Generating computer security audit logs with interposing libraries

Mustafa M. Paksoy '07, Prof. Benjamin A. Kuperman (advisor)
Computer Science Department, Swarthmore College, Swarthmore, Pennsylvania 19081

Introduction

Computer Security Monitoring (CSM) systems operate by analyzing audit logs for evidence of attack. Incidentally, CSM systems usually work with inadequate audit data sources that are available by default. Yet, any analysis scheme is only as good as the data it is working with-- garbage in, garbage out. Our audit log generator tries to mitigate this problem by generating audit data that will yield well to specific CSM analysis purposes.

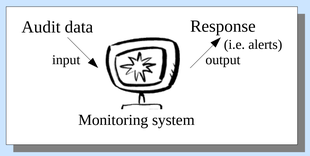


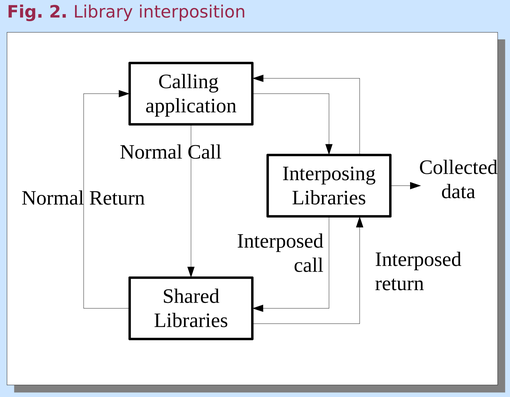
Fig. 1. Operation of Computer Security Monitoring (CSM) systems

Library Interposition

On modern operating systems, calls to functions in shared libraries are linked at runtime. Thus, any library interposed between shared libraries and the calling application can intercept calls to these dynamically loaded libraries (.dll files on Windows and .so on Linux).

Our implementation of library interposition uses the environment variable LD_PRELOAD to force the loader to resolve function calls to our libraries first.

Fig. 2. Library interposition



Application Structure

Over the summer we have redesigned and implemented an audit system with three interposing libraries, each designed to serve one of attack, intrusion and misuse detection.

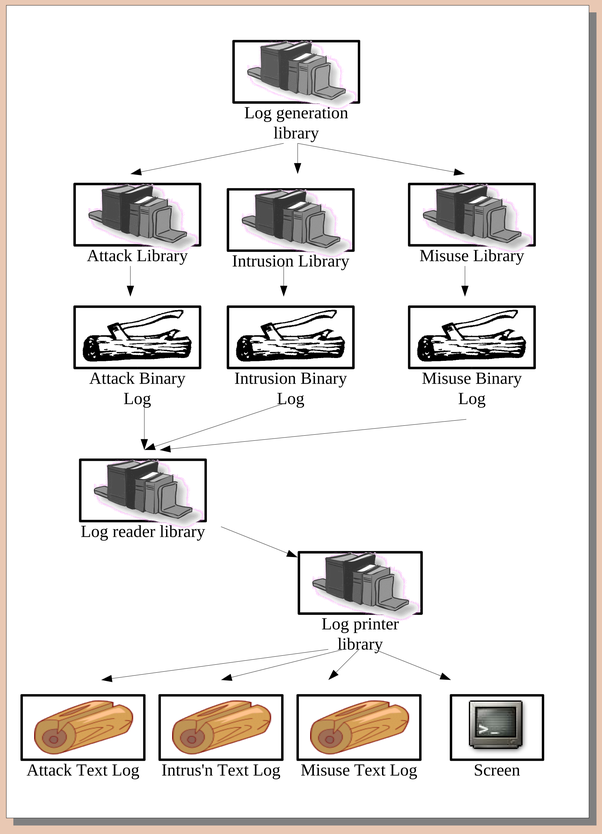
The sytem we started with was initially designed on the operating system Sun Solaris. We ported the core of it over to Linux Debian (used on Computer Science dept. computers) and rewrote much of the rest.

We pursued a highly modular approach in our redesign to make testing and maintenance of it easier. This way we could tweak a certain functionality easily without having to rewrite any other parts.

The interposing libraries (See section: *Library Interposition*) use functions in the log generation library to record data in binary form. This data is stored in log files associated with each library.

Although binary log files are an efficient way of storing data, they are not easy to handle. So we wrote a different application that can read these files into main memory, and yet another that can print them out in either human readable or parsable form. This output can then be recorded as text log files or printed to the screen.

Fig. 3. Organization of our application

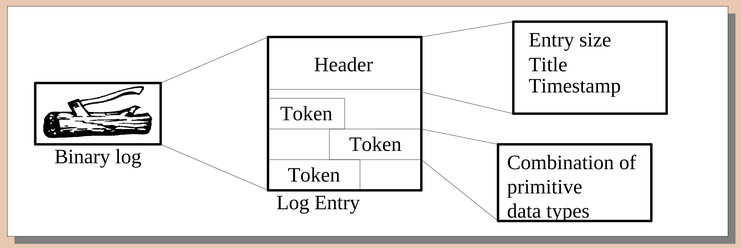


Log Structure

Our binary log files have a hierarchical structure similar to Sun Basic Security Module (BSM) logs.

Information collected by interposed libraries is composed into tokens that represent basic elements of data such as strings and integers. Tokens are combined into log entries that contain information relating to single events. These entries make up logs.

Fig. 4. Structure of binary logs



Future Work

Having completed the application design and implementation process we would now like to design a testing framework to make quantitative observations. Tests we would like to do include:

- Code coverage testing and profiling
- Comparison to other audit log generation methods in terms of performance impact, volume of logs generated, flexibility
- Run a set of known exploits (attacks) on the application and evaluate the usefulness of audit data generated about them

Further work using library interposition can include:

- Application sandboxing: limiting a non-trusted application's access to system resources
- Network monitoring through interposing TCP/IP libraries

Acknowledgments

Thanks to all Computer Science faculty, students and staff for their friendliness and helpfulness throughout the summer. Mustafa's work over the summer was made possible by Swarthmore College Office of the Provost through a Swarthmore College Student Summer Research Fellowship. Finally, thanks to Colin Purrington of Swarthmore College Biology dept. for the poster tempelate.

For more information contact: mpaksoy1@swarthmore.edu

SWARTHMORE