Log-Normalisierung in Echtzeit

Rainer Gerhards

Agenda

- Problemstellung
- Klassifikation von Log-Formaten
- Existierende Tools
- Wie arbeitet liblognorm?
- Praktische Anwendung

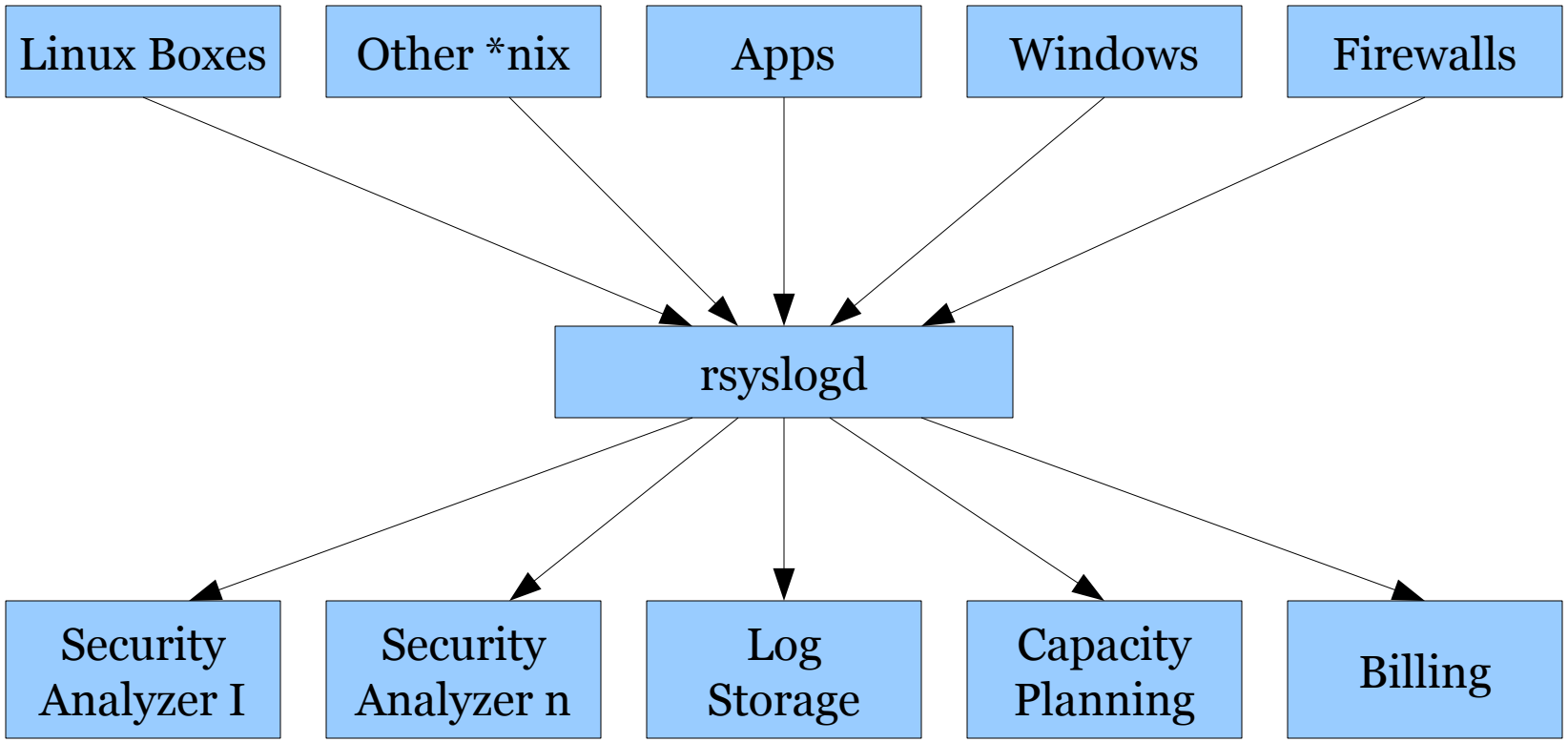
Problemstellung

- Das gleiche Ereignis (z.B. Login) wird sehr unterschiedlich dargestellt
- Manchmal bedeuten sehr ähnlich aussehende Meldungen auch Unterschiedliches (z.B. bei Firewalls)
- Ziel daher: die verschiedenen Ausgangsformate übersetzen in
  - Einheitliches Zielformat
  - Oder… verschiedene Zielformate, die diverse Tools verstehen

Ein reales Problem – aus meiner Mail...

