Strukturiertes Loggen mit rsyslog

Rainer Gerhards

Agenda

- Strukturiertes vs. Free-Form Logging
- CEE/Lumberjack
- Wie funktioniert das innerhalb rsyslog?
- Was gibt es Neues in rsyslog?
- Ein Normalisierungsszenario, ganz praktisch

- … bei dem mir dann wahrscheinlich die Zeit ausgeht
  - Gerne nach dem Vortrag
  - Anfang März auf www.rsyslog.com vollständig beschrieben

Logging ist einfach – oder?

- Wir erzeugen einfach einen log-record, wenn etwas Interessantes passiert…
- ABER
  - Was ist “interessant”?
  - Was ist erforderlich, um das Ereignis zu beschreiben?
  - Woher wissen wir, was die einzelnen Datenfelder bedeuten?
  - Wie sieht der Log-Record überhaupt aus?
- So… logging, wenn es Sinn machen soll, ist leider doch nicht ganz so einfach…

Das Logging Dilemma

- Es existiert kein allgemein akzeptiertes Format
- Gleich aussehende Logs beschreiben teilweise sehr unterschiedliche Ereignisse
- Das gleiche Ereignis wird durch sehr unterschiedlich aussehende Logs beschrieben
- Häufig wird (Pseudo)-Freitext verwendet
- Für “Log-Consumer” ist es schwierig, zumindest ein sinnvolles Subset von Logformaten zu unterstützen

Einige unausweichliche Wahrheiten I

- Logging ist langweilig
  - “Log quickly, log dirty, get over it”
  - Niemand wird bewundert für “gutes Logging”
- Logging “Economics”
  - Es **kostet** Geld, gut zu loggen
  - Man **verdient** gutes Geld mit der Interpretation von schlechtem Logging
  - Diejenigen, die die Kosten tragen müssen, sind andere als die, die die Gewinne erzielen!
- Nicht alles was neuer ist, ist auch besser…

Einige unausweichliche Wahrheiten II

- Die “unstrukturierte Text Log Dualität”:
  - Wenn ein log Format keinen Freitext unterstützt, wird es nicht genutzt (zumindest nicht mehr als unvermeidbar)
  - Wenn es (unter anderem) Freitext unterstützt, wird der Freitext missbraucht
  - → unstrukturierte Logs werden nicht verschwinden!
- Nichts schlägt die installierte Basis…
  - Wer glaubt tatsächlich, dass jeder seinen Code umschreibt, nur um einen neuen Logging-Standard zu unterstützen…?

Wie löst man das Schlamassel?

- Es wurden viele Versuche unternommen
  - Über viele Jahre…
  - Mit sehr beschränktem Erfolg
- Führten zu einem Projekt namens "Common Event Expression" (CEE)
  - Herstellerübergreifendes Team (sowohl OSS als auch closed source)
  - Unter Führung vom US MITRE
  - Baut auf **existierender** Infrastruktur auf

CEE: Kernideen

- KISS: Keep it stupid simple
- Aufbauend auf
  - existierenden Technologien
  - Bekannten & akzeptierten Verfahren
- Format
  - name/value Paare
  - Die Struktur wird so flach wie möglich gehalten, aber eine Hierarchie dennoch erlaubt
  - Dictionaries für Feldnamen, Syntax und Semantik
  - Profiles spezifizieren, was in bestimmten Ereignisstypen enthalten sein muss

Projekt Lumberjack

- Entstand vor einem Jahr auf der Fedora Developer Konferenz in Brünn
- Zielsetzung
  - Basierend auf CEE, entwickelt Ideen weiter, auch unabhängig von CEE (quasi ein “Fork”)
  - Kernfunktionalität als Open Source Lösung
  - Eine Lösung, die “einfach läuft”
- Voran getrieben von Logging Professionals von Red Hat, Balabit (syslog-ng) und Adiscon (rsyslog), offen für jeden Interessenten

facebook Suche nach Personen, Orten und Dingen

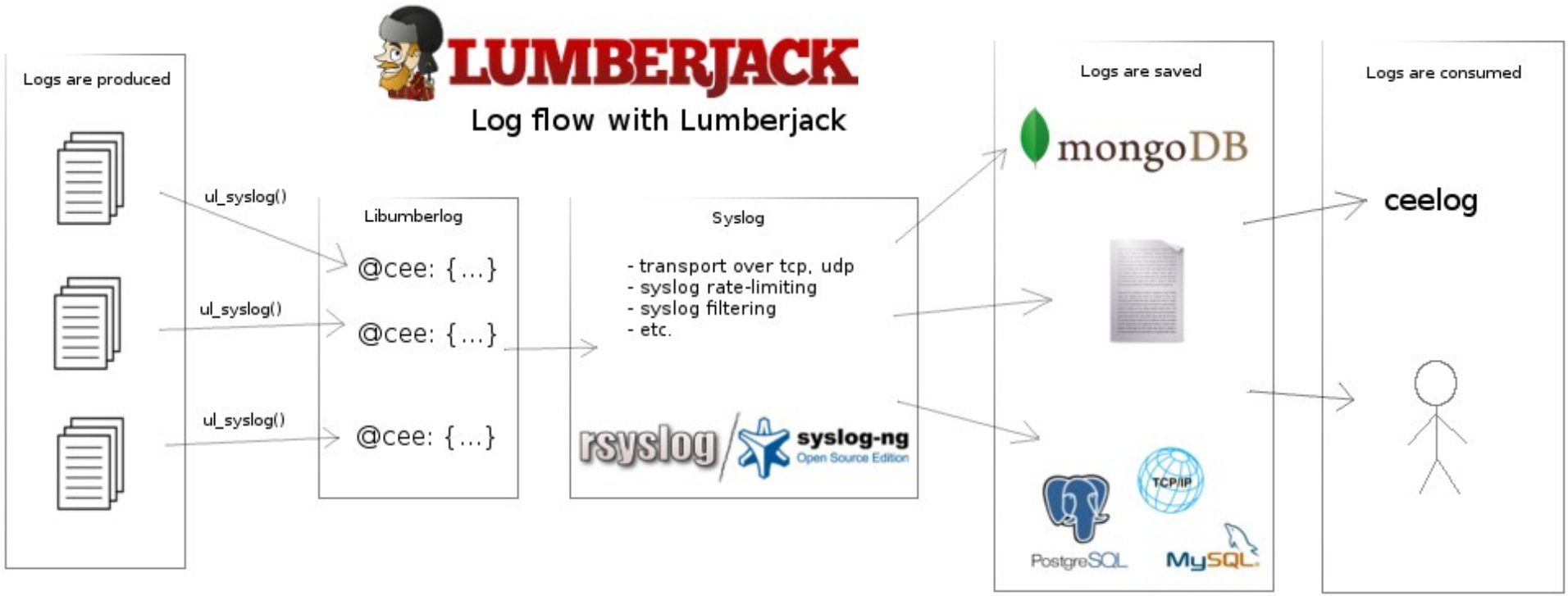
BalaBit IT Security · 706 gefällt das
vor 22 Stunden ·

