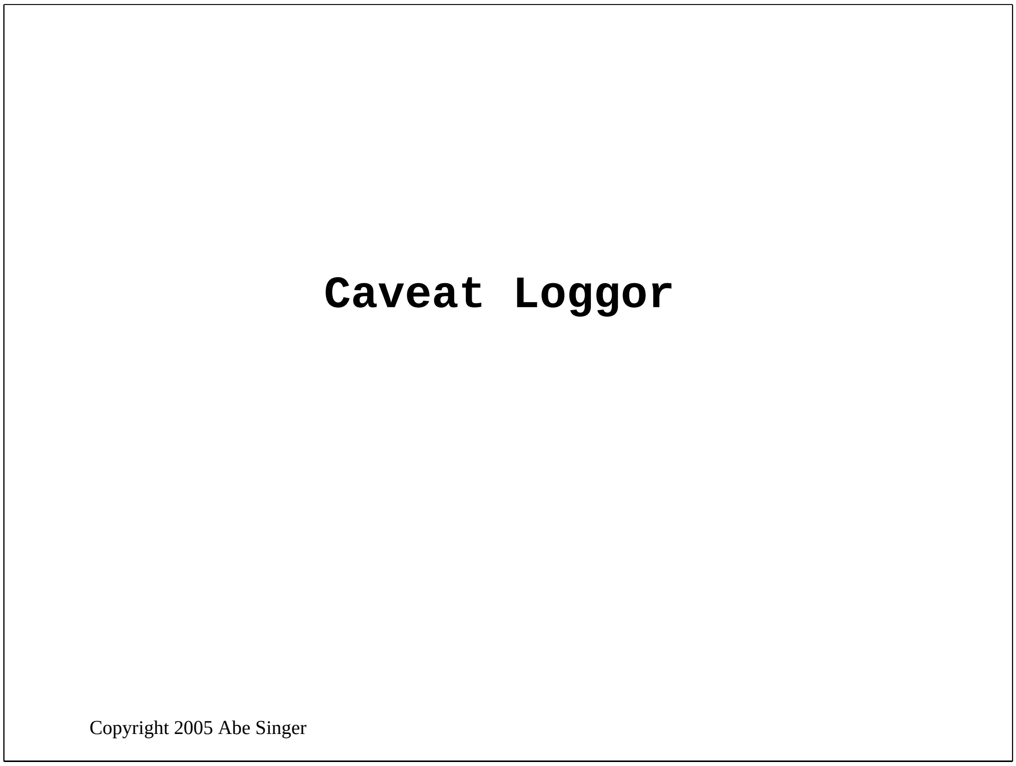
Many *syslogd*s require `<Tab>` as delimiter, not whitespace, & die gory, unpleasant, hard-to-detect deaths if `<Tab>`s are not present

Fixed in SDSC-syslog, syslog-ng, *sysklogd*, some OS implementations (FreeBSD)

SDSC High Performance *syslog* – http://security.sdsc.edu/software/sdsc-syslog/

sysklogd – http://freshmeat.net/projects/sysklogd

syslog-ng – http://www.balabit.com/products/syslog_ng/



Caveat Loggor

syslog only records what you've told it to record

Vast majority of events on a system are *not* recorded – events must generate logs to show up in log monitoring

Failed attacks often leave tracks;

successful attacks are often only recorded indirectly

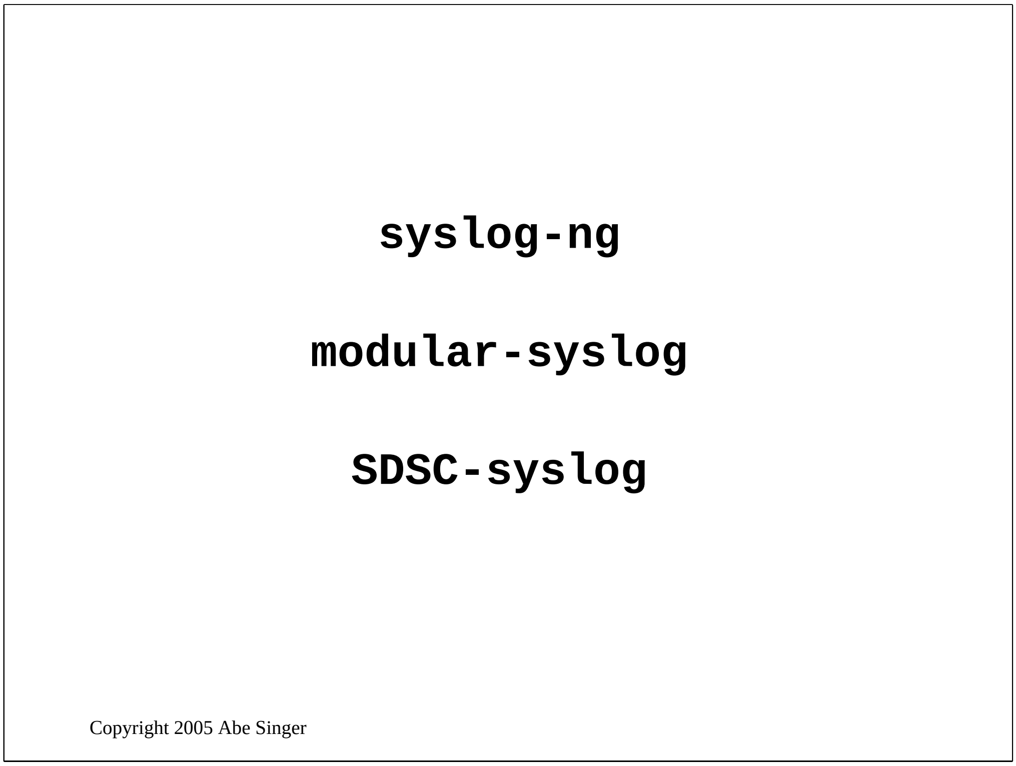
No default limitations on data sources (users or processes), so *all* log data is inherently unreliable

Nothing to prevent forged data from being inserted into data stream

Limited number of actions possible on receipt of a particular message

To put this another way, "you can't know what you don't know." Most default configurations of audit systems record relatively small amounts of information about the actions of users on the system – primarily because if all actions are recorded, the amount of audit information generated rapidly exceeds the system's storage capacity. We'll discuss a few ways to balance the need for forensic information on system use with the limited computing resources available within most organizations.

In addition, a lot of information that would be very useful when analyzing a successful attack – such as records of network connections – requires the addition of third party applications (like *tcpwrappers* and *tripwire*) to a



syslog-ng

modular-syslog

SDSC-syslog

syslogd replacements

Improved ability to filter and redirect inbound log messages

Integrity checks on locally-stored logfiles

Store more information about log data and events

Fix that whole `<Tab>` problem

Retain compatibility with classic *syslog*

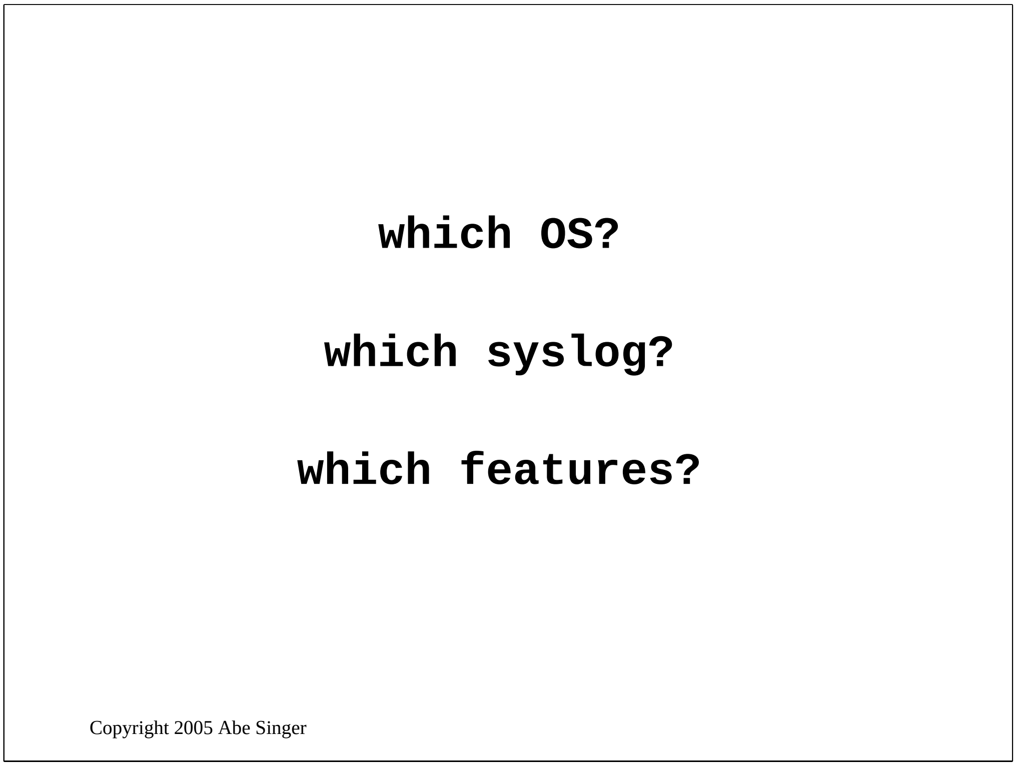
syslog-ng: most popular replacement; allows forwarding over TCP; remembers forwarding addresses; more granular message filtering

modular syslog: a syslog replacement that includes data integrity checks, easy database integration, and output redirection using regular expressions

syslog-ng: http://www.balabit.com/products/syslog_ng/

modular syslog: http://www.corest.com/products/corewisdom/CW01.php. The article at http://ezine.daemonnews.org/200112/log_protection.html is a good layman's introduction to the mechanisms *modular syslog* uses for its integrity checks.

Other *syslog* replacements are cataloged at http://www.loganalysis.org - click



which OS?

which syslog?

which features?

Decisions to make about loghost

Which operating system?\

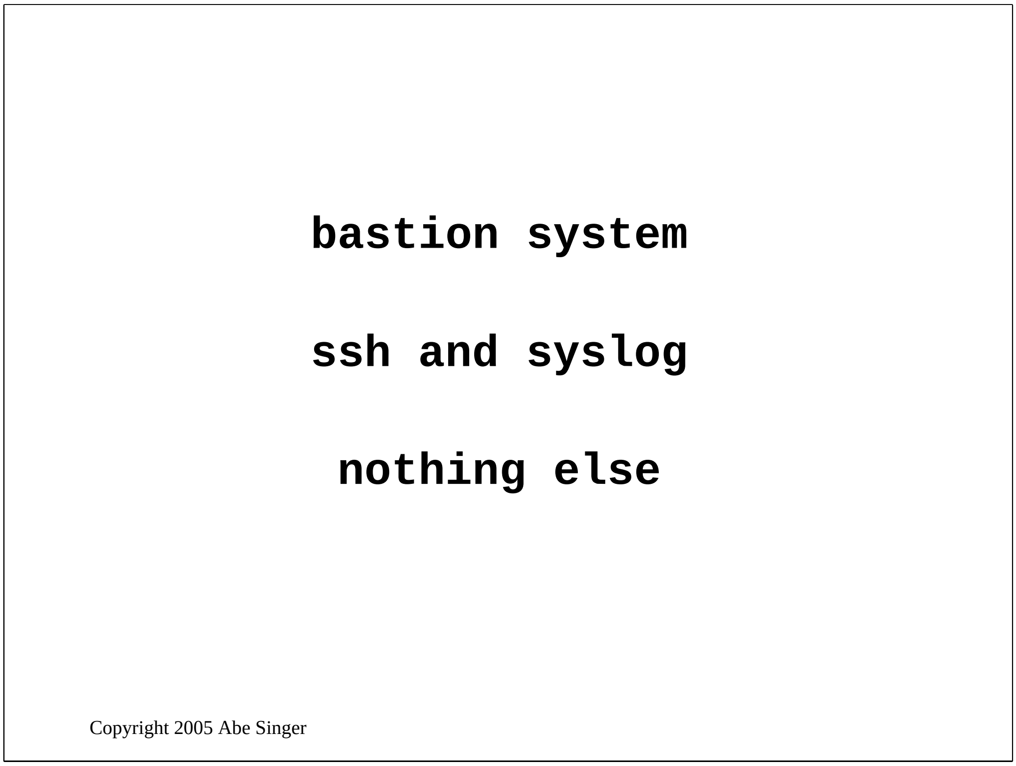
Most experience = easiest to harden vs.

Genetic diversity

Assuming *syslog*, which *syslog?*

Assuming *syslog*-the-protocol, out of the box, or (crypto, authentication) enhanced security?

A couple members of the Log Analysis mailing list use OpenVMS as the platform for their log servers. They collect the data over serial connections and use VMS tools to parse and monitor the data. It's certainly the case that there are far fewer hacker tools for breaking into VMS – despite its unfortunate association with Kevin Mitnick and friends – than there are for more commonly deployed operating systems like Solaris or Windows.



bastion system

ssh and syslog

nothing else

WHAT IS A LOGHOST?

Bastion system running limited services:

-----

archives and processes audit data

SSH or other secure protocol for administrative access

------

and does nothing else

For real paranoids: hide *syslog* configuration file

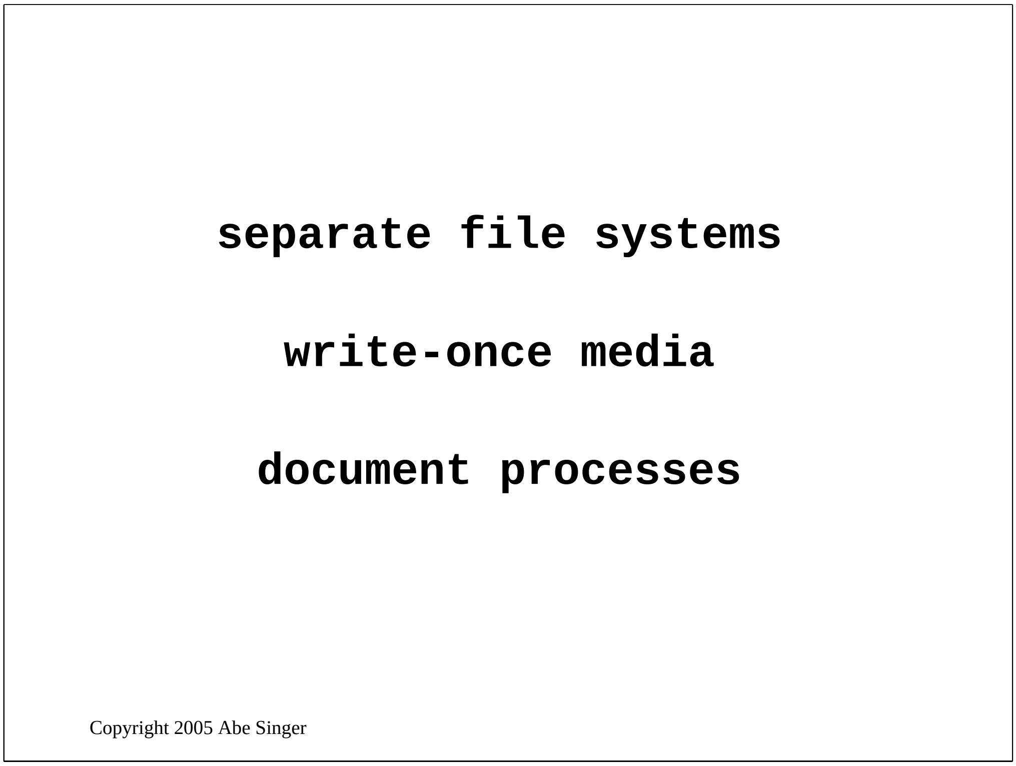
Or use a *syslog* replacement

Useful references: *Identify and enable system and network logging mechanisms*

http://www.cert.org/security-improvements/practices/p041.html

Configuring and using syslogd to collect logging messages on systems running Solaris 2.x

http://www.cert.org/security-improvements/implementations/i041.08.html



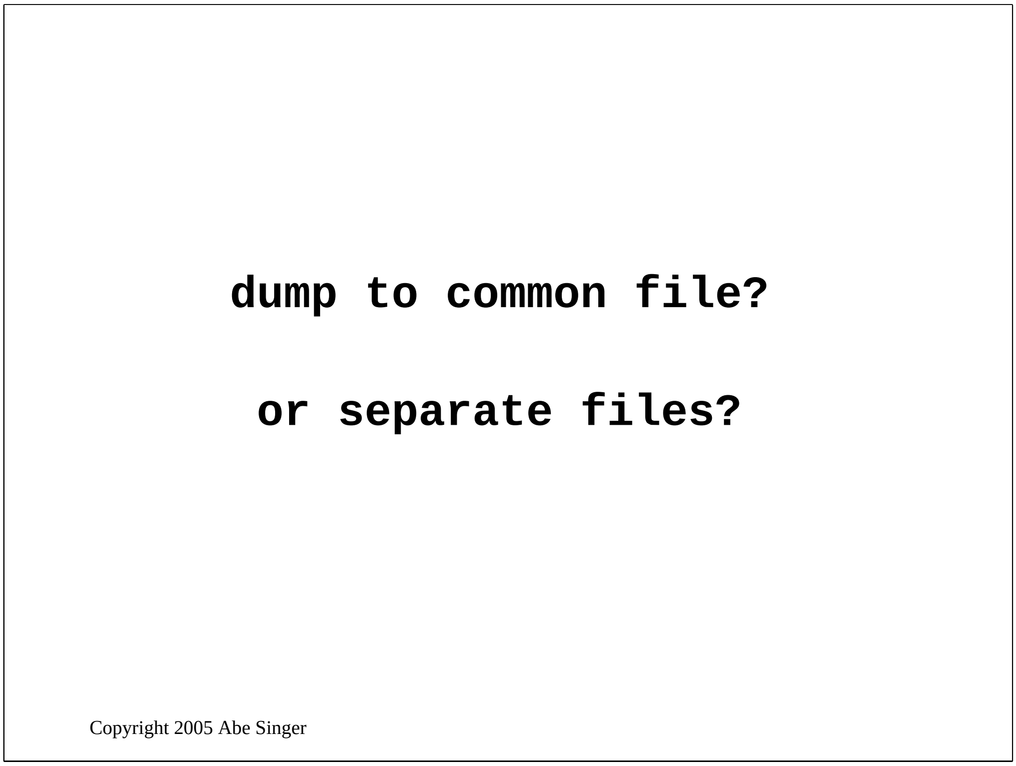
separate file systems

write-once media

document processes

Loghosts should have separate partitions for log data and operating system data and binaries. This way, an attacker can’t take down the entire loghost by filling its root partition, by sending spoofed *syslog* data.

It’s important to document all of the processes you use to build and manage your loghost. If you have to use your log data in court, this documentation helps convince judge and jury that you have a reliable datastream. For legal purposes, chain of evidence means that you can verify where the log data is at all times, that data transfers between personnel are documented at all times, and that data transfers between locations are documented at all times.



dump to common file?

or separate files?

syslog configuration on loghost: are new log messages dumped into common message file,

-----

or into specific files based on facility, or files/destinations based on severity?

