

# OpenXDAS

Version 0.4.226

## ***Novell Audit Instrumentation Conversion Manual***

Document Version 0.1

March 14, 2007

John Calcote



## Converting from Novell Audit to OpenXDAS

Most people converting from Novell Audit will be familiar with the Novell Audit SDK documentation, which is part of the Novell Developer's Kit. This conversion document is organized in a similar manner, considering each component discussed in the Novell Audit SDK documentation in the order they are discussed in that document, beginning in Chapter 2 of the Novell Audit SDK documentation entitled, "Instrumenting an Application".

Despite its existence in the OpenXDAS user's manual, some extremely relevant background information is in order. First, several of the fields in the Novell Audit Service event messages are generic fields designed to carry highly domain-specific event information. While it's true that this information is important, it is of secondary importance to the primary XDAS event fields, these being the XDAS Originator, Initiator and Target information. This XDAS primary data is used to properly autonomously classify audit events. The secondary Event information is important for later human analysis, if necessary.

The domain-specific Event Information field is comma (',') delimited, name-value pairs, where names and values are separated by an equal ('=') sign. Here's an example Event Information string which might be reported as part of an XDAS event:

```
char * event_info = "laf-id=1701,gid=3C45AD7B,file=15,system=dewey"
```

The names of each event data component identify the context of the value for purposes of both software and human analysis.

The values are completely free-form, but embedded commas or equal signs must be escaped with the XDAS escape character, percent ('%'). As usual, the escape character itself must also be escaped in order to be used literally in the string (eg., "%%" represents "%").

OpenXDAS itself defines several of these Event Information variables for its own purposes. The comprehensive list is found in the user's and developer's manual mentioned earlier, but some of these are better defined in this document in relevant sections.

### ***Event ID***

The Novell Audit Event Identifier is a 32-bit event discriminator made up of two 16-bit values: The application identifier, and the event identifier. Dividing the Novell Audit Event ID into application identifier and event identifier in this manner allows registered applications control over their own event identifier space, which is really convenient for the application developer, but at the expense of unnecessary added complexity in the event space itself. It dramatically broadens the possibility of overlapping semantic meanings among events in the entire event identifier space. In other words, most applications will likely define a significant portion of their audit events to mean exactly the same thing as similar events in most other applications.

This overlap causes a significant correlation problem in back-end analysis tools. These tools have to be familiar with every application's event space, instead of a single event space to which every application tries to conform. Required event data may or may not exist in each similar event type, but worse still, each event type has its own data format for the same data. This compounds the correlation problems of these analysis tools. They must not only correlate event types among applications, but they also have to

correlate data fields sent with these differing (but semantically equivalent) event types.

The XDAS taxonomy provides for a large portion of security-related event conditions. These include the following categories:

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

Additional event classes may be registered with the OpenGroup if required, but this should be a last-resort activity, as the true value of XDAS as an event taxonomy standard is found in the fact that the standard provides a normalized taxonomy, which can be analyzed in a domain-independent fashion by third-party analysis tools.

With this background, it should now be understood that an attempt must be made by the developer to convert application domain-specific Novell Audit Event ID values into normalized XDAS event taxonomy values. Please refer to the *OpenXDAS User's and Developer's Manual* for normalized XDAS event taxonomy values and symbolic definitions for these values.

### ***Log Level***

There is no direct XDAS equivalent to this Novell Auditing field value. The concept of log-level is inherited from the Unix Syslog facility, and other system logging services. In these types of logging facilities log messages are appended by applications for a variety of reasons. A distributed auditing service is designed to gather information about audit-worthy events that occur in security-relevant applications and environments. This mission statement basically implies that all recorded auditing events are of security relevance, and thus at the same general level of importance.

In addition, some degree of log level information is inherent in the XDAS event taxonomy. For instance, events within the following XDAS event classes can be said to map to various log levels defined by the Novell Auditing service:

#### **XDAS Event Classes**

#### **Novell Audit Log Level**

XDAS_AEC_ACCOUNT_MANAGEMENT	LE_INFO
XDAS_AEC_USER_SESSION	LE_INFO
XDAS_AEC_DATA_ITEM_MANAGEMENT	LE_INFO
XDAS_AEC_SERVICE_MANAGEMENT	LE_ALERT
XDAS_AEC_SERVICE_UTILIZE	LE_INFO
XDAS_AEC_PEER_ASSOC_MANAGEMENT	LE_INFO
XDAS_AEC_DATA_ITEM_CONTENT_ACCESS	LE_INFO
XDAS_AEC_EXCEPTIONAL	LE_ALERT / LE_EMERGENCY

This table shows a (rather arbitrary) classification of event classes within XDAS as related to Novell Audit Log Level, however nothing needs to be done by the instrumenting developer relative to this mapping. It is shown here only for the sake of interest.

### **Group ID**

The Novell Audit Group ID value is used to tag multiple events belonging to a single transaction. The Group ID is managed directly by the reporting application and is often set to zero, indicating that an event is not related to any other events.

There is no direct XDAS equivalent to the Novell Audit Service Group ID field. Primarily, this is due to the fact that XDAS audit events are high-level security events. As such, instrumented applications should generally not report several lower-level events which should be analyzed as a group in order to determine the security impact of the overarching event. However, when event grouping cannot be avoided the application developer should use a specific Event information data element defined by OpenXDAS for this purpose:

```
gid=<unique-identifier>
```

where <unique-identifier> is any UTF-8 text string that can simply be matched with the `gid` value of other potentially related events.

