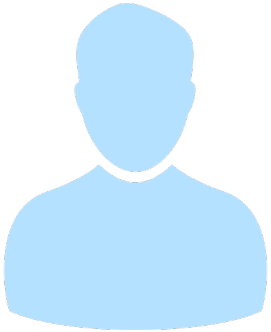
Syslog-ng, getting started, parsing messages, storing in Elasticsearch

Peter Czanik / syslog-ng, a One Identity business

ONE IDENTITY

About me



Peter Czanik from Hungary

Evangelist at One Identity: syslog-ng upstream

syslog-ng packaging, support, advocacy

syslog-ng originally developed by Balabit, now part of One Identity

Overview

- What you need

- What is syslog-ng / the four roles of syslog-ng
- Logging basics
- Configuration, testing
- Networking, relays
- Filters, parsers
- Elasticsearch
- Python (optional) / Q&A

What you need

- Laptop
- Syslog-ng 3.21+
- Elasticsearch & Kibana 7.X

- There is a ready to use VM for VirtualBox/Vmware
- USB key (vm image + slides)

- Copy to HDD, import
- root/workshop, workshop/workshop

syslog-ng

Logging

Recording events, such as:

Jan 14 11:38:48 linux-0jbu sshd[7716]: Accepted publickey for root from 127.0.0.1 port 48806 ssh2

syslog-ng

Enhanced logging daemon with a focus on portability and high-performance central log collection.
Originally developed in C.

Why central logging?

Ease of use

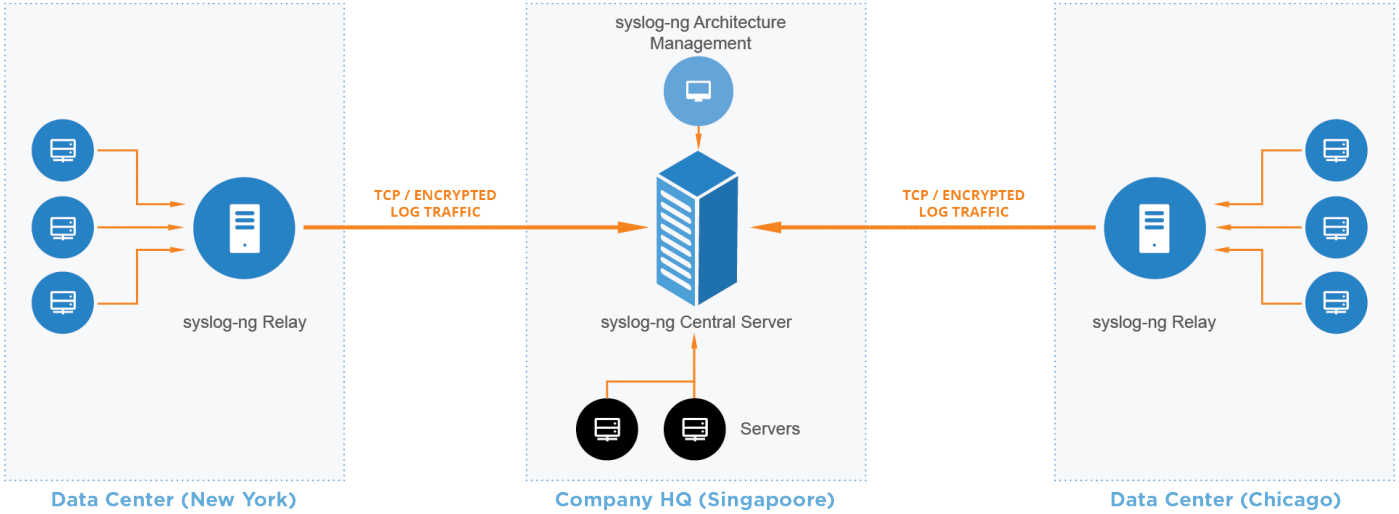
One place to check instead of many

Availability

Even if the sender machine is down

Security

Logs are available even if sender machine is compromised



Main syslog-ng roles

Collector

Processor

Filter

Storage
(or forwarder)

Role: data collector

Collect system and application logs together: contextual data for either side

A wide variety of platform-specific sources:

- /dev/log & co
- Journal, Sun streams

Receive syslog messages over the network:

- Legacy or RFC5424, UDP/TCP/TLS

Logs or any kind of text data from applications:

- Through files, sockets, pipes, application output, etc.

Python source: Jolly Joker

- HTTP server, Amazon CloudWatch fetcher, Kafka source, etc.

Role: processing

Classify, normalize, and structure logs with built-in parsers:

- CSV-parser, PatternDB, JSON parser, key=value parser

Rewrite messages:

- For example: anonymization

Reformatting messages using templates:

- Destination might need a specific format (ISO date, JSON, etc.)