“I am working with a customer who is deploying a large rsyslog environment for central logging. Basically they want a cluster of boxes to act as the "log of record". They would also like to have the logs fed to a couple security products for analysis. **The customer has a limited budget so having each vendor write parsers is cost prohibitive. A commonality for each of the additional destinations is the ability to ingest logs in <some common format>.** I believe rsyslog has the capability to alter the output...”

Beispiel: Rsyslog als Konverter



Lösungsansätze

- Custom-written Parsers (viele SIEMs)
- Grok/logstash
- Syslog-ng PatternDB
- Rsyslog mmnormalize (liblognorm v1)

Betrachten wir erst einmal Eingangsformate…

- Strukturierte Formate
  - JSON (inkl. CEE)
  - Apache CLF
  - WELF (Webtrends Enhanced Log Format)
- Semi-Strukturierte Formate
  - Werden allgemein als “strukturiert” bezeichnet
  - Haben aber sehr, sehr viele Varianten
  - Bsp: CSV: Feldtrenner, Anführungszeichen, …
- Freitext
  - Natürlichsprachiger Text

Log Repository for Research

- Log-Samples für Untersuchungen und Entwicklung
- Verfügbar unter http://git.adiscon.com/?p=log-samples-for-research.git;a=summary

- github Hosting war aufgrund der dortigen Größenbeschränkungen nicht möglich
- Log Contributions werden gerne genommen!

Freitext?

- “`pam_unix(gdm-password:session): session opened for user rger by (unknown)(uid=0)`”

- Ursprünglich für menschlichen Leser gedacht
- Ist aber nicht wirklich Freitext, vielmehr ein “strukturiertes Format, dessen Struktur sehr vielfältig und inkonsistent” ist.
- Dennoch automatisch auswertbar

Beispiel: Linux syslog()

```
syslog(LOG_CRIT, "Attempted login by %s on %s", user, tt);
syslog(LOG_ERR, "user: %s: shell exceeds maximum pathname size",
syslog(LOG_ERR, "tried to pass user \"%s\" to login",
syslog(LOG_DEBUG, "login name is +%s+, of length %d, [0] = %d\n",
syslog(LOG_ERR, "setlogin(%s): %m - exiting",
```

- API ermöglicht es nicht, Parameter zu Kennzeichnen
- Problemfall Leerzeichen
- inkonsistent

Genauere Problemdefinition

- Bei der Log-Normalisierung möchten wir
  - Freitextformate,
  - aber möglichst auch strukturierte
- mit einem einheitlichen Prozess in ein Ausgangsformat umformatieren,
- und das **hinreichend schnell für Realzeit-Anwendungen**.

Traditioneller Lösungsansatz: Iterieren über regex

- Für jedes Format wird ein regex angegeben
- Alle regex werden in Regel-DB gespeichert
- Algorithmus:

```
für jede Meldung:
    iteriere über die Regel-DB:
        führe aktuellen regex aus
        Passt?
            Ja → fertig
            nein → weiter mit nächstem regex
```

Pro und Con

- Pro
  - Sehr Benutzerfreundlich
  - Regex sind bekannt, leicht zu schreiben
- Con
  - Laufzeit hängt von der Anzahl der regex ab
  - Schlimmer noch: (reale) regex engines sind relativ langsam

Bsp: grok

- Häufig genutzt (z.B. Logstash)
- Sehr einfach, da umfangreiche Custom-Types existieren
- Aber: Durchsatz ist mehr als problematisch…