### **IP Address**

The Novell Audit IP Address field passes the IP address of the reporting host. This address information is reported in XDAS as part of the Originator information – specifically the <org\_location\_address> field of the Originator information string, as in the following example (the other fields in the Originator information string have been left blank for clarity):

```
char * org_info = ":192.168.17.2:::";
```

The XDAS specification refers to this field as a “communication service end point address”, implying that this address may be used to connect to a communication service end point. In today's world of web-based management initiatives and standards, a complete service end point address would consist simply of a URL. At the time the XDAS specification was written (ca 1998), the use of URL's was limited to web services, and were not considered a viable end point addressing scheme for general purpose use.

The need to connect to a service by address was still an issue at that time, so an additional field, <service\_type> was defined as part of the Originator information to allow the event reporter to specify additional service type context information of the end point address. URL's combine service type with service address and resource identifier information to create a complete end point access solution (eg, a unified or universal resource locator – URL). The following example shows the XDAS encoding of this address and service type information in the Originator information string:

```
char * org_info = ":192.168.17.2:ldap:::";
```

In this case, the service communication end point service type is “ldap”, which tag identifies a service protocol, as well as a service port. In case a non-standard port needs to be specified, add it in

parenthesis to the end of the service type string, as in the following example:

```
char * org_info = ":192.168.17.2:ldap(401)::";
```

**SIDEBAR:** A future version of the XDAS specification will undoubtedly combine these two fields into one, indicating that the format should be standard URL format.

### ***Client Time Stamp***

The client time stamp may be set manually for an event by calling the `xdas_timestamp_record()` function, or else it will be set automatically by the OpenXDAS client library at the time the event is committed by calling `xdas_commit_record()`.

There is no XDAS equivalent to the Novell Audit Server Time Stamp field. The only possible value the server time stamp field can have is to show the time it takes an event to travel from client to server, indicating any transport delays, such as unconnected audit server, etc.

Some advocates of a server time stamp might say that a server time stamp would also indicate a time differential between client and server – the server time would presumably be accurate. However, if both transport delay and time differential were present, then the analysis of client and server time stamps would indicate ambiguous conditions.

This information has little relevance to security auditing, and therefore XDAS does not incorporate a server time stamp.

### ***String and Numeric Values***

The Novell Audit general purpose string and numeric values (1, 2 and 3) are basically catch-all fields for domain-specific event information. This information is best captured in the Event Information string as name/value pairs, where the data type of the value is implied by the variable name.

The Novell Audit Service documentation suggests that the first two of these three string and numeric values be used to indicate additional sub-target information. However, target and sub-target information are encoded in the XDAS standardized Target information fields, as described in the OpenXDAS user's manual.

While these domain-specific name/value pairs can be interesting to a human reader, it should be recognized by the instrumenting application developer that automatic analysis tools cannot rely heavily on the existence of various name's in the variable list of the Event information string. Automatic analysis will best be facilitated by adding as much information to the standardized XDAS fields as possible.

### ***Target, Sub-Target and Target Type***

The Target, Sub-Target, and Target Type fields of the Novell Audit record are recorded in the Target information fields of an XDAS record. Target and Sub-Target information is encapsulated in the XDAS hierarchical name format of the `<tgt_location_name>` and `<tgt_location_address>` fields. Target Type is reported in the XDAS service in the `<tgt_service_type>` field.

## ***Originator and Originator Type***

The Originator and Originator Type fields of the Novell Audit record are recorded in the Originator information fields of an XDAS record. Target and Sub-Target information is encapsulated in the XDAS hierarchical name format of the `<org_location_name>` and `<org_location_address>` fields. Target Type is reported in the XDAS service in the `<org_service_type>` field.

## ***Data, Size and MIME Hint***

These fields are related, and contain domain-specific event information at the most fundamental level. The Novell Audit Service documentation indicates that these fields contain any event-specific data, the size of the data, and a MIME type hint about the nature of the data (respectively). This type of information is an obvious candidate for the XDAS Event information string.

Binary data must be encoded somehow in UTF-8. The most obvious way to do this is by encoding the binary data in base 64 encoding. However, the application developer should remember that the more human-readable this information is, the better it will be understood by human readers, as automated processes are not likely to spend many CPU cycles on such encoded data.

String values should not contain any control characters.

## ***Determining Event Generation Points and Event Data***

The Novell Audit Service documentation suggests that the next step is to collect events that need to be published. We concur with this approach, except that the XDAS taxonomy provides the event numbers, and the instrumenting application developer should only locate points within the application where security-relevant events might be generated, and map those events into the XDAS event taxonomy.

For developers simply switching from Novell Audit to OpenXDAS, this may already be done, as Novell Audit instrumentation already exists in the application, and those instrumentation points represent a collection of event generation points. However, the developer might wish to re-examine his or her code at this point in connection with the XDAS event taxonomy to see if other event generation points of security significance might exist. In addition, the developer might also wish to re-examine the existing points of audit event generation to see if they really are of security significance. The standard XDAS taxonomy makes evaluation of security-relevance easier.

Reporting event data (other than standard XDAS field values) should be considered carefully for security relevance.

## ***Using the XDAS API***

Event reporters should initialize an XDAS session once in the application's startup code path, and then tear down this session once in the shutdown code path. The OpenXDAS functions to use for session initialization and termination are `xdas_initialize_session()`, and `xdas_terminate_session()`, respectively.

At each point in the code where a Novell Audit event is reported, an XDAS record should be created with `xdas_start_record()`. The security-relevant data reported in the Novell Audit event should be reformatted into appropriate XDAS information strings and submitted in that same initial call to `xdas_start_record()`, or added to the record in one or more calls to `xdas_put_event_info()`. Then the completed record should be submitted to the XDAS service with `xdas_commit_record()`.