Structured logging offers a variety of advantages, including simpler parsing, easier format conversion, and more flexible classification and correlation of events, even across diverse log sources.

**About Structured Logging**

Each event is represented by an unrestricted set of key-value pairs

The BSD Syslog format provides only a very restricted set of metadata fields in the header. Any metadata that is specific to the event or log source, such as username or IP address, must be included in the free-form string.

A BSD Syslog event

```
<38>Nov 22 10:30:12 myhost sshd[8459]: Failed password for invalid user linda from 192.168.1.60 port 38176 ssh2
```

As a result, important metadata are not accessible during processing without resorting to extraction using regular expressions or other pattern matching. To support full classification and correlation of events, this additional metadata should also be available. The answer for this is structured logging, where each event is represented by an unrestricted set of key-value pairs. The format commonly used for this is JSON.

**The Syslog event as structured JSON**

```
{
  "SyslogFacility": "USER",
  "SyslogSeverity": "NOTICE",
  "EventTime": "2019-11-22 10:30:12",
  "Hostname": "myhost",
  "SourceName": "sshd",
  "ProcessID": 8459,
  "Message": "Failed password for invalid user linda from 192.168.1.60 port 38176 ssh2",
  "Status": "failed",
  "AuthenticationMethod": "password",
  "Reason": "invalid user",
  "User": "linda",
  "SourceIPAddress": "192.168.1.60",
  "SourcePort": 38176,
  "Protocol": "ssh2"
}
```

Notice that fields have been extracted from the free-form string as well as from the metadata fields in the
Syslog header. For messages generated by the source in Syslog format, this requires maintaining a collection of patterns. However, as logging systems move toward structured logging, pattern-based parsing can be avoided altogether, simply by taking full advantage of the structured log data available at the source.

Retaining Structured Data at the Source

Regular expression parsing is unnecessary when the data is already structured at the source

Windows EventLog stores events in a proprietary binary format which is not human-readable. A consumer, typically Event Viewer, renders the human-readable message by decoding the event records into, in this case, an XML format with each metadata value stored in a separate element. The System portion contains a standard set of values common to all events, while any additional key-value pairs logged by the event provider are stored in the EventData portion.

Windows EventLog XML (excerpt)

```xml
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft-Windows-Security-Auditing" Guid="{54849625-5478-4994-A5BA-3E3B0328C30D}" />
    <EventID>4625</EventID>
    <Level>0</Level>
    <TimeCreated SystemTime="2019-02-02T14:47:43.520229400Z" />
    <Channel>Security</Channel>
    <Computer>WIN-AQ35E3LLSVB.example.com</Computer>
  </System>
  <EventData>
    <Data Name="SubjectUserName">WIN-AQ35E3LLSVB$</Data>
    <Data Name="SubjectDomainName">EXAMPLE</Data>
    <Data Name="TargetUserName">Administrator</Data>
    <Data Name="TargetDomainName">EXAMPLE</Data>
    <Data Name="Status">0x0000006d</Data>
    <Data Name="FailureReason">%2313</Data>
    <Data Name="LogonType">2</Data>
    <Data Name="WorkstationName">WIN-AQ35E3LLSVB</Data>
    <Data Name="IpAddress">127.0.0.1</Data>
    <Data Name="IpPort">0</Data>
  </EventData>
</Event>
```

Some log collectors simply discard most of the fields in Windows EventLog XML
As shown in the example XML, the source event data is already structured as key-value pairs. There is no need to use pattern-based parsing when the data is already structured at the source. Unfortunately, however, some log collectors simply discard most of these fields in the EventLog source, collecting only the rendered message and a limited set of metadata. As a result, not only is some metadata lost, but what remains must now be parsed from the human-readable message. For example, here is the same event in the Snare tab-delimited format. The EventData fields are now only available in the free-form message, and some of the values have been replaced by human-readable strings.