Umgebung Durchsatztests

- Intel Core 2 Quad Core @ 2,4Ghz
- Cache: L1: 64KB, L2: 4MB
- 8 GB Hauptspeicher
- → total “Unspektakuläre Hardware”
- Ubuntu 14.04LTS
- Ansonsten unbenutztes System
- Ausgaben nach /dev/null geleitet
- Mehrere Durchläufe, Mittelbildung

Grok: Durchsatz

- Test-Datensatz
  - Cisco PIX
  - 20 Mio Log-Records
- Rule-Base
  - 34 Regeln, für alle vorkommenden Formate
- Ausführung mit Command-Line Tool:

1/34:	18.400mps	109 Sek
34/34:	695mps (!)	48 Minuten
1/1	103.500mps	19 Sek
real-life	2.200mps	15 Minuten

Grok: Fazit

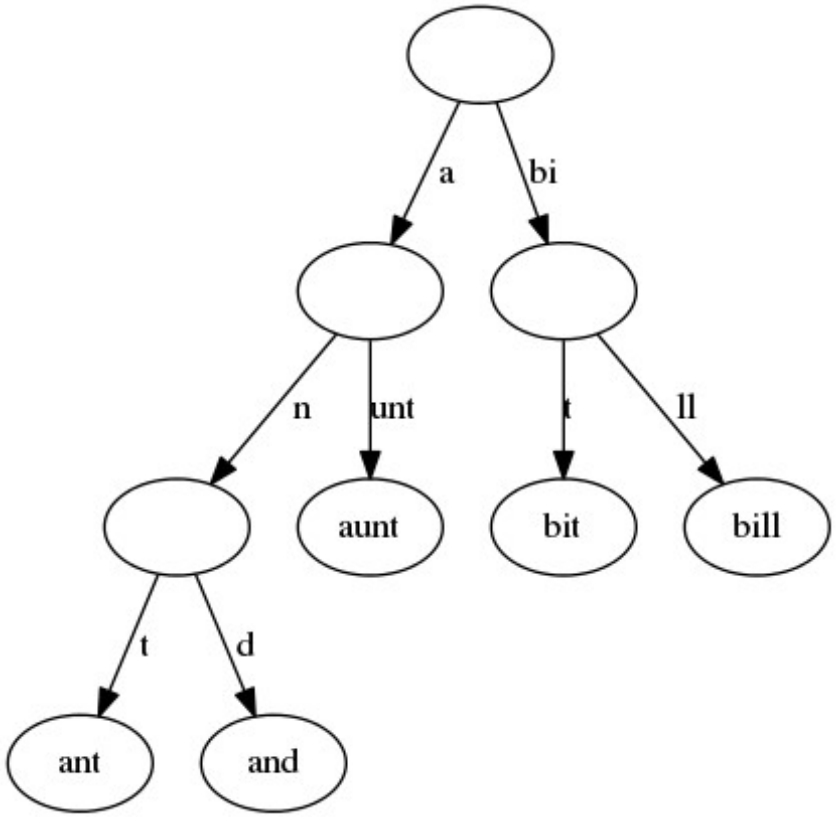
- Leicht zu benutzen
- Für große Datenmengen und Realzeit nur mit immensem Hardwareaufwand nutzbar

Alternative Ansätze: Prefix Match Normalizer

- Beginn ca. 2010, syslog-ng pattern db und liblognorm v1
- Basisidee altbekant
  - 1960 Edward Fredkin “trie”
  - Ursprünglich für rasche Suche entwickelt
  - Wurde in diesem Umfeld auch immer Fortentwickelt
  - “Radix Tree”

