



Published on Free Software Magazine (<http://www.freesoftwaremagazine.com>)

OpenXDAS

A free distributed audit service

By John Calcote

No one would argue that software auditing is not an important feature of mission critical applications. If a software based process is critical to the life of your company, then so is the security and access control surrounding resources managed by that software based process. Auditing is the way you track who did what to what and when it happened. Lately, however, the software industry has been lackadaisical at best regarding auditing. Off the shelf software developers either care about auditing, or they don't. When they do care, they either create elaborate custom auditing solutions that don't inter-operate with existing tools, or they simply send a few extra messages to the system logging facility and call it auditing—more often the latter. This article is about the benefits of the Open Group's Distributed Auditing Service (XDAS), and particularly about OpenXDAS, a free software implementation of the XDAS standard sponsored by Novell.

A quick history of software auditing

In the early days of computers, software was almost a by-product. When companies like IBM and Hewlett Packard were custom-manufacturing large mainframe computer systems for banks, businesses and institutions the software was free. Heck, if you bought a computer back then, you probably just blew any budget you might have had for software, anyway! But that's the way things were done. Computer hardware was so expensive that hardware manufacturers would write and maintain custom software for their customers—almost as a fringe benefit.

As requirements were collected, software developers would try to find a fit between existing software they'd already written, and the software system that would fall naturally out of the requirements gathering process. Sometimes, only small changes were necessary to the customer's process in order to reuse an existing package already written for a previous customer. At that point, negotiations between customers and developers would result in the ability to reuse some existing software, with perhaps a few customizations. But since computers were a big deal back then, customers were willing to change their processes significantly in order to accommodate the "requirements" of the computer.

Heck, if you bought a computer back then, you probably just blew any budget you might have had for software, anyway!

Over the last 30 years, the roles of hardware and software have become juxtaposed. As elaborate general purpose software systems were developed, hardware manufacturers found it more and more difficult to sell their wares to a customer based on the merits of the hardware. Customers began to expect all of the features of certain well-known software systems from their hardware vendors. Slowly vendors began to make the transition to selling the software for real money, while nearly giving away the hardware.

Free market pressures have pushed this paradigm to the limit in recent years with common off the shelf (COTS) software packages. It now costs far less for both hardware and software than it ever has, but the ratio of software prices to hardware prices has stabilized at around two to one (respectively). For a real world

example, a reasonably high-end Intel processor-based CAD/CAM system today costs about US\$3000. But a copy of AutoCAD 2006 costs twice that amount, at around US\$6000.

What's happened to our priorities?

Unfortunately, those same free market pressures that brought (relatively) cheap hardware and COTS software to market have also hurt software quality. Software features that are not readily visible to CEOs such as auditing have slipped through the cracks. Software security architects often know the advantages of software auditing, but don't have the clout they need with their engineering and product managers to ensure that auditing is a well-designed aspect of their company's software products. Auditing is not sexy and, much like insurance policies or regular backups, it's not missed until there's a problem.

There's no doubt about it—our priorities have degraded to the point where we're willing to settle for whatever we can get these days with regard to software auditing or other important back-end features.

Auditing vs. logging

Let me lay it on the line for you. Logging is not auditing. It's really that simple, and yet we're being sold a bill of goods these days by software vendors who don't really understand proper software auditing anymore than we consumers do.

By logging, I mean either the use of existing logging systems as the Unix Syslog facility, or the Windows application and security event logs. Or, even more simply, the use of a file-based logging system, where software sends messages to the log file whenever something happens that is considered worthy of such a message. Often what's considered worthy is extremely subjective—arbitrary really—as developers are given the task to “log something”, so sales people can spout terms like auditing and logging with some degree of integrity.

But auditing is really very much more than logging. Logging allows a system to send free-form informational messages to the console or to a log file. Log messages are really designed from the start to be read by humans. They contain informational text messages such as, “Xenobar system successfully started on 3 Oct, 2006 at 15:32”, or “Footech subsystem loaded properly”. Audit messages, on the other hand, should be machine readable. Not that they should be unreadable by humans, but rather that they should be easily parsed and categorized by software.

