Log Parsing 101

Introduction

At very high level, NetWitness for Logs can consume 'Log-based events' information from hundreds of Event Sources and transform it into actionable information which can be used for monitoring, reporting, and compliance purposes for the end users. They can use this information to keep track of everything happening in their Enterprise Network conveniently in one location on a real-time basis.

The Event Sources could vary from range from -

- Security devices like Firewalls, Antivirus, E-mail gateways and scanners, Data loss prevention (DLP) systems, Intrusion detection & prevention systems (IDS & IPS)
- Network components like Routers, Switches, Gateways
- Operating Systems
- Data Storage Systems
- Web Servers, etc.

Pre-Requisites

NetWitness Stack

At a minimum, following NW services are needed for NW Content Development. Preferably deploy the latest GA version available to the customers.

1. NetWitness Server
2. Log Collector + Log Decoder
3. Concentrator
4. NetWitness Live Account

Useful Development Tools

1. Notepad++
    - This is free to download tool. It is useful since it has lot of searching, editing options/plug-ins.
2. WinSCP
    - To transfer files between NW appliances and local systems
3. Remote Desktop
    - To connect to Windows-based systems in the lab
4. Putty client
    - To connect to NW appliances as well as UNIX-based systems in the lab

Log Parser Development for New Event Source

Log Parser Development process for New Event Source can be broadly divided into 4 main stages -

Research

1. *Function*
    - Understand the main functions of the product.
    - What does the Event Source actually do?
    - What among those functions would be relevant for the Security Monitoring and Analysis purposes?
2. *Product Version*
    - Find out the Latest version available in the market.
3. *Vendor Documentation*
    - Find the vendor documentation especially Administrator Guides and Configuration guides.
    - Type of Logging:
        - Find out what and how the event source is tracking all the activity that it is handling or the network traffic that is going through

Log Integration with NetWitness

Depending on how the event logs are being stored, the Log collection method can be decided using which the events are sent over to NetWitness.

Note:

For details on all configurations methods - refer to Log Collector Documentation here - RSA NW Log Collection Deployment Guide on RSA Link.

Log Collection Methods:

Here are some of the *most commonly* used collection methods -

Syslog

- This is the standard format in which NetWitness recognizes the events. Hence, there is no need of configuration any method of collection for that device.
- Make sure the Log Decoder service is running and the Capture is ON.

ODBC

- Few Event Sources store their events in their own database. ODBC collection method is used to pull out the events and send it to NetWitness.
- Here are a few links describing the configuration steps:-
  1. ODBC Collection Basics
  2. Collection Procedures
  3. ODBC DSNs Configuration Parameters
  4. ODBC Event Source Configuration Parameters

File Reader

- Most Event Sources either have stripped down versions of standard OS's or are installed on machines running standard OS's, in which case, the logs are written to a specific directory. These logs have to be delivered to an NetWitness using the in-house SFTP (Secure FTP) Agent from where the File reader collection does the task of picking up the delivered logs and transforming them as per the provided configuration.
  1. File Collection Basics
  2. File Collection: Configure Event Sources in SA
  3. File Collection: Procedures
  4. File Collection: Configuration Parameters

- Guidelines for configuring the SFTP agent -

  1. *Windows:* Install and Update the SFTP Agent
  2. *UNIX:* Configure SFTP Shell Script File Transfer

SNMP Traps

Sometimes the Event Sources create their events in the form of SNMP traps.

-
  1. SNMP Collection: The Basics
  2. SNMP Collection: Procedures
  3. SNMP Collection: Configure Event Sources on NW
  4. SNMP v3 User Manager Configuration Parameters