Enrich data:

- GeoIP
- Additional fields based on message content

Python parser:

- all of above, enrich logs from databases and also filtering

Role: data filtering

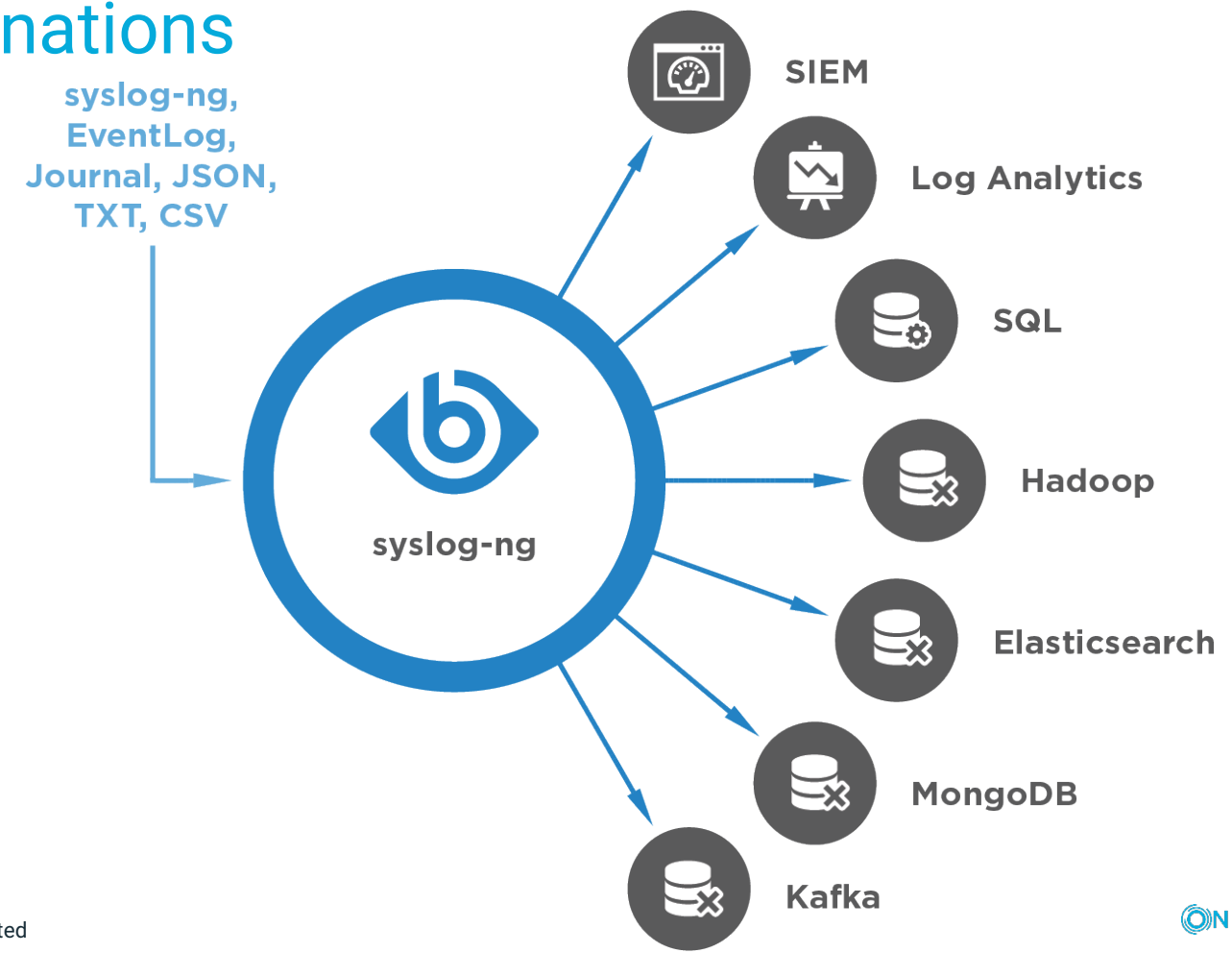
Main uses:

- Discarding surplus logs (not storing debug-level messages)
- Message routing (login events to SIEM)

Many possibilities:

- Based on message content, parameters, or macros
- Using comparisons, wildcards, regular expressions, and functions
- Combining all of these with Boolean operators

Role: destinations



Freeform log messages

Most log messages are: date + hostname + text

Mar 11 13:37:56 linux-6965 sshd[4547]: Accepted keyboard-interactive/pam for root from 127.0.0.1 port 46048 ssh2

- Text = English sentence with some variable parts
- Easy to read by a human
- Difficult to create alerts or reports

Solution: structured logging

Events represented as name-value pairs. For example, an ssh login:

app=sshd user=root source_ip=192.168.123.45

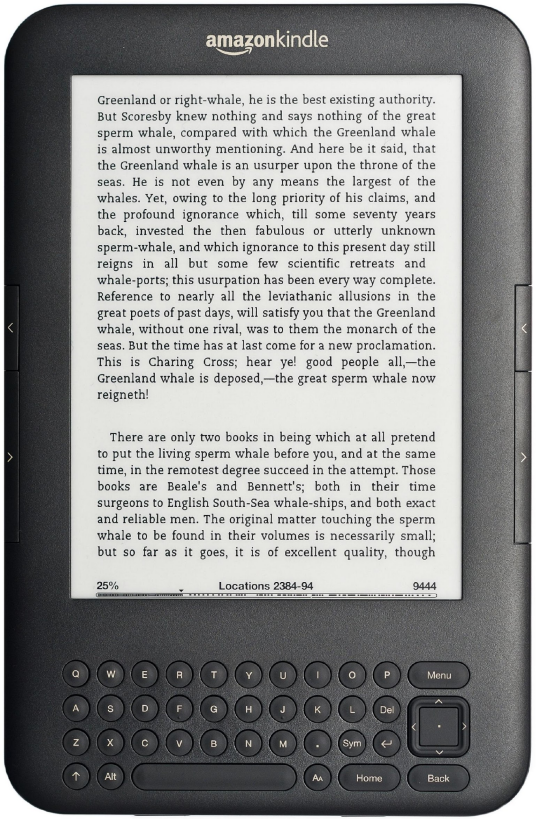
syslog-ng: name-value pairs inside

Date, facility, priority, program name, pid, etc.

Parsers in syslog-ng can turn unstructured and some structured data (CSV, JSON) into name-value pairs

Which is the most used version?

- Project started in 1998
- RHEL EPEL has version 3.5
- Latest stable version is 3.21 released a month ago



Kindle e-book reader

Version 1.6

Configuration & testing

- “Don't Panic”
- Simple and logical, even if it looks difficult at first

- Pipeline model:
  - Many different building blocks (sources, destinations, filters, parsers, etc.)
  - Connected into a pipeline using “log” statements

BASIC ELEMENTS

- **Source**: named collection of source drivers
  - **Source driver**: a tool that implements communication methods of log collection (for example reading /dev/log)
- **Destination**: named collection of destination drivers
  - **Destination driver**: a tool that implements communication methods of log message storing (for example writing messages into a file or sending them through TCP)
- **Log path**: combination of sources, destinations, filters, rewrite statements and parsers for routing messages from sources to destinations.

SOURCE DEFINITION

- Sources contain one or more source drivers where syslog-ng receives log messages:

```
source <identifier> {
  source-driver(parameters);
  source-driver(parameters);
  ...
};
```

- A simple file source:

```
source s_file {
  file("/path/to/the/file.log");
};
```

SOURCE DEFINITION

- Example source with multiple source drivers:

```
source s_files {
    internal();
    file("/path/to/the/first/file.log");
    file("/path/to/the/next/file.log");
    unix-stream("/dev/log");
};
```

SOURCE FLAGS

- Source drivers can have flags:
  - no-parse: disables syslog message parsing, the whole incoming message is stored on the MESSAGE field
  - syslog-protocol: expects RFC5424 message format
  - Further flags → documentation

SOURCE DRIVERS

- internal(): internal messages of syslog-ng
- unix-stream(), unix-dgram(): unix domain sockets
- systemd-journal(): reads systemd's journal files
- file(): opens one file and reads the messages
- pipe(): reads a named pipe
- network(): reads legacy sources
- syslog(): reads the RFC5424 syslog family standard
- sun-stream(): reads streams on Sun Solaris
- program(): runs a program and reads standard output

A COMMON MISTAKE

- Duplicating sources can cause errors:
  - binding twice on the same IP and port
  - multiplicating incoming messages
- Solution:
  - Define a source once and use it twice in different log paths

THE SYSTEM() SOURCE

- Collect system-specific log messages of the host
  - not required to discover all the possible sources of a system
  - standard configuration files are available (one source discovers the current system)
  - A complete replacement of sytemd-journal, /dev/log /proc/kmsg
- Usage:

```
@include "scl.conf"

source s_all {

    system();

};
```

DESTINATION DEFINITION

- Destinations contain one or more destination drivers where syslog-ng sends (stores) log messages:

```
destination <identifier> {
  destination-driver(parameters);
  destination-driver(parameters); ...
};
```

- A simple file destination:

```
destination d_file {
  file("/var/log/syslog");
};
```

DESTINATION DRIVERS

- file(): writes to a file
- pipe(): writes to a named pipe
- unix-stream() and unix-dgram(): writes to a socket
- network(): sends legacy messages over the network
- usertty(): writes to a logged in user terminal
- program(): writes to a program's standard input
- sql(): writes to an sql database
- syslog(): writes the RFC5424 syslog family standard

THE LOG PATH

- Defines the route of the incoming log messages:

```
log {
  source(s_id1);
  destination(d_id1);
};
```

- The log path can contain flags, filters and other objects:

```
log {
  source(s_id1); source (s_id2);...
  filter(f_id1); filter(f_id2);...
  destination(d_id1); destination(d_id2);...
  flags(flag1[,flag2...]);
};
```

A SIMPLE LOG PATH EXAMPLE

```
@version:3.21
source s_devlog {
   unix-stream("/dev/log");
};
destination d_syslog {
   file("/var/log/syslog");
};
log {
  source(s_devlog);
  destination(d_syslog);
};
```

FURTHER ELEMENTS

- Options: set global behavior of syslog-ng
- Macro: element of a parsed log message. They can be used for reconstructing messages.
- Template: user-defined expression for reformatting (restructuring) log messages (for example, adding timezone)
- Filter: expression for selecting (filtering) messages
- Parser: separates message into smaller parts by a separator. The result can be used as a name-value pair in templates.
- Rewrite: a sed-like tool that modifies a part of the message.

