Evaluation of a few security audit tools for Linux
Technical Report

A. Apvrille
Open Systems Lab, Ericsson Research Canada,
8400 Décarie Blvd, Town of Mount-Royal, (QC) Canada H4P 2N2.
{Axelle.Apvrille}@Ericsson.ca

April, 11 2003

1 Introduction

[DSI], a security framework for distributed environments, offers various security services such as Distributed Access Control (DisAC), Distributed Confidentiality and Integrity (DisCI).

The project now plans to add another service: a *secure auditing service*, designed for carrier-class clusters.

As a first step, this report summarizes features of current major auditing tools for Linux, trying to highlight their pros and cons. This report is intended as *background* information to design an adequate auditing service for DSI, it does not detail how DSI's auditing service will be developped.

Warning: the opinions expressed in this technical report are only mine, from my own comprehension of the tools I have evaluated. They may be wrong: please do not hesitate to send feedback or comments.

2 SNARE

2.1 In brief

Name	SNARE System iNtrusion Analysis and Reporting Environment
URL	`https://sourceforge.net/projects/snare/`
Tested version(s)	snare-core 0.9.1 and snare 0.9
Support	Sourceforge's website does not seem very active, but mail can be sent to IntersectAlliance.

2.2 Installation

Two packages are needed: the `snare-core` package (which has the *core* functionalities) and the `snare` package if you want a GUI.

For the `snare-core` package, no configure script, however the usual `make clean, make, make install` work without any problem. Basically, the following files get installed:

- `/usr/bin/auditd`: the audit daemon
- `/etc/init.d/audit`: script to start/stop auditing. Inserts an `auditmodule` kernel module.
- `/etc/audit/audit.conf`: the configuration file.

As for the GUI `snare` package, strangely, the installation failed with the source tar.gz package[1], so the RPM binary packaged was used.

2.3 Auditing modes

SNARE offers two different auditing modes:

- the *kernel* mode, where a selection of system calls are audited ;
- and the *objective* mode where user can define filters of various events to be logged.

2.3.1 Auditing kernel activity

User can select which system calls he wishes to audit. Depending on which system calls are selected (and how many) this may introduce heavy logging. For instance, the simple fact of loading `emacs` generated 227 logs, more precisely 1 `execve` system call and 226 `open` system calls of various libraries, configuration files etc.

Logs are sent to a file in human readable format, and may be displayed by the GUI (with a coloured light according to severity). Logs in figure 1 show an `execve()` event launched by user `lmcaxpr`, consisting in a call to `/usr/bin/emacs` from a bash shell.

Logs in figure 2 show logs generated by an `ls` command. In this example, the auditing of the `open()` system call has been turned off, so we only see a log for the `execve()` call, and one for the `exit()`.

Finally, a buffer overflow exploit from Aleph One [Al0] was tested over SNARE. The buffer overflow simply calls the *vulnerable* program with a very specific string as argument, that overflows an internal buffer of the program. This well-chosen string makes vulnerable spawn a *bash* shell. Generated logs are shown at figure 3. In this case, we do detect a new shell is launched by *vulnerable*.

[1]An error at build occurs, even when using the right versions of autotools.

```
glacier LinuxAudit event,execve(),
Thu Feb 20 17:04:40 2003 user,lmcaxpr(500),install(502),
lmcaxpr(500),install(502) process,1838,bash
path,/usr/bin/emacs
arguments,emacs return,0 sequence,4297

glacier LinuxAudit event,open(O_RDONLY),
Thu Feb 20 17:04:40 2003 user,lmcaxpr(500),install(502),
lmcaxpr(500),install(502) process,1838,emacs
path, /etc/ld.so.preload return,-2 sequence,4298
...

glacier LinuxAudit event,open(O_RDONLY),
Thu Feb 20 17:04:51 2003 user,lmcaxpr(500),install(502),
lmcaxpr(500),install(502) process,1838,emacs
path, /usr/share/emacs/21.2/etc/splash.xpm return,3
sequence,4524
```

Figure 1: Sample kernel logs: emacs is launched from a shell.

2.3.2 Objective auditing

In this mode, the administrator can configure special events to log such as access to given files, from given users, executing a given program, accessing the network, changing identity etc. Those filters are customizable. The amount of logs is still pretty heavy, specially if you're using a graphical environment (for instance, if you've got a clock, `date` is called every minute etc). See logs figure 4.