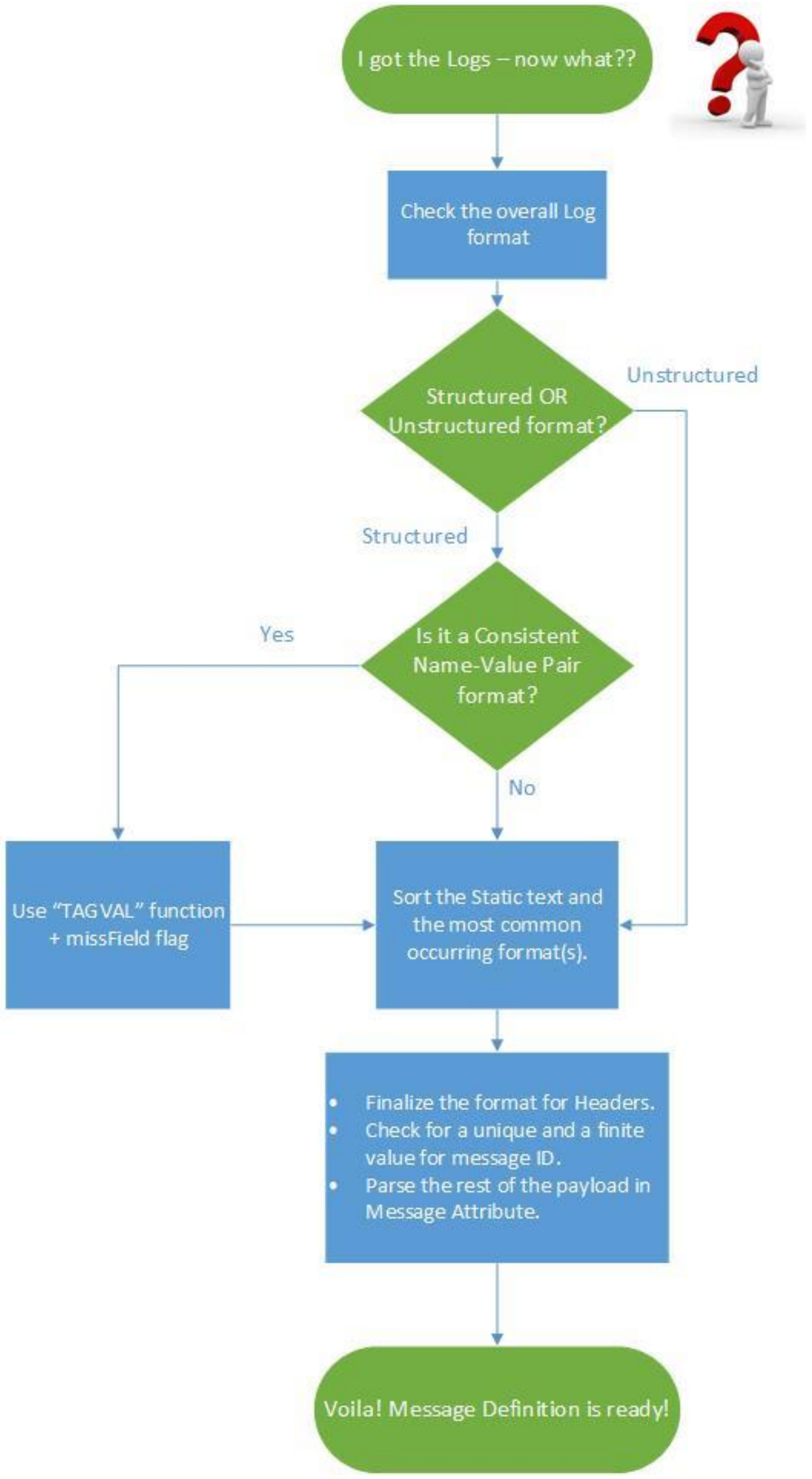
Windows

- A separate collection method is used to collect Windows OS logs which are commonly called Windows Event Logs.
- The in-house collection method is using the WIN *RM* (Windows Remote Management).
  - Windows Collection: The Basics
  - Windows Event Source Configuration Parameters
  - Useful Docs for Win RM Configuration and Troubleshooting

        - Microsoft WinRM Configuration Guide
        - Test and Troubleshoot Microsoft WinRM Guide

- Third party tools are used for Windows events' collection -
    - *Snare*
    - *Event Reporter (ER)*

Parser Development


I got the Logs – now what??
Check the overall Log format
Structured OR Unstructured format?
Unstructured
Structured
Is it a Consistent Name-Value Pair format?
Yes
No
Use "TAG VAL" function + missField flag
Sort the Static text and the most common occurring format(s).
Finalize the format for Headers.
Check for a unique and a finite value for message ID.
Parse the rest of the payload in Message Attribute.
Voila! Message Definition is ready!