/etc/syslog-ng/syslog-ng.conf: getting started

```
@version:3.19
@include "scl.conf"

# this is a comment :)

options {flush_lines (0); keep_hostname (yes);};

source s_sys { system(); internal();};
destination d_mesg { file("/var/log/messages"); };
filter f_default { level(info..emerg) and not (facility(mail)); };

log { source(s_sys); filter(f_default); destination(d_mesg); };
```

SCL: syslog-ng configuration library

- A collection of configuration snippets
- Work like any syslog-ng driver
- Application Adapters (automatic message parsing)
- Credit-card number anonymization
- elasticsearch-http() destination
- and a lot more

Starting syslog-ng

- By default starts in the background
- systemctl [stop|start] syslog-ng
- Stop it now: syslog-ng-ctl stop
- Important options:
  - -s: syntax check
  - -F: start in foreground
  - -v: verbose
  - -d: debug
  - -f path/to/config: use alternate configuration

Testing syslog-ng

- Test it in the foreground
  - Easier to see configuration problems
  - Easier to stop (^C)
- Tools:
  - logger: sends a single message
  - loggen: benchmarking, sending logs from files

Practice the basics

- Backup /etc/syslog-ng/syslog-ng.conf
- Minimal config
- Starting and stopping syslog-ng

syslog-ng.conf: minimal

```
@version:3.19
source s_sys { system(); internal();};
destination d_mesg { file("/var/log/messages"); };
log { source(s_sys); destination(d_mesg); };
```

Testing syslog-ng

- Check the syntax
- Start in the foreground
- Start in the foreground with debugging enabled
- Send some test messages
- Check /var/log/messages

Networking

- RFC 3164 (legacy syslog)
- Three modes of operation: client → relay → server

RFC3164

```
<123>Aug  1 10:28:22 host syslog-ng[12446]: syslog-ng starting up;
version='6.0.0'
```

- Three parts: <PRI>HEADERS MESSAGE
- PRI=8*Facility+Severity
- HEADERS: timestamp, hostname, process and process ID e.g.,

  `Aug  1 10:28:22 host syslog-ng[12446]:`
- MSG: the log message itself
  - e.g., `syslog-ng starting up; version='6.0.0'`

MODES OF OPERATION

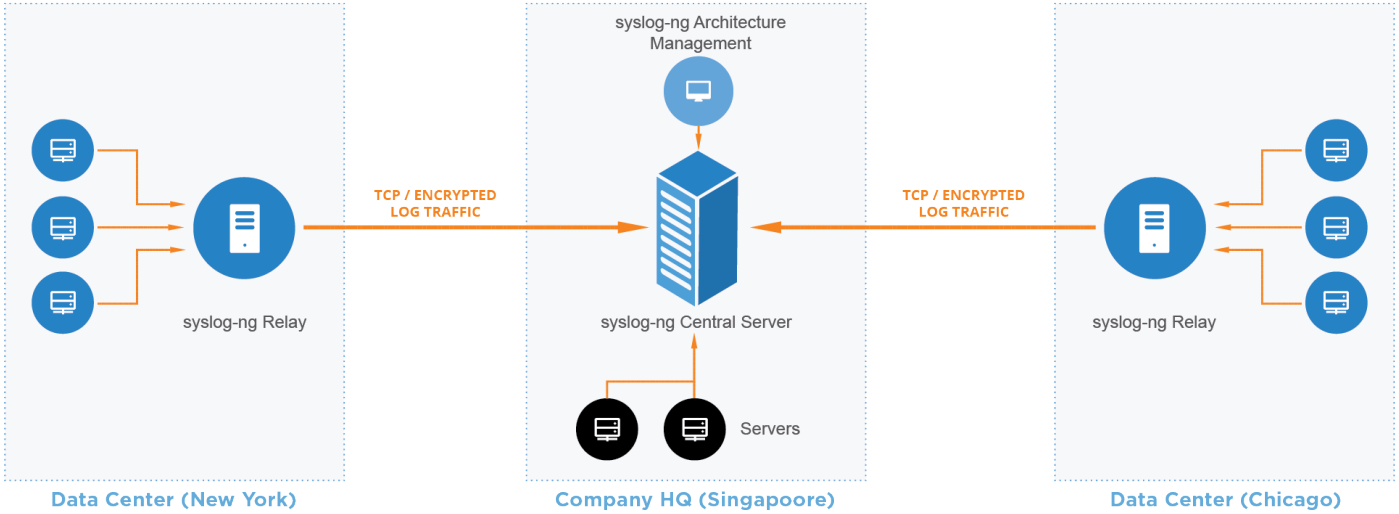
- Client mode: collecting logs from the client and sending them to the remote server (directly or through a relay)
- Relay mode: collecting logs from the clients (through the network) and sending them to the remote server (directly or through another relay)
- Server mode: collecting logs from the clients and storing them locally or in a database

Why relays?