Logging is not auditing. It's really that simple

The software I'm talking about here is called security analysis software. Entire companies have become prosperous by providing security analysis software that spends literally 90 percent of its CPU cycles just parsing and categorizing human readable log messages. They spend most of their time trying to heuristically extract minuscule bits of security-relevant information from near useless log messages pumped out by mission critical software packages. The Sentinel product by eSecurity (recently acquired by Novell) is an example of such a security analysis software package.

And now, consider what happens when a new version of Xenobar is shipped, and the latest developer working on this version decides to change all of the log messages throughout the program because he doesn't like the format used by the previous developer. After all, they're just log messages, right?

“But,” you say, “if a company recognized that security analysis software would be designed to read their log messages, could they not provide a well-defined format, and then ensure that it never changes from version to version?” Sure, and this has been done before. Novell’s eDirectory product, for example, contains a trace facility that is widely used by eDirectory customers for many reasons—including system auditing. This trace facility was originally intended to be used only as a debugging tool by eDirectory engineers, but they soon discovered that several key customers were relying on third-party tools to parse eDirectory trace logs, in order to extract information critical to their business processes. After that, the developers were ordered not to change the format of most of the trace messages between versions. It’s sad really, because now this debugging tool can no longer be used effectively for its original purpose—to help engineers find and fix defects in the product.

Proper auditing demands three key features:

1. An application programmer’s interface or API, entirely separate from the logging facility.
2. A common standardized audit event record format.
3. A common standardized audit event taxonomy (a set of events with associated well-defined and documented semantic meanings).

The Bandit Project—the future of identity?

Novell has always been a market leader in enterprise-level identity services. Since the acquisition of SuSE Linux several years ago, Novell has also become a leader in providing standards-based free software solutions for the enterprise.

The Bandit Project (<http://www.bandit-project.org>) is a free software effort sponsored by Novell to support the best free software identity infrastructure components on the internet today. If Bandit Project administrators can’t find a required identity infrastructure component, then they create a free software project around the missing concept, using existing open networking standards whenever possible.

Besides OpenXDAS, the Bandit Project sponsors a role calculation engine based on the RBAC standards, an authentication credential store called CASA, and an identity attribute service (IDAS) provider for the Higgins identity framework. Higgins is an Eclipse technology project designed to bring identity providers together into a single authentication realm that can be used by service providers and consumers on the internet, as well as within the enterprise.

Novell and the Bandit Project are primary sponsors of the OpenXDAS project. Bandit Project members consider software auditing a required part of a well-rounded identity services infrastructure component set. Bandit Project identity infrastructure components are designed to work equally well in the free software world and at the enterprise level. By providing free identity services infrastructure components, Novell promotes one of its primary goals—to grow the industry.

Enter OpenXDAS and the XDAS standard

In 1998, the value of proper auditing was being reconsidered by a group of major software vendors around the world. These companies approached the Open Group, an open standards organization famous for world-wide computing standards such as the Single Unix Specification, the Distributed Computing Environment (DCE), CORBA and the XWindow system, and many other well-known standards and certifications. Representatives formed a working group called the Distributed Audit Service (DAS) working group, in an effort to define a common standard audit record format for both storage and transmission. Their goals included the definition of a record format that was both easy for machines to parse, and for humans to read.

OpenXDAS

The results of this year-long effort were published in the initial XDAS preliminary specification in 1999. Unfortunately, a series of coincidental events culminated in the discontinuation of the project. It's interesting to note, however, that nothing better than XDAS has ever been suggested to any standards body since that time. Other network and device management standards have arisen, as well as proprietary specifications, which have incorporated some of the features of XDAS. But no auditing standard as concise and focused has ever been attempted since the dissolution of the XDAS working group.

In 1998, the value of proper auditing was being reconsidered by a group of major software vendors around the world

XDAS defines a basic C language application programmer's interface (API). This API was designed with two goals in mind—simplifying the transition from the traditional use of logging facilities as an auditing tool, and capturing all of the required bits of information. The XDAS C language API is designed to be as simple as possible, but no simpler. Developers accustomed to instrumenting their applications for auditing with a logging facility might be disinclined to use a complex auditing interface when they've been using a simple, single-function formatted string logging interface in the past. While the XDAS C language API is not as simple as, say Syslog for example, neither is it difficult.