Tests of objective mode have also been made using the same buffer overflow exploit as for kernel tests (§2.3.1). Unfortunately, the default filters provided with SNARE were unable to detect either the opening of the *vulnerable* program nor its launching of a new shell. To detect the anomaly, one should remove all default filter and add (1) one filter for starting /stopping programs for all users, and (2) another filter for opening programs (for all users).

2.4 A word about how system calls are audited

SNARE does not require kernel patching (which is really cool ;-)), it is just loaded as a *kernel module*. To audit system calls, it works the following way:

- **until 2.4.17 (included):** the kernel (`kernel/ksyms.c`) exports a table called *sys_call_table* which contains a list of pointers to various system calls (open, close, execve...). This table is made available to external modules:

  ```
  #ifndef __mips
  EXPORT_SYMBOL(sys_call_table);
  ```

```
glacier LinuxAudit      event,execve(),
Thu Feb 20 17:16:48 2003 user,lmcaxpr(500),install(502),
lmcaxpr(500),install(502) process,1874,bash
path,/bin/ls    arguments,ls --color=tty
 return,0       sequence,6610

glacier LinuxAudit      event,exit(),Thu Feb 20 17:16:48 2003
user,lmcaxpr(500),install(502),lmcaxpr(500),install(502)
process,1874,ls return,0       sequence,6611
```

Figure 2: Sample kernel logs generated by the ls UNIX command, with logs to open() turned off

```
glacier LinuxAudit event,execve(),
Mon Feb 24 11:06:03 2003 user,lmcaxpr(500),
install(502),root(0),install(502) process,2634,bash
path,/home/lmcaxpr/prog/bof/vulnerable arguments,./vulnerable
return,0 sequence,72128
...

glacier LinuxAudit event,execve(),
Mon Feb 24 11:06:03 2003 user,lmcaxpr(500),
install(502),root(0),install(502)
process,2634,vulnerable path,/bin/bash arguments,
/bin/sh return,0 sequence,72132
```

Figure 3: Kernel logs when launching a buffer overflow exploit.

```
#endif
```

As Snare is loaded as a kernel module, and as this table is available to such modules, it just simply replaces the pointers of the various system call by their own audited system calls. And their own audited system calls are basically a call to the original system call and auditing information.

- **from 2.4.18 +:** unfortunately the system call table is no longer made available to external modules (for security reasons concerning rootkits). It's quite simple to patch the kernel and export the table again, but SNARE has chosen another (trickier) solution. They have noticed that the system call table can actually be retrieved from the `boot_cpu_data` structure in `asm/processor.h`. This fix is a real hack - of course, probably one day it won't work any longer - but it's a workaround.

```
glacier LinuxAudit      objective,clear,
Thu Feb 20 17:02:27 2003,The program /usr/bin/emacs has been
executed by the user lmcaxpr       event,execve(),
Thu Feb 20 17:02:27 2003 user,lmcaxpr(500),install(502),lmcaxpr(500),
install(502) process,1830,bash       path,/usr/bin/emacs
arguments,emacs return,0       sequence,3665

glacier LinuxAudit      objective,clear,
Thu Feb 20 17:02:29 2003,The process emacs, owned by the user lmcaxpr
, has exited        event,exit(),Thu Feb 20 17:02:29 2003
user,lmcaxpr(500),install(502),lmcaxpr(500),install(502)
process,1830,emacs       return,0       sequence,3889
```

Figure 4: Sample objective logs when launching emacs

2.5 Bugs

Noticed a few bugs concerning uninstallation that does not erase everything (and then induces error when re-installing), and a major bug concerning `socketcall` not being logged. This bug is known, and is being currently fixed.

Globally, however, SNARE works pretty well.

2.6 Benchmarks

Benchmarks [VS96] have been run on a Pentium IV, 2.4 Ghz, running a Redhat 7.3 Linux. Results compare a system without SNARE, and a system that runs snare-core 0.9.1 with kernel logs (open, rename, chmod, setuid, setgid, chown, truncate, execve, socketcall, reboot, exit, create, mknod, link, mkdir, unlink, rmdir, symlink, create_module activated). Benchmarks have been run ten times without SNARE, and ten times with SNARE.

In most cases, results were roughly similar, except for a few particular tests shown at table 2.6. Please note that for open & close, fork, exec and sh tests, results are in microsecond (smaller number is the better), and for mmap and bcopy results are in MB/s (biggest number is the better).