UDP source
Collect as close as possible

Scalability
Distributing processing

Structure
A relay for each site or department



Using logger with a network source

- logger can generate network messages
- logger -T -n 127.0.0.1 -P 514 bla bla bla bla bla
- Important options
  - -T: TCP
  - -n: hostname or IP
  - -P: port
  - Log message

syslog-ng.conf: netsource.conf

```
@version:3.19
source s_sys { system(); internal();};
destination d_mesg { file("/var/log/messages"); };
log { source(s_sys); destination(d_mesg); };

source s_tcp { tcp(port(514)); };
destination d_file { file("/var/log/fromnet"); };
log { source(s_tcp); destination(d_file); };
```

Practice networking

- Network source
- Using logger / loggen

Testing networking (netsource.conf)

- Check the syntax
- Start in the foreground
- Send logs using logger
- Check /var/log/fromnet

Macros and filtering

- Macros are values parsed (or related to) messages

- Routing / discarding log messages
- Tons of filtering functions
- Boolean operators
- Advanced: if / else makes filtering easier

MACROS

- Macros are variables defined by syslog-ng
  - As one syslog message arrives, syslog-ng parses it
  - Macros contain parsed message parts or converted formats
- Example syslog-ng macros:
  - `$FACILITY, $PRIORITY`
  - `$DATE, $ISODATE, $YEAR, $MONTH, $WEEK, $DAY, $HOUR, $MINUTE` etc.

TEMPLATES

- Templates can be used to create standard message formats or filenames.
- A simple message formatting template and its usage:

```
template t_syslog {
  template("$ISODATE $HOST $MSG\n");
};
destination d_syslog {
  file("/var/log/syslog" template(t_syslog));
};
```

TEMPLATES

- A simple file path defined by template:

```
destination t_demo1 {
  file("/var/log/$HOST/messages.log" create_dirs(yes));
};

destination t_demo2 {
  file("/var/log/$HOST_messages.log");
};
```

LOG ROTATION

- Log rotation using syslog-ng macros:

```
destination d_messages {
File(
"/var/log/$R_YEAR/$R_MONTH/$HOST_$R_DAY.log"
create_dirs(yes));
};
```

Declaring filters

- Just like any other building block:
  - filter name { filterfunction(); };
  - filter f_default { level(info..emerg) and not (facility(mail)); };

AVAILABLE FILTERS

- level: filters for the severity
- facility: filters for the facility
- host: filters hostname
- program: filters for the running program
- match: filters by regular expression
- netmask: filters by sender IP or subnet
- filter: uses a different filter
- tags: filters for a classified message tag

/etc/syslog-ng/syslog-ng.conf: filter

```
@version:3.19
@include "scl.conf"

source s_sys { system(); internal();};
destination d_mesg { file("/var/log/messages"); };
filter f_default { level(info..emerg) and not (facility(mail)); };

log { source(s_sys); filter(f_default); destination(d_mesg); };
```

THE INLIST() FILTER

Filtering based on white- or blacklisting

- Compares a single field with a list of values
- One value per line in text file

Use cases

- Poor man's SIEM: alerting based on spammer / C&C / etc. IP address lists
- Filtering based on a list of application names

If/else

- Conditional expressions in log path
- Makes it easier to use the results of filtering
- if (filter()) { do this }; else { do that };
- For example, use different parsers on different logs

/etc/syslog-ng/syslog-ng.conf: iftest

```
@version:3.21
@include "scl.conf"

source s_sys { system(); internal();};
destination d_mesg { file("/var/log/messages"); };
log { source(s_sys); destination(d_mesg); };

filter f_sudo {program("sudo")};

destination d_sudoall {
  file("/var/log/sudo.json"
  template("$(format-json --scope nv_pairs --scope dot_nv_pairs --scope rfc5424)\n\n"));
};

log {
    source(s_sys);
    filter(f_sudo);
    if (match("workshop" value(".sudo.SUBJECT"))) {
        destination { file("/var/log/sudo_filtered"); };
    };
    destination(d_sudoall);
};
```

Practice filtering

- Filter functions
- If/Else

Practice filtering

- Send logs using logger and different priority setting to the simple filter (filter.conf)
- Filter sudo logs to a separate file and format it to JSON (iftest1.conf)
- Save logs from user “workshop” to a separate file (iftest2.conf)

Parsing

- Structuring, classifying and normalizing log messages
- PatternDB for unstructured logs
- JSON, XML, CSV, etc. parsers for structured log messages
- Advantages:
  - More precise filtering (alerting)
  - Save only relevant data

PATTERNDB PARSER

Extracts information from unstructured messages into name-value pairs

- Add status fields based on message text
- Message classification (like LogCheck)

Needs XML describing log messages

Example: an ssh login failure:

- Parsed: app=sshd, user=root, source_ip=192.168.123.45
- Added: action=login, status=failure
- Classified as “violation”

JSON PARSER

Turns JSON-based log messages into name-value pairs

{"PROGRAM":"prg00000","PRIORITY":"info","PID":"1234","MESSAGE":"seq: 0000000000, thread: 0000, runid: 1374490607, stamp: 2013-07-22T12:56:47 MESSAGE... ","HOST":"localhost","FACILITY":"auth","DATE":"Jul 22 12:56:47"}