Might want mail, Web (client & server), FW network connection logs handled separately

Mail server logs, proxy and Web server access logs, and firewall network connection records tend to be large, and relatively uninteresting – at least, they tend not to have a lot of *oh-my-god-page-me-at-3:13am* events. So parsing them off line is relatively safe, and will save processing capacity on your parsing system for the “real” events.

```
Single loghost

relay

stealth
```

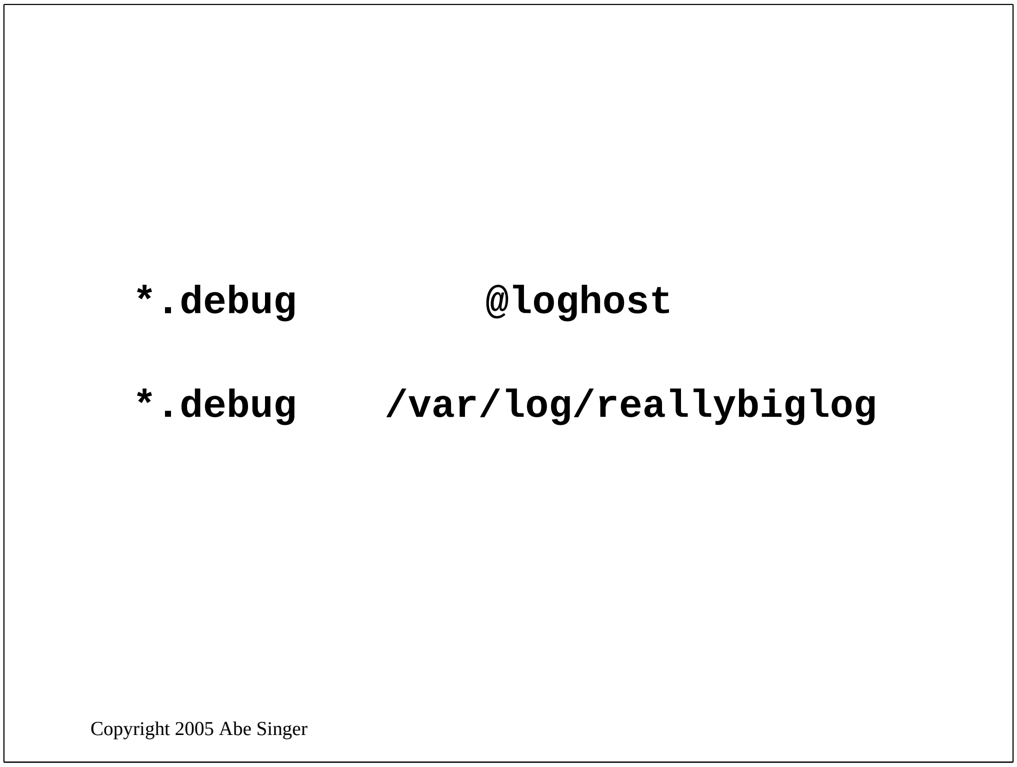
Popular Architectures

Single Central Loghost

Relay architecture – remote systems report to loghosts in branch; branch loghosts forward to central location for processing & archiving

Stealth logging for DMZ networks – monitoring Web, email servers

Logging over SSH for confidential data collection within private network



```
*.debug        @loghost

*.debug    /var/log/reallybiglog
```

setup for a single central loghost

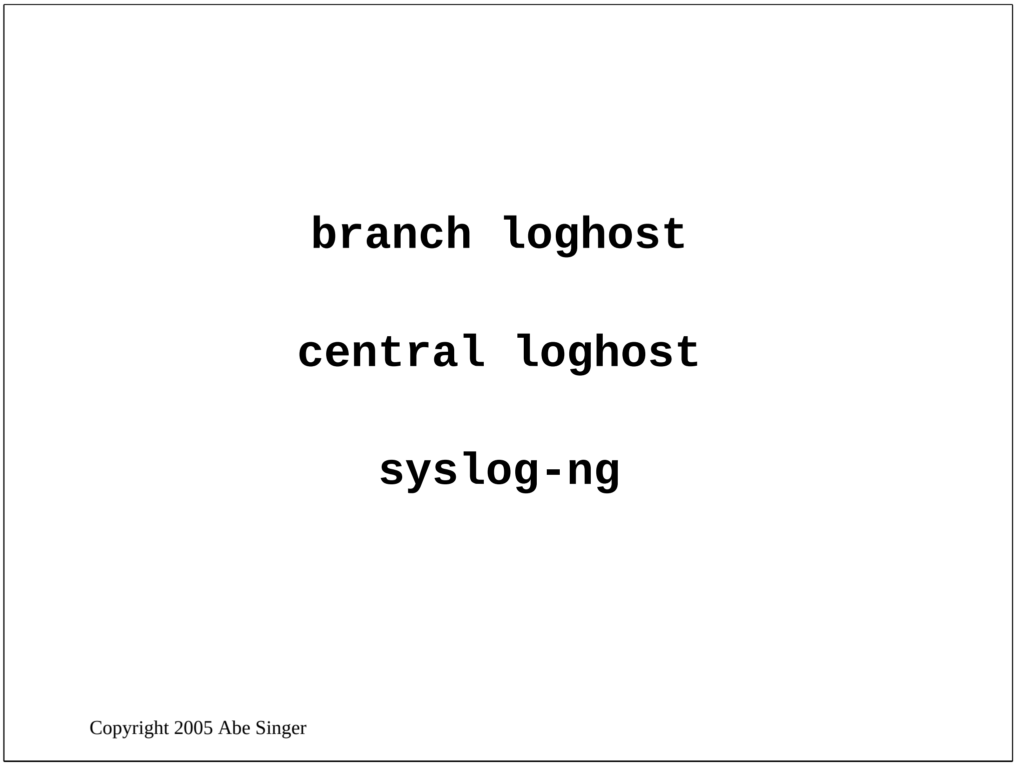
clients send logs to loghost

-----

and loghost dumps them to a file

------

can be a good idea to also log locally on the client machines



branch loghost

central loghost

syslog-ng

relay architecture described

Branch office loghosts: receive data from branch office servers, localhost; forward to central loghost

-----

Central loghost: receive data from branch office loghosts; write to archive; process data

-----

syslog-ng

-----

to preserve source info

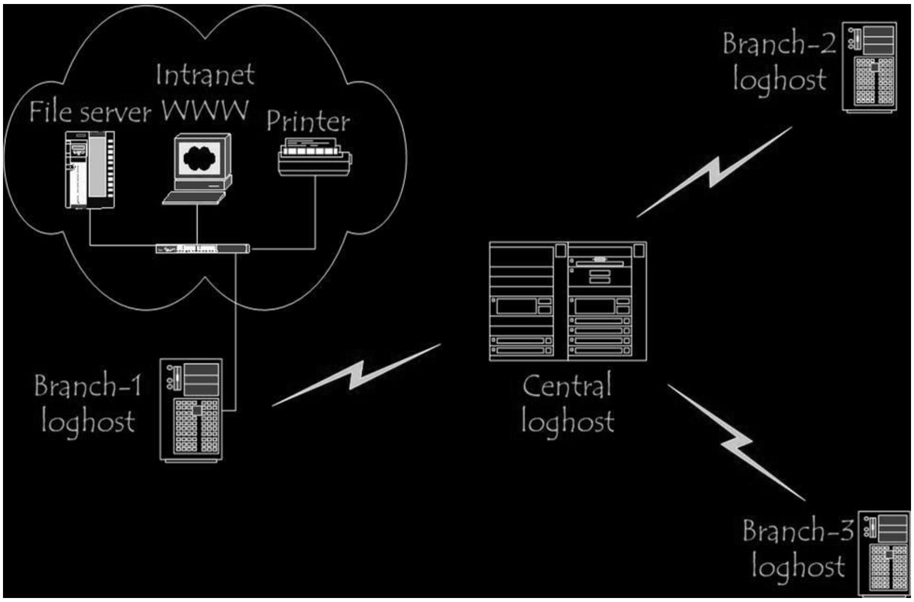
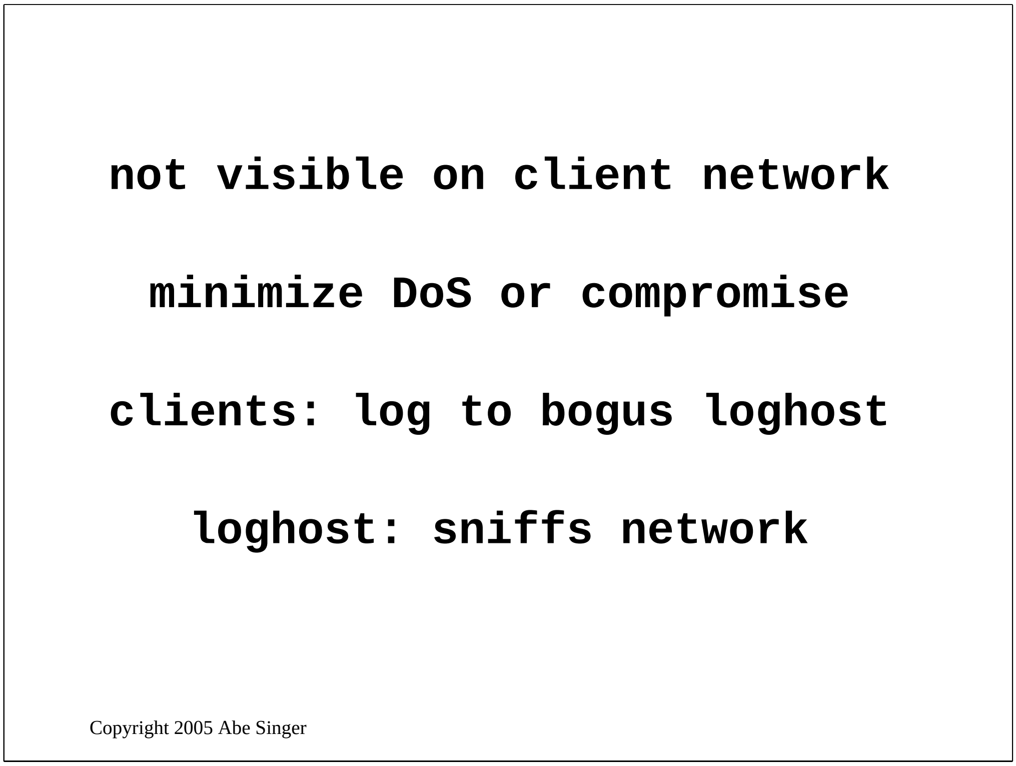


diagram of a relay architecture



not visible on client network

minimize DoS or compromise

clients: log to bogus loghost

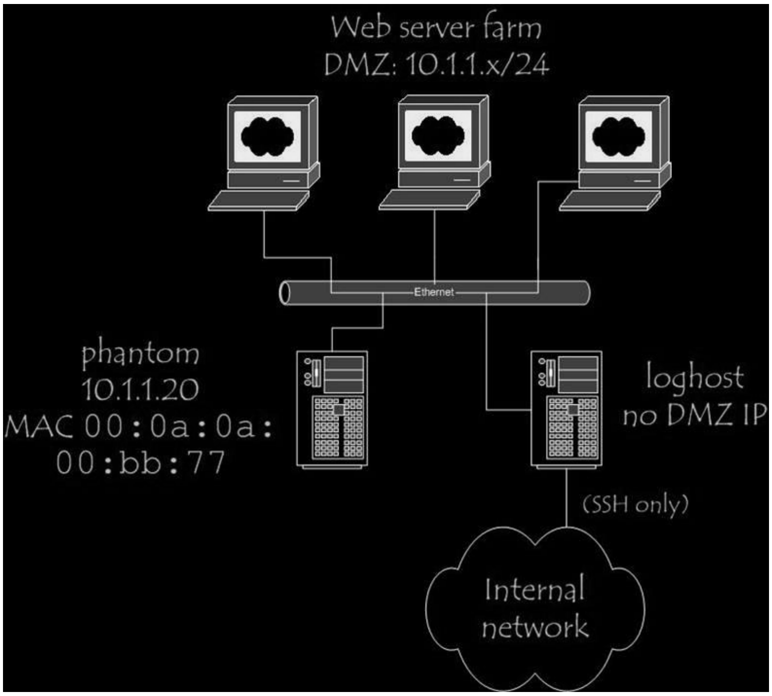
loghost: sniffs network

Stealth loghost

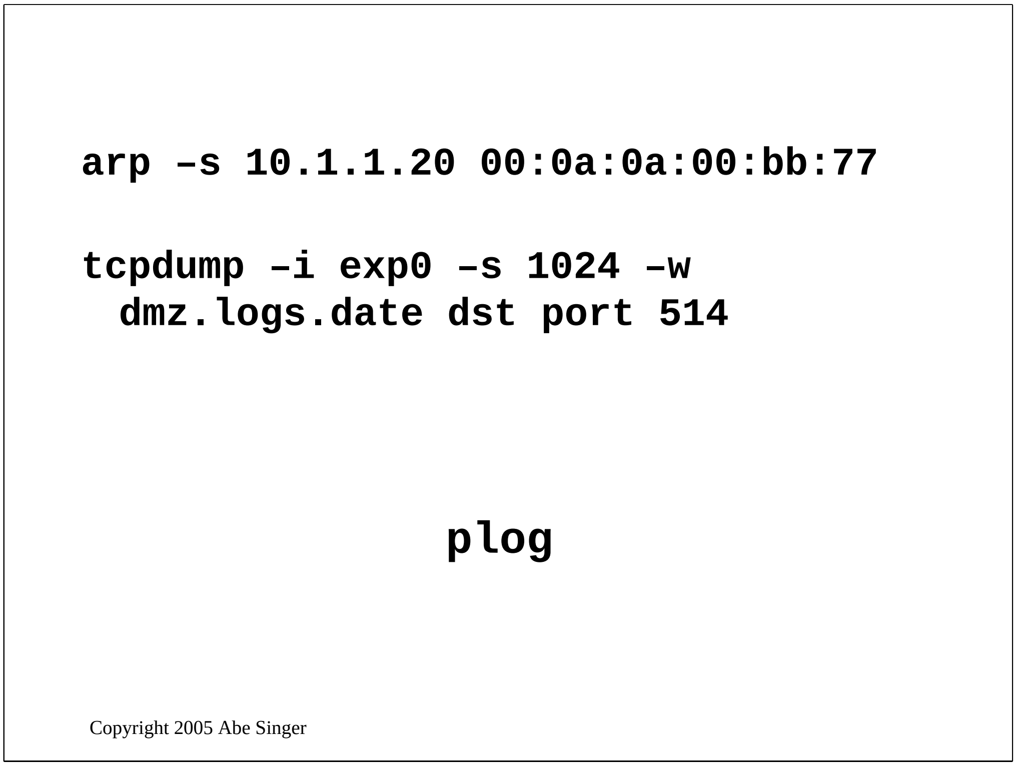
To collect data in places where you need to minimize chance of network-based DoS, or compromise of log server

Configure hosts and applications to log to a non-existent but valid IP address on DMZ

One more pretty good idea from Lance Spitzner, especially useful for honeypot servers (decoy systems set up to collect information about malicious activity) and production DMZ environments. An introduction to the idea is on line at http://www.linuxjournal.com/modules.php?op=modload&name=NS-lj-issues/issue92&file=5476s2



The idea is that the publicly-visible systems – the Web servers – will be configured to log to a "phantom" loghost, with IP address 10.1.1.20 and MAC address 00:0a:0a:00:bb:77. That system's not really there, making it pretty nearly impossible to break into (I'd say *completely impossible*, but we all know that's unrealistic). The "real" loghost has a network interface in promiscuous mode on the DMZ network – it eavesdrops on the UDP *syslog* traffic and records everything using *tcpdump* or Snort or something like that.



```
arp –s 10.1.1.20 00:0a:0a:00:bb:77

tcpdump –i exp0 –s 1024 –w
  dmz.logs.date dst port 514
```

plog

Configure Web servers with bogus *arp* entry for phantom logserver:

Loghost DMZ interface – no IP address, in promiscuous mode, connected to hub or span port on switch

Don’t forget to add your static *arp* entry to the system’s local start-up scripts, so it will continue to log successfully after reboots.

tcpdump puts interface into promiscuous mode unless told otherwise.

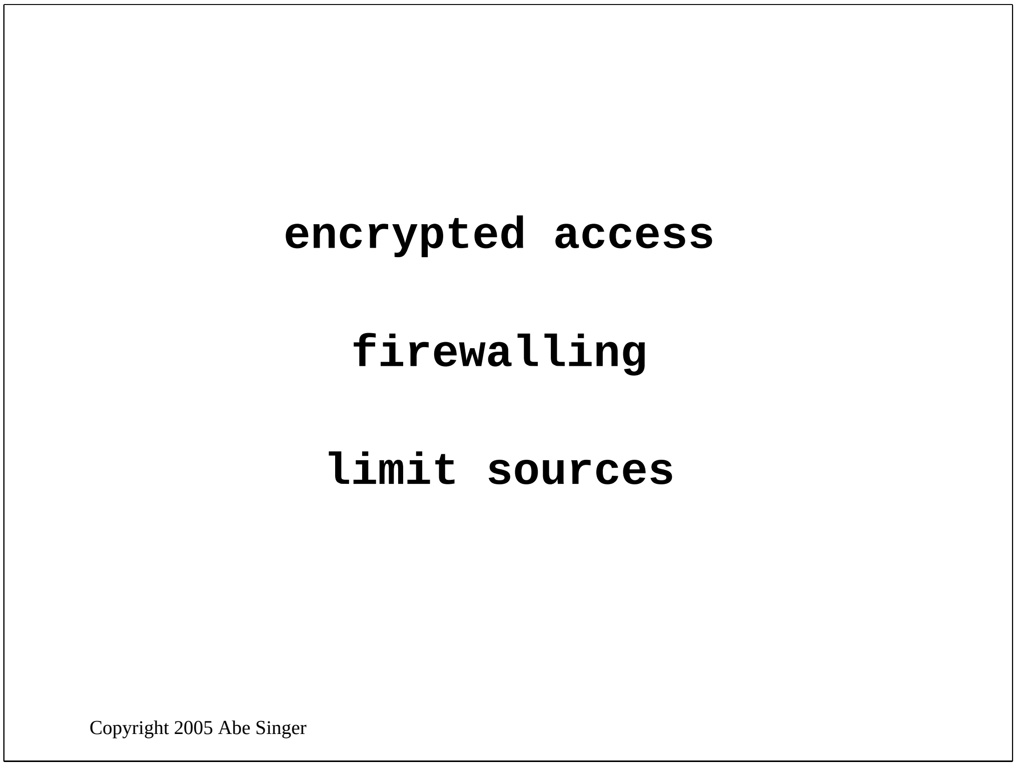
Assume loghost’s stealth interface is *exp0*