Grundidee

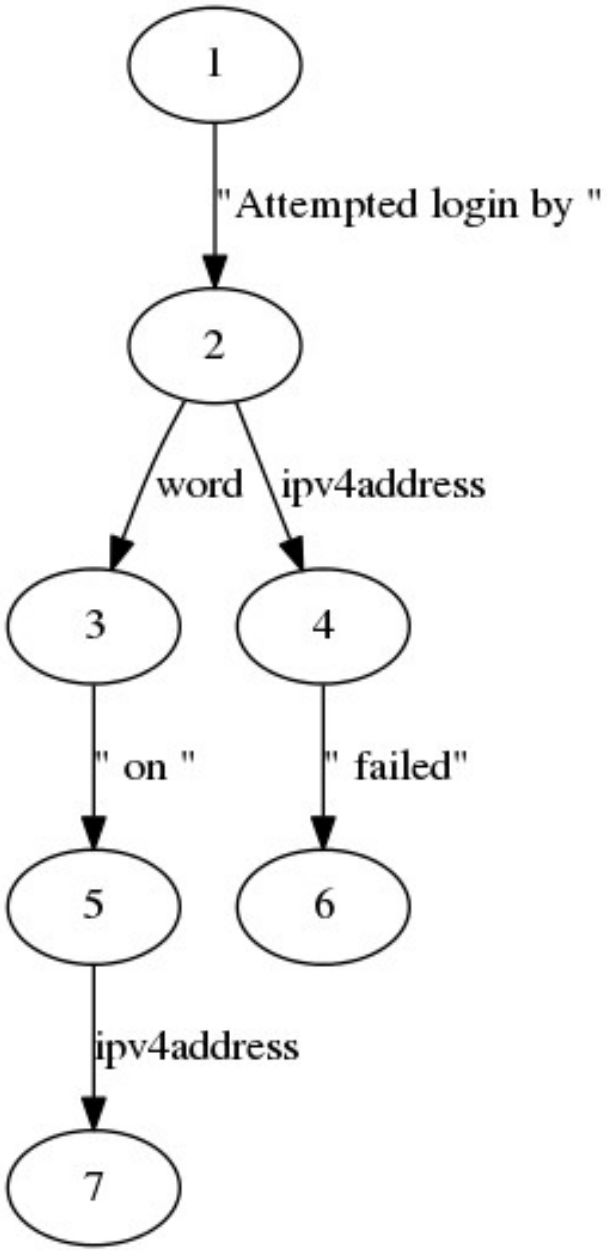
- Alle Regeln werden in **einen** Baum aufgenommen
- Die Kanten mit den Suchbegriffen beschriftet
- Man sucht ab der Wurzel
- Suchzeit abhängig nur von Baumtiefe



Anpassung an die Log-Normalisierung

- Wurzel ist der Anfang der Log-Meldung! (keine Iteration innerhalb der Meldung)
- aber: wir haben nicht nur konstanten Text
- Es gibt auch immer wiederkehrende “Objekte”, “Datentypen” oder “Parser”
  - IP-Adressen
  - Port-Nummern
  - Zahlen (in div. Formaten)
  - Benutzernamen
- Parser sind unterschiedlich aufwändig (z.B. Benutzer, WELF)

Beispiel für einen “Normaliserungs-baum” (PRT)



Nachteile durch die Anpassung

- Laufzeit wird nicht mehr nur durch die Tiefe des Baumes bestimmt, sondern hängt auch von seiner Struktur ab:
  - Welche Parser werden verwendet?
  - Wie viele alternative Parser gibt es auf einer Baumebene?
- Backtracking kann erforderlich werden, somit theoretisch exponentielle Laufzeit (wie regex) → praktisch kommt das nicht vor…

Ergebnisse PRT

- Mit syslog-ng pdb, liblognorm v1
- Gleicher PIX 20m Datensatz wie bei grok
- Ausführung mit Command-Line Tool:

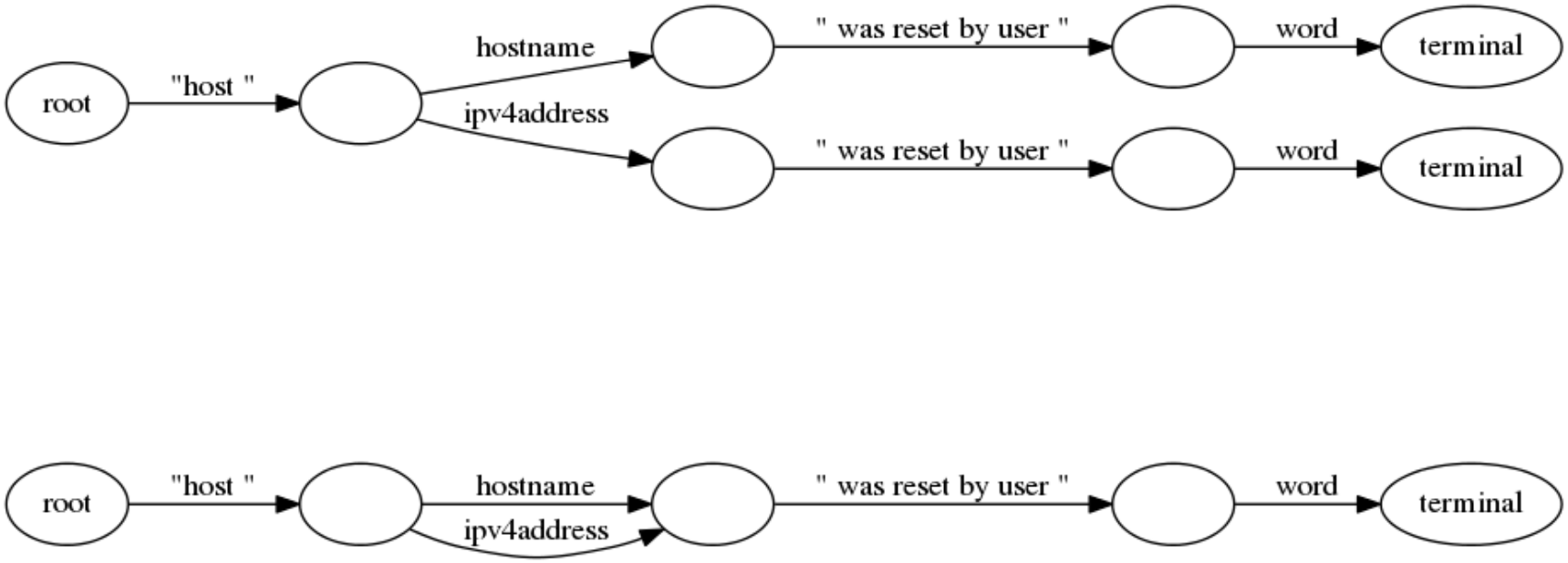
grok	2.200mps	15 Minuten
syslog-ng	80.500mps	4Min., 5Sek.
LognormV1	149.200mps	2Min., 14Sek.

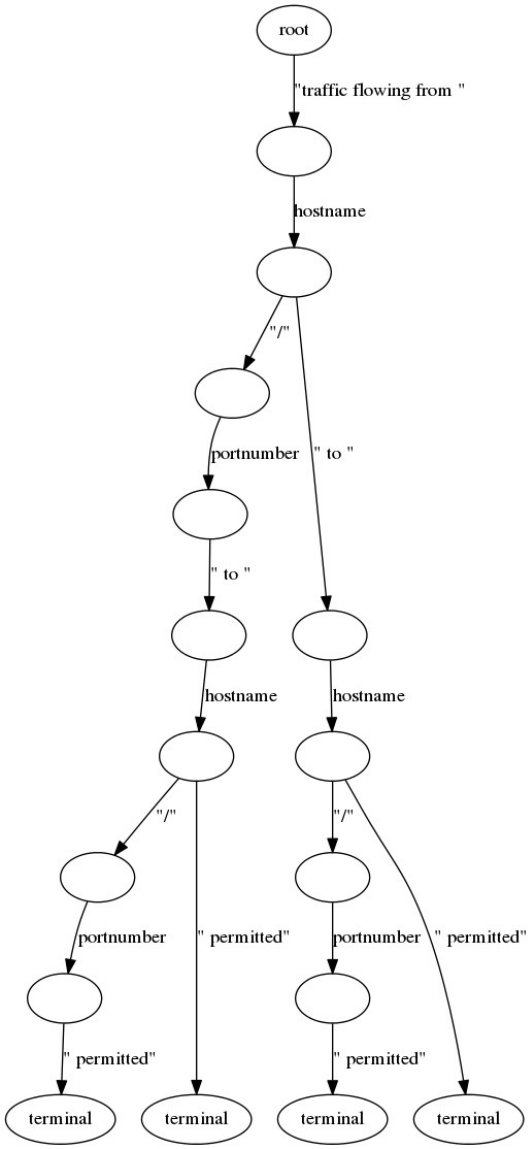
- Deutlich bessere Ausführungszeiten
- Realzeit-fähig zumindest für mittlere Lasten, bzw. bei akzeptablem Hardware-Einsatz