Gefällt mir

It is the beginning of a beautiful friendship...between rsyslog and syslog-ng. Although we are competitors, we cooperate in log standardization efforts. That makes someone a syslog-ng Superhero :-)

Finally back from DevConf with some gifts ;-)

Von der Lumberjack Web Site



Quelle: https://fedorahosted.org/lumberjack/, Abruf 2013-02-27, 11:50

Lumberjack-supporting APIs

- libumberlog
  - Native Library für Lumberjack
  - Legacy syslog() kann mittels preload erweitert werden
- liblogging
  - Ursprünglich für RFC3195 entwickelt
  - Emulator für systemd journal API
  - cross-logging API (libstdlogging)
    - Einfach zu benutzen
    - Log-Provider Auswahl dynamisch zur Laufzeit
    - Multiple Log-Kanäle innerhalb einer App

Das CEE/Lumberjack Format

- “@cee:” Cookie kennzeichnet Lumberjack Format
- Rest der Meldung muss gültiges JSON sein
  - Falls nicht --> keine Lumberjack Meldung (verhindert False Positives!)
  - Geschachtelte Objekte sind erlaubt
  - Arrays sind nativ nicht in der Spec, werden aber künftig erwaretet…
- Einfach zu benutzen über eine **Vielzahl** von existierenden “Transports” (nicht nur syslog!)

```
@cee: { "feld1": "value1", "feld2": "value2" }
```

Rsyslog Erweiterungen für Project Lumberjack

- Vollständige Unterstützung für JSON
- Neue message modification Module: audit support, mmjsonparse, mmnormalize
- Neue Outputs: elasticsearch, mongodb
- input für journal structured log data in Kürze (merge request existiert schon)
- Eine komplett neue Konfigurationssprache (aber die Alte wird weiterhin supported!)
  - Custom Variablen
  - Geschachtelte if … then … else; call-statements
  - Neue Template-Typen

Strukturiert vs Free-Form

- Es gibt Tools á la Splunk, die direkt auf unstrukturierten Logs arbeiten
- ABER
  - Sie sind sehr rechenzeitintensiv
  - Sie arbeiten (noch?) nicht gut mit SIEM Tools zusammen
- High-speed Filtering auf Zwischensystemen (relays!) funktioniert viel besser (nur?) mit strukturierten Logs
- Nahezu alle aktuellen Tools erwarten Name/Value Paare

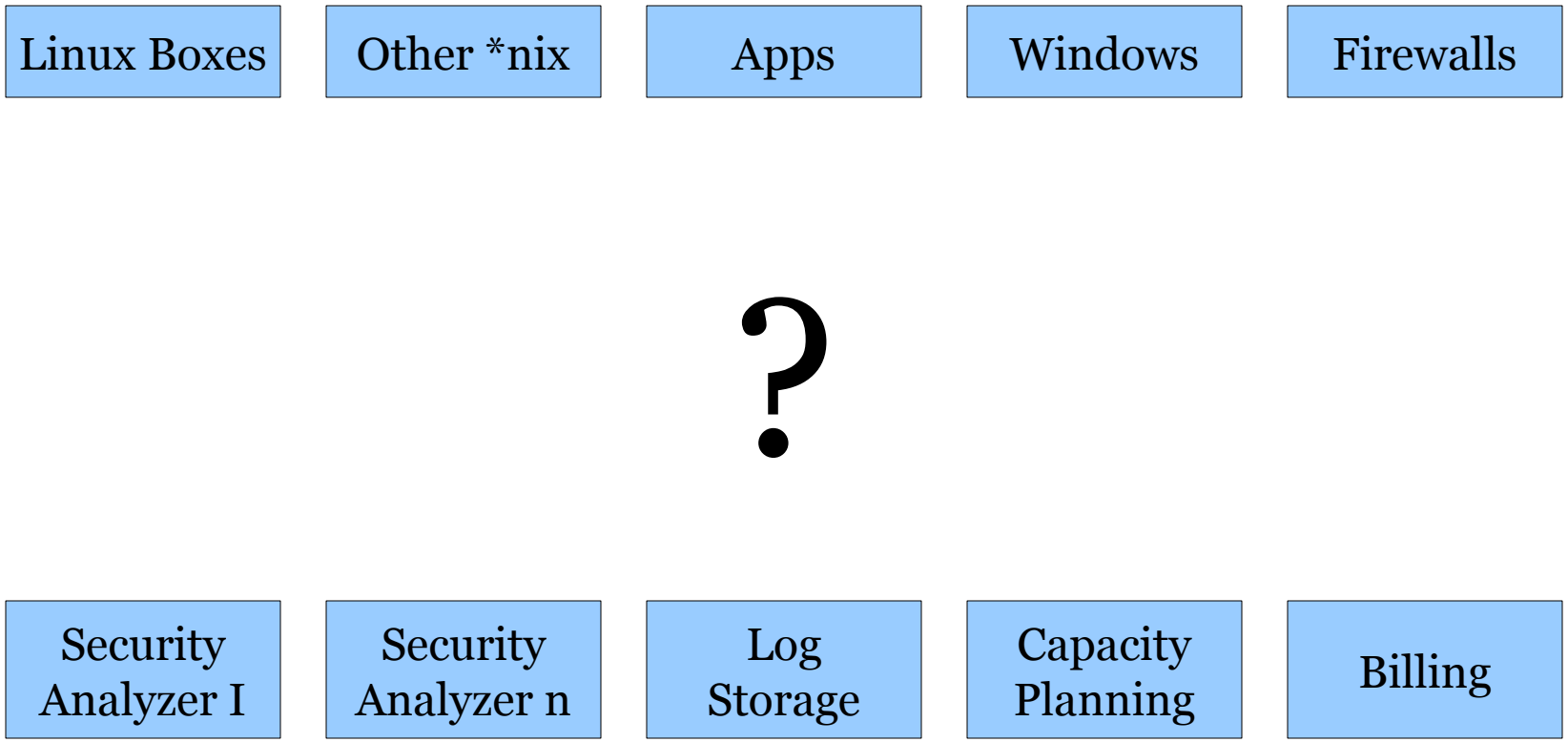
Einige Ankündigungen: rsyslog

- Linux Journal support wird verbessert
  - Input plugin (merge-request vorhanden)
  - Output Plugin (schon im Tree, 7.3.7)
  - Optional rsyslog-Statusmeldungen ins Journal
- Starke Signaturen für Log-Files
  - Im Laufe des ersten Halbjahres (erster Code schon im Tree)
  - Standards-basierend (RFC3161, OpenKSI)
  - Langfristig multiple Crypto-Provider Plugins
- Langfristig geplant
  - Dynamische Config (Red Hat+Adiscon)