-----

tcpdump –i exp0 –s 1024 –w dmz.logs.*date* dst port 514

tcpdump – http://www.tcpdump.org

OR USE PLOG



encrypted access

firewalling

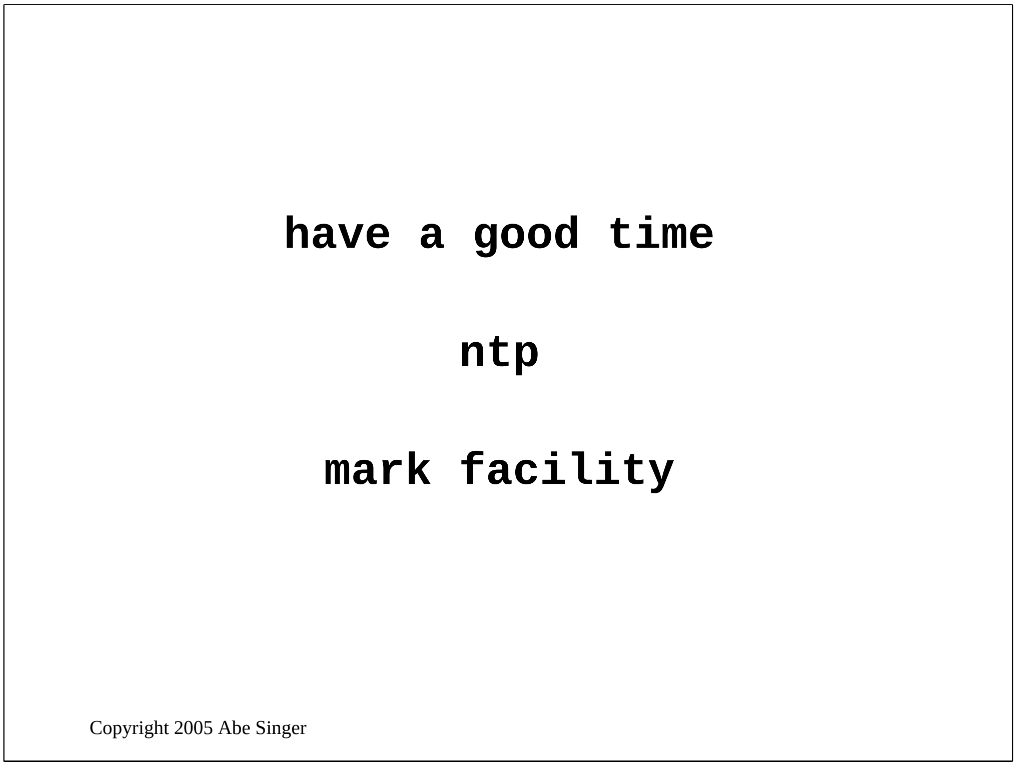
limit sources

some things to secure your loghost

Use encryption to limit access (via SSH tunnel, or one of the secure *syslog* replacements)

Built in firewalling on loghost (*ipchains*, *iptables,* etc)

Limiting which machines are allowed to send data to the loghost makes it a little harder for people to try to fill your loghost disks with garbage data. If you do this, be sure you allow SSH traffic to the server, at least from your own workstation! Again, configuration details are available in the linuxsecurity.com article referenced above.



have a good time

ntp

mark facility

time is important

Accurate time-keeping simplifies analysis and event correlation

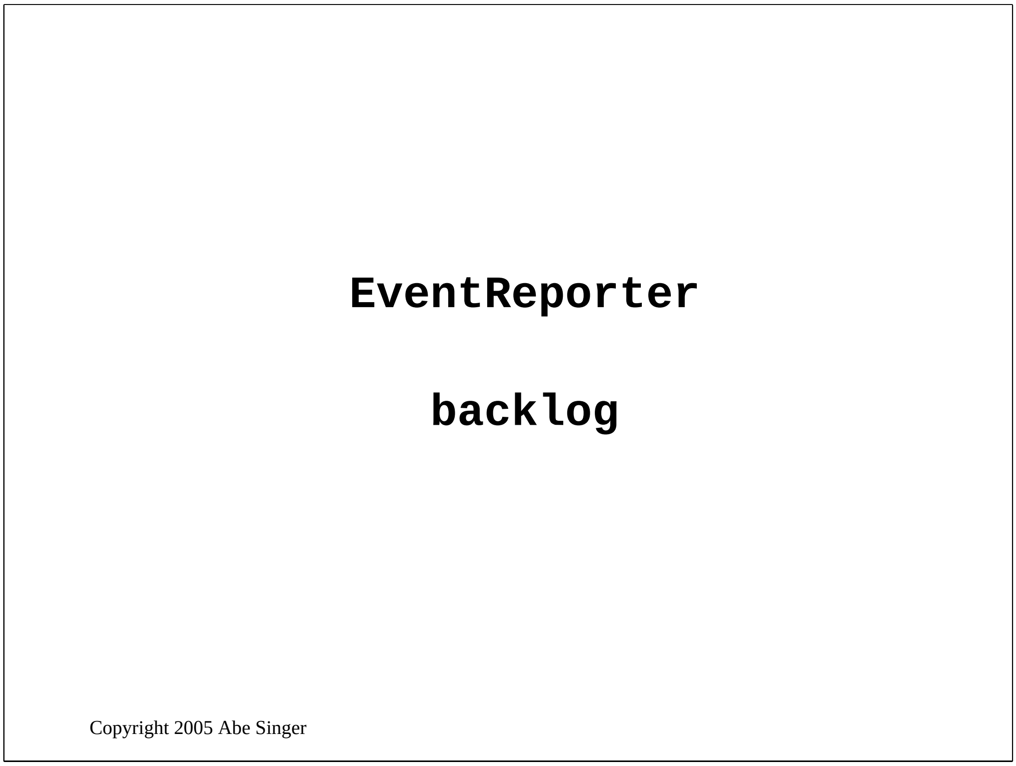
UNIX and Windows implementations of Network Time Protocol at
http://www.ntp.org/downloads.html

List of public timeservers at
http://www.eecis.udel.edu/~mills/ntp/clock1a.html

or use a GPS dongle like me ☺

`mark` facility produces internal timestamps at intervals selected by the admin

useful for verifying that logging is up and running, estimating lags in message delivery or other time synchronization problems\



EventReporter

backlog

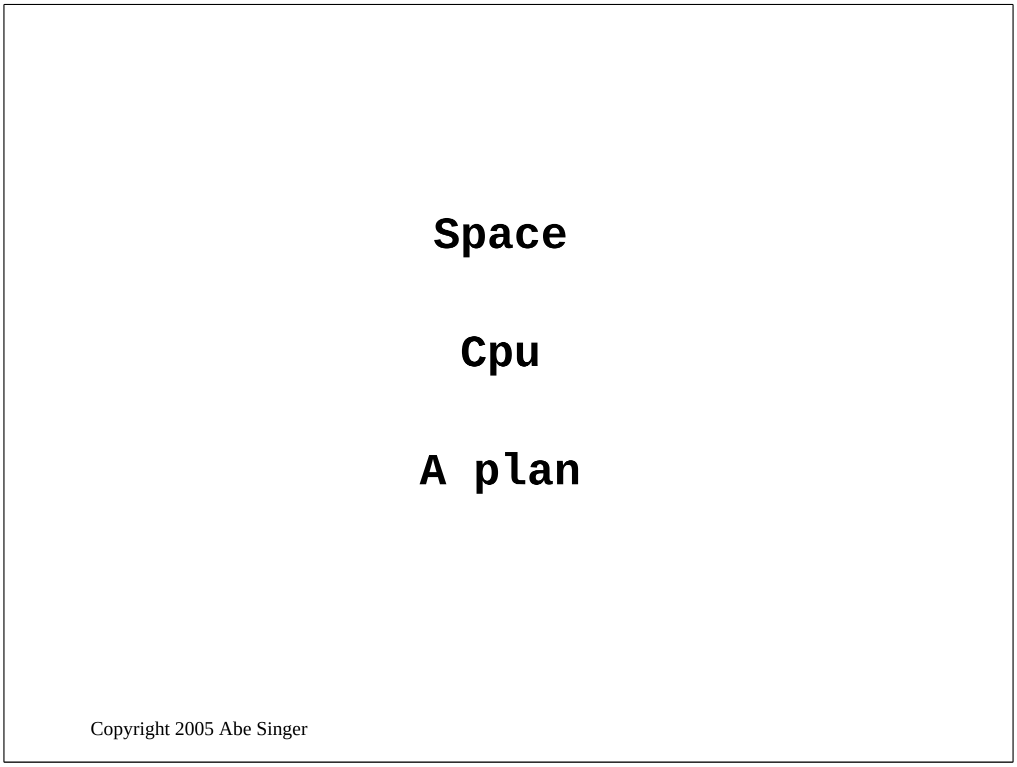
Third-party tools required to send Event Log data to remote loghost

Pure *syslog* clients:

- http://www.eventreporter.com
- BackLog http://www.intersectalliance.com/projects
- http://www.sabernet.net/software/ntsyslog.html
  - Perl module Win32::EventLog – allows external access to EventLog API

EventReporter and BackLog both provide a graphical interface as well as command line or registry edit capability. NTsyslog is command line only and is not (apparently) being maintained any more.

Log Analysis



Space

Cpu

A plan

basic log analysis strategy

you’re gonna need some CPU

-----

some disk space

-----

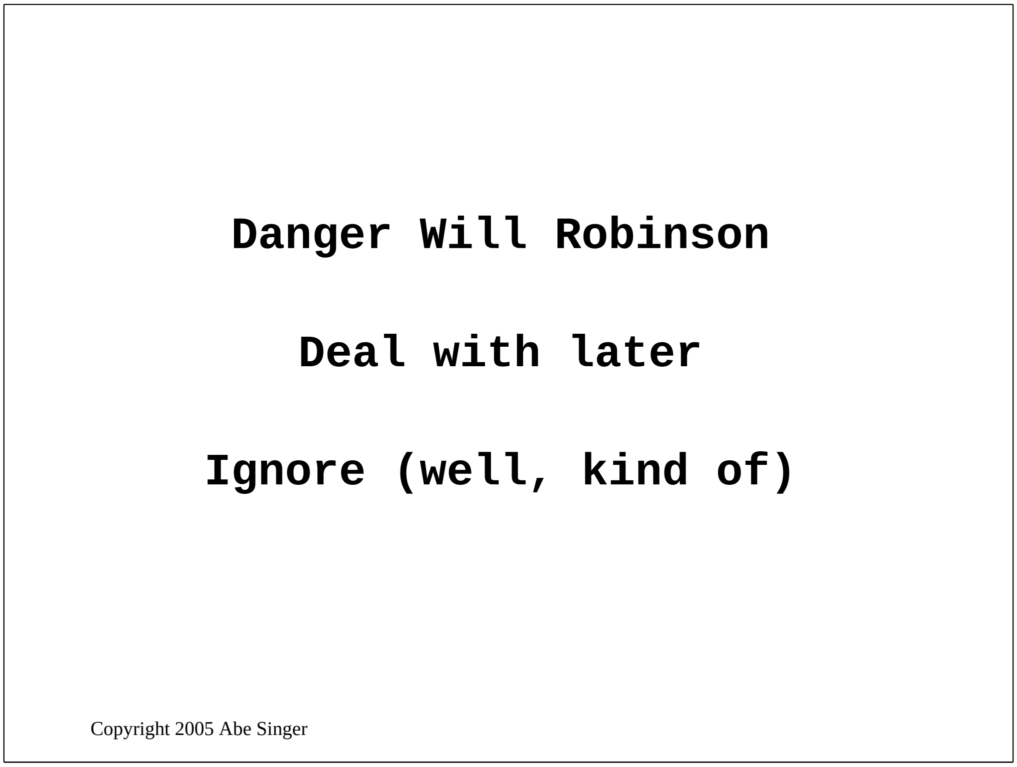
and a plan for what to do with it

-----

triage

parse-reduce-parse

analyze



Danger Will Robinson

Deal with later

Ignore (well, kind of)

triage means a first pass at weeding out the important stuff

separate into

stuff you need to know about right away, like attacks

-----

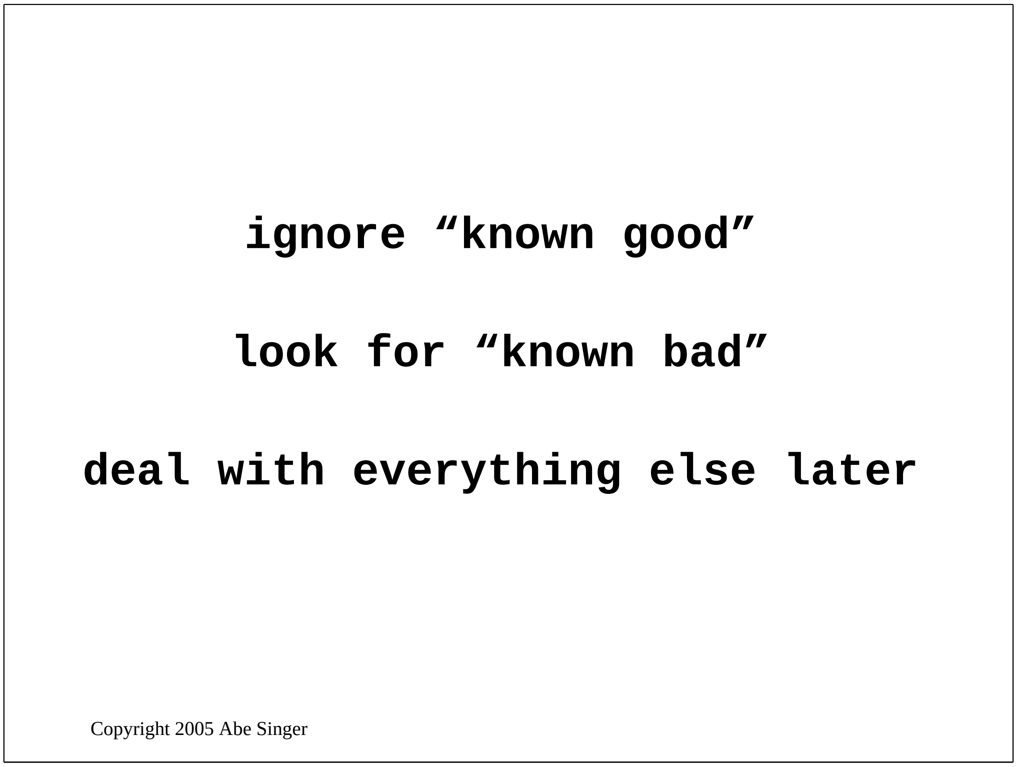
stuff to deal with later: status information, resource utilization

-----

and stuff to ignore

------

although you don’t really want to ignore it for some reasons

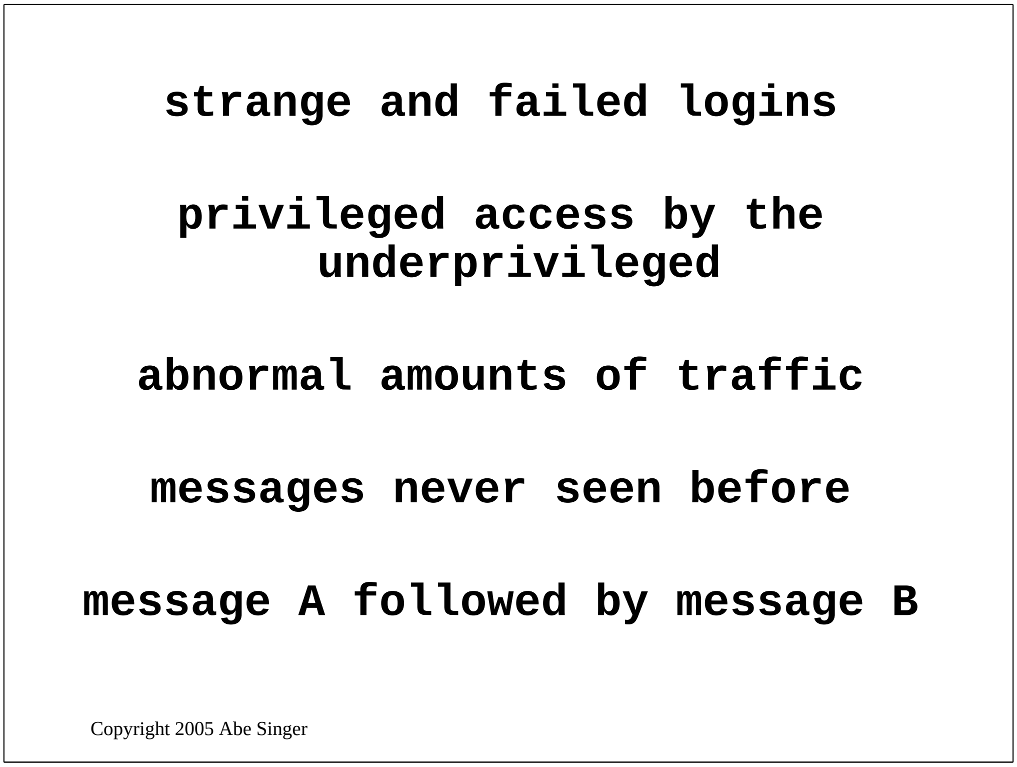


ignore “known good”

look for “known bad”

deal with everything else later

Then there’s Marcus Ranum’s



strange and failed logins

privileged access by the underprivileged

abnormal amounts of traffic

messages never seen before

message A followed by message B

some not-so-easy things to look for....

strange and failed logins

---

privileged access by the underprivileged

---

abnormal amounts of traffic

---

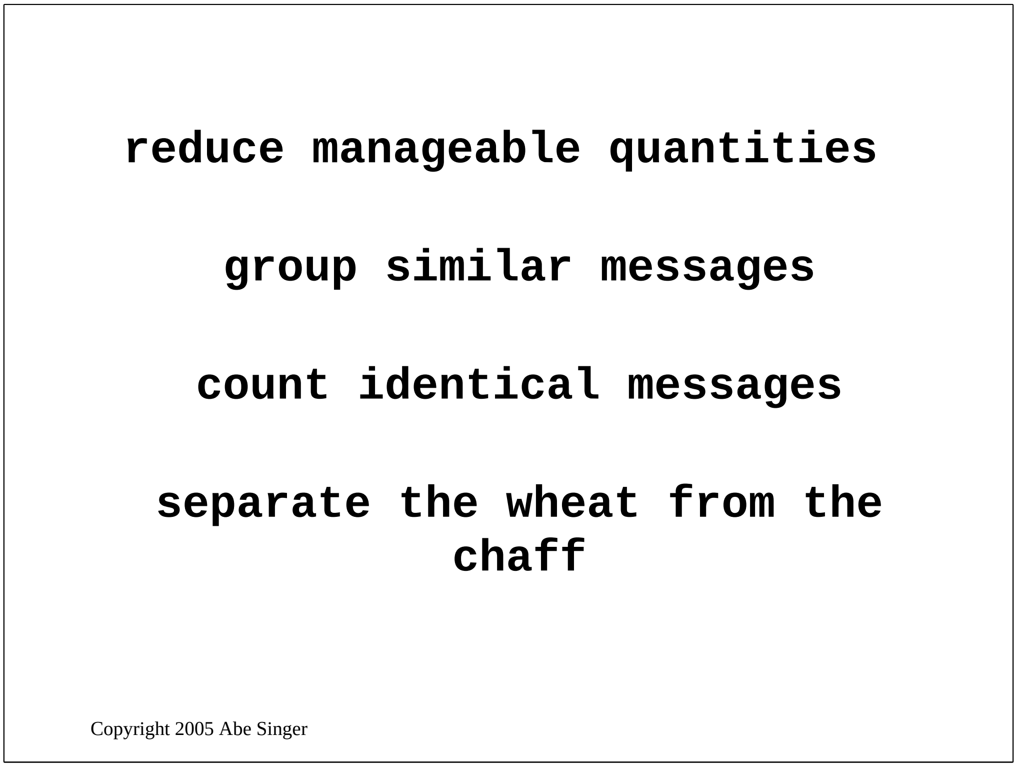
messages never seen before

---

message A followed by message B

---

or not followed by B



reduce manageable quantities

group similar messages

count identical messages

separate the wheat from the chaff

log reduction is about getting log data down to more manageable quantities

-----

the basic steps are to....