Weiterentwicklung des PRT: liblognorm v2

- Teils hoher Speicherbedarf
- Nicht optimal im Hinblick auf Cache-Nutzung
- Benutzerunfreundlich durch
  - Spezielle Regelsprache
  - Notwendigkeit, Parser in C zu implementieren
- Fortentwicklung von
  - Datenstruktur (→ Parse DAG, “PDAG”)
  - Konfiguration

Optimierungen: Alternativen und “Comon Suffix”

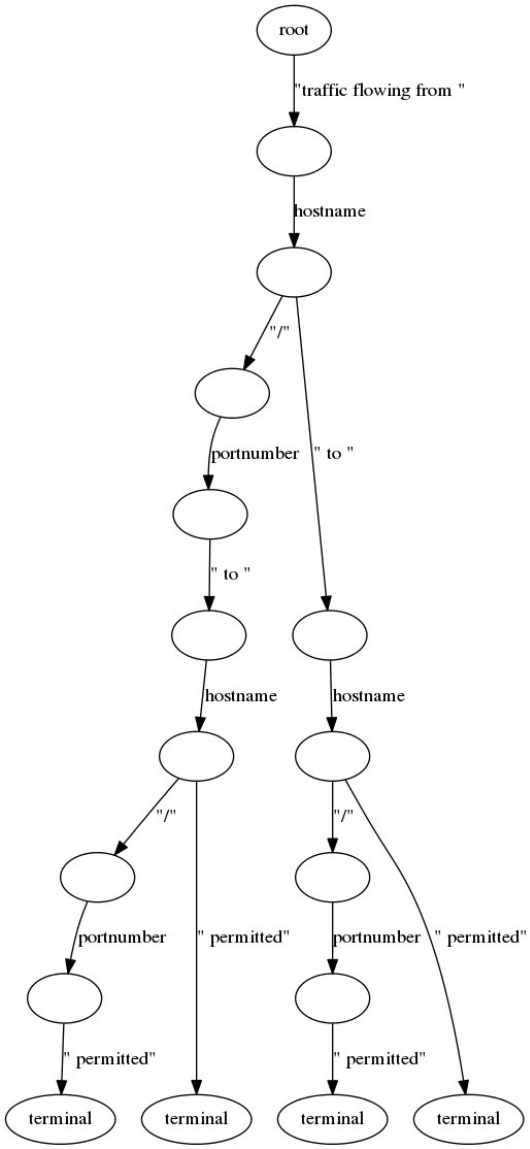


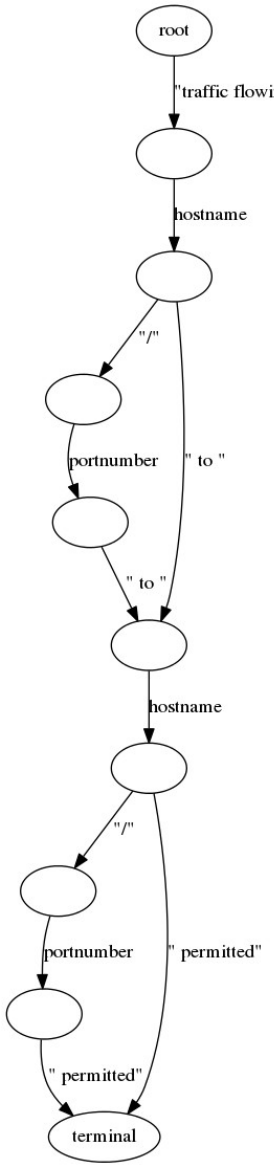


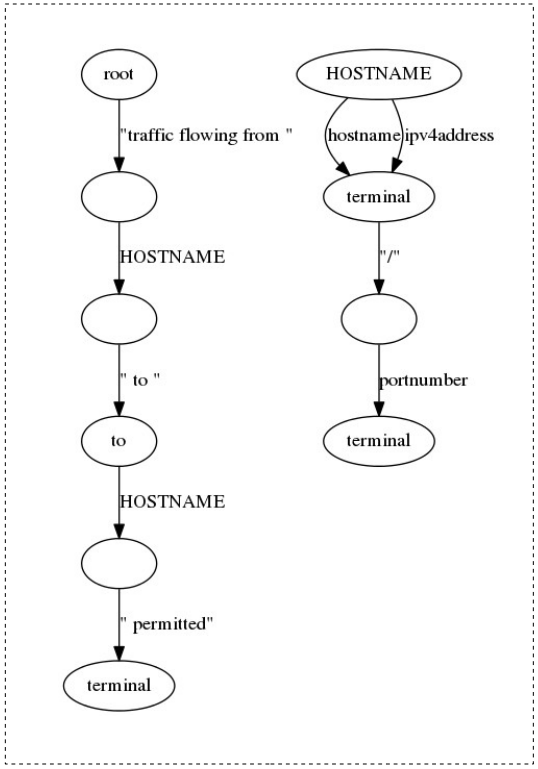
Beispiel (etwas “fabriziert”):

```
traffic flowing from
host[/port] to host[/port]
permitted
```