Normalisierung ist ein “real-world problem”!
Aus meiner Mailbox:

*“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.** ”*

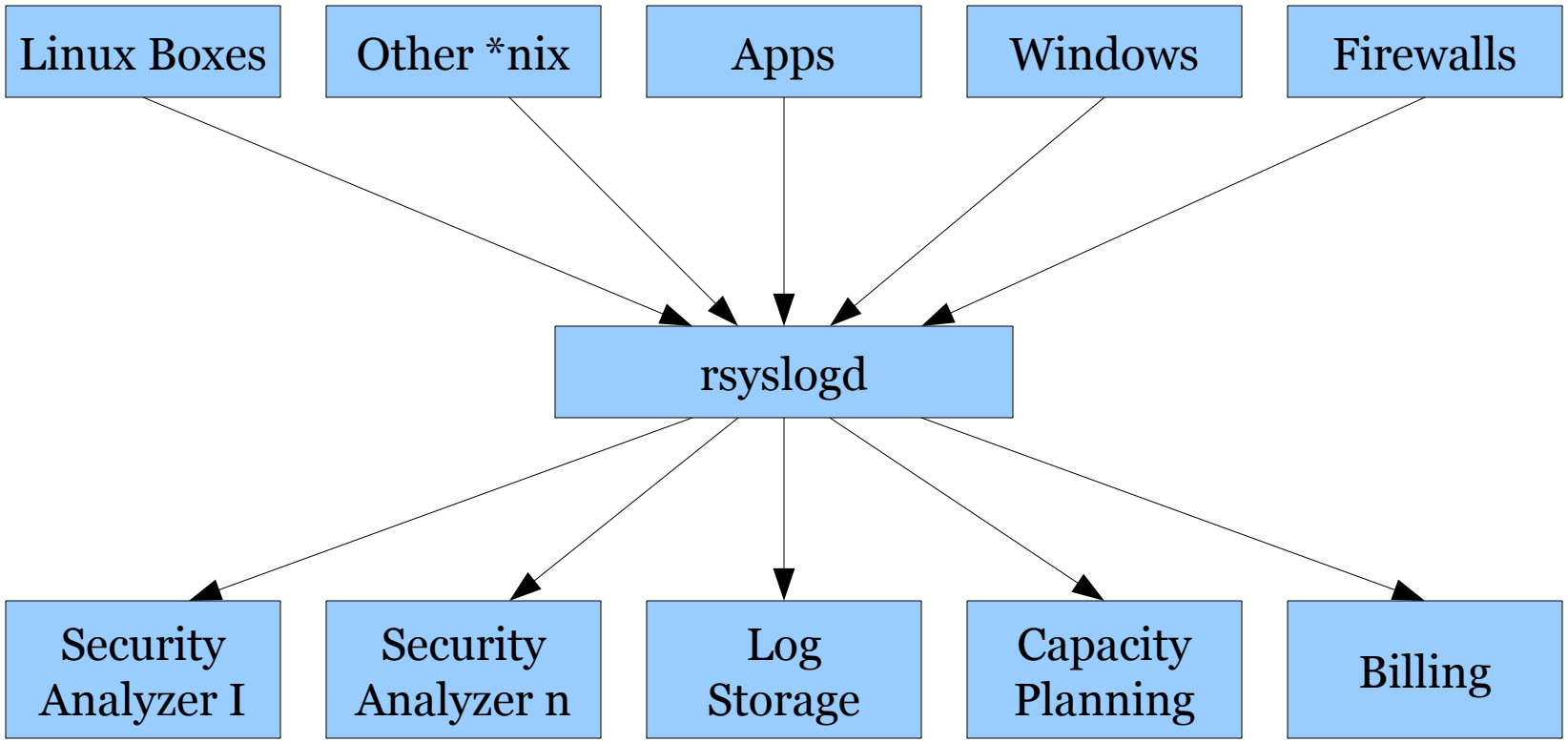
Log Erzeuger und Konsumenten



Einige wichtige Log-Quellen

- Unstrukturierte Text Formate
  - Traditionelle syslog Meldungen
  - Application Text Log Dateien
- Strukturierte Formate
  - Windows Event Log
  - Linux Journal (z.Zt. Jedoch primär Freitext)
  - Application Text Log Dateien (XML, CSV, WELF, Apache CLF, wasAuchImmer, ...)
  - SNMP traps
  - New-style syslog (lumberjack)

Rsyslog als Konverter



Zurück zu meiner Mailbox...

"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..."*

Einige rsyslog Basics...

- Ruleset
  - Entspricht grob einer Funktion in Programmiersprachen
  - Besteht aus (bedingten) Anweisungen und Aktionen
  - Aufruf aus einem anderen Ruleset oder Bindung an einen Listner
- Variablen
  - Meldungs Variablen (z.B. $msg, $rawmsg)
  - System Variablen (z.B. $$now)
  - Strukturierte Variablen: bilden eine Baumstruktur, z.B. $!usr!somevar

Ein kleines Beispiel…

- Ziel: vereinheitlichte logs für logon/logoff Report
  - Zur Verarbeitung durch ein Analysetool (hier nicht gezeigt)
  - Im Beispiel nur 4 Felder: Hostsystem, Empfangszeit, Username, logon/logoff Status
- Eingaben
  - Linux: traditionelle text log Meldungen
  - Windows: verschiedene Agents
- Ausgabe
  - Lumberjack JSON
  - CSV

Konfiguration der Eingangsserver

```
module(load="imtcp")

/* We assume to have all TCP logging (for simplicity)
 * Note that we use different ports to point different sources
 * to the right rule sets for normalization. While there are
 * other methods (e.g. based on tag or source), using multiple
 * ports is both the easiest as well as the fastest.
 */
input(type="imtcp" port="13514" Ruleset="WindowsRsyslog")
input(type="imtcp" port="13515" Ruleset="LinuxPlainText")
input(type="imtcp" port="13516" Ruleset="WindowsSnare")
```

Das Linux Eingabebeispiel

- Unstrukturierter Text

Jan 16 09:28:33 rger-virtual-machine sudo: pam_unix(sudo:session): session opened for user root by rger(uid=1000)
Jan 16 09:28:33 rger-virtual-machine sudo: pam_unix(sudo:session): session closed for user root
Jan 24 02:38:49 rger-virtual-machine sshd[2414]: pam_unix(sshd:session): session opened for user rger by (uid=0)
Jan 24 02:41:22 rger-virtual-machine sshd[2414]: pam_unix(sshd:session): session closed for user rger

Parsing von des Textes: mmnormalize

- Benutzt eine “sample rule base”
  - Ein Beispie für jeden erwarteten Meldungstyp
  - Beispiel enthält Text (muss “passen”) und Feldbeschreibungen (z.B. IPv4 Address, char-matches, …)
  - Wenn das Beispiel zutrifft, werden die entsprechenden Felder extrahiert
  - Spezieller Parser für iptables
- Implementiert als rsyslog-action
- Sehr schneller Algorithmus (viel schneler als regex)
- Basiert auf liblognorm

Die Rulebase für unser kleines Beispiel...