Test	Basic	With SNARE	Overhead
Open, Close	2.5ms	215 ms	+8500%
Fork	112 ms	374 ms	+234%
Exec	392 ms	1365 ms	+248%
Sh proc	2165 ms	2863 ms	+32%
Mmap	1734 MB/s	796 MB/s	-54%
Bcopy	440 MB/s	221 MB/s	-50%

Table 1: Benchmarking of SNARE on a Pentium IV 2.4GHz

From those results, we see that kernel auditing is **very** demanding.

On the other hand, the nice thing about snare is that it rarely consumed more than 5% of the host's CPU (and more often around 1%).

2.7 Pros and cons

Type	Pros	Cons
Ease of use	No kernel patch needed. SNARE is loaded as a kernel module	Impossible to set contextual log conditions such as *"if process A launches process B, then log. If B is launched directly, ignore"*. In objective mode, information you need tends never to be logged... Difficult to set really good filters.
Performance	No bottleneck	There's at least one indirection for each system call. General slowdown of the machine.
Security	Logs use a sequence number. The idea of ordering logs is good.	Logs are not digitally signed and can be tampered with Useless on a security point of view, as sequence numbers are predictible. Time of logs is not guaranteed as there is no secure timestamp. Logs severity is written in the logfile. If severity levels are changed, old logs are not re-tagged.

3 SDSC Secure Syslog

3.1 In brief

Name	SDSC Secure Syslog
URL	`http://security.sdsc.edu/software/sdsc-syslog`
Tested version(s)	sdscsyslog 1.0.0 RC6
Support	Through mailing-list at `http://lists.sdsc.edu/mailman/listinfo/sdscsyslog`

3.2 Goal of Secure Syslog

Secure Syslog is not an auditing system as Snare, and even less an IDS. Secure Syslog is "simply" an enhanced syslog package we pay attention to because:

- it is designed for high volume of system logs,

- it is designed to secure logging.

3.3 Installation and configuration

This package uses the regular `./configure`, `make` and `make install` process. The following files are installed.

- `/usr/local/sbin/syslogd`: SDSC's syslog daemon.

- `/usr/local/share/SDSCSyslogd`: default package installation path, containing sample configuration files and README. Documentation is available through `man syslogd` and `man syslogd.conf`, but is not up-to-date.

- `/usr/local/etc/syslogd.conf`: default location for the configuration file. Otherwise, launch binary with `-c` option and path to config. file.

Note: create /usr/local/var/run if it doesn't exist or syslogd will not launch.

```
Feb 24 17:22:23 glacier su(pam_unix)[4306]: session closed for user root
Feb 24 17:22:27 glacier su(pam_unix)[4532]: authentication failure;
                                            logname=lmcaxpr uid=500
                                    euid=0 tty= ruser=lmcaxp
                                            rhost=  user=root
Feb 24 17:22:33 glacier su(pam_unix)[4534]: session opened for user root
                                            by lmcaxpr(uid=500)
Feb 24 17:22:45 glacier su(pam_unix)[4534]: session closed for user root
Feb 24 17:23:48 glacier su(pam_unix)[4579]: session opened for user root
                                            by lmcaxpr(uid=500)
```

Figure 5: Sample SDSC Syslog logs.

3.4 Pros and cons

Type	Pros	Cons
Ease of use	Capable of logging logs sent by various hosts. An XML format of logs is offered via the *cooked* format type	Very poor documentation. This is a syslog utility. Absolutely no way to specify precisely under what conditions event should be logged.
Security		Security features of Secure Syslog are meant to be provided by implementing BEEP [RFC3080], which offers message integrity, confidentiality and authentication through [TLS] or [SASL]. However, currently only reliable delivery of logs [RFC3195]of logs is implemented.

4 Secure Auditing for Linux

4.1 In brief

Name	SAL: Secure Auditing for Linux
URL	`http://secureaudit.sourceforge.net`
Tested version(s)	SAL 1.0 RC3
Support	Through mailing-list `secureaudit-general@lists.sourceforge.net`, but not very active.

4.2 How SAL works

SAL is a research project funded by the DARPA. Its goal is explicitly to provide US Defense departments with a C2 compliant Linux, implementing kernel auditing.

SAL's architecture is based on a log server, named *SAL Log Server* (SLS), multiple clients (*SAL Instrumented Client* (SIC)), and secure tunnels between each SIC and the SLS (*Secure Network Transmission Tunnel* (SNTT)).

On each SIC, the kernel source is patched to instrument a given selection of system calls. Each time an audited system call is called, audit events get generated and stored in kernel level system buffers. Periodically, an audit daemon (`auditd`) dumps those kernel buffers onto the disk in so-called *little files*.

