Syslog nach RFC

Martin Schütte

4. März 2010

Sinn von Logfiles

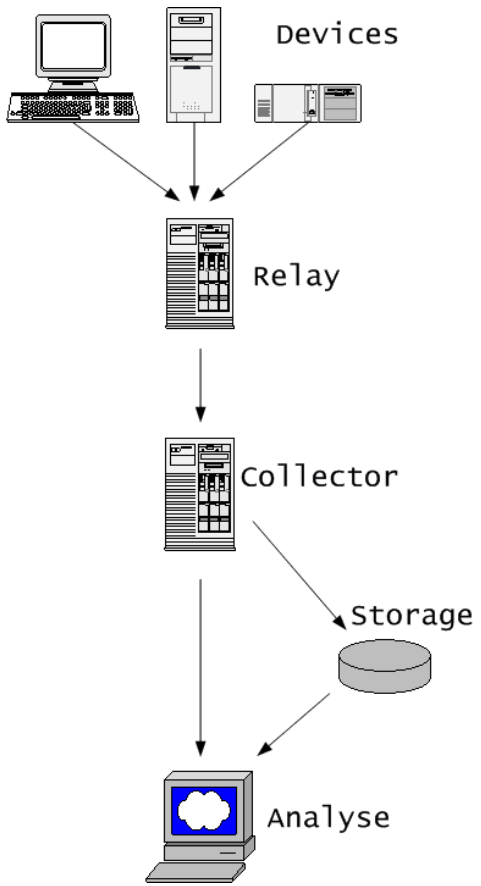
- Fehlersuche
- Systemauslastung
- Verantwortung/Zurechenbarkeit von Aktivitäten
- Warnung vor laufenden Angriffen
- Untersuchung nach Angriffen

BSD Syslog

Design

- ursprünglich als Programmier-/Debug-Hilfe
- einfach zu benutzen und zu konfigurieren
- minimale Ressourcen, *best effort* Ansatz
- mit UDP auch auf andere Rechner loggen
- de facto Standard für Log-Daten unter Unix

Eric wrote syslogd so that all the logs could be brought to one place and cleanly thrown away at once.



BSD Syslog

Bestandteile

Kombination aus:

- API
- Nachrichtenformat
- Daemon
- IPC Protokoll

API

```
SYSLOG(3)          NetBSD Library Functions Manual      SYSLOG(3)

NAME
    syslog, vsyslog, openlog, closelog, setlogmask -- control system log

LIBRARY
    Standard C Library (libc, -lc)

SYNOPSIS
    #include <syslog.h>

    void
    syslog(int priority, const char *message, ...);

    void
    openlog(const char *ident, int logopt, int facility);

    void
    closelog(void);

    int
    setlogmask(int maskpri);
```

Syslog Message Format

```
<38>Mar 17 21:57:57 frodo sshd[701]: Connection from 211.74.5.81 port 5991
<52>Mar 17 13:54:30 192.168.0.42 printer: paper out
```

- Priority
  - Facility
  - Severity
- Header
  - Timestamp
  - Hostname
- Message
  - Tag
  - Content

Konventionen:

- Priority nicht in Logdatei
- nur ASCII
- bis zu 1024 Zeichen pro Zeile

Daemon: syslogd

- Liest Nachrichten von Kernel, Anwendungen, Netzwerk
- Filter nach Priority
- schreibt in Dateien, Pipes, auf Terminals oder Netzwerk (UDP)
- neue Implementierungen mit mehr Features
  (z. Bsp. *BSD syslogd, syslog-ng, rsyslog)
  - Filter nach Host-/Programmname oder RegExp
  - Konvertieren verschiedener Nachrichten und Zeitstempel
  - Schreiben in Speicherpuffer, TCP-Verbindungen oder SQL-DBs

“Transport Protocol“

Nachrichten von . . .

- Anwendungen: `socket(AF_UNIX, SOCK_DGRAM, 0);`
- Netzwerk: `socket(AF_INET, SOCK_DGRAM, 0);`
- Kernel: `open("/dev/klog", O_RDONLY, 0);` (Ringspeicher)

⇒ Eine Nachricht/Zeile je `recvfrom()`/`read()`.