CSV PARSER

Parses columnar data into fields

```
parser p_apache {
    csv-parser(columns("APACHE.CLIENT_IP", "APACHE.IDENT_NAME", "APACHE.USER_NAME",
        "APACHE.TIMESTAMP", "APACHE.REQUEST_URL", "APACHE.REQUEST_STATUS",
        "APACHE.CONTENT_LENGTH", "APACHE.REFERER", "APACHE.USER_AGENT",
        "APACHE.PROCESS_TIME", "APACHE.SERVER_NAME")
         flags(escape-double-char,strip-whitespace) delimiters(" ") quote-pairs("""[]')
         );
};
destination d_file { file("/var/log/messages-${APACHE.USER_NAME:-nouser}"); };
log { source(s_local); parser(p_apache); destination(d_file);};
```

KEY=VALUE PARSER

Finds key=value pairs in messages

Introduced in version 3.7.
Typical in firewalls, like:
Aug 4 13:22:40 centos kernel: IPTables-Dropped: IN= OUT=em1 SRC=192.168.1.23 DST=192.168.1.20 LEN=84 TOS=0x00 PREC=0x00 TTL=64 ID=0 DF PROTO=ICMP TYPE=8 CODE=0 ID=59228 SEQ=2
Aug 4 13:23:00 centos kernel: IPTables-Dropped: IN=em1 OUT= MAC=a2:be:d2:ab:11:af:e2:f2:00:00 SRC=192.168.2.115 DST=192.168.1.23 LEN=52 TOS=0x00 PREC=0x00 TTL=127 ID=9434 DF PROTO=TCP SPT=58428 DPT=443 WINDOW=8192 RES=0x00 SYN URGP=0

FURTHER PARSERS

XML, Linux Audit, Date

XML

Linux Audit

- /var/log/audit/audit.log
- MSG often parsed further for extra info

Date

- Uses templates
- Saves to sender date

SCL: syslog-ng configuration library

Apache, Cisco

Apache access logs

- Combines CSV and date parsers

Cisco

- Cisco logs are similar to syslog messages
- Can parse many but not all Cisco logs

PARSERS WRITTEN IN PYTHON

Python parser

- Released in syslog-ng 3.10

- Parse complex data formats
- Enrich logs from external data sources, like SQL, whois, etc.

- Slower than C
- Does not need compilation or a development environment

Application adapters, Enterprise wide message model

Application adapters

- Parse messages easily
- Syslog and a few sample parsers (Cisco, sudo), more coming
- Enabled by default from 3.13

Enterprise wide message model

- Forward name-value pairs between syslog-ng instances (JSON)
- Can preserve original message

/etc/syslog-ng/syslog-ng.conf: application adapter

```
@version:3.21
@include "scl.conf"

source s_sys { system(); internal();};
destination d_mesg { file("/var/log/messages"); };
log { source(s_sys); destination(d_mesg); };

filter f_sudo {program(sudo)};

destination d_test {
  file("/var/log/sudo.json"
  template("$(format-json --scope nv_pairs --scope dot_nv_pairs --scope rfc5424)\n\n"));
};

log {
    source(s_sys);
    filter(f_sudo);
    if (match("czanik" value(".sudo.SUBJECT"))) {
        destination { file("/var/log/sudo_filtered"); };
    };
    destination(d_test);
};
```

Enriching log messages

- Additional name-value pairs based on message content
- PatternDB
- GeoIP
- add-contextual-data

ENRICHING LOG MESSAGES

PatternDB

GeoIP: find the geo-location of an IP address

- Country name or longitude/latitude
- Detect anomalies
- Display locations on a map

Add metadata from CSV files

- For example: host role, contact person
- Less time spent on locating extra information
- More accurate alerts or dashboards

Using loggen with a network source

- loggen can generate logs or post existing log file
- loggen -i -S -d -R /root/iptables_nohead_short localhost 514
- Important options
  - -i: Internet
  - -S: TCP and unix-stream
  - -d: don’t parse
  - -R /path/to/file : read log messages from a file
  - Host & port

Iptables sample logs

Feb 27 14:31:01 bridge kernel: INBOUND UDP: IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1 SRC=212.123.153.188 DST=11.11.11.82 LEN=404 TOS=0x00 PREC=0x00 TTL=114 ID=19973 PROTO=UDP SPT=4429 DPT=1434 LEN=384

Feb 27 14:34:41 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1 SRC=206.130.246.2 DST=11.11.11.100 LEN=40 TOS=0x00 PREC=0x00 TTL=51 ID=9492 DF PROTO=TCP SPT=2577 DPT=80 WINDOW=17520 RES=0x00 ACK FIN URGP=0

Feb 27 14:34:55 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1 SRC=4.60.2.210 DST=11.11.11.83 LEN=48 TOS=0x00 PREC=0x00 TTL=113 ID=3024 DF PROTO=TCP SPT=3124 DPT=80 WINDOW=64240 RES=0x00 SYN URGP=0