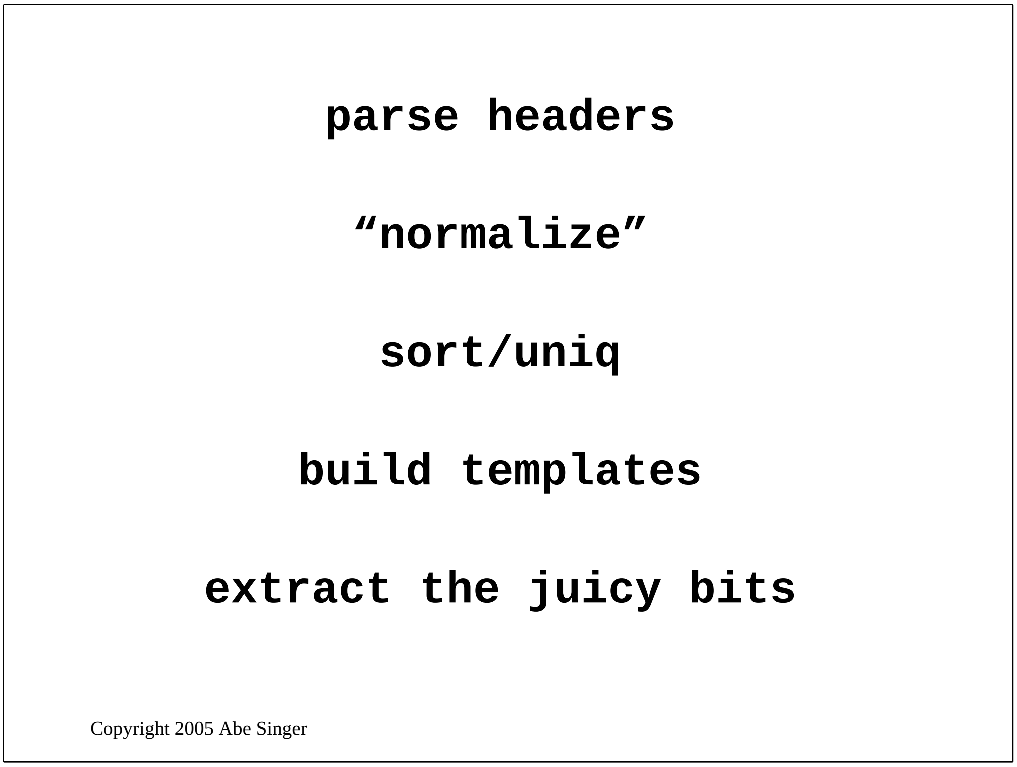
group similar messages for further processing

------

reduce identical messages to a sample and the number of occurences

-----

and basically separate the wheat from the chaff, get down to the useful pieces of information and get rid of the entropy



parse headers

“normalize”

sort/uniq

build templates

extract the juicy bits

an approach to data reduction that I use:

first, separate out the headers, which is harder than it looks.

-----

then normalize (for lack of a better word)

-----

lower-case, remove whitespace, replace IP addrs with placeholders

then you can sort/uniq it

------

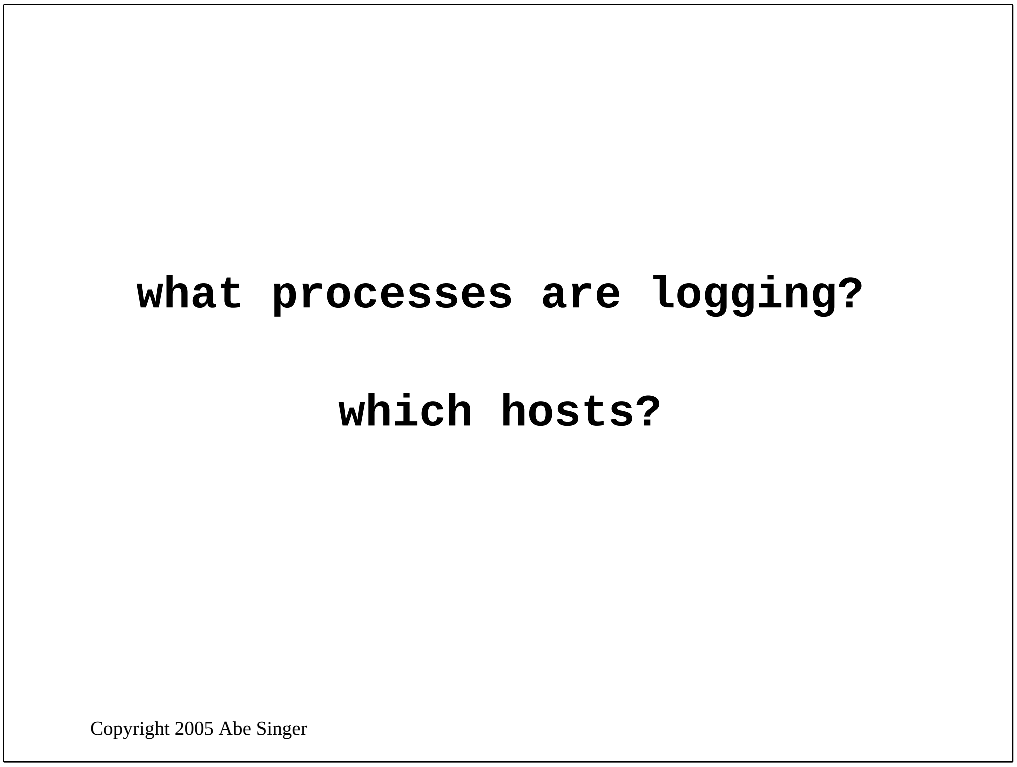
doing this much can reduce the data a *lot*

-----

then you can start building patterns to match messages

----

cuz the ultimate goal is to get the meaty parts, like remote IP address



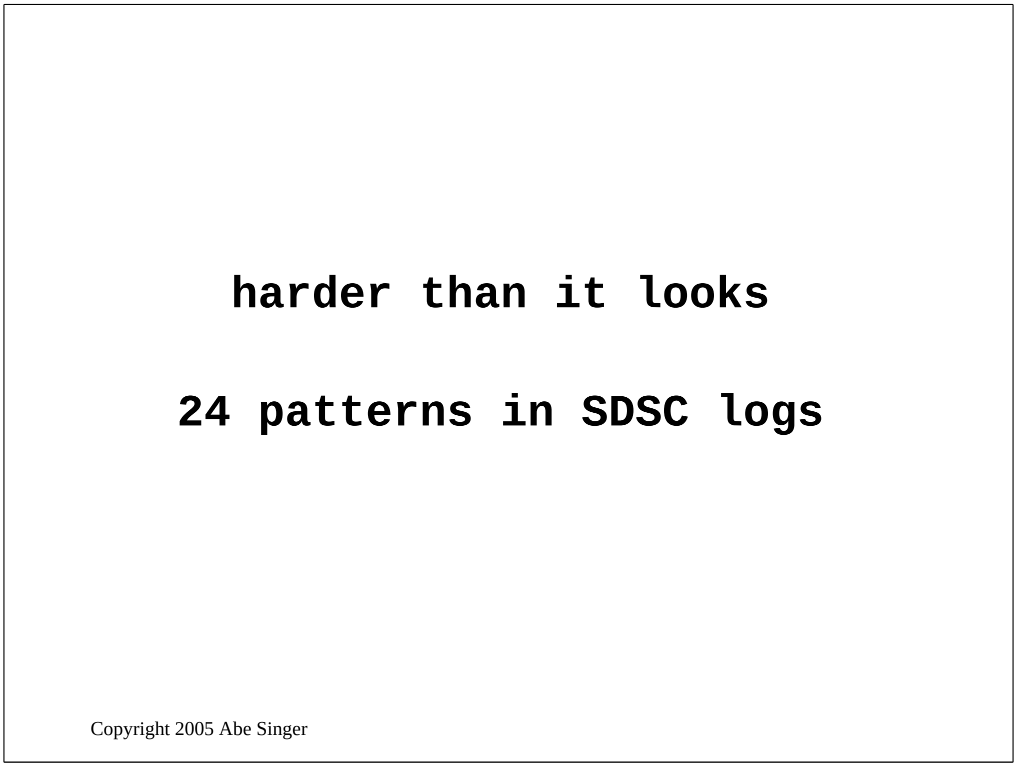
what processes are logging?

which hosts?

one of hte first things you can do to get a handle on your logs

what process are logging?
------
how many?
(375 at SDSC)

and which hosts are logging?
------
this takes a good ability to parse log headers



harder than it looks

24 patterns in SDSC logs

identifying headers is hard

-----

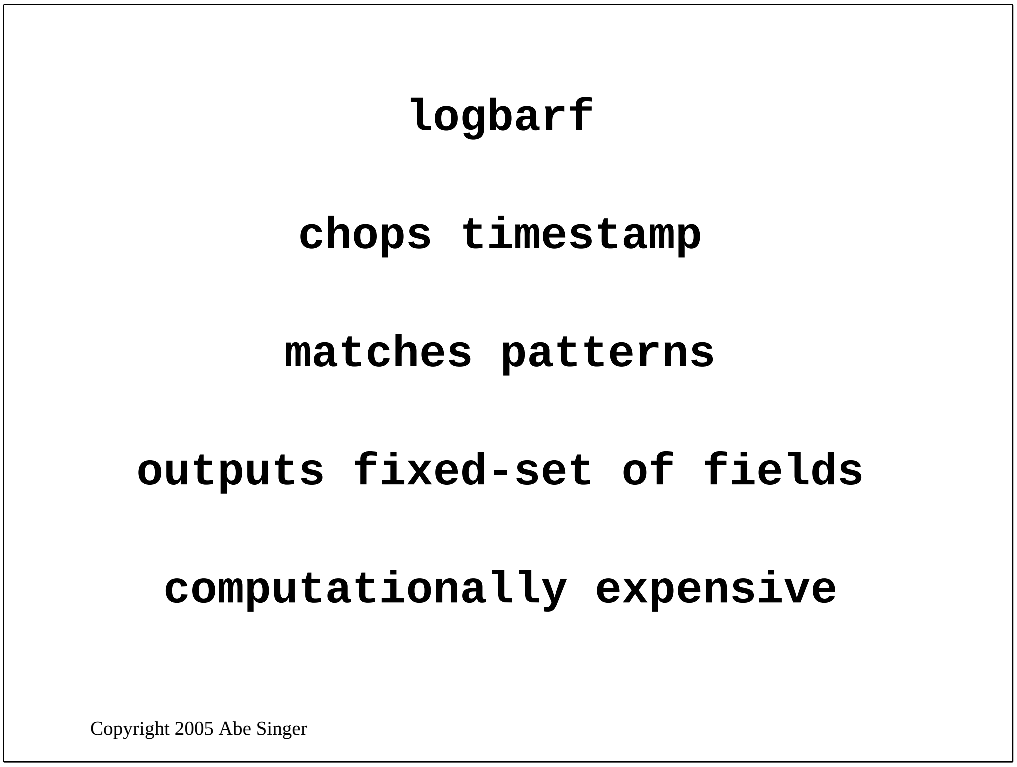
cuz there's a lot of different formats

I've identified 24 different types in SDSC's logs

and that's not perfect

```
HOST last message repeated NUM time
FQDN Message forwarded from HOST: last message
  repeated NUM time
FQDN Message forwarded from FQDN: SERVICE[PID]:
FQDN Message forwarded from HOST: SERVICE[PID]:
FQDN Message forwarded from HOST: SERVICE:
FQDN Forwarded from HOST: SERVICE[PID]:
FQDN Forwarded from HOST: SERVICE:
FQDN WORD: [id NUM kern.WORD]
FQDN -- SERVICE[PID]: root ?login on \S+
FQDN \^Mpanic[cpuNUM]
FQDN FQDN SERVICE[PID]:
```

```
FQDN HOST SERVICE[PID]:
REAL] SERVICE[PID]:
FQDN WORD SERVICE[PID]:
FQDN SERVICE[PID]:
FQDN NUM:
HOST NUM:
FQDN SERVICE:
HOST SERVICE[PID]:
HOST SERVICE:
FQDN SERVICE version:
HOST SERVICE version:
FQDN
FQDN HOST WORD ... NUM
```



logbarf

chops timestamp

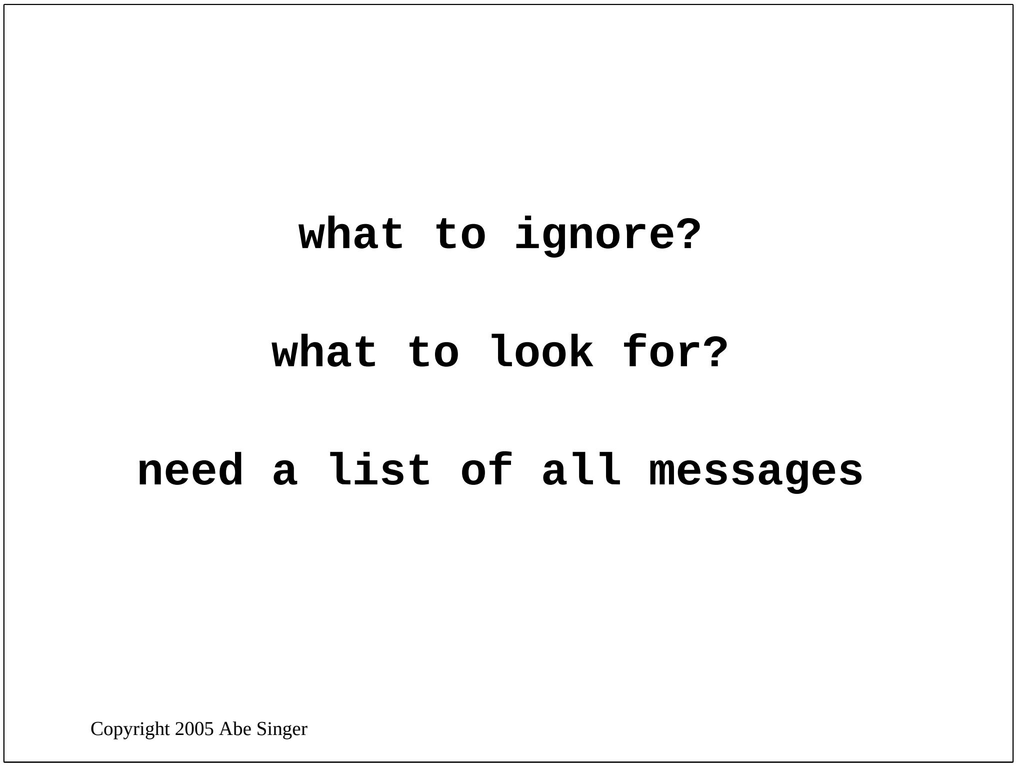
matches patterns

outputs fixed-set of fields

computationally expensive

so I wrote a tool called logbarf
-----
which chops off the timestamp
----
matches against all the patterns
----
outputs delimited fields
-----
in a variety of formats

but it’s pretty computationally expensive
------
4 hours to process a week’s worth of data
but useful to parse on the lfy
and very useful for separating out message bodies



what to ignore?

what to look for?

need a list of all messages

so eventually you realize that you need to know

------

what do I want to ignore?

------

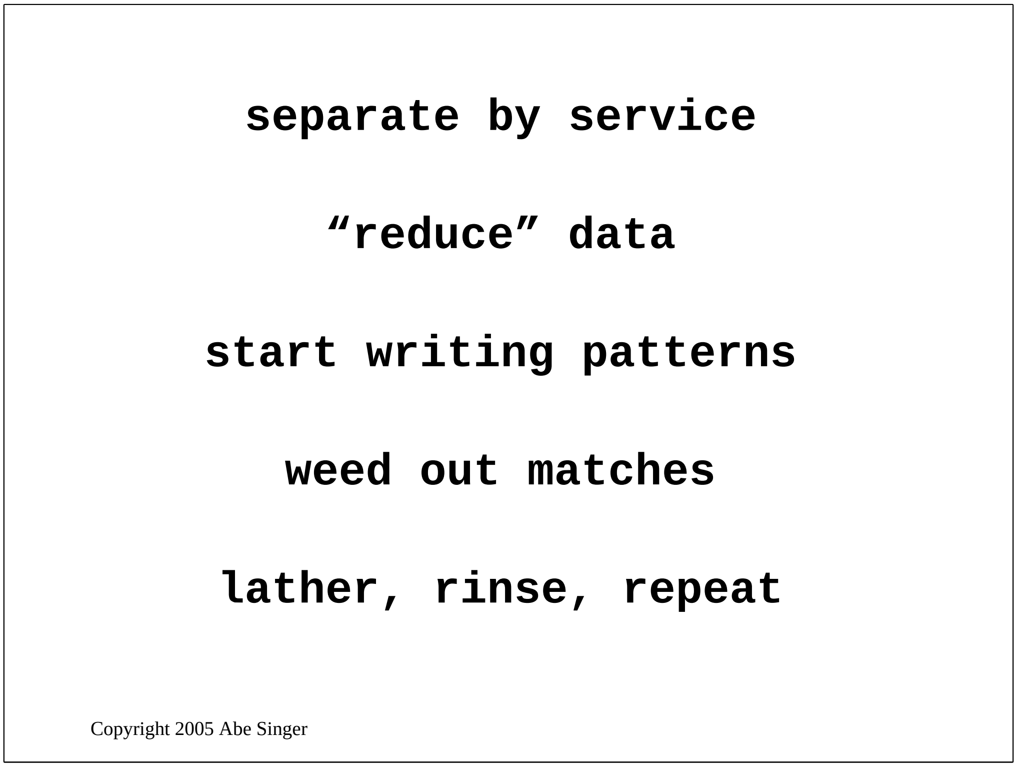
what do I want to look for?