IETF Working Group

"Security Issues in Network Event Logging"

- RFC 3164: The BSD Syslog Protocol (informational)
- RFC 3195: Reliable Delivery for Syslog

- RFC 5424: The Syslog Protocol
- RFC 5425: TLS Transport Mapping for Syslog
- RFC 5426: Transmission of Syslog Messages over UDP
- RFC 5427: Textual Conventions for Syslog Management

- Internet Drafts:
  - Signed Syslog Messages (RFC 5428)
  - DTLS Transport Mapping for Syslog
  - Syslog Management Information Base

RFC 5424: The Syslog Protocol

Neues Nachrichtenformat

```
<165>1 2003-10-11T22:14:15.003Z frodo.example.com prog 1234 ID47↩
 [exampleSDID@32473 iut="3" eventID="1011" eventSource="Application"]↩
 BOMUne entré du journal des événements . . .
```

- Header
  - Priority
  - Version *(new)*
  - ISO Timestamp *(extended)*
  - Hostname/FQDN
  - Application Name
  - Process ID
  - Message ID *(new)*
- Structured Data *(new)*
- Message text *(UTF-8, SHOULD support $\geq$ 2048 octets)*

API-Vorschlag

bisher nur in NetBSD-current

- Syslog-Protocol in IPC ab `syslog(3)`
- `syslog(3)` unverändert (MSGID und SD leer)
- `syslogp(3)` als erweiterte API:

```
void syslogp(int priority, const char *msgid,↩
    const char *sdfmt, const char *message, ...);

syslog(LOG_INFO, "foobar error: %m");
syslogp(LOG_INFO, NULL, NULL, "foobar error: %m");

syslogp(LOG_INFO, "ID%d", "[meta language=\"de-DE\"]",↩
    "Ereignis: %s", 42, EventDescription);
```

RFC 5426: UDP Transport

- UDP-Übertragung wie bisher
- Portnummer: udp/514
- eine Nachricht pro UDP-Datagramm
- keinerlei Sicherung/Authentifizierung

"The ability to log messages received in UDP packets is equivalent to an unauthenticated remote disk-filling service ..."

RFC 5425: TLS Transport

- Punkt-zu-Punkt Verschlüsselung, Integrität und Authentifizierung
- Portnummer: tcp/6514
- Erfordert Client- und Serverzertifikate
- Authentifizierung per CA oder Zertifikat/Subject/Fingerprint

- *neues Protokoll; inkompatibel zu TCP in rsyslog/syslog-ng*
- *auf Anwendungsebene nur Simplex, keine ACKs o.ä.*

Draft: DTLS Transport

- Verschlüsselung, Integrität und Authentifizierung wie TLS
- eine Nachricht pro UDP/DCCP-Datagramm
- Portnummer: udp/6514

Internet-Draft: Signed syslog Messages

Übersicht

- digital signierte Syslog-Nachrichten
- Ende-zu-Ende Authentifizierung, Integrität, korrekte Reihenfolge und Vollständigkeit
- DSA für Signaturen
- SHA-1 oder SHA-256 als Hashfunktion
- OpenPGP- oder X.509-Zertifikate

nicht spezifiziert, aber selbst einzufügen: signierte Zeitstempel

Internet-Draft: Signed syslog Messages

Konstruktion

- Signature Groups für mehrere Nachrichtenströme
- Voraussenden des öffentlichen Schlüssels
- Übertragen der Folge von Nachrichten in Folge von Hash-Werten
- Einfügen der Signaturnachrichten in den Nachrichtenstrom
- Signieren der eigenen Kontrollnachrichten

Payload Blocks

Öffentlichen Schlüssel versenden

```
<110>1 2009-05-03T14:00:39.519307+02:00 host.example.org syslogd 2138 - ↩
 [ssign-cert VER="0111" RSID="1" SG="0" SPRI="0" TPBL="587" ↩
 INDEX="1" FLEN="587" FRAG="2009-05-03T14:00:39.519005+02:00 K ↩
 BACsLMZNCV2N...2Rg==" SIGN="AKAQEUiQptgp...L7+c="]
```

Beim Systemstart, enthält:

- Signature Group (`VER`, `RSID`, `SG`, `SPRI`)
- Fragmentierung (`TBPL`, `INDEX`, `FLEN`)
- Payload Block (`FRAG`) mit
  - Zeitstempel
  - Art des Schlüssels
  - öffentlicher Schlüssel (base64)
- DSA Signatur (`SIGN`)