- Refer to *"Creating an Event Source XML File"* section in the ESI Tool **Overview Guide**: for the basics on XML development.

- Refer to "Log Parser Best Practices" section on this page for understanding some more intricate details behind building a quality Log parser.

Deliverables

Following artifacts are expected to be delivered for "releasing" a Log Parser -

1. Log Parser XML
2. Sample log file
3. In case of Collection methods like ODBC, File Reader - their respective collection configuration files are to be delivered as well.
4. Event Source Configuration Guide
    - This is a user-guide explaining the detailed steps to configure the collection from the event source and getting those events parsed successfully onto the Log Decoder
    - Current Documentation for RSA-supported Event Sources

Log Parser Best Practices

How to define a Header and Message ID?

Header

- The Header is used to help identify the Event Source by identifying and defining the main theme in its Event log format.
- The header identifies what the message is and where does it start.
- Each header element requires the following attributes:
    1. *id1*
    2. *id2*
    3. *content*

```
<HEADER
        id1="0001"
        id2="0001"
        content="%PIX-&lt;hlevel&gt;-&lt;messageid&gt;: &lt;!payload&gt;" />
```

ID1 & ID2

- ID1 and ID2 are unique identifiers for a header.
- ID1 should be same as ID2.

The **BEST PRACTICE** is to use a 4 digit number as the ID beginning with "0001".

Content

- Content attribute defines Log format before the payload.
- The variables <*messageid*> and <*!payload*> must be present in the content attribute.

<messageid>:

Defines the location of an "event identifier".
Each unique value of this field becomes ID2 in the following message element.

Sample for messageid using STRCAT

```
<HEADER
        id1="0001"
        id2="0001"
        content="%McAfeeNAC-&lt;messageid&gt;: &lt;!payload&gt;"/>
```

The <*messageid*> can also be a concatenation of several strings when needed.
A function call STRCAT() is used to join the strings in this case.

Sample for messageid - Common Use case

```
<HEADER
   id1="0001"
    id2="0001"
    messageid="STRCAT(msgIdPart1, '_', msgIdPart2)"
    content="&lt;hmonth&gt; &lt;hday&gt; &lt;htime&gt; &lt;hfld1&gt;
,&lt;msgIdPart1&gt;, &lt;hfld2&gt;, &lt;hdate&gt; &lt;hdatetime&gt;,
&lt;msgIdPart2&gt;, &lt;!payload&gt;" />
```

<payload>:

Defines the section of the Log event which is passed on the the Message element for further parsing.
It can be a section of the Log event after the <messageid> or it can even be a section before which is defined using a "Payload rewind" specifier as shown below.

Sample for messageid - Using Payload rewind

```
<HEADER
            id1="0001"
            id2="0001"
            content="%reconnex:
&lt;hdatetime&gt;^^&lt;id&gt;^^&lt;hostname&gt;^^&lt;haddress&gt;^^&lt;fl
d4&gt;^^&lt;fld5&gt;^^&lt;messageid&gt;: &lt;!payload:hostname&gt;" />
```

Message ID:

- Look for *unique 'Identifier field'* in the Log events for a particular event source which has a *finite set of values* throughout all the occurrences in the log events.
- The logic behind choosing the message ID is to group events together and analyze *which value makes each event unique.* Preferably, the unique value will be located *in the same place* for *each* event.
- The message ID that is referenced in the Header is registered as *'id2'* field in the message definition or '*vid*'.
- Some commonly found useful fields could be 'an event type', 'action meta are unique to a set of events and/or have a list defined by the vendor

  Note: A message ID value *cannot contain* spaces, periods, or commas

Bad choices for Message IDs:

Fields *which usually do not have a Finite Set of values* are not recommended to be used as message IDs.

Examples -

- Usernames,
- Hostnames,
- Event-ids (which are more of Serial numbers and not unique IDs associated with particular event type(s)),
- Process IDs (Mostly in Unix environments - these are again more of Serial numbers and not unique IDs associated with particular processes/daemons)
- Initial words of sentences

If such fields are used as Message IDs then that increases the chances of getting unknowns as every new variance of such fields will result in a new 'identifier' which is not been supported in the parser.