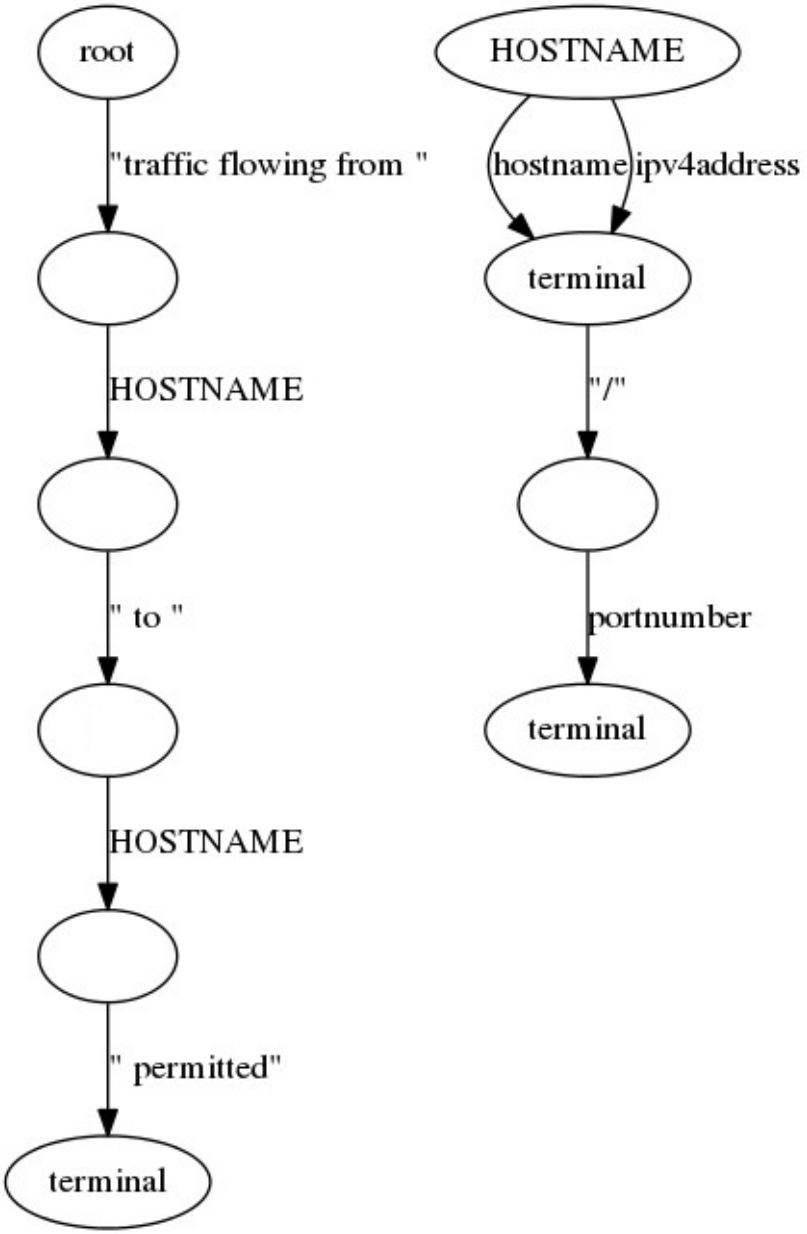
Evolution zum PDAG







Evolution zum PDAG


root
"traffic flowing from "
HOSTNAME
" to "
HOSTNAME
" permitted"
terminal
HOSTNAME
hostname
ipv4address
terminal
"/"
portnumber
terminal

Liblognorm v2 Skalierbarkeit

- Gleicher PIX Datensatz
- 37 Regeln, die alle Fälle abdecken
- Ausführung mit Command-Line Tool:

1/37:	381.770mps	52,39 Sek
37/37:	381.685mps	52,41 Sek
1/1	386.614mps	51,73 Sek
real-life	*192.521mps*	*104 Sek*

- Abweichung “real-life” durch sehr große Meldungen (alle Lösungen sensitiv zu Meldungsgröße)

Liblognorm v2 aus Anwendersicht

- Schneller
- Leichtere, an JSON angelehnte Beschreibungssprache
- Benutzerdefinierbare Typen gestatten eine ähnlich einfache Nutzung wie bei grok
  - Es ist eine Standard-Typ -Bibliothek geplant
- Release mit rsyslog in April
  - Achtung: opt-in in Rulebase erforderlich! (“version=2” in erster Zeile)

Rulebase-Format

- Früher auch “samples” genannt
- Grundidee: Beispiel (sample) einer Log-Meldung
  - Konstanter Text primär zu Erkennung