Signature Blocks

SHA-1 Hash-Werte sammeln ...

```
<46>1 2009-05-03T14:00:29.285912+02:00 host.example.org syslogd 2138 - - restart
<13>1 2009-05-03T14:00:31.156355+02:00 host.example.org app1 9521 - - Message 1
<13>1 2009-05-03T14:00:32.163839+02:00 host.example.org app2 2100 - - Message 2
<13>1 2009-05-03T14:00:33.173021+02:00 host.example.org app3 9649 - - Message 3
<13>1 2009-05-03T14:00:34.182776+02:00 host.example.org app2 15695 - - Message 4
<13>1 2009-05-03T14:00:35.192952+02:00 host.example.org app1 25046 - - Message 5
<13>1 2009-05-03T14:00:36.202981+02:00 host.example.org app2 18971 - - Message 6
```

```
K6wzcombEvKJ+UTMcn9bPryAeaU= zrkDcIeaDluypaPCY8WWzwHpPok=
zgrWOdpx16ADc7UmckyIFY53icE= XfopJ+S8/hODapiBBCgVQaLqBKg=
J67gKMFl/OauTC2OibbydwIlJC8= M5GziVgB6KPY3ERU1HXdSi2vtdw=
Wxd/lU7uG/ipEYT9xeqnsfohyH0=
```

Signature Blocks

. . . und Hash-Werte & Signatur versenden

```
<110>1 2009-05-03T14:00:39.529966+02:00 host.example.org syslogd 2138 - ⏎
 [ssign VER="0111" RSID="1" SG="0" SPRI="0" GBC="2" FMN="1" CNT="7" ⏎
 HB="K6wzcombEvKJ+UTMcn9bPryAeaU= zrkDcIeaDluypaPCY8WWzwHpPok= ⏎
 zgrWOdpx16ADc7UmckyIFY53icE= XfopJ+S8/hODapiBBCgVQaLqBKg= ⏎
 J67gKMFl/OauTC2OibbydwIlJC8= M5GziVgB6KPY3ERU1HXdSi2vtdw= ⏎
 Wxd/lU7uG/ipEYT9xeqnsfohyHO=" SIGN="AKBbX4J7Qkrw. . . uMyfM="]
```

Eine Syslog-Nachricht, enthält:

- Signature Group (`VER`, `RSID`, `SG`, `SPRI`)
- Global Block Counter (`GBC`, zählt syslog-sign Nachrichten)
- First Message Number (`FMN`, zählt normale Nachrichten)
- `CNT` Hash Blocks (`HB`)
- DSA Signatur (`SIGN`)

Offline Verifikation

1. Log teilen in Certificate Blocks (CB), Signature Blocks (SB) und normale Nachrichten
2. für normale Nachrichten Hashes berechnen
3. CBs und SBs sortieren
4. CBs zusammensetzen und verifizieren
   $\Rightarrow$ ergibt öffentliche Signaturschlüssel
5. SBs verifizieren
6. Folge aller Hash-Werte aufbauen (mittels `FMN`)
7. zu jedem Hash-Wert die zugehörige Nachricht raussuchen
   $\Rightarrow$ ergibt Folge verifizierter Nachrichten

verify.pl

```
$ perl verify.pl -i test.log
reading input...
processing CBs...
decoding SGs...
got PKIX DSA key
verifying CBs...
verified CB and got key for SG: (host.example.org,1217632162,0111,3,0),↩
    start: 2008-08-02T01:09:27.773464+02:00
now process SBs
signed messages:
...
host.example.org,1217632162,0111,3,0,11 <15>1 ... test 6255 - - msg10
host.example.org,1217632162,0111,3,0,12 <15>1 ... test 6255 - - msg11
host.example.org,1217632162,0111,3,0,13 **** msg lost
host.example.org,1217632162,0111,3,0,14 <15>1 ... test 6255 - - msg13
host.example.org,1217632162,0111,3,0,15 <15>1 ... test 6255 - - msg14
host.example.org,1217632162,0111,3,0,16 <15>1 ... test 6255 - - msg15
...

messages without signature:
<15>1 2008-08-02T02:09:27+02:00 host.example.org test 6255 - - modified msg12
```