# SSH and sudo logins
prefix=**%rcvdat:date-rfc3164% %rcvdfrom:word%**

rule=: sshd[**%-:number%**]: pam_unix(sshd:session): session **%type:word%** for user **%user:word%** by (uid=**%-:number%**)

rule=: sshd[%-:number%]: pam_unix(sshd:session): session %type:word% for user %user:word%rule=: sudo: pam_unix(sudo:session): session %type:word% for user root by %user:char-to:(%(uid=%-:number%)

rule=: sudo: pam_unix(sudo:session): session %type:word% for user %user:word%

So sieht's dann in rsyslog.conf aus:

```
/* plain Linux log messages (here: ssh and sudo) need to be
 * parsed - we use mmnormalize for fast and efficient parsing
 * here.
 */
ruleset(name="LinuxPlainText") {
        action(type="mmnormalize"
       rulebase="/home/rger/proj/rsyslog/linux.rb" userawmsg="on")
        if $parsesuccess == "OK" and $!user != "" then {
                if $!type == "opened" then
                        set $!usr!type = "logon";
                else if $!type == "closed" then
                        set $!usr!type = "logoff";
                set $!usr!rcvdfrom = $!rcvdfrom;
                set $!usr!rcvdat = $!rcvdat;
                set $!usr!user = $!user;
                call outwriter
        }
}
```

Windows Horrors: SNARE

- Tab-Getrenntes Chaos:

<131>Feb 10 15:48:12 Win2008StdR2x64_vm MSWinEventLog#0111#011Security#0114#011Tue Feb 05 16:39:27 2013#0114624#011Microsoft-Windows-Security-Auditing#011WIN2008STDR2X64\Administrator#011N/A#011Success Audit#011Win2008StdR2x64_vm#011Anmelden#011#011Ein Konto wurde erfolgreich angemeldet. Antragsteller: Sicherheits-ID: S-1-5-18 Kontoname: WIN2008STDR2X64$ Kontodomäne: WORKGROUP Anmelde-ID: 0x3e7 Anmeldetyp: 2 Neue Anmeldung: Sicherheits-ID: S-1-5-21-3148105976-3029560809-1855765213-500 Kontoname: Administrator Kontodomäne: WIN2008STDR2X64 Anmelde-ID: 0x1d1feb Anmelde-GUID: {00000000-0000-0000-0000-000000000000} Prozessinformationen: Prozess-ID: 0xc40 Prozessname: C:\Windows\System32\winlogon.exe Netzwerkinformationen: Arbeitsstationsname: WIN2008STDR2X64 Quellnetzwerkadresse: 127.0.0.1 Quellport: 0 Detaillierte Authentifizierungsinformationen: Anmeldeprozess: User32 Authentifizierungspaket: Negotiate Übertragene Dienste: - Paketname (nur NTLM): - Schlüssellänge: 0 Dieses Ereignis wird beim Erstellen einer Anmeldesitzung generiert. Es wird auf dem Computer

Nun denn: Extrahierung basierend auf Feldposition...

```
ruleset(name="WindowsSnare") {
        set $!usr!type = field($rawmsg, "#011", 6);
        if $!usr!type == 4634 then {
                set $!usr!type = "logoff"; set $!doProces = 1;
        } else if $!usr!type == 4624 then {
                set $!usr!type = "logon"; set $!doProces = 1;
        } else  set $!doProces = 0;
        if $!doProces == 1 then {
                set $!usr!rcvdfrom = field($rawmsg, 32, 4);
                set $!usr!rcvdat = field($rawmsg, "#011", 5);
                /* we need to fix up the snare date */
                set $!usr!rcvdat = field($!usr!rcvdat, 32, 2) & " " &
                                   field($!usr!rcvdat, 32, 3) & " " &
                                   field($!usr!rcvdat, 32, 4);
                set $!usr!user = field($rawmsg, "#011", 8);
                call outwriter }
}
```

Windows: rsyslog Agent

- natives Lumberjack Format mit Windows Feldnamen
- Ein stukturiertes Chaos ;-)

<133>Feb 05 11:15:56 win7fr.intern.adiscon.com EvntSLog: **@cee:** {"source": "win7fr.intern.adiscon.com", "nteventlogtype": "Security", "sourceproc": "Microsoft-Windows-Security-Auditing", "id": "4634", "categoryid": "12545", "category": "12545", "keywordid": "0x8020000000000000", "user": "N\\A", "TargetUserSid": "S-1-5-21-803433813-209592097-1264475144-8733", "TargetUserName": "fr", "TargetDomainName": "ADISCON", "TargetLogonId": "0xb8c7aed", "LogonType": "7", "catname": "Logoff", "keyword": "Audit Success", "level": "Information", "msg": "An account was logged off.\r\n\r\nSubject:\r\n\tSecurity ID:\t\tS-1-5-21-803433813-209592097-1264475144-8733\r\n\tAccount Name:\t\tfr\r\n\tAccount Domain:\t\tADISCON\r\n\tLogon ID:\t\t0xb8c7aed\r\n\r\nLogon Type:\t\t\t7\r\n\r\nThis event is generated when a logon session is destroyed. It may be positively correlated with a logon event using the Logon ID value. Logon IDs are only unique between reboots on the same computer."}

Parsing Lumberjack Data: mmjsonparse

- Überprüft, ob die Meldung Lumberjack structured data enthält
  - Falls ja
    - Felder werden ausgelesen
    - Es werden die Feldnamen aus der Meldung genutzt
  - Falls nicht: fülle das Lumberjack “msg” Feld
- Implementiert als rsyslog Action
  - Kann basierend auf Regeln aufgerufen werden, und daher auch für nur bestimmte Ereignisse

rsyslog.conf:

```
/* the rsyslog Windows Agent uses native Lumberjack format
 * (better said: is configured to use it)
 */
ruleset(name="WindowsRsyslog") {
        action(type="mmjsonparse")
        if $parsesuccess == "OK" then {
                if $!id == 4634 then
                        set $!usr!type = "logoff";
                else if $!id == 4624 then
                        set $!usr!type = "logon";
                set $!usr!rcvdfrom = $!source;
                set $!usr!rcvdat = $timereported;
                set $!usr!user = $!TargetDomainName &
                         "\\" & $!TargetUserName;
                call outwriter
        }
}
```

Was haben wir bisher erreicht?