----

which leads you to realize that you want a list of all

----

the messages that you could see, so you can pick the “interesting’ ones



separate by service

“reduce” data

start writing patterns

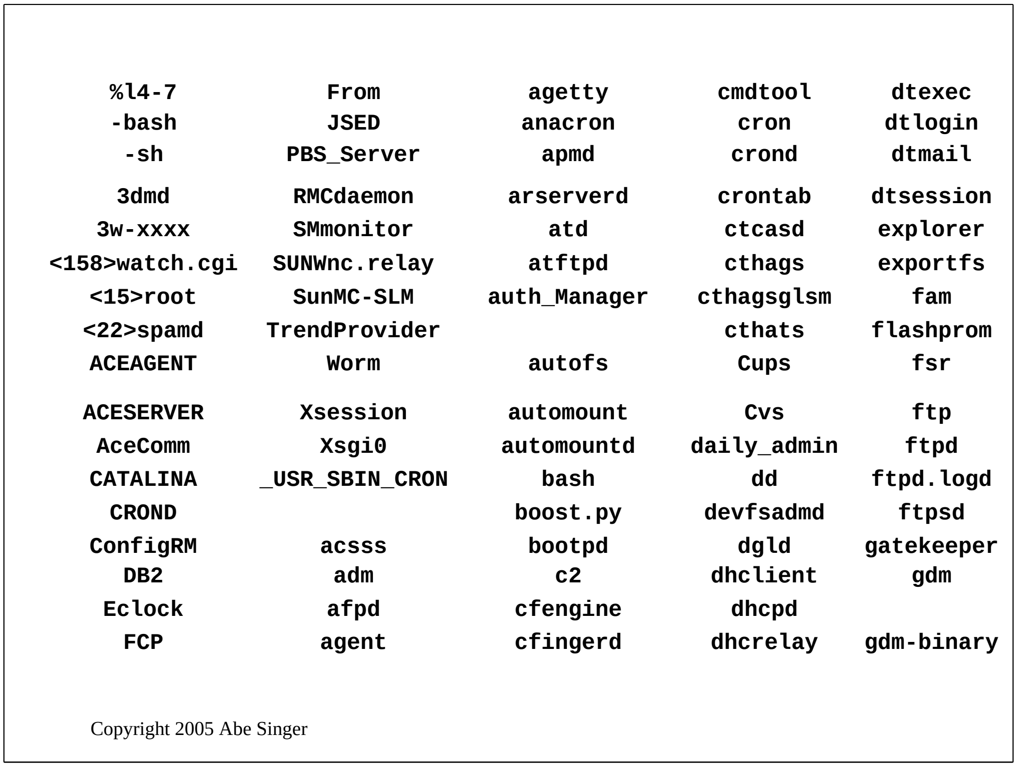
weed out matches

lather, rinse, repeat

so my approach to getting to the list of messages is to divide and conquer

first, separate out message by service.
----
much easier to pattern match that way

then reduce the data as described above
-----
then start writing patterns
-----
weed out the matches from your file with grep –v –f
-----
and repeat the process until you have a complete list
it takes some work



```
%l4-7
-bash
-sh
3dmd
3w-xxxx
<158>watch.cgi
<15>root
<22>spamd
ACEAGENT
ACESERVER
AceComm
CATALINA
CROND
ConfigRM
DB2
Eclock
FCP
From
JSED
PBS_Server
RMCdaemon
SMmonitor
SUNWnc.relay
SunMC-SLM
TrendProvider
Worm
Xsession
Xsgi0
_USR_SBIN_CRON
acsss
adm
afpd
agent
agetty
anacron
apmd
arserverd
atd
atftpd
auth_Manager
autofs
automount
automountd
bash
boost.py
bootpd
c2
cfengine
cfingerd
cmdtool
cron
crond
crontab
ctcasd
cthags
cthagsglsm
cthats
Cups
Cvs
daily_admin
dd
devfsadmd
dgld
dhclient
dhcpd
dhcrelay
dtexec
dtlogin
dtmail
dtsession
explorer
exportfs
fam
flashprom
fsr
ftp
ftpd
ftpd.logd
ftpsd
gatekeeper
gdm
gdm-binary
```

a sampling of some of the services running at SDSC

Subject

hardmon

s3khc

rz

slapd

spnkeyman

^Mpanic[cpu386]_thread=2a10409dd40

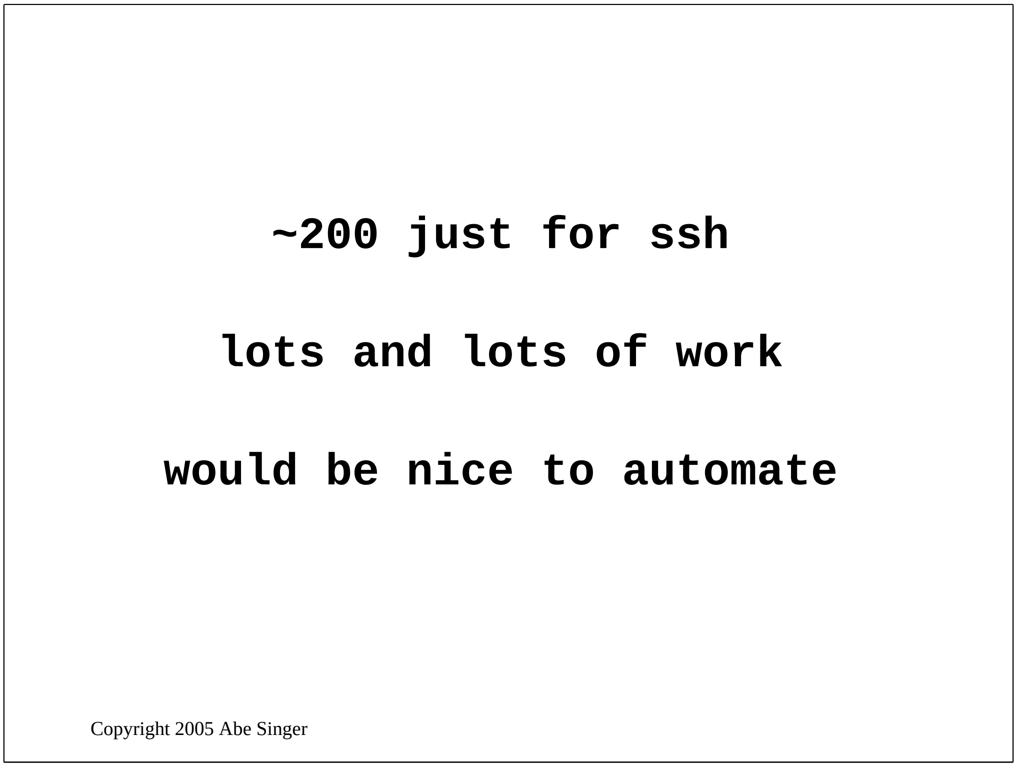
_usr_sdsc_apps_sudo-1.6.3p3_NOKRB_bin_sudo

Content-Transfer-Encoding

sshd.cfsaved

xemacs-20.4

userhelper



~200 just for ssh

lots and lots of work

would be nice to automate

so how many patterns are we talking about?

I came up with 200 for SSH

-----

in one weeks worth of data

but it took a lot of work

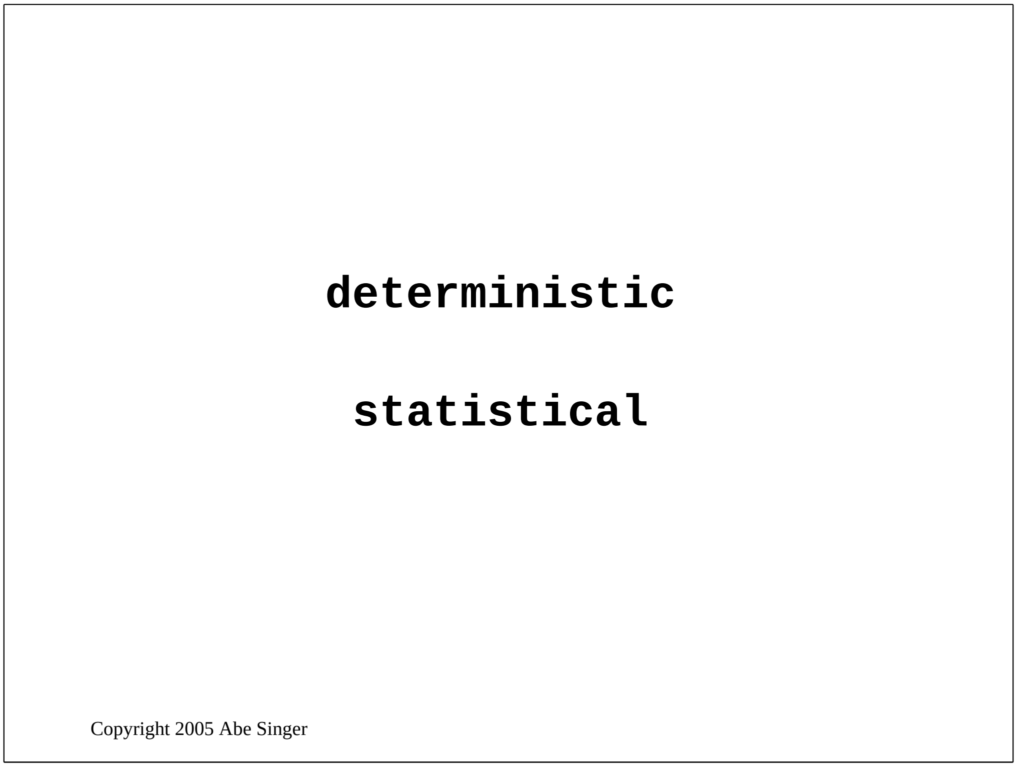
-----

it would be nice to have something better that was automated

------

so you can focus on other stuff

but it’s a really hard problem



deterministic

statistical

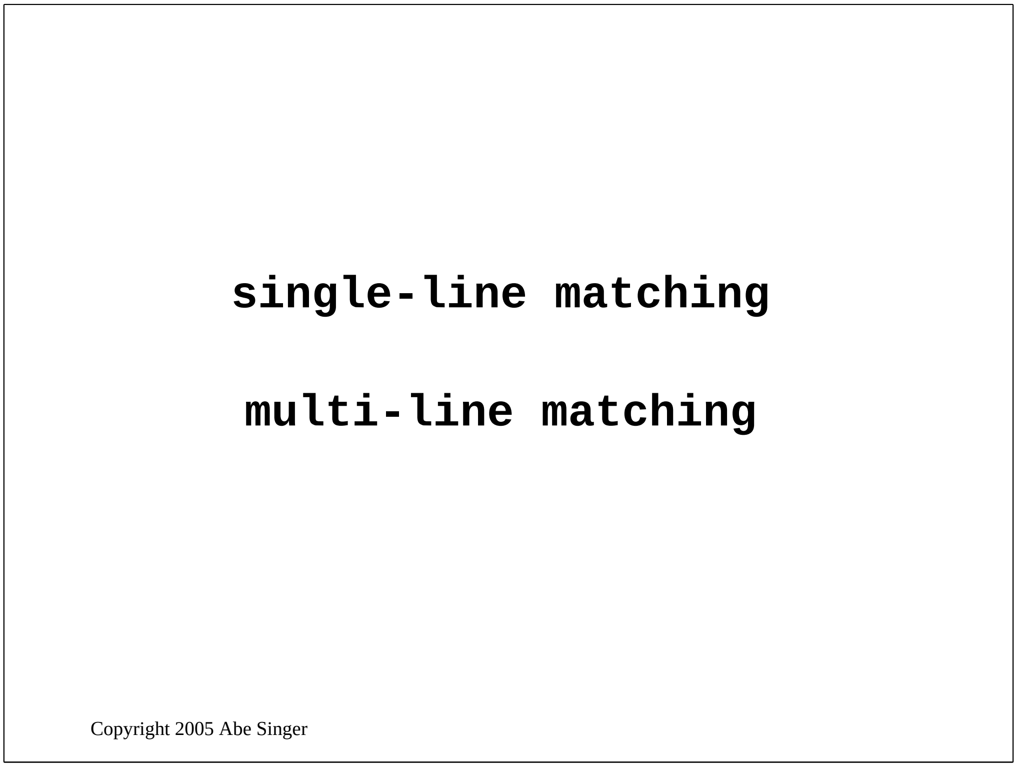
so now that you have your data where you want it, let’s tal about some analysis techniques

there’s two categories of analysis

deterministic
-----
where you know what you’re looking for

and statistical
------
where you’re looking for relationships in the data more than just “look for these things together”

data mining and the more complicated things are usually statistical



single-line matching

multi-line matching

deterministic analysis

basically looking for a known message

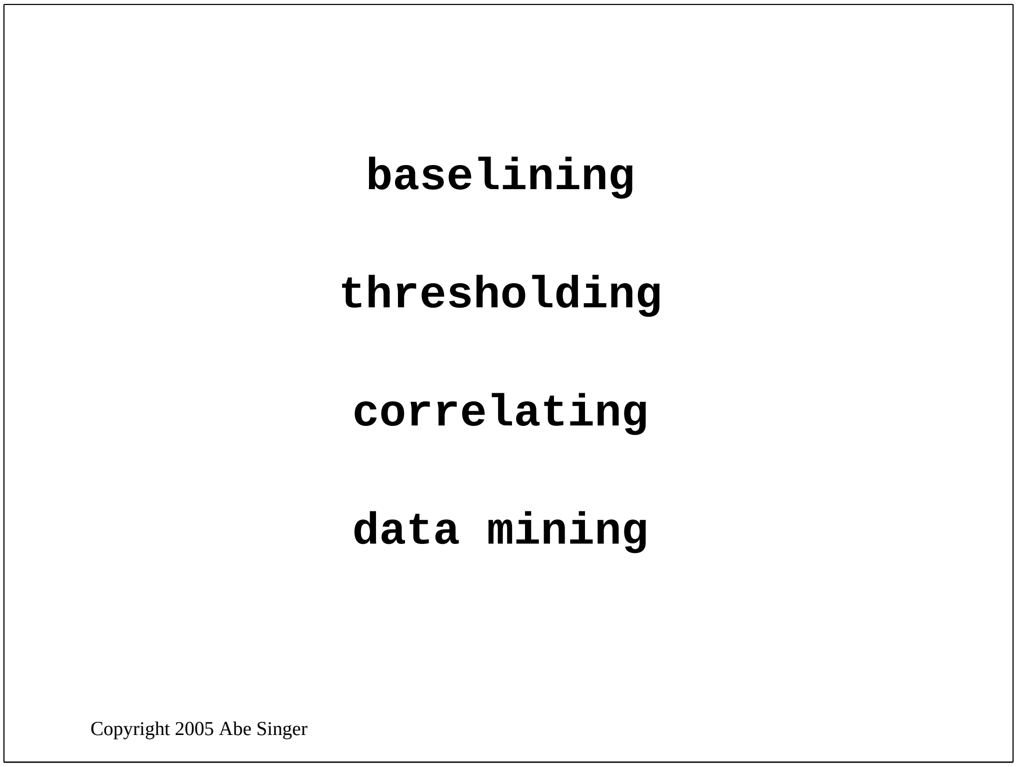
-----

and doing something, including counting and summarizing

and then the same sort of thing with multi-line matching

-----

which still involves knowing what you’re looking for



baselining

thresholding

correlating

data mining

statistical analysis is used for things like

baselining

-----

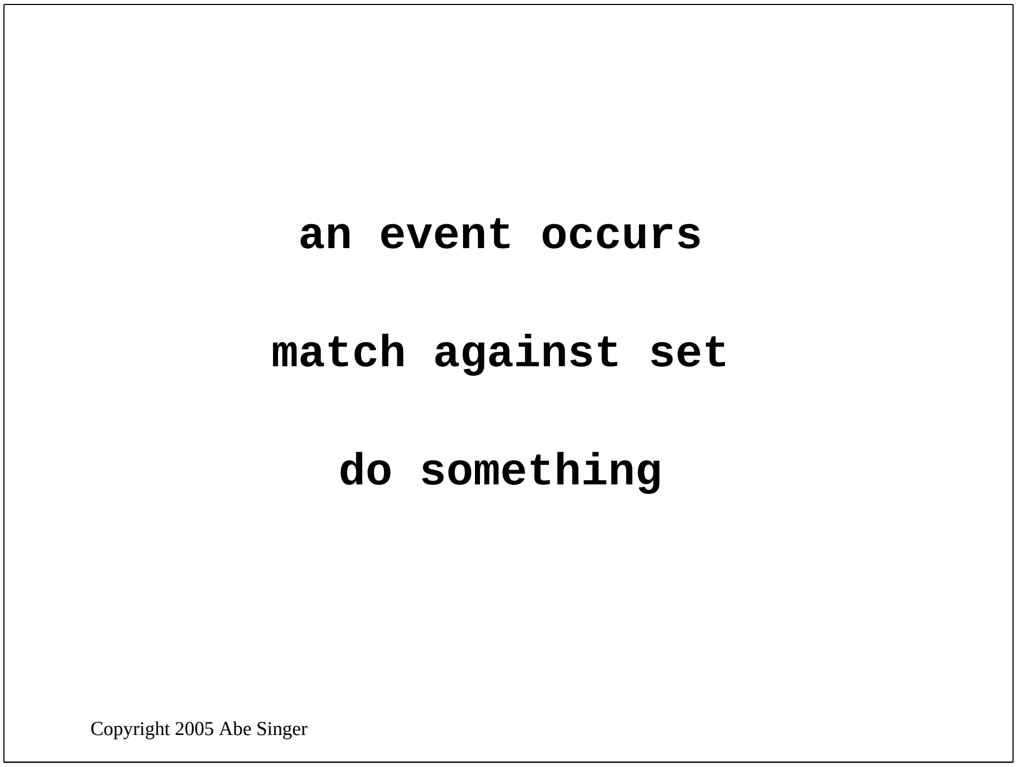
thresholding

-----

correlating

-----

and other data mining techniques such as clustering, pattern discover, etc.