The patching of the kernel is done automatically by calling the `patches/sal-conf.pl` script. Basically, this Perl script patches the `kernelDir/arch/i386/kernel/entry.S` file. It adds two system calls (implemented in `dev/audit.c`):

- syscall_audit which is called after each system call to audit (a `je auditsys` jump is inserted after each system call in entry.S). This system call collects information about the audited system call (in a `struct syscall_buf`), and stores it in a internal kernel buffer.

- and sys_audit, which retrieves information from the kernel buffer and copies it to user space buffer. This is typically called by the the audit daemon `auditd`.

Then, an application named `auditclient` is in charge of reading those *little files* and sending them to the SLS, through a tunnel encrypted and authenticated by OpenSSL tools.

On the SLS, the `sald` application listens to connections from various *auditclient* programs, and receives the audited data. It adds some further information (such as time of receipt), and stores that data onto a storage media (disk, sequential device...). Furthermore, an application named `archiver` is responsible for continuously verifying the integrity of previously recorded data. If there's evidence logs have been tampered with, alarms can be raised.

4.3 Installing SAL

UNDER PROGRESS.

4.4 Pros and cons

Type	Pros	Cons
Ease of use	Excellent design documentation	Installation process is rather difficult (and manual), and not that much documented.
	Capable of auditing multiple clients.	Kernel needs to be *recompiled* at each change of system calls to audit.
		Rather complicated system.
Security	SIC are authenticated by the SLS	Is SLS authenticated by SICs ?
	The idea of writing logs on sequential (and optionally remote) storage data is good.	No explanation is provided concerning how data can be prevented from tampering. As a matter of fact, this is a big issue, and even using CD-Rs or optical disks do not solve all security issues [WORM02, WormFTP02].
		Little files are not sealed nor signed ?
	Difficult for an attacker to intentionally stop the auditing as the audited data is gathered from the kernel: (1) stopping the auditing applications will not stop the kernel from gathering events, and (2) there's no kernel module to unload.	
Performance	SAL handles network failures between SLS and SICs	Documentation says they have demonstrated SAL does not lose events, but they do not provide any numerical results, nor benchmarks of the whole system.
	If `auditd` crashes, events will be lost only once the kernel buffer are full	No numerical impact provided regarding this fact.
	Use of multiple processes (and threads) make it possible to do work when there's time to do so. This reduces bottleneck probabilities	No numerical results.
	Auditing code is compiled statically at kernel level, which should reduce overhead	No numerical results provided.

5 EVLOG

5.1 In brief

Name	EVLOG: Linux Event Logging for the Enterprise
URL	`https://sourceforge.net/projects/evlog`
Tested version(s)	
Support	Through mailing-list at `evlog-developers@lists.sourceforge.net`

5.2 Goal of EVLOG

In short, EVLOG's goal is to provide a greatly improved open-source logging facility, that'd be more *'professional'* than syslogd (i.e targetting medium to large servers of entreprises). To do so, its implementation complies to the current POSIX draft for logging [PX1003].

Applied to the context of security auditing, EVLOG provides some nice features compared to syslogd:

- kernel logs can be sent to any facility, whereas syslogd only send them to `LOG_KERN`.
- it's possible to notify end-user of a given event.
- both a standardized set of data (textual information) and customized data (binary format) may be logged.
- tools to extract, filter of view logs are provided

A more extensive comparison may be found at `http://evlog.sourceforge.net/why_not_just_use_syslog.html`.

5.3 Pros and Cons

Type	Pros	Cons
Ease of use	A lot of documentation concerning design and event logging standards Capable of logging events of multiple hosts Capability to log data in text and/or binary format	Kernel needs to be patched and recompiled (BTW: 2.4.17 is not supported). More complicated.
Security		Security is not taken into account. Basically, EVLOG does not offer more security than syslogd.
Performance	logs may be gzipped	... but at significant processing cost Not much information provided

To summarize, to my opinion, EVLOG really focuses on the logging system (as syslogd), but it does not deal at all with auditing (for instance, which system calls to audit or not, and how), nor with security (guaranteeing integrity of logs, reliability, ordering etc).