- Wir akzeptieren Eingaben von verschiedenen Quellen
  - Unstrukturierter Text
  - Tab-getrennt, semi-strukturiert
  - Nativ Lumberjack
- Wir extrahierten exakt die gleichen Felder aus allen Meldungen
- Und haben diese in unsren Variablen im $!usr Zweig gespeichert

Nun müssen wir nur noch das normalisierte Output schreiben

```
/* this ruleset simulates forwarding to the final destination */
ruleset(name="outwriter"){
            action(type="omfile"
           file="/home/rger/proj/rsyslog/logfile.csv" template="csv")
            action(type="omfile"
           file="/home/rger/proj/rsyslog/logfile.cee" template="cee")
}
```

Templates machen die eigentliche Arbeit

```
template(name="csv" type="list") {
        property(name="$!usr!rcvdat" format="csv")
        constant(value=",")
        property(name="$!usr!rcvdfrom" format="csv")
        constant(value=",")
        property(name="$!usr!user" format="csv")
        constant(value=",")
        property(name="$!usr!type" format="csv")
        constant(value="\n")
}

template(name="cee" type="string"
        string="@cee: %$!usr%\n")
```

Die kombinierte CEE Datei:

@cee: { "type": "logon", "rcvdfrom": "rger-virtual-machine", "rcvdat": "Jan 16 09:28:33", "user": "root" }

@cee: { "type": "logoff", "rcvdfrom": "rger-virtual-machine", "rcvdat": "Jan 16 09:28:33", "user": "root" }

@cee: { "type": "logon", "rcvdfrom": "Win2008StdR2x64_vm", "rcvdat": "Feb 05 16:39:27", "user": "WIN2008STDR2X64\\Administrator" }

@cee: { "type": "logoff", "rcvdfrom": "WIN-VSBQP2NOITT", "rcvdat": "Jan 25 15:44:35", "user": "WIN-VSBQP2NOITT\\te" }

@cee: { "type": "logoff", "rcvdfrom": "win7fr.intern.adiscon.com", "rcvdat": "Feb 5 11:15:56", "user": "ADISCON\\fr" }

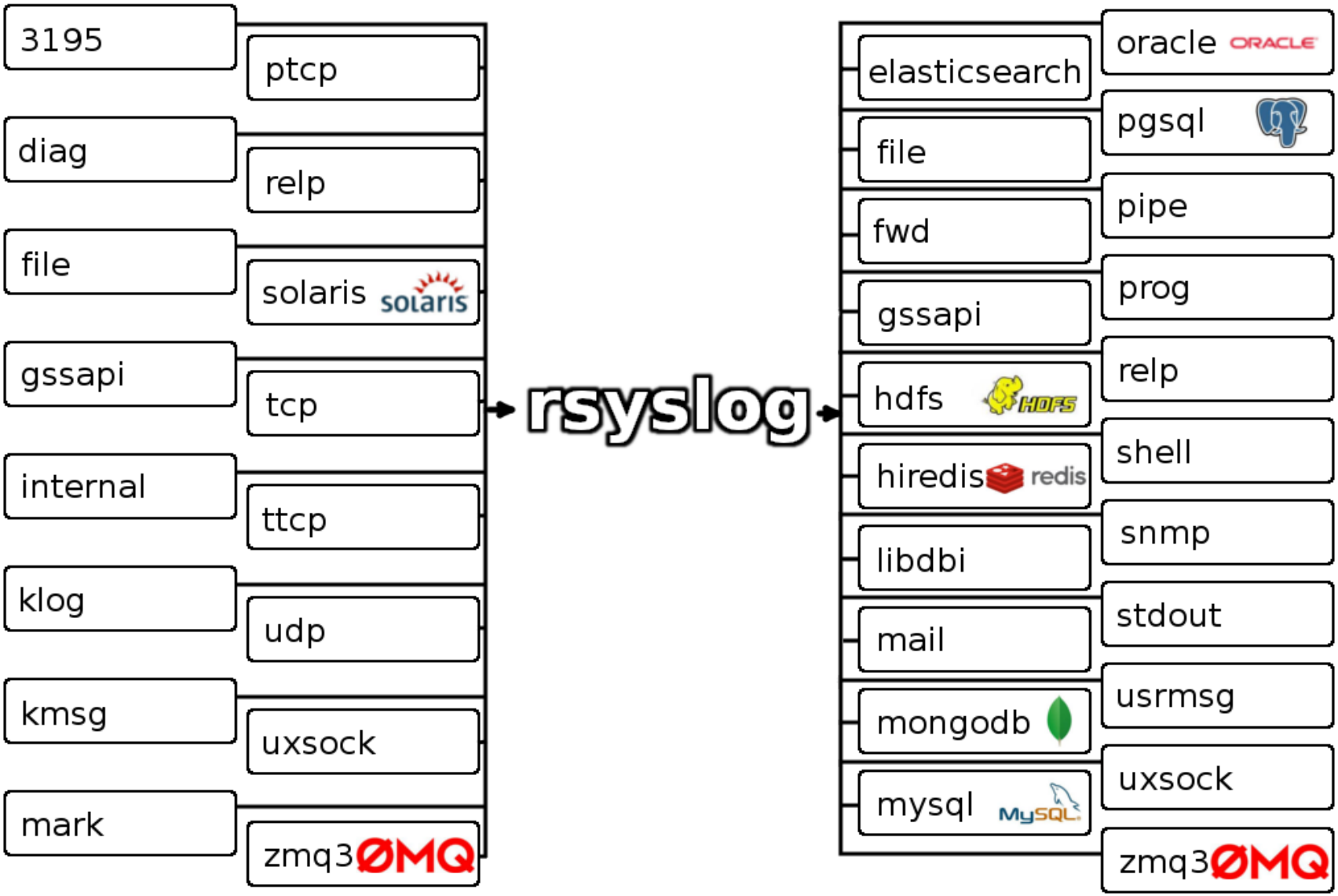
@cee: { "type": "logon", "rcvdfrom": "win7fr.intern.adiscon.com", "rcvdat": "Feb 5 13:41:28", "user": "NT AUTHORITY\\SYSTEM" }