**The Windows event in the Snare format**

```
win-aq35e3llsvb.example.com → MSWinEventLog → 1 → Security → 971 → Sat Feb 02 06:47:43 2019 → 4625 → Microsoft-Windows-Security-Auditing → EXAMPLE\Administrator → N/A → Failure Audit → WIN-AQ35E3LLSVB.example.com → Logon → → An account failed to log on. Subject: Security ID: S-1-5-18 Account Name: WIN-AQ35E3LLSVB$ Account Domain: EXAMPLE Logon ID: 0x3E7 Logon Type: 2 Account For Which Logon Failed: Security ID: S-1-0-0 Account Name: Administrator Account Domain: EXAMPLE Failure Information: Failure Reason: Unknown user name or bad password. Status: 0xC000006D Sub Status: 0xC000006A Process Information: Caller Process ID: 0x4d0 Caller Process Name: C:\Windows\System32\svchost.exe Network Information: Workstation Name: WIN-AQ35E3LLSVB Source Network Address: 127.0.0.1 Source Port: 0 Detailed Authentication Information: Logon Process: User32 Authentication Package: Negotiate Transited Services: - Package Name (NTLM only): - Key Length: 0 This event is generated when a logon request fails. It is generated on the computer where access was attempted. The Subject fields indicate the account on the local system which requested the logon. This is most commonly a service such as the Server service, or a local process such as Winlogon.exe or Services.exe. The Logon Type field indicates the kind of logon that was requested. The most common types are 2 (interactive) and 3 (network). The Process Information fields indicate which account and process on the system requested the logon. The Network Information fields indicate where a remote logon request originated. Workstation name is not always available and may be left blank in some cases. The authentication information fields provide detailed information about this specific logon request. - Transited services indicate which intermediate services have participated in this logon request. - Package name indicates which sub-protocol was used among the NTLM protocols. - Key length indicates the length of the generated session key. This will be 0 if no session key was requested. -
```

Rather than discarding the structure of the event data at the source and then parsing metadata from the rendered message, the fields should be collected directly from the event source. This is the same event in JSON format (with the rendered message truncated for brevity):
The event XML converted to JSON (excerpt)

```json
{
    "EventTime": "2019-02-02T06:47:43.520229-08:00",
    "Hostname": "WIN-AQ35E3LLSVB.example.com",
    "EventType": "AUDIT_FAILURE",
    "Severity": "ERROR",
    "EventID": 4625,
    "SourceName": "Microsoft-Windows-Security-Auditing",
    "ProviderGuid": "{54849625-5478-4994-A5BA-3E3B0328C30D}",
    "Channel": "Security",
    "Message": "An account failed to log on. [...]",
    "Category": "Logon",
    "SubjectUserName": "WIN-AQ35E3LLSVB$",
    "SubjectDomainName": "EXAMPLE",
    "TargetUserName": "Administrator",
    "TargetDomainName": "EXAMPLE",
    "Status": "0xc000006d",
    "FailureReason": "%2313",
    "LogonType": "2",
    "AuthenticationPackageName": "Negotiate",
    "WorkstationName": "WIN-AQ35E3LLSVB",
    "IpAddress": "127.0.0.1",
    "IpPort": "0"
}
```

Classifying and Correlating Events Across Log Sources

Events can be classified and correlated using any of the available metadata

With structured logs, accessible event metadata is not limited to basic metadata such as timestamp and severity. As a result, events can be classified and correlated using any of the values provided by the event set.

For example, an attack originating from a particular IP address may result in events across many different systems. With unstructured logs, the IP address is not available without pattern-based parsing:

A basic unstructured Syslog event, difficult to correlate

```
<38>Nov 22 10:30:12 myhost sshd[8459]: Failed password for invalid user linda from 192.168.1.60 port 38176 ssh2+
```

The attack can be analyzed much more easily, even across a diverse set of log sources, if all events are
structured and include a **SourceIPAddress** field:

An excerpt of the same event as structured data, easier to correlate

```json
{
  "SyslogFacility": "USER",
  "SyslogSeverity": "NOTICE",
  "EventTime": "2019-11-22 10:30:12",
  "Hostname": "myhost",
  "SourceName": "sshd",
  "ProcessID": 8459,
  "Status": "failed",
  "AuthenticationMethod": "password",
  "Reason": "invalid user",
  "User": "linda",
  "SourceIPAddress": "192.168.1.60",
  "SourcePort": 38176,
  "Protocol": "ssh2"
}
```

In order to correlate and filter events across diverse log sources, it is important to use a common set of fields. These field names can be used when parsing unstructured logs, or can be mapped from equivalent fields in a structured log source.