6 Conclusion

6.1 What seems to be lacking in existing auditing systems

- impossible to audit several machines *together*. Sometimes, it's possible to forward all logs to a single host, but in all cases, machines are always audited *independently*. This means that if a host A accesses to host B, logs on A will say "A accesses IP address x", and "B receives connection from IP address y" but both logs will not be linked. In a cluster, that would mean the administrator would have to check audit of all nodes seperatly (and analyze logs to understand their links).
- lack of contextual logs. For instance, it is impossible to differentiate launching a new shell from a given program, or from another existing shell.
- most of the time, security of logs is missing (or only partially implemented). See §6.2 for a list of security requirements for logging systems. Usually, the problem is to find a compromise between performance (high volume of logs being received) and security (which consumes CPU...). It is very likely that specific dedicated protocols and algorithms should be used for that very issue.

6.2 Secure logging requirements

If DSI is involved in developping a new logging system, particular attention should probably be taken concerning the following:

- **secure programming**, such as ignored malformed messages ([RFC3164, §6.1]).
- **authenticating sender of messages** ([RFC3164, §6.2]), including detecting message forgery (an intruder sending fake messages [RFC3164, §6.2]).
- **message integrity** ([RFC3164, §6.5]). In practice, it is very difficult to guarantee that logs have not been tampered with (for instance, what happens if somebody forges from scratch completely new data ?). Research has already been conducted in that area by StorageTek [WORM02, WormFTP02].
- **message confidentiality**: under some circumstances, encryption of log messages may be required during transmission, or once logged.

- **guaranteeing order of logs** at reception. It is difficult to guarantee that if two logs are sent one after the other, they will be received and logged in the same order (for instance, because syslog uses UDP, and also because hosts may not be synchronized). However, at least, it should be possible to guarantee that events that are logged are in the right order of reception. An intruder should not be able to tamper successfully this order, ie there should be a way to check order of received logs is correct.
- **reliable delivery**: being able to guarantee logs are not lost. Actually, this is a difficult task ([RFC3164, §6.4]).

6.3 Actions

1. **Seperate auditing and logging**. Dealing altogether with both is too heavy a task. Maybe we should focus on one of those first. Probably, auditing would be chosen first as DSI can contribute more in that area, however there is a real need for a secure logging system. At first, messages could be processed by a basic syslog or klog system.
2. **Contextual and homogeneous logs** seems to be areas where DSI could contribute.
3. **Logging and law**: there is a strong need on a legal point of view to be able to prove logs of a machine are correct. This is important and has been pointed out by multiple projects [SAL, WORM02, WormFTP02].

7 Contact people

The DSI team:

- DSI on Sourceforge [DSI],
- Axelle Apvrille : Axelle.Apvrille@Ericsson.ca,
- Makan Pourzandi : Makan.Pourzandi@Ericsson.ca,
- Gabriel Ioan Ivascu : Gabriel-Ioan.Ivascu@polymtl.ca,
- Marc Chatel: Marc.Chatel@Ericsson.ca.

References

[Al0] Aleph One, *Smashing the Stack for Fun and Profit*, Phrack 49, Vol. 7, file 14/16.

[DSI] *The Distributed Security Infrastructure Project*, `http://sourceforge.net/projects/disec`.

[PX1003] *Standard for Information Technology - Portable Operating System Interface (POSIX) - Event Logging*, System API - Services for Reliable, Available, and Serviceable Systems, P1003.25 (draft). `http://www.ieee.org`

[RFC3080] M. Rose, *The Blocks Extensible Exchange Protocol Core*, Network Working Group, RFC 3080, March 2001.

[RFC3164] C. Lonvick *The BSD syslog protocol*, Network Working Group, RFC 3164, August 2001.

[RFC3195] D. New, M. Rose, *Reliable Delivery for syslog*, Network Working Group, RFC 3195, November 2001.

[SAL] *Secure Auditing for Linux, Software Design Document*, Version 1.0, February 28 2003, `http://secureaudit.sourceforge.net`

[SASL] Myers, J., *Simple Authentication and Security Layer (SASL)*, RFC 2222, October 1997.

[TLS] Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and P. Kocher *The TLS Protocol Version 1.0* RFC 2246, January 1999.

[VS96] Mc Voy L., Staelin C. *LmBench: portable tools for performance analysis*, in Proceedings of the 1996 USENIX Annual Technical Conference, `http://www.bitmover.com/lmbench`.

[WORM02] Apvrille A., Hughes J., *A Time Stamped Virtual WORM System*, Workshop SECI02 SEcurité de la Communication sur Intenet, September 2002, Tunis, Tunisa.

[WormFTP02] Apvrille A., Hughes J., Girier. V., *Streamed or Detached Triple Integrity for a Time Stamped Secure Storage System*, First IEEE International Security In Storage Workshop, December, 2002, Greenbelt, Maryland, USA.