Und das gleiche für CSV:

"Jan 16 09:28:33","rger-virtual-machine","root","logon"
"Jan 16 09:28:33","rger-virtual-machine","root","logoff"
"Jan 24 02:38:49","rger-virtual-machine","rger","logon"
"Feb 05 16:39:27","Win2008StdR2x64_vm","WIN2008STDR2X64\Administrator","logon"
"Jan 25 15:44:35","WIN-VSBQP2NOITT","WIN-VSBQP2NOITT\te","logoff"
"Feb 5 11:15:56","win7fr.intern.adiscon.com","ADISCON\fr","logoff"
"Feb 5 13:41:28","win7fr.intern.adiscon.com","NT AUTHORITY\SYSTEM","logon"

Kernfakten

- Rsyslog ist breits **heute** ein **universeller Log Format Übersetzer**
- Wir hoffen, dass Analyse-Tools künftig mehr Gebrauch vom lumberjack format machen
- Aber falls nicht: auch egal!
  - Wir können bereits heute die für viele wichtige Tools notwendige Konvertierung vornehmen
  - Neue Ausgabeformat können schnell hinzu gefügt werden
  - Syslogd-Interoperabilität gestärkt!

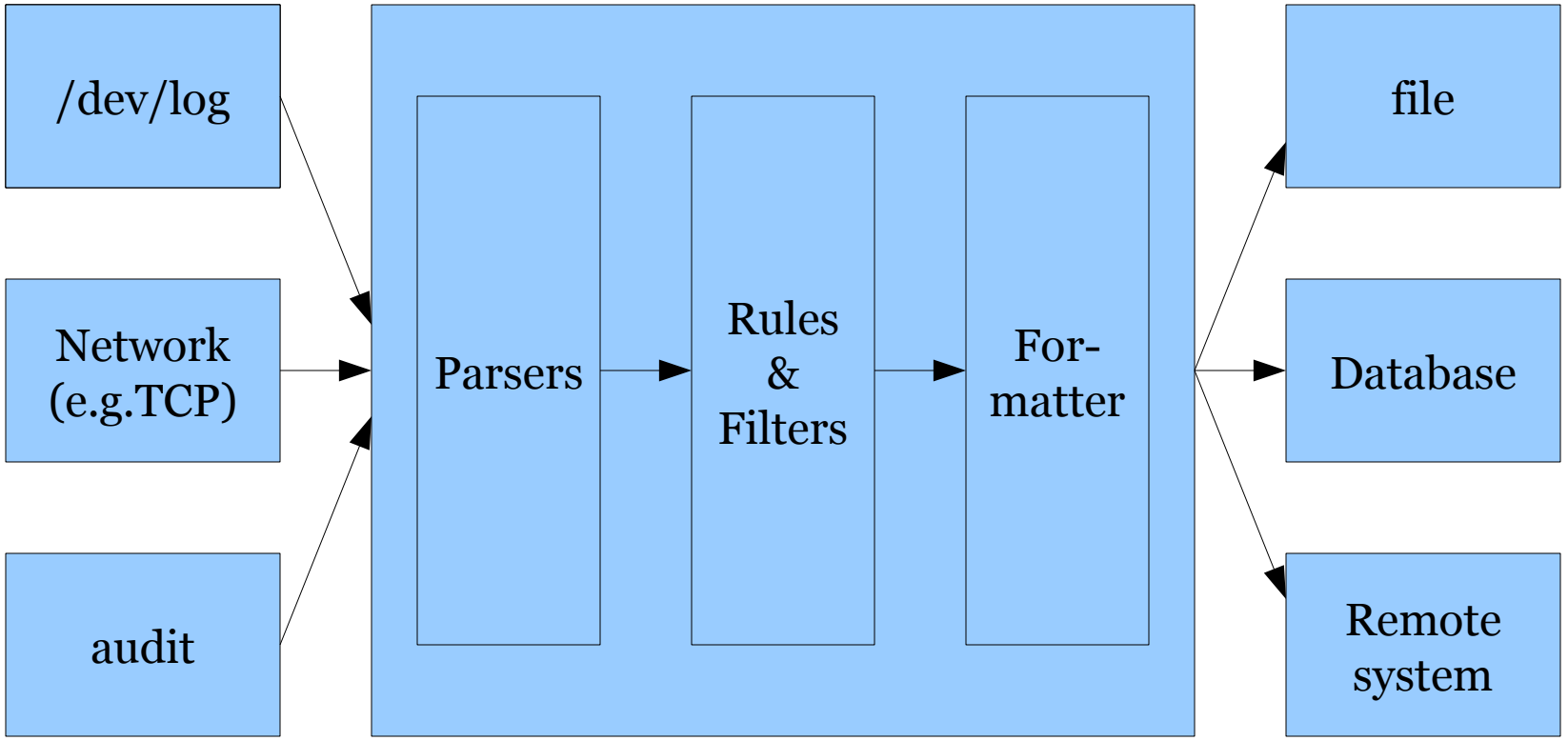

3195
diag
file
gssapi
internal
klog
kmsg
mark
ptcp
relp
solaris
tcp
ttcp
udp
uxsock
zmq3
rsyslog
elasticsearch
file
fwd
gssapi
hdfs
hiredis
libdbi
mail
mongodb
mysql
oracle
pgsql
pipe
prog
relp
shell
snmp
stdout
usrmsg
uxsock
zmq3

Fragen?