## Converting Between Formats for Integration

**Structured data can be serialized to JSON and used as a Syslog payload**

Structured logging often simplifies conversion between formats, allowing event data to be more easily integrated with logging dashboards, analytics platforms, and other log processors. Structured logs can also help work around limitations in log formats.

For example, to transport structured data using Syslog, the key-value pairs can be serialized to JSON and used as the payload. In the case of BSD Syslog, which lacks a proper timestamp, the full timestamp with year can be preserved in the serialized JSON data (see **EventTime** below).

**BSD Syslog event with JSON payload**

```plaintext
<13>Nov 22 10:30:12 myhost sshd[8459]:
```
Or the key-value pairs can be used as the structured data portion of the IETF Syslog format:

**IETF Syslog event with structured data**

```plaintext
<13>1 2019-11-22T10:30:12.000000-06:00 myhost sshd 8459 - [NXLOG@14506
EventReceivedTime="2019-02-06 21:04:36" SourceModuleName="in"
SourceModuleType="im_file" Status="failed" AuthenticationMethod="password"
Reason="invalid user" User="linda" SourceIPAddress="192.168.1.60" SourcePort="38176"
Protocol="ssh2"] Failed password for invalid user linda from 192.168.1.60 port 38176
ssh2
```

This can be done for many other log formats also, by either serializing fields into a free-form message area or using a key-value area specified by the format.

**Increasing the Reliability of Event Parsing**

Structured logging reduces the dependence on pattern matching that must be updated anytime there are changes in generated log messages.

Because unstructured logs contain metadata in free-form messages, pattern-based parsing is often implemented to extract this metadata for analysis. When there are multiple points of analysis, for example by both an intrusion detection system (IDS) and a log monitoring application, there may be multiple pattern databases used and these patterns may be implemented in different formats.

When the message format of an unstructured log source changes, the patterns stop matching and message parsing fails. The failed parsing must be investigated and all applicable pattern databases updated to correspond with the new message format. In particular, if a log collector reads the rendered message from Windows EventLog rather than the structured data, an added field can break parsing when it is included in the human-readable message.

For log sources such as Windows EventLog that are available as structured data, it is not necessary to maintain pattern databases. Instead, the structured data should be collected directly. For unstructured logs, the parsing should be done as soon as possible and the events handled as structured data. In either case, once the log data is structured, fields can be freely added to events at any point in the log processing chain.

**NXLog: Designed for Structured Logging**

Modern enterprise systems generate an overwhelming amount of log data which must be collected, transported, managed, and monitored. Environments with complex logging requirements can benefit greatly from structured logging.
NXLog is a modular log collector that was designed around the concept of structured logging, and NXLog continues to embrace this approach.

**Internally structured events**

When NXLog receives an event, it is represented by a set of fields. Any changes to an event are performed by reading from and writing to those fields. See Event Records and Fields in the NXLog User Guide for more details.

**Direct collection and storage of structured logs**

Many modules read directly from, or write to, structured sources. For example, the im_msvistalog module reads XML data from the EventLog API and maps it directly to NXLog fields. Other modules connect to databases, read or write structured formats such as JSON, or interface with other platform-specific sources.

**Event classification and correlation**

NXLog can classify, correlate, and filter events according to fields, as required for log processing and alerting. Further analytics, if necessary, can be performed after log data has been collected and processed by NXLog.

**Parsing and generating log data**

Modules provide functionality for reading, writing, and converting log data between many different formats and transports. See Available Modules for a full list of the modules available with NXLog Enterprise Edition.

**Field extraction**

Further extraction of fields can be done with regular expressions, XML pattern databases, or Grok patterns. See Extracting Data.

**Flexible configuration**

NXLog’s thoroughly documented configuration is flexible and tailored to structured logging. Conversion between formats is straightforward. Structured data can in many cases be easily serialized and encapsulated to preserve structured data across legacy formats.

---

NXLog Ltd. develops multi-platform log collection tools that support many different log sources, formats, transports, and integrations. The tools help administrators collect, parse, and forward logs so they can more easily respond to security issues, investigate operational problems, and analyze event data. NXLog distributes the free and open source NXLog Community Edition and offers additional features and support with the NXLog Enterprise Edition.

This document is provided for informational purposes only and is subject to change without notice.