an event occurs

match against set

do something

basic determinist approach:

something happens, causes one or more log messages to be generated

-----

those messages are matched against a set of know messgaes

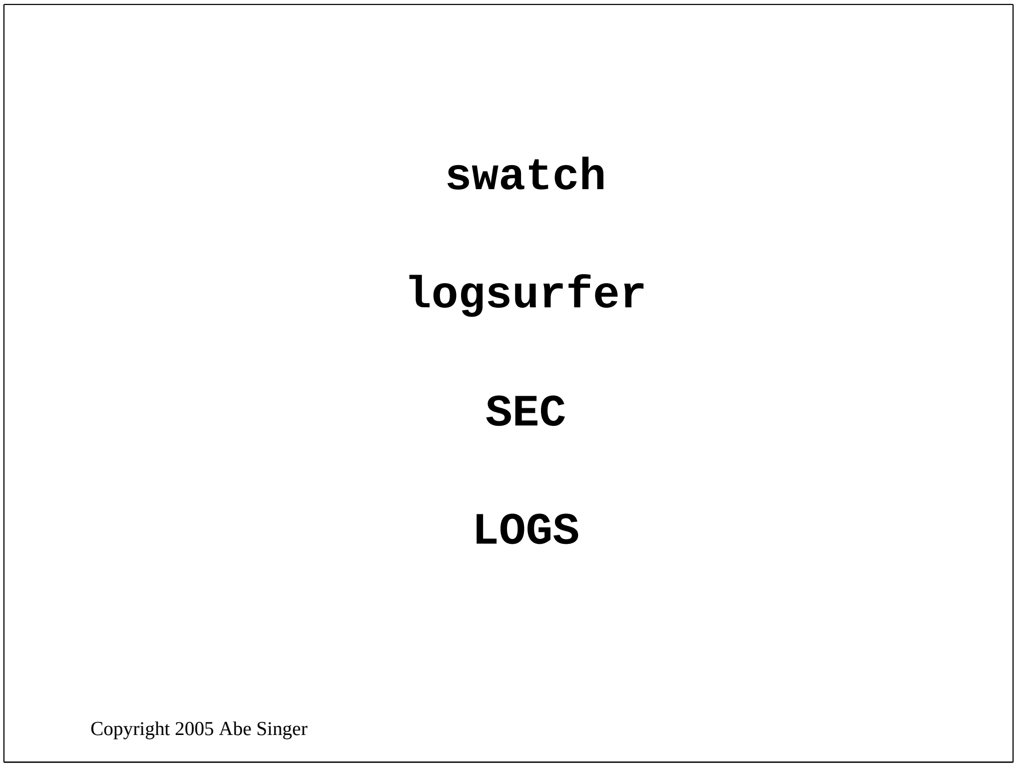
-----

when a match is made, do something

----

the something could be:

page
send email
trigger process
ignore



swatch

logsurfer

SEC

LOGS

some “analysis tools”, although
they don’t really do analysis

swatch

-----

logsurfer

-----

sec

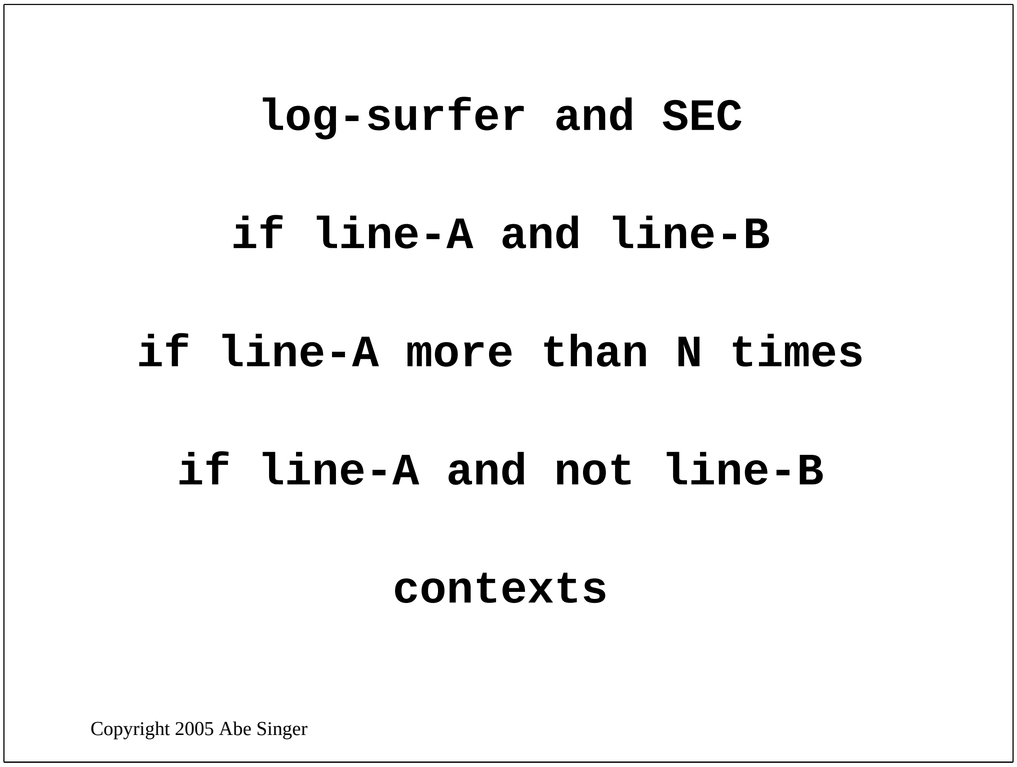
LOGS

----

many others at loganalysis.org

```
ignore /Media load or eject failed/

ignore /named-xfe4r .* connect for zone .* failed: Connection
   (refused|timed out)/

watchfor /fail/ mail address=abe subject=“Alert: fail”

watchfor /statd: attempt to create/ mail address=alerts
   subject=“”tmp/bob attempt”

watchfor /./ pipe “cat - >> /somewhere/unknown’
```

examples using swatch



log-surfer and SEC

if line-A and line-B

if line-A more than N times

if line-A and not line-B

contexts

if ya wanna do multi-line matching and anything fancy, you need logsurfer or SEC (or LOGS)

-----

they can do things like

if line-A and line-B

-----

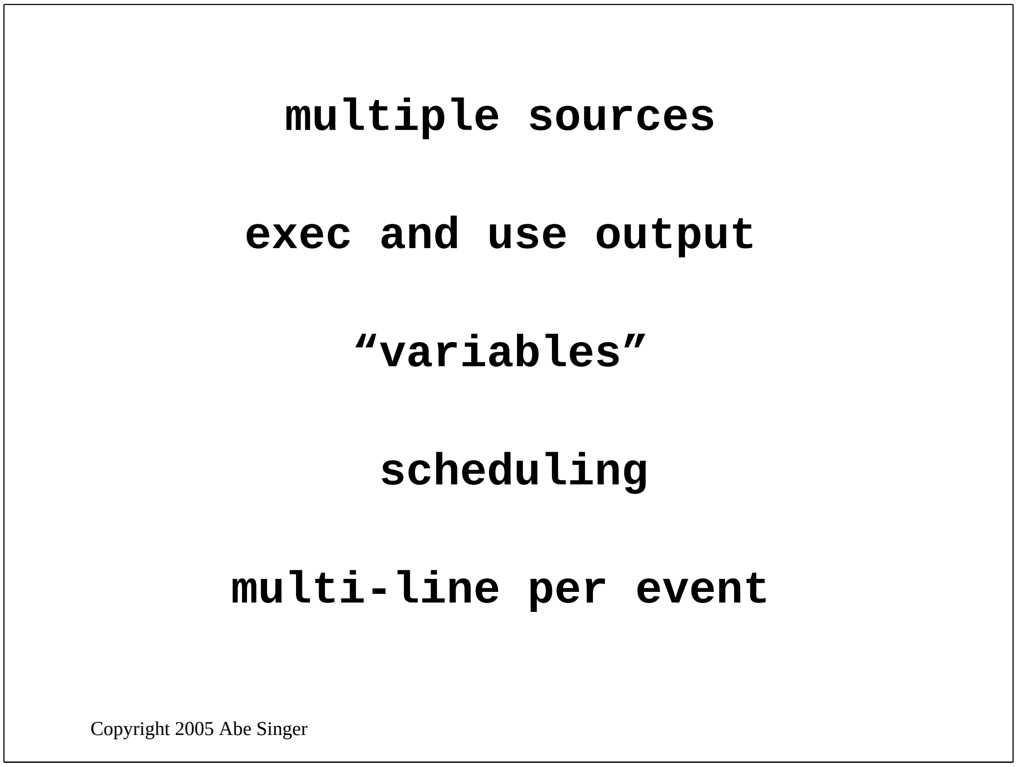
if line-A more than N times

-----

if line-A and not line-B

-----

contexts



multiple sources

exec and use output

“variables”

scheduling

multi-line per event

some cool things about SEC

multiple sources for input

----

exec and use output

----

user-definable “variables”

----

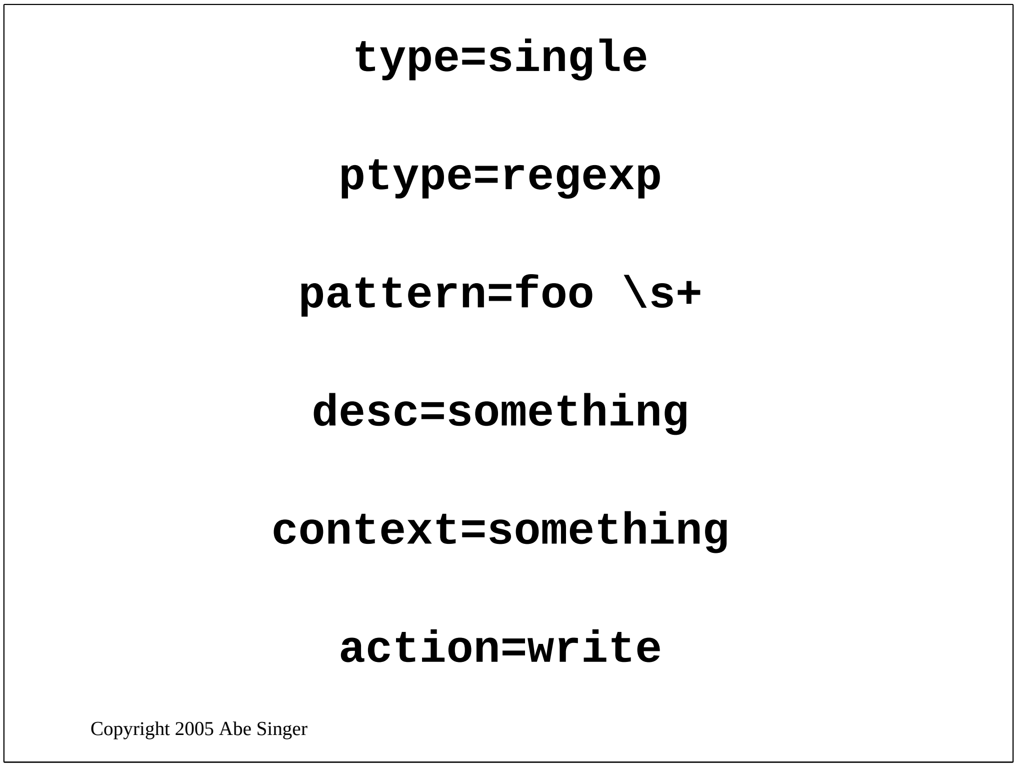
cron-type scheduling

----

multi-line config per even match

-----

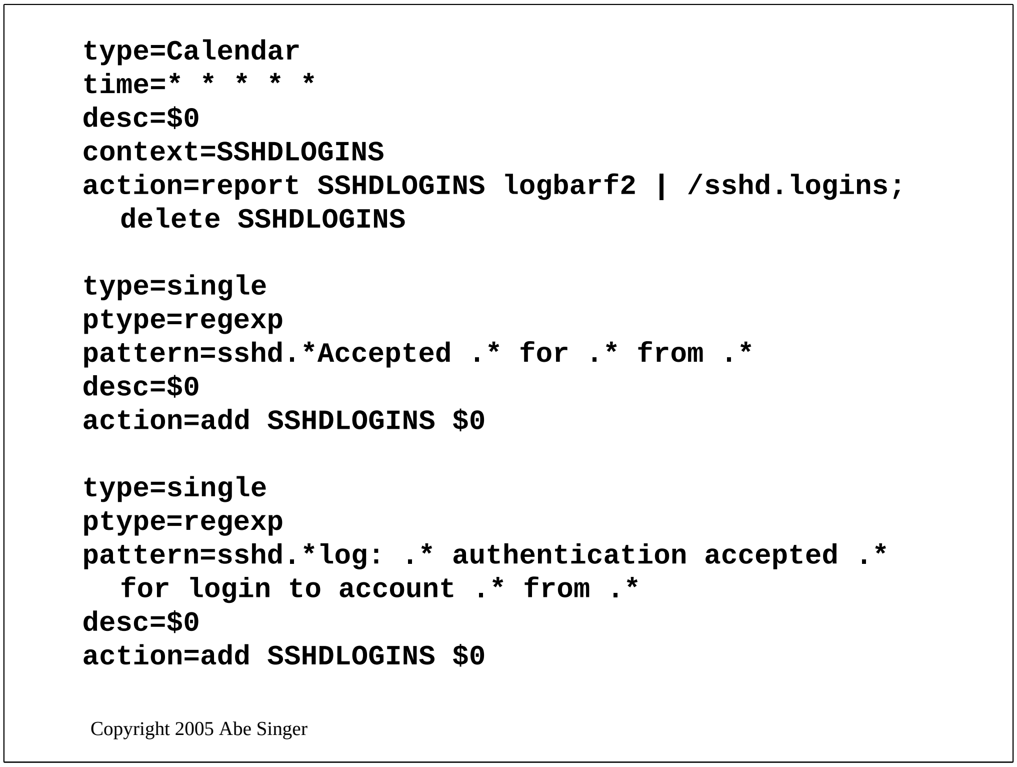
and it’s written in Perl!



```
type=single

ptype=regexp

pattern=foo \s+

desc=something

context=something

action=write
```

SEC syntax looks like this:

```
type=single
-----
ptype=regexp
-----
pattern=foo \s+
-----
desc=something
-----
context=something
-----
action=write
```



```
type=Calendar
time=* * * * *
desc=$0
context=SSHDLOGINS
action=report SSHDLOGINS logbarf2 | /sshd.logins;
  delete SSHDLOGINS

type=single
ptype=regexp
pattern=sshd.*Accepted .* for .* from .*
desc=$0
action=add SSHDLOGINS $0

type=single
ptype=regexp
pattern=sshd.*log: .* authentication accepted .*
  for login to account .* from .*
desc=$0
action=add SSHDLOGINS $0
```

some example code that track ssh logins

data is put in a context as it’s matched, then once per

minute (due to volume), it’s piped to a program that parses the data, matches against some databases, and outputs the result.

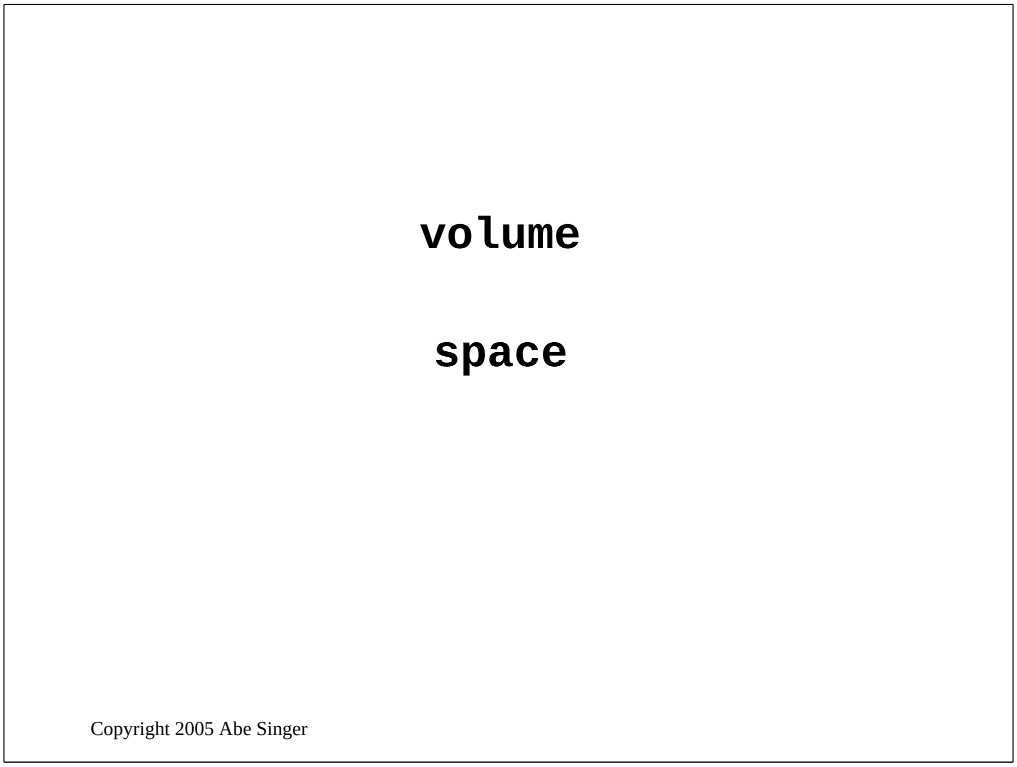
Eventually it will pipe right into a database

http://www.ranum.com

retail

nbs

some other tools that might be found useful for this sort of things



volume

space

a quick note on sticking things in a database

does it scale?

------

we’re takling millions of records per day

can the database even handle indexing and loading them?

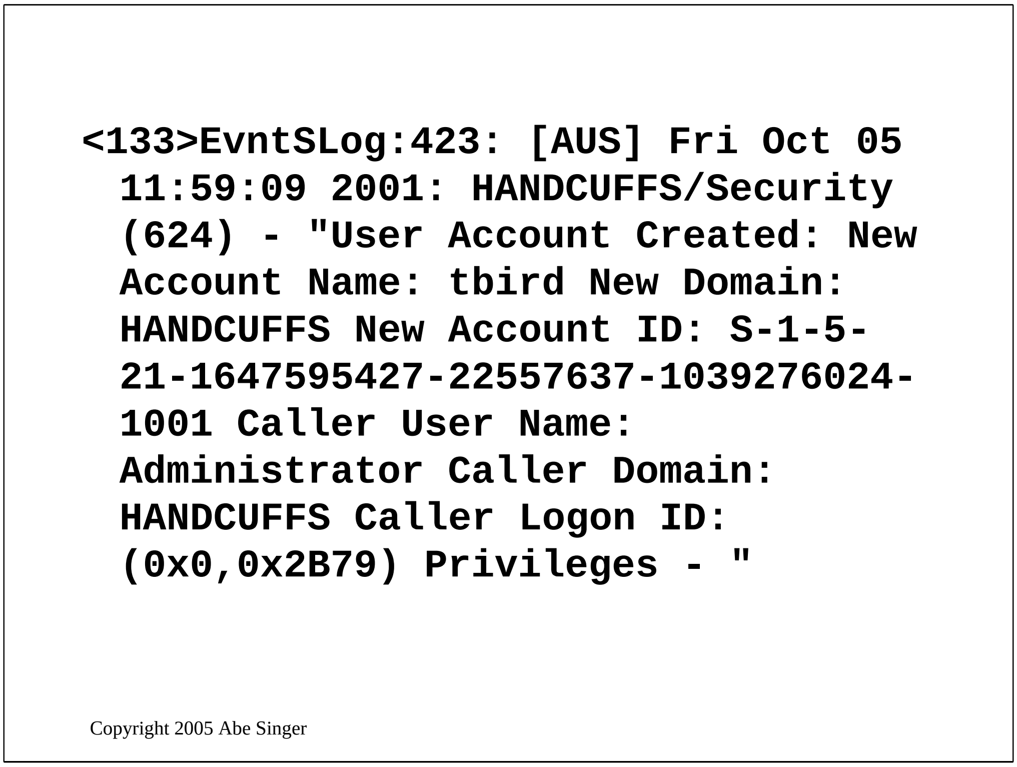
can it do queries efficiently enough?