  - Parser zur Extraktion der Daten

```
rule=:%date:date-rfc3164% %hostname:word%
%date2:date-rfc3164% %tag:word% Built
%direction:word% TCP connection %connr:number% for
faddr %faddr:cisco-interface-spec% gaddr
%gaddr:cisco-interface-spec% laddr %laddr:cisco-
interface-spec%
```

Erweiterungen des Rulebase-Formats

- Zeilenumbrüche gestattet
- Unterschiedliche Beschreibungsformate
- Erweiterungen
  - alternative
  - Repeat
  - Parser-Prioritäten
- Neue Parser
  - cisco-interface-spec
  - checkpoint-lea
  - cef …
- Parameter-Eingrenzungen (z.B. maxint)

Custom Types

- Ermöglicht es, eigene “parser” zu definieren
- Typnamen beginnen mit “@”
- Typdefinition weitgehend “normales” rule format
- Können verwendet werden wie Basistypen
- Typebibliothek im Aufbau begriffen

```
type=@IPaddr:%ip:ipv4%
type=@IPaddr:%ip:ipv6%

rule=:host %host:@IPaddr% connected
```

Wie anwenden?

- rsyslog mmnormalize
- Command line Tool “lognormalizer”
- Als Bibliothek in eigenen (C-)Anwendungen

Fragen?

- rgerhards@adiscon.com