Implementierungen

- rsyslog
  sehr modular, default-syslogd auf mehreren Linux-Systemen, implementiert syslog-protocol und tls-transport ab Version 3.19
- syslog-ng
  flexible Konfiguration, implementiert syslog-protocol und tls-transport in Open Source Edition (GPL) ab Version 3.0
- NetBSD syslogd
  syslog-protocol, tls-transport und syslog-sign implementiert als Google Summer of Code 2008 Projekt, bislang nur in -Current

rsyslog.conf Beispiel

```
# certificate files
$DefaultNetstreamDriverCAFile /etc/ca.pem
$DefaultNetstreamDriverCertFile /etc/cert.pem
$DefaultNetstreamDriverKeyFile /etc/key.pem

$ModLoad imtcp                                  # load TCP listener
$InputTCPServerStreamDriverMode 1               # TLS-only mode
$InputTCPServerStreamDriverAuthMode anon # client NOT authenticated
$InputTCPServerRun 6514                         # listen at port 6514

# set up the action
$DefaultNetstreamDriver gtls                # use gtls netstream driver
$ActionSendStreamDriverMode 1               # require TLS
$ActionSendStreamDriverAuthMode anon  # server NOT authenticated

*.* @@(o)server.example.net:6514          # send (all) messages
```

syslog-ng.conf Beispiel

```
source src_tls { syslog( ip(10.100.20.40)
    transport("tls")
    tls(peer-verify(required-trusted)
        ca_dir('/etc/ca.d/')
        key_file('/etc/client_key.pem')
        cert_file('/etc/client_certificate.pem')
    )  ); };

filter f_notice { level(notice..emerg); };

destination dst_tls { syslog("10.100.20.41"
    transport("tls")
    port(6514)
    tls(peer-verify(required-trusted)
        ca_dir('/etc/ca.d/')
        key_file('/etc/client_key.pem')
        cert_file('/etc/client_certificate.pem')
    ) ); };

log { source(src_tls); filter(f_notice); destination(dst_tls); };
```

NetBSD syslog.conf Beispiel

```
tls_ca="/etc/my.cacert"
tls_cert="/etc/localhost.crt"
tls_key="/etc/localhost.key"
tls_gen_cert=on

tls_server=on
tls_bindhost="192.168.1.2"
tls_bindport="syslog-tls"
tls_allow_clientcerts="/etc/somehost.crt"

*.*     @[2001:db8::fe41:be53]:syslog-tls(↩
                fingerprint="MD5:00:A2:...:27")
```

Umstieg auf neue Protokolle

Änderung an allen Schnittstellen:

- lokal in `/var/run/log`
- im Netzwerk zwischen Hosts und Logserver
- Format für Archivierung
- Format zur Auswertung

Vorteile der Protokolle?

- TLS statt UDP sinnvoll?
- Ende-zu-Ende-Signaturen notwendig?
- ideales Format zur Archivierung?
- ideales Format zur Auswertung?

Vorteile der Protokolle?

```
Feb 28 23:34:08 mail amavis[90040]: (90040-06) Passed CLEAN, ↩
  [140.211.11.3] [216.139.236.158] <users-return-87524-lists=↩
  mschuette.name@spamassassin.apache.org> -> <mschuett@mail.asta.↩
  uni-potsdam.de>, Message-ID: <27738584.post@talk.nabble.com>, ↩
  mail_id: OOmMiHIeWCKq, Hits: -13.65, size: 3109, ↩
  queued_as: B5501502423, 17412 ms
```

besser/schlechter als?

```
2010-02-28T23:34:08.285912+02:00 mail.asta.uni-potsdam.de ↩
  amavis 90040 passed [amavisResult@32473 proc="90040-06" ↩
  result="Passed CLEAN" original_IP="140.211.11.3" ↩
  est_orig_IP="216.139.236.158" orig_env_sender="<users-return-↩
  87524-lists=mschuette.name@spamassassin.apache.org>" ↩
  recip_list="<mschuett@mail.asta.uni-potsdam.de>" ↩
  msg_id="<27738584.post@talk.nabble.com>" ↩
  mail_id="OOmMiHIeWCKq" hits="-13.65" size="3109" ↩
  queued_as="B5501502423" proc_time="17412"]
```

Links

- IETF Syslog WG: Status Page
- rsyslog
- syslog-ng
- NetBSD & GSoC08: Improve syslogd
- NetBSD CVS Repositories
- SEC - simple event correlator
- logsurfer+