|  |
| --- |
| Title : Parsing Cases General Training |
| URL Name |
| **PART I of VI - Parsing Cases Overview**  **What this article is:**  While working as a Technical Support Engineer you will run into a variety of parsing questions and case types. This is the first Knowledge Article of six to address the topic. This article will help you to understand at a high level what parsing is, how regex is used in the product, and provide a basis of knowledge to follow the other case-specific Knowledge Articles in this group.  **What this article is NOT:**  - A comprehensive deep dive into parsing. We have that. See here:  <https://confluence.logrhythm.com/display/GS/DD+-+Data+Processor+-+MPE>  - A guide for new device requests. The confluence page on that can be found here and is straight forward.  <https://confluence.logrhythm.com/display/GS/MDI+Case+Handling#MDICaseHandling-NewDeviceRequest>  **PART I - A: What is regex and how is it utilized here at LogRhythm?**  Regex is short for “regular expression”. Based on a Turing definition it is not technically a language but you will hear from different sources that it is and is not. The important part of the definition as a Support Engineer would be: ***“Regular Expressions are a particular kind of*** [***formal***](http://en.wikipedia.org/wiki/Formal_language) ***grammar used to parse strings and other textual information”.*** See the example below as this is one of the most common regular expressions you will see here at LogRhythm.  ^.\*?:(?<tag1>(?<severity>\w+))>  The above regex sample is what LogRhythm refers to as a “syslog catch-all”. It is one of thousands of regex strings that we use to “parse” logs. Parsing in this context refers to the act of identifying logs and tabulating them into correct categories. This particular example is used to parse the first part of a syslog formatted log known as the header.  There is plenty more to learn about parsing but understanding how the catch-all functions will give you a great understanding. The name “catch-all” is literal. It suggests that it looks for something that all syslog formatted logs will have in common and can be used to identify a log as a potential match for that log parsing policy. The following log is an example from TrendMicro which outputs its log in Syslog format. The part of the log that is called “the header” can be found in green.  04 16 2021 14:20:06 10.71.4.124 <LOC3:INFO> CEF:0|Trend Micro|Trend Micro Web Security|3.3.1.2882|100000|Access Log|0|wrsScore=81 companyID=f25a4fcc-f918-4ea0-aede-5dd1717748d0 app=2 upStreamSize=1915 userDepartment=880 - FINANCE scanType=0 malwareType=0 httpTrans={"http\_req": {"headers": {"host": "r2---sn-p5qlsndr.c.2mdn.net:443", "proxy-connection": "keep-alive", "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.114 Safari/537.36"}, "host": "r2---sn-p5qlsndr.c.2mdn.net", "scheme": "https", "method": "CONNECT", "path": ""}, "ver": "1.1", "http\_response": {"headers": {"true-file-type": "0"}, "status\_code": -1}} malwareName= rt=Apr 16 2021 18:15:55 +0000 policyName=Allow all - [default] severity=65 filehash= logType=1 dst=172.217.135.39 appName=The Secure HyperText Transfer Protocol groupName= fname= adDomain=ads-pipe.com gatewayName=roaming user principalName=kristin.moyer@ads-pipe.com downStreamSize=3915798 profileName= userName=Kristin Moyer src=75.118.215.46 transportBytes=3917713 domainName=r2---sn-p5qlsndr.c.2mdn.net cat=Computers/Internet act=allow  Below in Screenshot 1 you are looking at MPERuleBuilder. This will be your home-base for parsing type questions. Using this tool, you will have access to all our log source parsing polices, regex strings, and test center to verify/refute customer claims regarding parsing performance. You can even build custom rules in here if that is an interest piece (but keep in mind that most custom parsing qualifies as Professional Services).  **Screenshot 1**  Graphical user interface, application  Description automatically generated  The major things to notice about screenshot 1 is that the log above has been manually queued up for testing. I did that by right clicking and loading an llx file. Notice how it appears in LogMsg and how test result states “no match” and how *matched rule* and *common event* are blank. These are blank because while I have uploaded the log I have not yet clicked test to have the regex go up against the log in order to parse it.  You can also see that I selected the appropriate log parsing policy using the yellow folder in the top left corner where I went to “Trend Micro” and onto “Catch-All Level 1”. This loads the regex into the middle box leaving you from top to bottom: parsing policy, information, actual regex, and test center.  **Screenshot 2**    Notice how screenshot 2 differs from screenshot 1. You can see that the *TestResult* is now “match” and that *MatchedRule* and *Common Event* are now populated with “General Information”.  This isn’t a ton of information but obtaining a ton of information is not the point of a “catch-all”. The idea of a catch-all is to use as little information as possible to identify it as a genuine log so we can then build and subject it to more complex regex that can parse out more information. More complex regex will be included later in the series.  A great example of “more-complex” in terms of catch-all would be the catch-all rules that are used in the parsing of Windows logs. Take for example the MS Event XML – System Policy. The below log is a typical MS Event XML – System log.  **<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-WAS' Guid='{524b5d04-133c-4a62-8362-64e8edb9ce40}' EventSourceName='WAS'/><EventID Qualifiers='16384'>5186</EventID><Version>0</Version><Level></Level><Task>None</Task><Opcode></Opcode><Keywords></Keywords><TimeCreated SystemTime='2021-04-27T14:19:15.000000000Z'/><EventRecordID>713295</EventRecordID><Correlation/><Execution ProcessID='0' ThreadID='0'/><Channel>System</Channel><Computer>BG-WSUS-01.southcentralbank.biz</Computer><Security/></System><EventData><Data Name='ProcessID'>17172</Data><Data Name='AppPoolID'>DefaultAppPool</Data><Data Name='Minutes'>20</Data></EventData></Event>**  We start with the same process as we used in screenshot 2 with selecting the yellow folder, electing the correct policy, and selecting the correct rule. You can see in screenshot that I have selected the “Catch-All Level 2” rule for the MS Event XML – System Policy and uploaded the same log example from above.  **Screenshot 3**  **Graphical user interface, text  Description automatically generated**  It is clear in screenshot 3 that the Catch-All Level 2 on the MS Event XML – System Policy is quite a bit more complex than the standard catch-all seen in policies serving syslog formatted logs. You can see that *TestResult* = Match, *MatchedRule* = Catch All Level 2 , *CommonEvent* = General Information, *VMID* = , *DName* = glsp01.southcentralbank.biz , *Session* = 0 , *Process* = Microsoft-Windows-WAS, and *ProcessID =* 0  Using the MPERuleBuilder to test catch-all rules is a great way to understand the fundamentals of parsing. You can see in the two regex strings in screenshots 1 & 2 that the regex is essentially taking the important information out and placing it where we want to place it. Once you can grasp this concept you are ready to move onto actual troubleshooting when the intended functionality of this process breaks down. |