Developers create an OpenXDAS session once at process startup, and close the session once at process termination. During the life of the process, each time an audit-worthy event occurs, the program should call an OpenXDAS function to create a new record, and another function to commit that record.

XDAS defines a standard record format containing what the DAS working group deemed to be a canonicalized form of the key elements of an audit record:

- Event
- Originator
- Initiator
- Target
- Source
- Data

These standard audit record elements make up the auditing equivalents of what we might recognize as the key verification elements present in human interactions. For instance, when a journalist quotes someone regarding a witnessed event, he or she includes the following information in the article:

- The event (the nature of the news-worthy occurrence)
- The journalist's own name and credentials
- The person or organization that initiated the event
- The person or organization affected by the event
- The person reporting the event (or sometimes just "anonymous trustworthy sources")
- Additional interesting data about the event

Another analysis of this critical information set shows that it parallels human conceptual events. Take the following sentence, for example:

Joe rode the bus.

There are three critical bits of information in this simple sentence. These are formally called the "subject", the "action" and the "object". The subject is usually found in the first part of the sentence and refers to the initiator of the action. The action itself is the verb—what's happening—the occurrence that's being reported.

OpenXDAS

Finally, the object is generally at the end of the sentence and represents the target of the action. In this example, “Joe” is the subject who did something. “rode” is the action, or what it was that Joe did, and “the bus” is the object, or what was affected by Joe’s action.

OpenXDAS is a free software implementation of the Open Group’s XDAS standard. The OpenXDAS project is still in its infancy. What this means is that OpenXDAS is not yet a complete implementation of the XDAS standard. So is it useful? The answer is a resounding yes.

The XDAS standard defines several categories of API functionality. One of these is called the “Event Submission” API. Event submission implies the ability to generate complete audit event records in the XDAS common record format.

In truth, OpenXDAS attains about 80 percent of its most useful functionality just by implementing the event submission API. OpenXDAS implements the XDAS standard as a C language client instrumentation library, a Java language client instrumentation library, and a client-side service or daemon process that provides a common point of client-side filtering and configuration management.

The initialization API provides the following functions. `xdas_initialize_session` must be called once at application startup, and `xdas_terminate_session` must be called once at shutdown:

```
int xdas_initialize_session(
    int * minor_status,
    const char * org_info,
    xdas_audit_ref_t * das_ref);

int xdas_terminate_session(
    int * minor_status,
    xdas_audit_ref_t * das_ref);
```

The event submission API provides five functions, but don’t let this number deceive you—the submission API can be effectively used with only two of them:

```
int xdas_commit_record(
    int * minor_status,
    xdas_audit_ref_t das_ref,
    xdas_audit_rec_desc_t * audit_record_descriptor);

int xdas_discard_record(
    int * minor_status,
    xdas_audit_ref_t das_ref,
    xdas_audit_rec_desc_t * audit_record_descriptor);

int xdas_put_event_info(
    int * minor_status,
    xdas_audit_ref_t das_ref,
    xdas_audit_rec_desc_t * audit_record_descriptor,
    unsigned event_number,
    unsigned outcome,
    const char * initiator_information,
    const char * target_information,
    const char * event_information);

int xdas_start_record(
    int * minor_status,
    xdas_audit_ref_t das_ref,
    xdas_audit_rec_desc_t * audit_record_descriptor,
    unsigned event_number,
```

OpenXDAS

```
    unsigned outcome,  
    const char * initiator_information,  
    const char * target_information,  
    const char * event_information);  
  
int xdas_timestamp_record(  
    int * minor_status,  
    xdas_audit_ref_t das_ref,  
    xdas_audit_rec_desc_t audit_record_descriptor);
```

Developers must call `xdas_start_record` and `xdas_commit_record`. The remaining functions exist for convenience in special cases.

The instrumentation library sends messages to the service through a local IPC channel, which applies filters and then forwards remaining records to installable logging facilities. It might surprise you to read that after all of the above discussion of the lack of merit in the use of system logging facilities as audit frameworks, one of the supported OpenXDAS logging facilities is none other than Syslog. The problems with using a system logging facility as an audit framework are not related to the data store, but rather to the record format. When OpenXDAS applies the XDAS common record format to syslog messages, they become so much more useful to security analysis software, which is already designed to read syslog message logs.