-----

and then there’s the space it takes up, remember it’s some multiple of the amount of data you put in, inlcuding indexes and stuff

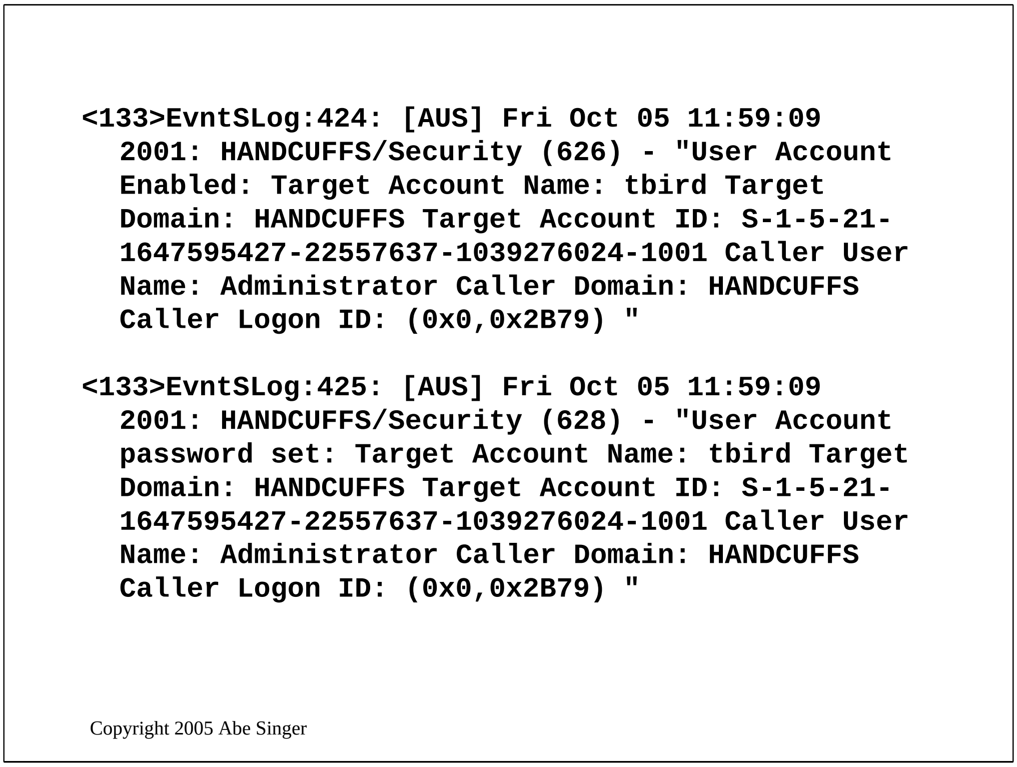
the answer is “maybe” but test it out before you hose your production database system

Attack Signatures



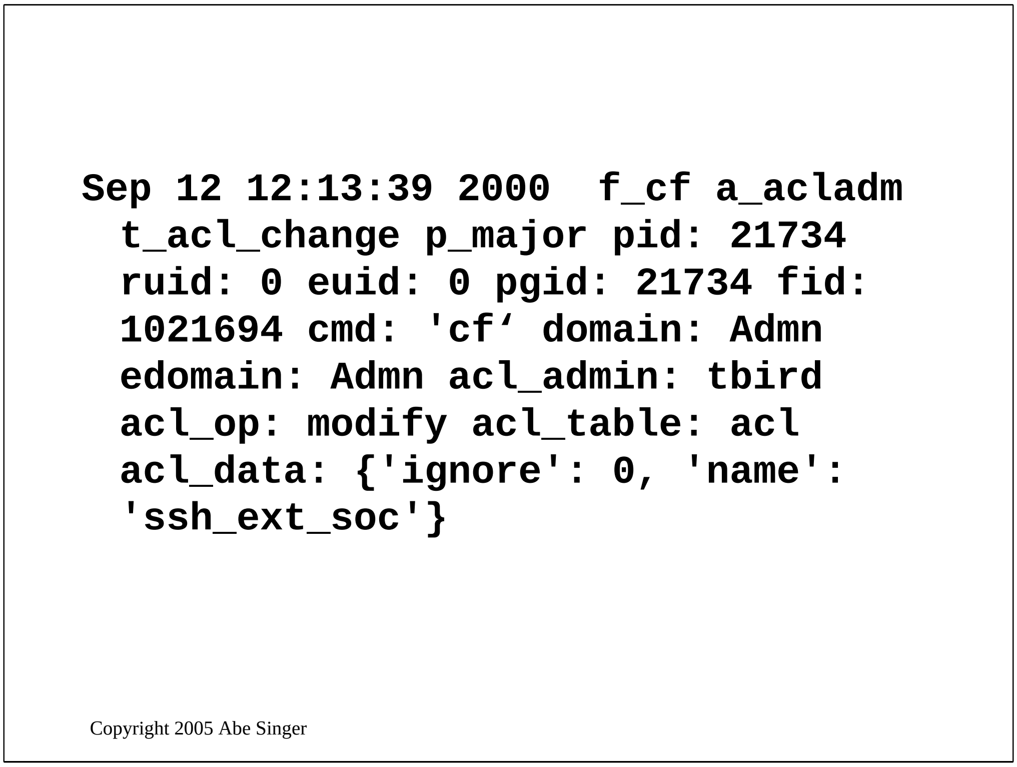
```
<133>EvntSLog:423: [AUS] Fri Oct 05
  11:59:09 2001: HANDCUFFS/Security
  (624) - "User Account Created: New
  Account Name: tbird New Domain:
  HANDCUFFS New Account ID: S-1-5-
  21-1647595427-22557637-1039276024-
  1001 Caller User Name:
  Administrator Caller Domain:
  HANDCUFFS Caller Logon ID:
  (0x0,0x2B79) Privileges - "
```

This space intentionally left blank.



```
<133>EvntSLog:424: [AUS] Fri Oct 05 11:59:09
  2001: HANDCUFFS/Security (626) - "User Account
  Enabled: Target Account Name: tbird Target
  Domain: HANDCUFFS Target Account ID: S-1-5-21-
  1647595427-22557637-1039276024-1001 Caller User
  Name: Administrator Caller Domain: HANDCUFFS
  Caller Logon ID: (0x0,0x2B79) "

<133>EvntSLog:425: [AUS] Fri Oct 05 11:59:09
  2001: HANDCUFFS/Security (628) - "User Account
  password set: Target Account Name: tbird Target
  Domain: HANDCUFFS Target Account ID: S-1-5-21-
  1647595427-22557637-1039276024-1001 Caller User
  Name: Administrator Caller Domain: HANDCUFFS
  Caller Logon ID: (0x0,0x2B79) "
```

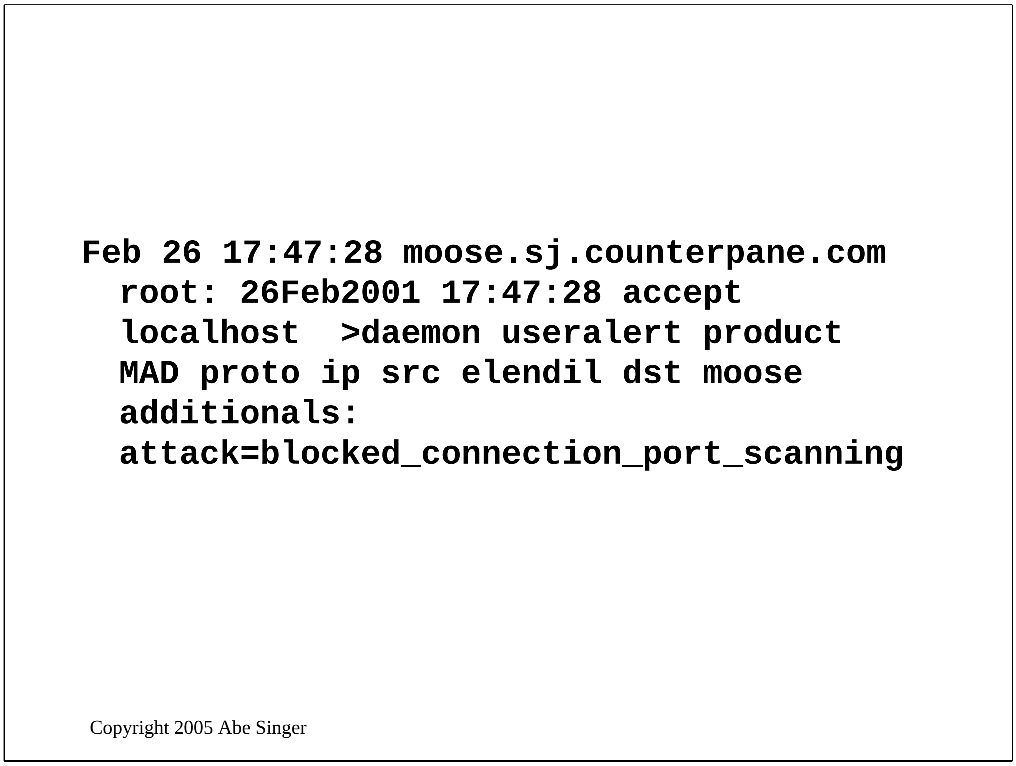
This space intentionally left blank.



```
Sep 12 12:13:39 2000  f_cf a_acladm
  t_acl_change p_major pid: 21734
  ruid: 0 euid: 0 pgid: 21734 fid:
  1021694 cmd: 'cf‘ domain: Admn
  edomain: Admn acl_admin: tbird
  acl_op: modify acl_table: acl
  acl_data: {'ignore': 0, 'name':
  'ssh_ext_soc'}
```

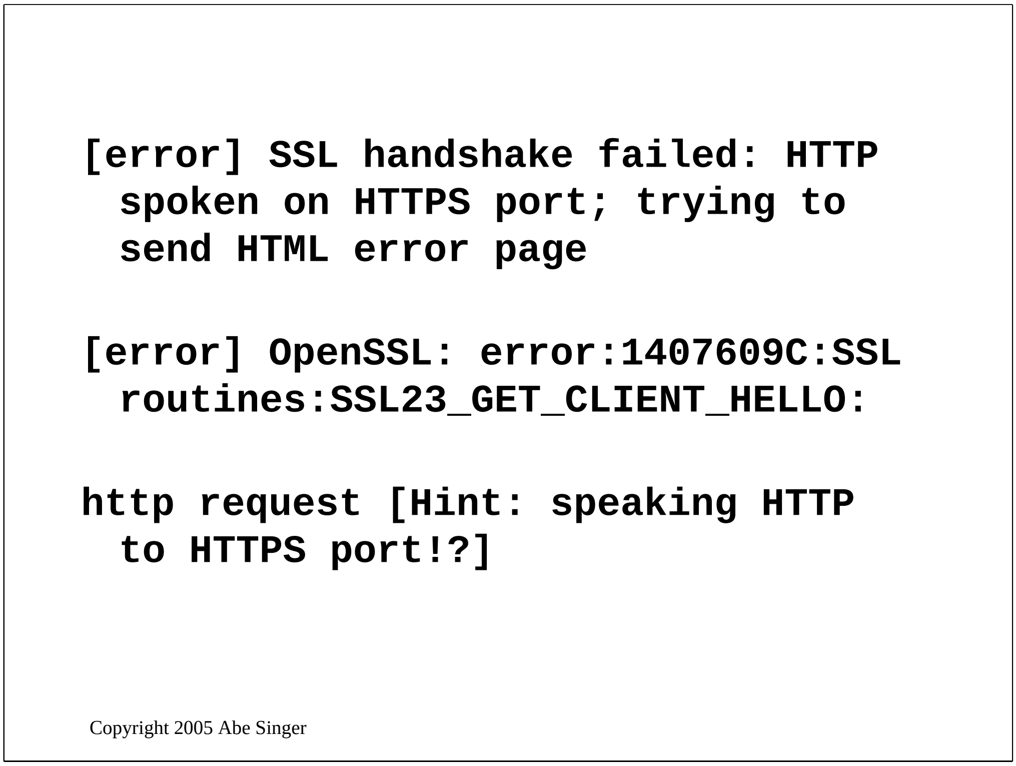
You wouldn’t know it by looking, but this is an audit message generated by a Sidewinder firewall. An access control list entry was modified – a rule named ‘ssh_ext_soc’ was changed from “ignore” status (i.e., don’t enforce this rule) to “active.” The majority of information in this log entry relates to process identifications on the firewall and the security compartments which are proprietary to Sidewinder.

These kinds of messages are subtle. If the configuration change was something you knew about, that’s one thing – but if it happened without your knowledge or whatever level of authorization you require, it could be very bad news indeed.



```
Feb 26 17:47:28 moose.sj.counterpane.com
  root: 26Feb2001 17:47:28 accept
  localhost  >daemon useralert product
  MAD proto ip src elendil dst moose
  additionals:
  attack=blocked_connection_port_scanning
```

MAD = Malicious Activity Detection, Checkpoint’s module to scan its own log files and look for signs of, well, malicious activity. It’s not very widely used, but it is there. MAD uses the FW-1 management console to perform thresholding tests on the network connection data it’s collecting, allowing it to for instance detect X denied connections in a given period of time from the same source IP address.

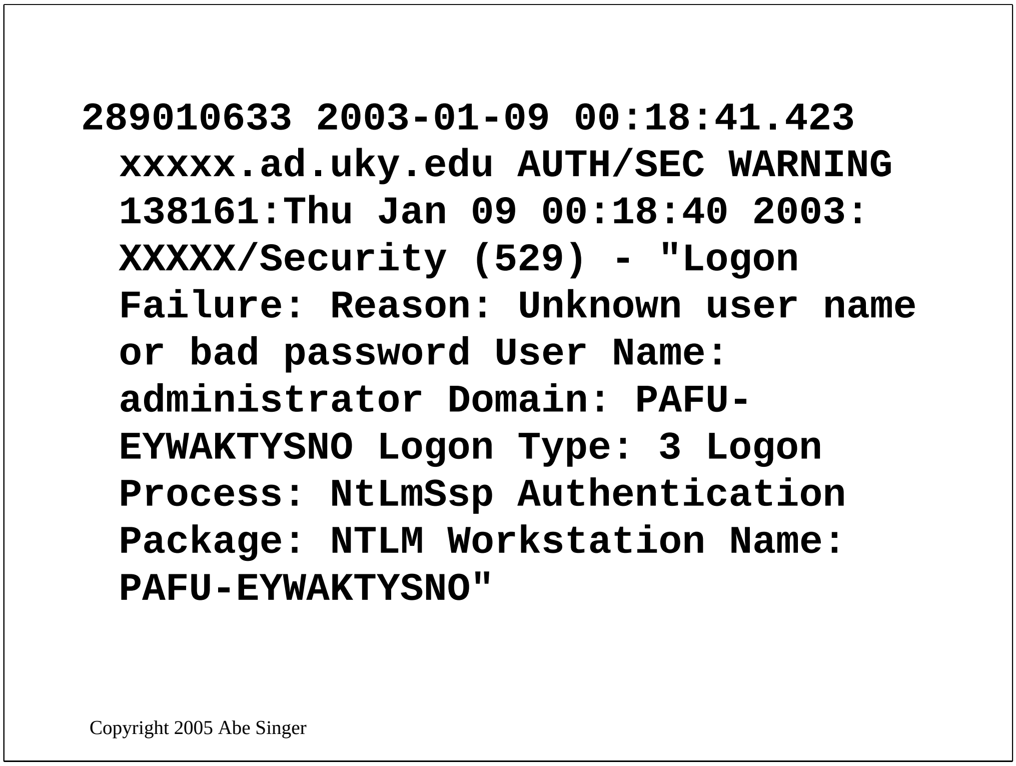


```
[error] SSL handshake failed: HTTP
  spoken on HTTPS port; trying to
  send HTML error page

[error] OpenSSL: error:1407609C:SSL
  routines:SSL23_GET_CLIENT_HELLO:

http request [Hint: speaking HTTP
  to HTTPS port!?]
```

Apache/mod-SSL worm discovered 13 Sept 2002; exploits buffer overflow in SSL v2

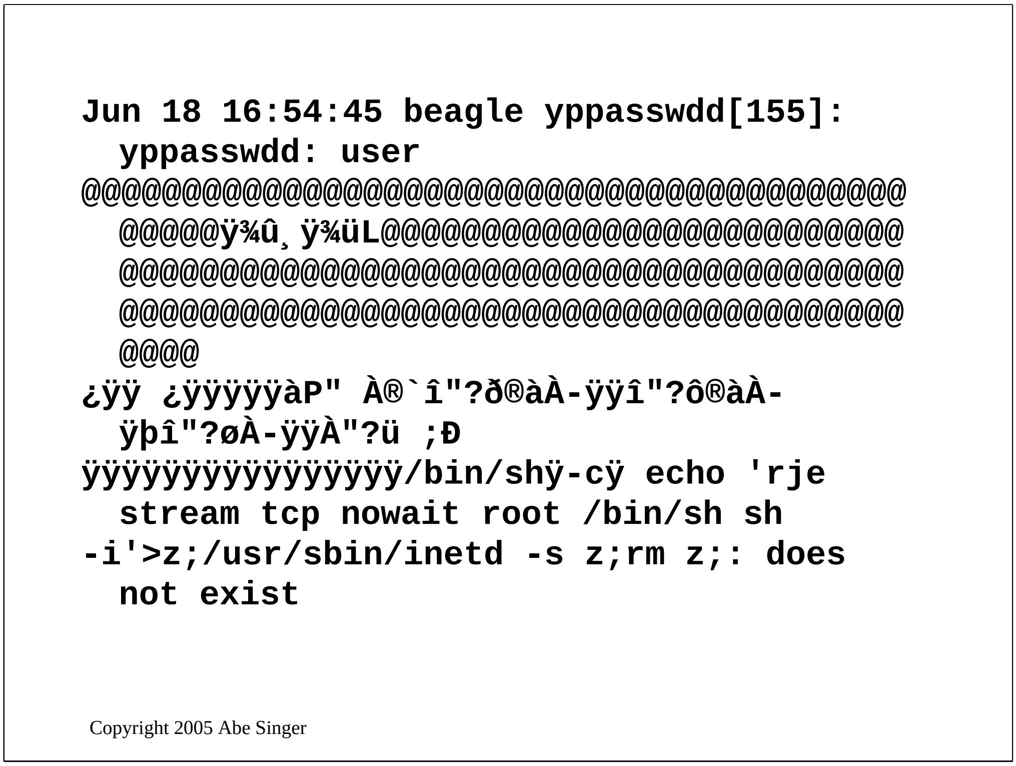
This little gem only shows up when the Slapper exploit probes an Apache server running the then-current, *patched* version of OpenSSL, 0.9.6g. The new code handles the buffer overflow data correctly. As is frequently the case, when the exploit hits a vulnerable system, it works and leaves no evidence in the Apache logs or in *syslog.*

For more information: http://www.counterpane.com/alert-i20020915-001.html



```
289010633 2003-01-09 00:18:41.423
  xxxxx.ad.uky.edu AUTH/SEC WARNING
  138161:Thu Jan 09 00:18:40 2003:
  XXXXX/Security (529) - "Logon
  Failure: Reason: Unknown user name
  or bad password User Name:
  administrator Domain: PAFU-
  EYWAKTYSNO Logon Type: 3 Logon
  Process: NtLmSsp Authentication
  Package: NTLM Workstation Name:
  PAFU-EYWAKTYSNO"
```