/etc/syslog-ng/syslog-ng.conf: kv parser & GeoIP

```
@version:3.19
source s_sys { system(); internal();};
destination d_mesg { file("/var/log/messages"); };
log { source(s_sys); destination(d_mesg); };

parser p_kv {kv-parser(prefix("kv.")); };
parser p_geoip2 { geoip2( "${kv.SRC}", prefix( "geoip2." ) database( "/usr/share/GeoIP/GeoLite2-City.mmdb" ) ); };

source s_tcp { tcp(port(514)); };
destination d_file {
  file("/var/log/fromnet" template("$(format-json --scope rfc5424
       --scope dot-nv-pairs --rekey .* --shift 1 --scope nv-pairs
       --exclude DATE --key ISODATE @timestamp=${ISODATE})\n\n")  );
};
log {
  source(s_tcp);
  parser(p_kv);
  parser(p_geoip2);
  destination(d_file);
};
```

Practice parsing & enrichment

- GeoIP
- template

Practice parsing & enrichment

- Send iptables logs to network source (geoip1.conf)
- Parse using kv parser (geoip2.conf)
- Parse using GeoIP parser (geoip3.conf)

Elasticsearch

- Old: Java-based destination
- Can not be included in distros
- New: wrapper around the http() destination
- Might be more resource intensive at extreme load

/etc/syslog-ng/syslog-ng.conf: elasticsearch-http

```
destination d_elasticsearch_http {
    elasticsearch-http(
        index("syslog-ng")
        type("")
        url("http://localhost:9200/_bulk")
        template("$(format-json --scope rfc5424
        --scope dot-nv-pairs --rekey .* --shift 1 --scope nv-pairs
        --exclude DATE --key ISODATE @timestamp=${ISODATE})")
    );
};
```

/etc/syslog-ng/syslog-ng.conf: GeoIP rewrite

```
rewrite r_geoip2 {
    set(
        "${geoip2.location.latitude},${geoip2.location.longitude}",
        value( "geoip2.location2" ),
        condition(not "${geoip2.location.latitude}" == "")
    );
};
```

Mapping

```
{
  "mappings" : {
      "properties" : {
        "geoip2" : {
          "properties" : {
            "location2" : {
              "type" : "geo_point"
            }
          }
        }
      }
  }
}
```

Practice Elasticsearch

- System logs
- GeoIP
- All together (if conditional)

Practice Elasticsearch & Kibana

- Send system logs to Elasticsearch (elastic1.conf)
- Send firewall logs to Elasticsearch (elastic2.conf)
- Add kv parser and GeoIP (elastic3.conf)
- Combine the two with an if conditional (elastic4.conf)

Python in syslog-ng

- Python bindings: configuration + code
- Can pass parameters to Python code
- Only the class name is mandatory in config
- Python code can be in-line in a python {} block, or stored in external file(s)

Python destination: mandatory

- Only the class name is mandatory in config
- Only send() method is mandatory
- Name-value pairs as
  - object – all
  - dict – only those configured

Python destination: optional

- Many non-mandatory options, like disk-buffer, etc.

- init() and deinit()
  - When syslog-ng started or reloaded
- open() and close()
  - start/reload or when sending fails

A simple file destination

```
destination d_python_to_file {
    python(
        class("TextDestination")
    );
};

log {
    source(src);
    destination(d_python_to_file);
};

python {
class TextDestination(object):
    def send(self, msg):
        self.outfile = open("/tmp/example.txt", "a")
        self.outfile.write("MESSAGE = %s\n" % msg["MESSAGE"])
        self.outfile.flush()
        self.outfile.close();
        return True
};
```

Python parser

- Only parse() method is mandatory
- Name-value pairs only as object
  - Can create new: log_message['hostname.dest'] = 'myname’

- <38>2018-10-03T18:00:17 localhost prg00000[1234]: seq: 0000001451, thread: 0000, runid: 1538582416, stamp: 2018-10-03T18:00:17 PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD PADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADDPADD PADDPADDPADDPADDPADD

Python parser: config

```
parser my_python_parser{
    python(
        class("SngRegexParser")
        options("regex", "seq: (?P<seq>\\d+), thread: (?P<thread>\\d+), runid: (?P<runid>\\
d+), stamp: (?P<stamp>[^ ]+) (?P<padding>.*$)")
    );
};

log {
    source { tcp(port(5555)); };
    parser(my_python_parser);
    destination {file("/tmp/regexparser.log.txt" template("seq: $seq thread: $thread runid:
$runid stamp: $stamp my_counter: $MY_COUNTER\n"));
    };
};
```

Python parser: code

```
python {

import re

class SngRegexParser(object):
def init(self, options):
        """
        Initializes the parser
        """
        pattern = options["regex"]
        self.regex = re.compile(pattern)
        self.counter = 0
        return True
```

Python parser: code continued

```
    def deinit(self):
        pass
    def parse(self, log_message):
        decoded_msg = log_message['MESSAGE'].decode('utf-8')
        match = self.regex.match(decoded_msg)
        if match:
            for key, value in match.groupdict().items():
                log_message[key] = value
            log_message['MY_COUNTER'] = str(self.counter)
            self.counter += 1
            return True
        return False
};
```

Python source

- Options, like time zone handling
- Name-value pairs as object
- Two modes
  - server
  - fetcher (syslog-ng handles the eventloop)

- Server: the run() and request_exit() methods are mandatory
- Fetcher: only the fetch() method is mandatory