Best Practices:

- Try to define Header in such a way so as to cover maximum possible variations in the Log Format.
- Use as many Static tokens aka static words that **commonly** occur within the log formats that makes the Header strong and device discovery becomes more efficient.
- The more generic headers should always be placed *after* the specific ones.
- Parse the meta values in the header itself if there are certain fields that are of significant value and are *not* going to be parsed in the <payload>.

Message Element

Sample Message Definition

Each message element in the XML file has the following attributes:

- *Id1*
- *Id2*
- *Eventcategory*
- *Functions*
- *Content*

Sample ref: Cisco Ironport WSA parser

```
    <MESSAGE
        id1="CONNECT:01"
        id2="CONNECT"
        eventcategory="1204000000"

         functions="&lt;@domain:*URL($DOMAIN,url)&gt;&lt;@fqdn:*URL($FQDN,url)&gt;&lt;@
web_domain:*URL($FQDN,url)&gt;&lt;@web_root:*URL($ROOT,url)&gt;&lt;@web_ref_domain:*UR
L($DOMAIN,web_referer)&gt;&lt;@web_ref_page:*URL($PAGE,web_referer)&gt;&lt;@web_ref_qu
ery:*URL($QUERY,web_referer)&gt;&lt;@web_ref_root:*URL($ROOT,web_referer)&gt;&lt;@webp
age:*URL($PAGE,url)&gt;&lt;@event_time:*EVNTTIME($MSG,'%X',fld1)&gt;&lt;@msg:*PARMVAL(
$MSG)&gt;"
        content="&lt;fld1&gt;.&lt;fld2&gt; &lt;duration_string&gt; &lt;saddr&gt;
&lt;action&gt;/&lt;resultcode&gt; &lt;sbytes&gt; &lt;web_method&gt; &lt;url&gt;
&lt;username&gt; &lt;fld4&gt;/&lt;fld5&gt; &lt;content_type&gt; &lt;policyname&gt;
&lt;&lt;&lt;info&gt;&gt; &lt;fld7&gt; s-ip= &lt;daddr&gt; s-port= &lt;dport&gt;
webcat-code= { - | &lt;filter&gt; } cs-version= &lt;fld11&gt; cs-auth-group=
&lt;group_object&gt; c-port= &lt;sport&gt; cs-bytes= &lt;rbytes&gt; wbrs-score= { ns |
&lt;reputation_num&gt; } wbrs-threat-reason= &lt;result&gt; wbrs-threat-type=
&lt;category&gt; cs-user-agent= { - | &lt;user_agent&gt; } cs-referer= { - |
&lt;web_referer&gt; } cs-cookie= { - | &lt;web_cookie&gt; }" />
```

id2 (NW Key: <vid>)

- This is the <messageid> key that is identified in the Header.
- This *should match exactly* as that in the Log *including the Case Sensitivity* - else it can result in mis-parsing.
- This can be repeated to cover the variations for the SAME message id.

id1 (NW Key: <msg.id>)

- This is usually same as id2.
- But if there are variations/ different flavors for the same id2, then it should be appended with :01, :02, :03.... and so on.

Event Categories

The Event category should be assigned *based on that specific event*.

NOTE: Do not decide on event category based on the overall function of the event source.

EXAMPLE**:**

If it's a router and the event is about successful user authentication, use the event category for User Authentication and not related to Network Activity or Communication.

Content

- The meat of the parsing is handled in the Content tag.
- The Payload passed from the Header is parsed in this attribute by defining the Meta keys and functions.
- The idea is to parse all the 'Money Meta' as much as possible.
  - *IP Addresses*
  - *Usernames*
  - *Network Port numbers*
  - *Protocols*
  - *Web Services*

Fine Parsing:

Usage of Static tokes help in making the message definition strong and unique.

Example:

The words marked in brown would repeat hence can be kept static. While the values in green are variables and useful information which need to be captured into Meta keys.

- Jun 26 14:18:47 boe_cmsd[19640]: [ID 121500 user.info] The password for user**xyz123**(**#1398**) has expired.
- Jun 26 14:10:11 boe_cmsd[19640]: [ID 162767 user.notice] User**xyz123**(**#2066**) tried to change password.