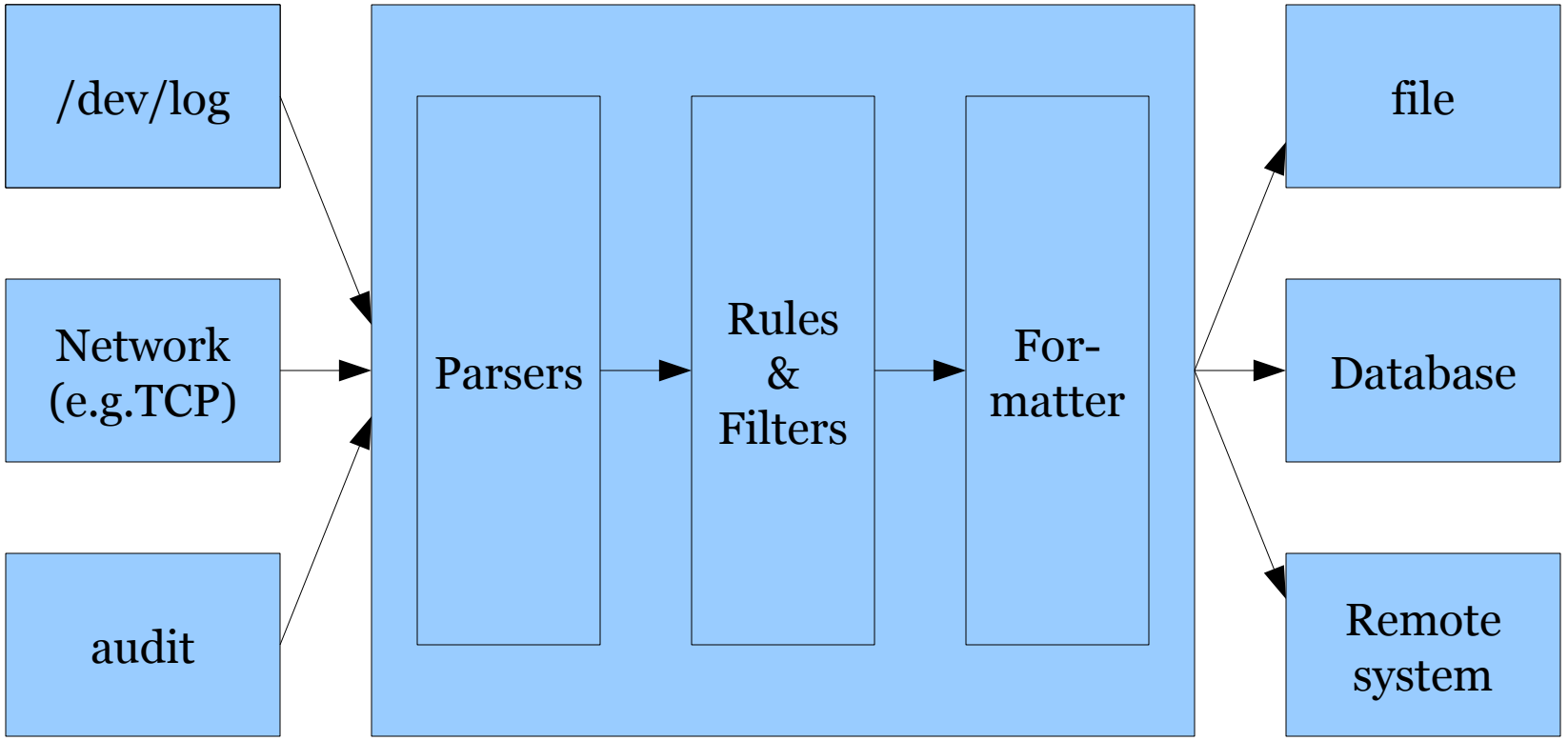
- rgerhards@adiscon.com

- Unterlagen unter
  http://www.rsyslog.com/ffg2013
  - Werden im Laufe der kommenden Woche noch vervollständigt

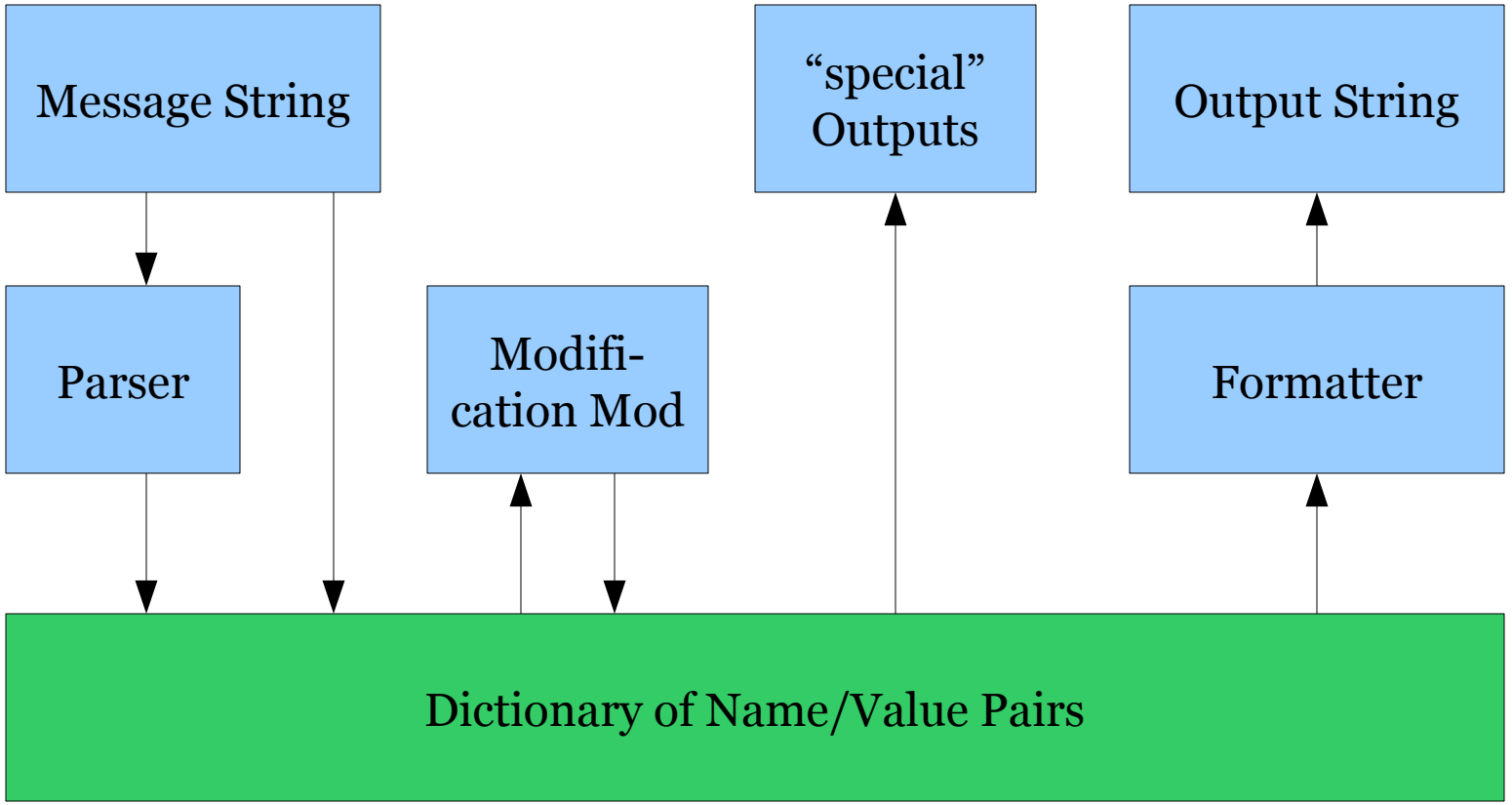
rsyslog Architecture



rsyslog Architecture



“String” Handling in rsyslog



Of course, this is just a small example, but

- It shows how all the pieces can be put together
- mmnormalize is a very important building block to integrate free-form text logs, no matter what the source is
- The output format is highly flexible
- Of course, structured outputs like MongoDB or Elasticsearch are also supported
- We can emit almost all output formats, new ones requires relatively little work in rsyslog's engine

Once again back to my inbox...