The benefits of a taxonomy

Another aspect of a well-designed audit system is a common event taxonomy. Remember when I said that machines must be able to easily parse and classify audit events? Well, a common record format makes parsing easy, but a common event taxonomy makes event classification easy. An event taxonomy is nothing more than a standardized set of common audit events, sometimes organized into a suitable hierarchy. XDAS defines a standard set of events that are general-purpose enough to capture a high percentage of audit-worthy events in a software system. Interestingly, there are only about 30 standard events in the XDAS event taxonomy. The events in the taxonomy are divided into nine major classes as follows:

- Account Management Events
- User Session Events
- Data Item and Resource Element Management Events
- Service or Application Management Events
- Service and Application Utilization Events
- Peer Association Management Events
- Data Item or Resource Element Content Access Events
- Exceptional Events
- Audit Service Management Events

Events in the Account Management Event class include the following:

- Create account
- Delete account
- Disable account
- Enable account
- Query account attributes
- Modify account attributes

A standard event taxonomy allows security analysis software to quickly and efficiently capture the essence of an event. The software may be configured to generate an alarm or execute an external program on receipt of a

specific type of event. This process is completely deterministic with a standard event taxonomy, but somewhat arbitrary and subjective with free-form log messages. A certain log message may have been intended to convey the desired information, but was simply not formatted according to the pattern matching criteria specified by the analysis software. A standard event taxonomy alleviates this problem.

Often, developers come to me, ready to instrument their applications, with questions like, “My application doesn’t manage accounts! How can you say these events fit my needs?” Developers must realize that this is a very generic set of high-level security-relevant events. While account management may be a perfect fit for software systems that keep track of users through user accounts, account management might also be thought of more generally as a set of events related to the creation and management of the persistent storage of attributes associated with users in any fashion. Some applications simply may not have anything to do with account management in any form, so they won’t use the account management events.

Security-relevant, or just stuff that happens

The XDAS standard is aimed at creating a common security infrastructure in the face of existing security or pseudo-security systems already widely in use. For instance, one of the XDAS C language API functions is designed to import records from native logging facilities. This import process sorts through the myriad of log messages in a system log, extracting messages that have security relevance, and reformatting them into XDAS format. Remember that one of the fields of the common XDAS record format is the “Source” field. This field is generally left blank when XDAS instrumented applications commit audit records. It’s used during import when log records in other systems are converted to XDAS common record format and then submitted to the XDAS auditing framework. The Source field contains a referral to the original record from which an XDAS record was generated. It could be any type of referral, but in today’s world, a URL seems an appropriate vehicle for source record information.

So often, developers submit reams of log records containing nothing useful to security analysis software

So often, developers submit reams of log records containing nothing useful to security analysis software. The XDAS common event taxonomy helps developers determine what’s security-relevant, and what’s just interesting stuff that happened.

Getting the package

OpenXDAS is available in several package formats, including SuSE and RedHat binary RPM’s for Intel and AMD 32 and 64 bit platforms, Windows installer packages, simple “tarballs”, and of course full source code archives in both tar.bz2 and zip formats. You can download the latest version of the OpenXDAS packages from [SourceForge](#), and you’ll find the [project page](#) there too. Both pages are cross-linked, so you should have no trouble getting to one from the other.

There is also very complete documentation in both OpenOffice (2) format and Adobe PDF format. Documentation currently includes a complete user’s manual targeted toward developers wanting to instrument their applications to submit XDAS audit records, and even a conversion guide for developers who have previously instrumented their applications using the Novell Nsure Audit SDK.

Summary

OpenXDAS

More than ever, corporate software consumers are begging vendors to include more robust auditing features in their software packages. OpenXDAS provides a well-designed, standards-based approach to relatively simple audit instrumentation. As time passes, consumers will demand better auditing functionality. This is not an opinion, but a fact driven by recent business world events such as the Enron and Arthur Anderson fiascos. The US Securities Exchange Commission is cracking down with tougher requirements for corporate security policy. A better auditing system is coming, and Novell and the developers of OpenXDAS hope to pave the way through this new legal requirements jungle with a well-designed common audit record.

Biography

[John Calcote](/user/28810) (/user/28810" title="View user profile.):

Copyright information

Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.

Source URL:

<http://www.freesoftwaremagazine.com/articles/openxdas>