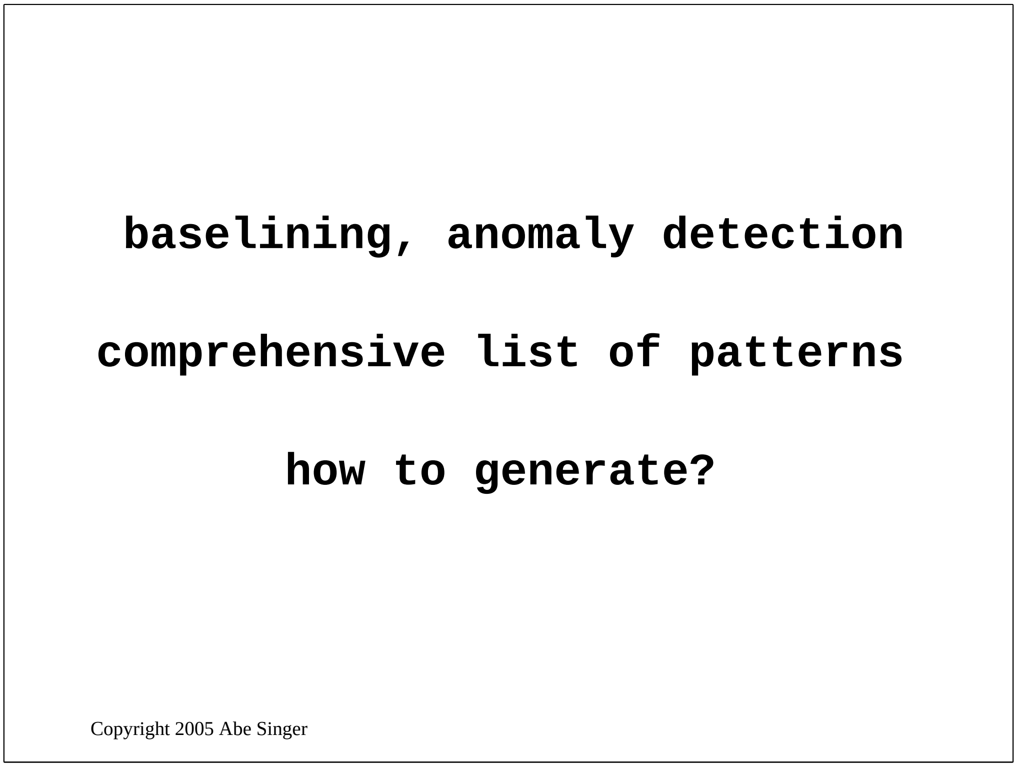
Frank Solomon of the University of Kentucky points out that failed login attempts from non-local or unknown domains are almost always a sign of someone conducting a brute force atttack on user accounts. Don’t you have a domain called PAFU-EYWAKTYSNO????



```
Jun 18 16:54:45 beagle yppasswdd[155]:
  yppasswdd: user
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @@@@@ÿ¾û¸ÿ¾üL@@@@@@@@@@@@@@@@@@@@@@@@@@
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @@@@
¿ÿÿ ¿ÿÿÿÿÿàP" À®`î"?ð®àÀ-ÿÿî"?ô®àÀ-
  ÿþî"?øÀ-ÿÿÀ"?ü ;Ð
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ/bin/shÿ-cÿ echo 'rje
  stream tcp nowait root /bin/sh sh
-i'>z;/usr/sbin/inetd -s z;rm z;: does
  not exist
```

This is, of course, all one line.

Current Work



baselining, anomaly detection

comprehensive list of patterns

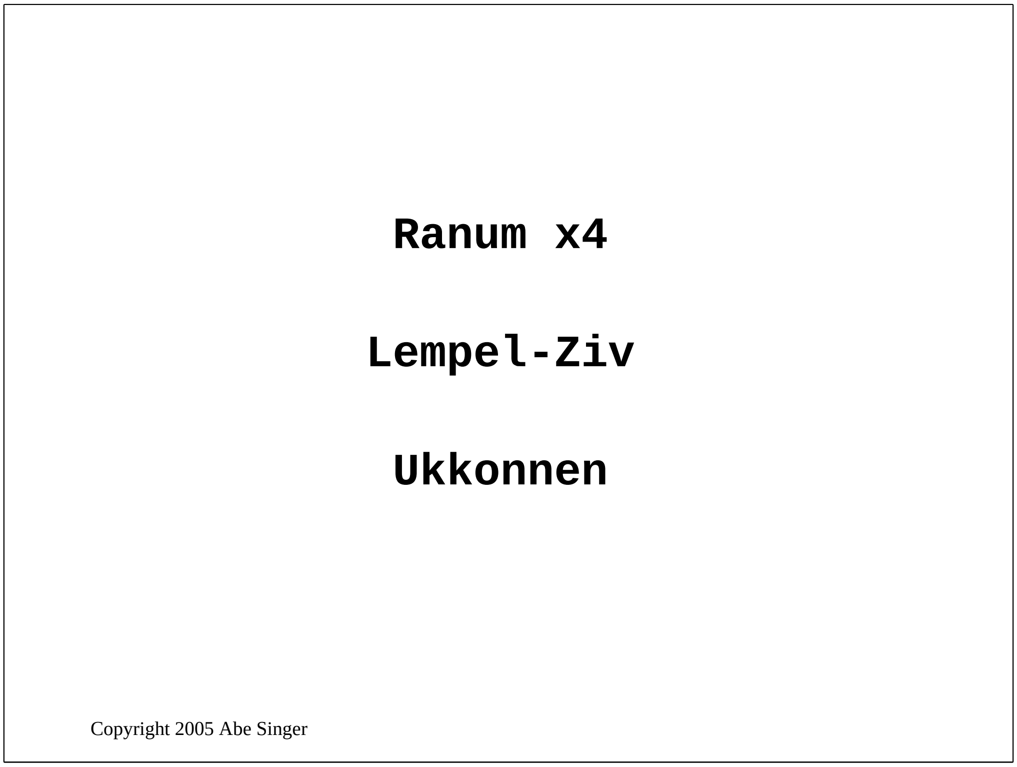
how to generate?

remember how I said it would be nice to automagically generate patterns?

let me tell you about some work I’m doing

first, my goals are to get baselining and anomaly detection
-----
but to get that, I need a comprehensive list of patterns

I have a *lot* of data to work with, 8 years worth
so how can i get a list of patterns bvy analysing my data?



Ranum x4

Lempel-Ziv

Ukkonnen

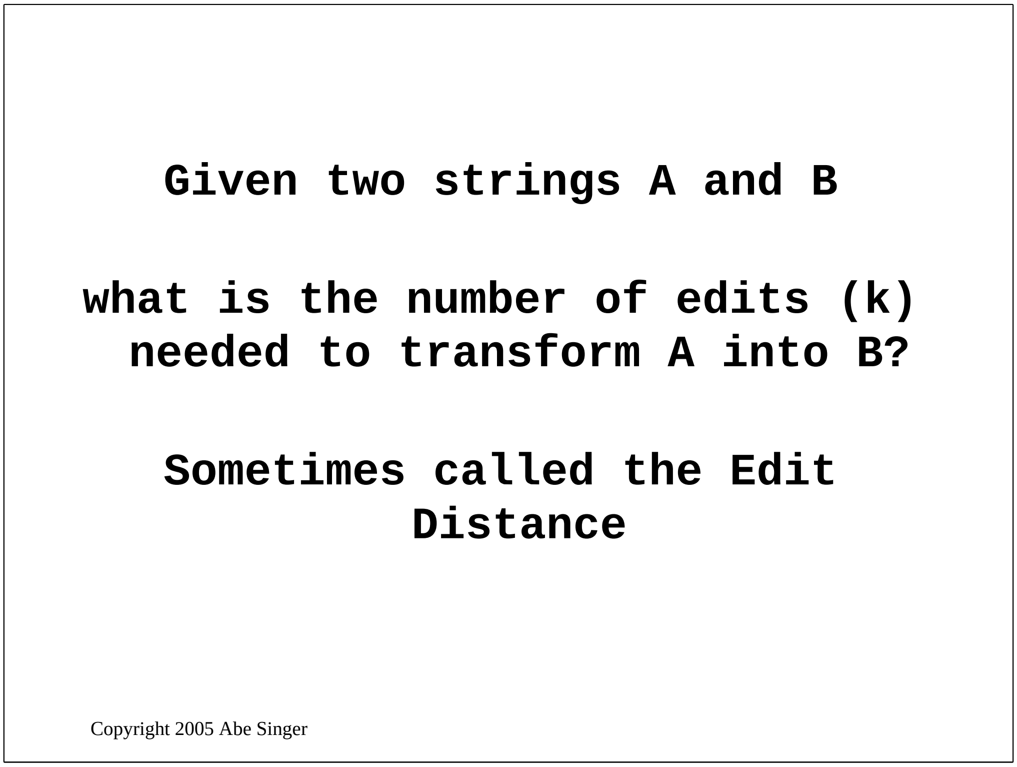
Some approaches that were tried

Marcus Ranum tried some statistical analysis
statistical analysis of tokens by position
-----
called x4. The remains of it are available on his website

I tried looking at lempe-ziv compression
compress repeating sequences of tokens

-----
which gets you partway there, but not enough

Then I found a paper by Ukkonnen on approximate string matching



Given two strings A and B

what is the number of edits (k) needed to transform A into B?

Sometimes called the Edit Distance

this is ASM:

given two strings

-----

how many edits (replacements, insertions, deletions)

does it take to get from A to B

-----

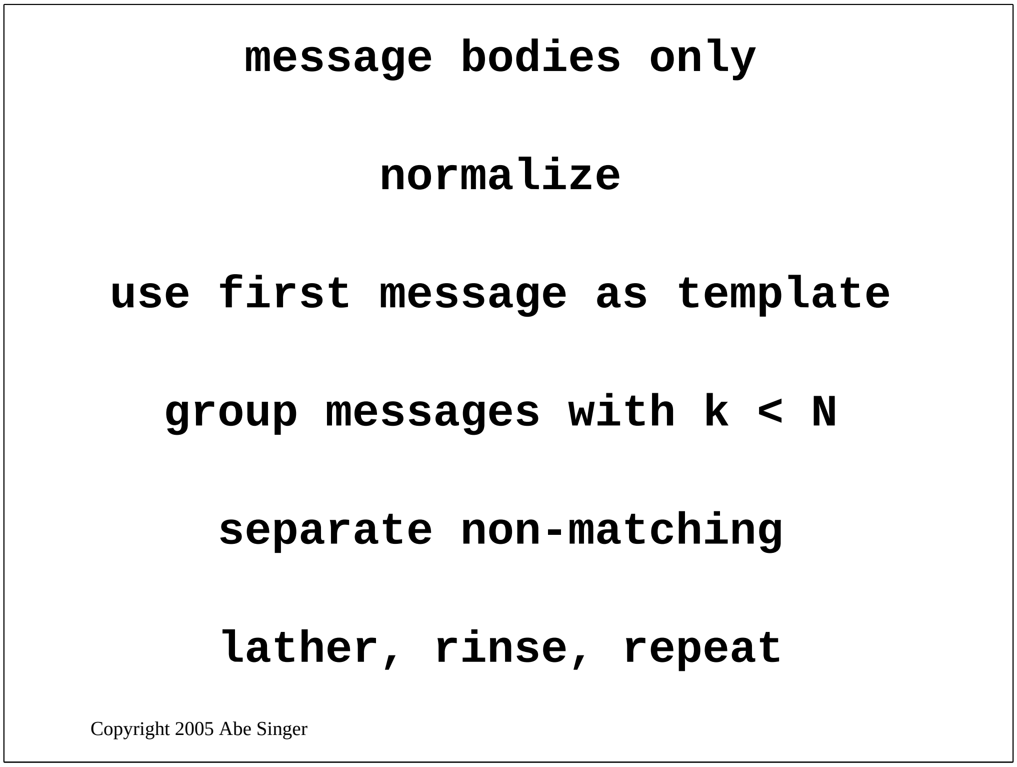
this is known as the edit distance

-----

or hamming distance, or just “k”

ukkonnen applied to strings

I did it to tokenized logs



message bodies only

normalize

use first message as template

group messages with k < N

separate non-matching

lather, rinse, repeat

and this is my approach

message bodies only, for one service

-----

normalize as described above

-----

use first message as template

-----

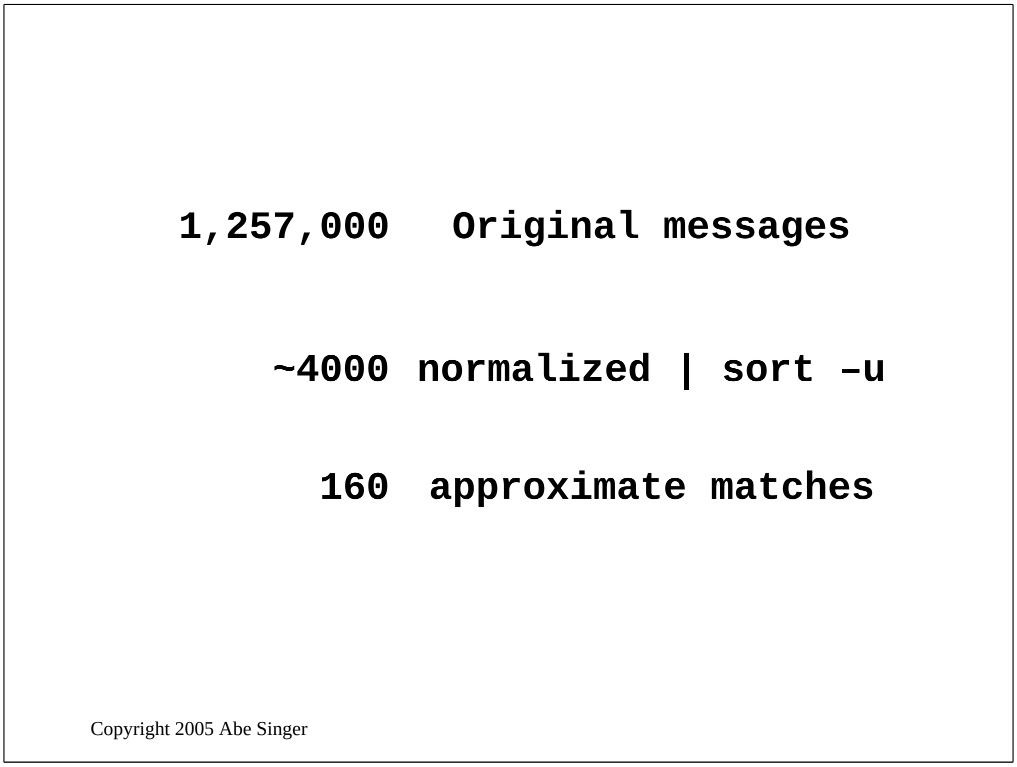
group messages with k < N

-----

separate non-matching

-----

lather, rinse, repeat



1,257,000   Original messages

~4000 normalized | sort –u

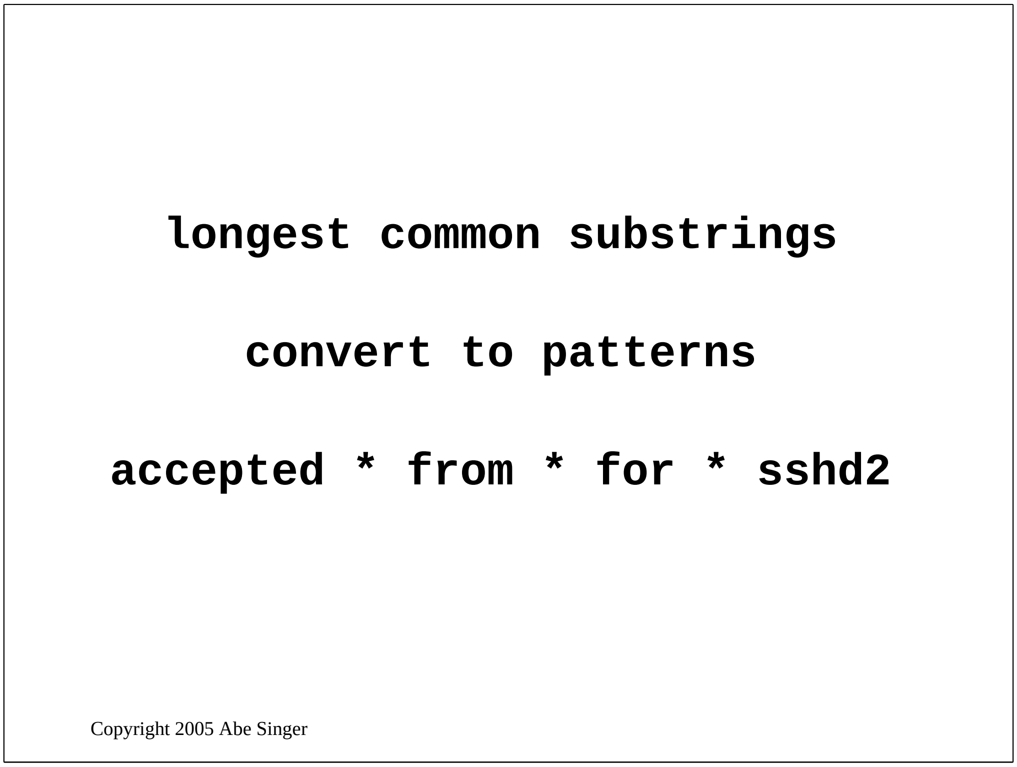
160 approximate matches

and these were my results with a weeks worth of ssh messages

1,275,000 original messages
over 4000 normalized
down to 160, which is close to what I got by hand (different set)

```
accepted external-keyx for inca from NUM.NUM.NUM.NUM port NUM ssh2
accepted gssapi for inca from ::ffff:NUM.NUM.NUM.NUM port NUM ssh2
accepted keyboard-interactive/pam for baden from ::ffff:NUM.NUM.NUM.NUM port NUM
    ssh2
connection closed by NUM.NUM.NUM.NUM
did not receive identification string from NUM.NUM.NUM.NUM
error: pam: authentication failure
gsi user /c=us/o=npaci/ou=sdsc/cn=inca user account/userid=inca is authorized as
    target user inca
gsi user /c=us/o=sdsc/ou=sdsc/cn=keith thompson/userid=kst is authorized as target
    user kst
[id NUM auth.info] accepted password for antoine from NUM.NUM.NUM.NUM port NUM
    ssh2
[id NUM auth.info] did not receive identification string from NUM.NUM.NUM.NUM
[id NUM auth.info] received disconnect from NUM.NUM.NUM.NUM: NUM: disconnect
    requested by windows ssh client.
```

```
[id NUM auth.info] subsystem request for sftp
[id NUM auth.info] warning: /etc/moduli does not exist, using fixed modulus
[id NUM auth.info] wtmp_write: problem writing /var/adm/wtmp: no such file or
    directory
lastlog_openseek: /var/log/lastlog is not a file or directory!
lastlog_perform_login: couldn't stat /var/log/lastlog: permission denied
received disconnect from NUM.NUM.NUM.NUM: NUM: disconnect requested by windows ssh
    client.
setting tty modes failed: invalid argument
subsystem request for sftp
userauth_hostbased mismatch: client sends tg-master.sdsc.teragrid.org, but we
    resolve ::ffff:NUM.NUM.NUM.NUM to tg-master
```



longest common substrings

convert to patterns

accepted * from * for * sshd2

So here are my next steps:

derive longest common substrings

-----

produce patterns for future matching

-----

they should look like:

```
     accepted * from * for * sshd2
     -----
     stay tuned for smoe results and working code
```

That’s All Folks!