- "*I know this is asking a lot since rsyslog would have to do a bunch of processing. I also understand there may be a delay in log delivery due to the processing.*"
- Well ... actually it's far from being as bad as described:
  - Structured logs are ingested very quickly
  - Liblognorm/mmnormalize is extremely fast in converting classical text logs
  - Reformatting is done always in any case, so... ;-)

Long-Term Vision

- There NEVER will be a single format
  - Political reasons (vendors, projects, history, …)
  - Need for new features/functionality
- BUT: use as few as possible
  - Less hassle for producer and consumer devs
  - Forces closed source vendors to support these standard, making it easier for the OSS guys
  - Big win for Enterprise folks who get plug&play
- We hope that Lumberjack will be dominant
  - Stack already in place
  - Good & simple solution
  - Rsyslog converts everything running on Linux

libumberlog vs. journal

```
r = ul_syslog(LOG_NOTICE, "Logged in user: %s", username,
              "service", "%s", service,
              "auth-method", "%s", auth_method,
              "sessionid", "%d", session_id,
               NULL);
```

```
r = sd_journal_send("MESSAGE=Logged in user: %s", username,
                    “PRIORITY=%d", LOG_NOTICE,
                    “SERVICE=%s", service,
                     "AUTH-METHOD=%s", auth_method,
                     "SESSIONID=%d", session_id,
                     NULL);
```

Liblogging abstraction API

- Will be modeled similar to libumberlog
  - Traditional PRI handling (e.g. LOG_NOTICE)
  - Faster parsing of parameter types
  - Will support multiple log channels at the same time
- Not yet finalized, but along the lines of

```
r = ll_log(channel, LOG_NOTICE,
        "Logged in user: %s", username,
        "service", "%s", service,
        "auth-method", "%s", auth_method,
        "sessionid", "%d", session_id,
        NULL);
```

New-Style syslog via libumberlog

- Emit lumberjack-style syslog messages
  - Traditional message text in “msg” property
  - Arbitrary name/value pairs can be specified
  - Ready for consumption by rsyslog mmjsonparse
- Can “convert” old applications using “just” the regular syslog() calls via preloaded lib

Was ist im letzten Jahr passiert?

- Agree on the log format
- Made rsyslog fully lumberjack-aware
- Made Adiscon's Windows Products fully lumberjack-aware
- Made syslog-ng fully lumberjack-aware
- Create new syslog API --> libumberlog

Common Event Expression

A Unified Event Language for Interoperability

Current Release: Version 1.0-beta1

About CEE
Documents
FAQs

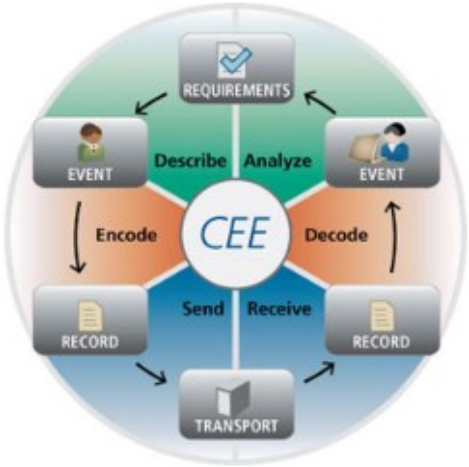
CEE Language
Current Release
Previous Releases

CEE Community
CEE Board
Discussion List
Discussion Archive

News & Events
Calendar
Free Newsletter

Contact Us
Search the Site

Unifying the Audit & Event Lifecycle for Electronic Systems



Users:

- ✓ Find events quickly using common fields and classification tags
- ✓ Easy to understand, easy to use
- ✓ Compatible with existing processes and products
- ✓ Simplifies event handling and audit compliance

Developers:

- ✓ Unifies event structures and content
- ✓ Interoperable with most platforms, event syntaxes, and APIs
- ✓ Easy to use, simple XML structure
- ✓ Extensible event architecture

Unified

CEE unifies event representation and classification by combining support for multiple event syntaxes and log protocols in a single event standard.

Agile

The future is difficult to predict. That's why CEE defines a base set of event record requirements and allows event authors to create their own custom event and event field definitions.

Open

CEE is an open, community-driven effort built on open standards. Interested parties are invited to contribute. All CEE components are freely licensed at no cost.

Compatible

CEE is capable of representing any observable computer or network event and has been developed to be compatible with existing event management products and processes.

Related Efforts

Cyber Observables (CybOX)
Vulnerabilities (CVE)
Configurations (CCE)
Attack Patterns (CAPEC)
Assessment Language (OVAL)
Malware (MAEC)

Latest News

4 CEE Language Specifications Updated to Version 1.0-beta1: *CEE Overview, CEE Profiles, CEE Log Syntax,* and *CEE Log Transport*

CEE/Making Security Measurable booth at *2012 Information Assurance Expo*

CEE Briefing Slides from *Security Automation Developer Days 2012* Now Available

CEE Main Topic of Article on *NetworkWorld*

More News »

Upcoming Events

CEE/Making Security Measurable booth and SwA/SCAP briefings at *IT Security Automation Conference 2012*, October 3-5

More Events »

Status Report

Version 1.0-beta1 CEE Specification documents now available:

- CEE Overview
- CEE Profiles
- CEE Log Syntax (CLS)
- CEE Log Transport (CLT)

More Information
cee@mitre.org

Member of Making Security Measurable

liblogging

- Originally developed to provide RFC3195 reliable syslog
- Just became enhanced to emulate journal API
- Planned: abstraction layer
  - **Core idea: developer doesn't need to care about the actual logging system**
  - Simple to use new structured API
  - Permits dynamic logging provider selection at runtime (syslog, journal, whoKnowsWhat... ;))
  - Inspired by log4j idea, but probably slimmer
  - On TODO list for this year

Some announcements :-)

- Lumberjack just got a web front-end: Adiscon LogAnalyzer now supports lumberjack format!
- Liblogging provides replacement for journal logging API
  - Only log creation subset
  - Permits use of Journal API on platforms without Journal
  - Will potentially in the future permit to redirect Journal calls directly to enterprise syslog
- New plugin to read journal soon available
- Rsyslog will get very strong log signature system within first half of 2013