Simple “server” source

```
source s_python {
    python(
        class("MySource")
        options(
            "option1" "value1",
            "option2" "value2"
        )
    );
};

destination d_file { file("/var/log/python.txt"); };

log { source(s_python); destination(d_file); };
```

Simple “server” source continued

```
python {
from syslogng import LogSource
from syslogng import LogMessage

class MySource(LogSource):
    def init(self, options): # optional
        print("init")
        print(options)
        self.exit = False
        return True

    def run(self): # mandatory
        print("run")
        while not self.exit:
            msg = LogMessage("this is a log message")
            self.post_message(msg)

    def request_exit(self): # mandatory
        print("exit")
        self.exit = True
};
```

Simple “fetcher” source: config

```
source s_loadavg {
    python-fetcher(
      class("loadavg.Loadavg")
      options("interval" "1")
    );
};

destination d_file {
    file("/var/log/loadavg"
      template("$(format-json --scope rfc5424 --scope nv-pairs)\n")
    );
};

log {
  source(s_loadavg);
  destination(d_file);
};
```

Simple “fetcher” source: code

```
import time
from syslogng import LogFetcher
from syslogng import LogMessage

class Loadavg(LogFetcher):
    def __init__(self): # optional
        print("constructor")
        self.fname = '/proc/loadavg'
        self.interval = 0

    def init(self, options): # optional
        print(options)
        try:
            self.interval = int(options["interval"])
            return True
        except:
            print("configure 'interval' in syslog-ng.conf as a positive number")
            return False
```

Simple “fetcher” source: code continued

```
def open(self): # optional
    """
    opens the file
    """
    print("open")
    self.fhandle = open(self.fname)
    return True

def close(self): # optional
    """
    closes the file
    """
    print("close")
    self.fhandle.close()
```

Simple “fetcher” source: code continued

```
def fetch(self): # mandatory
    time.sleep(self.interval)

    self.fhandle.seek(0, 0)
    line = self.fhandle.readline()
    loadavgtmp = line.split()
    runtmp = loadavgtmp[3].split("/")

    msg = LogMessage()
    msg["loadavg.load1"] = loadavgtmp[0]
    msg["loadavg.load5"] = loadavgtmp[1]
    msg["loadavg.load15"] = loadavgtmp[2]
    msg["loadavg.runcurr"] = runtmp[0]
    msg["loadavg.runproc"] = runtmp[1]
    msg["loadavg.lastpid"] = loadavgtmp[4]
    return LogFetcher.FETCH_SUCCESS, msg
```

Debugging

- Logging to internal() from Python code
- From syslog-ng 3.20

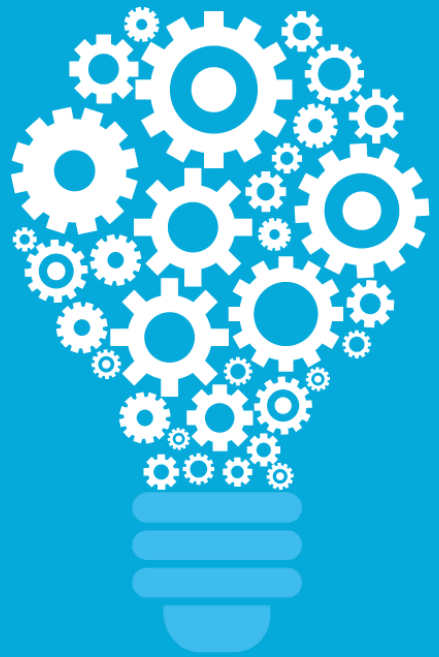
```
import syslogng
logger = syslogng.Logger()
logger.error("plain text message: ERROR")
logger.warning("plain text message: WARNING")
logger.info("plain text message: INFO")
logger.debug("plain text message: DEBUG")
```

Further examples

- MQTT destination: https://www.syslog-ng.com/community/b/blog/posts/writing-python-destination-in-syslog-ng-how-to-send-log-messages-to-mqtt
- Parsers: https://www.syslog-ng.com/community/b/blog/posts/parsing-log-messages-with-the-syslog-ng-python-parser
- HTTP source: https://www.syslog-ng.com/community/b/blog/posts/creating-an-http-source-for-syslog-ng-in-python

What’s new in syslog-ng

- Disk-based buffering
- Grouping-by(): generic correlation
- Python bindings
- HTTP(s) destination:
  - Splunk, Elasticsearch
  - Telegram, Slack, etc.
- Wildcard file source
- Performance and memory usage improvements
- Many more :-)



syslog-ng benefits

High-performance reliable log collection

Simplified architecture

Single application for both syslog and application data

Easier-to-use data

Parsed and presented in a ready-to-use format

Lower load on destinations

Efficient message filtering and routing

Join the community!

- syslog-ng: http://syslog-ng.org/
- Source on GitHub: https://github.com/balabit/syslog-ng
- Mailing list: https://lists.balabit.hu/pipermail/syslog-ng/
- Gitter: https://gitter.im/balabit/syslog-ng

Questions?

syslog-ng blog: https://syslog-ng.com/community/

My e-mail: peter.czanik@oneidentity.com

Twitter: https://twitter.com/PCzanik

ONE IDENTITY