Generic Parsing:

If some logs do not present much information then they can be parsed generically - meaning multiple fields can be parsed to few Meta keys creating buckets.

Example:

- The text highlighted in green can be parsed to a single Meta key (<<fld>) because it is not giving any security related information.
- %CISCOIPORTWSA-4: Mon Jul 6 12:30:54 2009 Info: **Begin Logfile**
- Aug 20 23:17:18 backup_cubs[5948]: [ID 702911 local3.info] **Status of clone-split is 0**

But in this case, these need to be ordered below the "more - specific" message definitions.

Functions

Functions are used for further processing the Meta values that are extracted from the payload.

- When using any form of function in a content string, it is **Best Practice** to place the functions in a Separate 'functions' attribute.
- These are evaluated from *Left to Right.*

```
id1="CONNECT:01"
id2="CONNECT"
functions="&lt;@msg:*PARMVAL($MSG)&gt;&lt;@event_time:*EVNTTIME($MSG,'%D/%B/%W:%N:%U:%
O',fld1)&gt;"
content =" ...................." />
```

NOTE:

- Every message should contain at least this function: "<@msg:*PARMVAL($MSG)>" to place the <payload> into the <msg> meta key.
- This way, if any data that is not parsed from the payload will at least be searchable via the <msg> meta key.
- Log Decoder throws an error if a function is referencing a meta that is not used in the <content> line.

INFO:

Refer to this link for more information various functors available in Log Decoder ----> Log Parsers Functors

Formatting Best Practices

- All Device XML files should be formatted by placing starting lines ("<") with no spacing, and XML contents (such as the content line) two tabs in.
- Messages should ideally be ordered alphabetically (by id2).

Choosing Meta Keys

Meta Keys from Table Map/ Master Variable Guide

- It is extremely important to use these Meta Keys *consistently* as this data is further used in for Threat Analytics such as Application Rules, Reports, ESA rules etc.
- All the keys must be chosen from the **Table Map XML**
- Log Decoder recognizes the keys that are specified in Table Map xml file only.
- All of these meta keys are described in the **Master Variables Guide**

Meta Keys not present in Table-map

- While creating the Device XML there may be fields which contain information, but do not map to any variables in the Table Map. In this situation, a variable named **"fld#"** can be used as a place holder where # is a number.
    - The data in <fld> meta is not stored anywhere on Log Decoder or shown on Investigation UI unless it is indexed.
    - Whenever a situation arises where an fld variable should be used, make sure you are **consistent!** If the same data appears in multiple messages, use the same fld# for that data
- Behind the scenes, *every* meta should be used from Master Variable Guide, or else Syntax Checker tool (described later) will throw errors.
- Back to the point! . If a different piece of data appears, use a different fld*#. Make sure to document the fld usage at the top of the XML file*, so that it is understood on our end what purpose the variable served (for when an XML needs an update). Note that even if the fld is used to capture junk info (a varying number of spaces, for example) consistency should be maintained and the variable use should be documented.

Unit Testing Log parser

NW Log player

This tool is used to inject/replay log events directly into the Log Decoder

- Prior to installing NWlogplayer, you may want to verify the dependency - Universal C Runtime: https://support.microsoft.com/en-us/kb/2999226

REST API

Log Decoder logs are a good source to track the parsing going on -

- Use REST API to connect to LD - http://<*Log Decoder IP address*>**:50102**/
- Go to http://<*Log Decoder IP address*>**:50102**/logs/config
- Set the following values for "*Log Levels (log.levels)* (*)" parameter

    info,audit,warning,failure,LogParse=debug|info|audit|warning|failure,Parse =debug|info|audit|warning|failure

ESI Tool

https://community.rsa.com/community/products/netwitness/blog/2017/04/24/rsa-netwitness-esi-10-beta-3

Syntax Checker

This tool is used to test the Parser XML syntax for any issues that might not work in the LD environment

NW Console

- This is installed along with NW Log player.
- This can also be used to debug any issues with Log parsing as it simulates a Log Decoder environment locally itself without actually connecting to a LD.