|  |
| --- |
| Title : Parsing Defects/Parsing Improvements: Where To Start When You Don’t Know |
| URL Name |
| Identification: This article is written to help the average Technical Support Engineer work their way through a standard variety parsing defect and or parsing improvement case.  **What this article is:**  - Simple steps that you can self-address to resolve basic parsing improvement or parsing defect cases.  -A brief explanation as to why these are the steps and how parsing works in layman’s terms.  **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>  - A guaranteed solution for every parsing issue that you will come across. These are basics to help wrap your mind around the problem you are solving.  ---------------------------  The two most common ways that you will run into parsing improvement and parsing defect cases will be:  - a customer directly identifying a parsing error (i.e. a case saying “look at this parsing issue”)  - while troubleshooting other items such as AIE, Alarms, or GLPRs you might find that KBs have shifted how items parsed for a particular rule can end up making changes on the macro performance level.  **PART I : What is regex and how is it utilized here at LogRhythm?**  Regex is short for “regular expression”. Based on a Turing definition it is a not language but you will hear from different sources that it is and is not. To do you job just understand this: ***“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 is what LogRhythm refers to as a “syslog catch-all”. The term “catch-all” is literal in that it is designed to read a the first bit of a log (known as the header) to determine if it can parse any general information. The part of the log that is called “the header” can be found on the next page in green.  There is plenty more to learn about parsing but understanding how the catch-all functions will give you a great understanding. The expression above when applied to the entirety of the sample log is basically saying “I am looking for logs in this format…if they are in this format then they should be syslog”. The following log is an example from TrendMicro which outputs its log in Syslog format.  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  **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. 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.  **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 in all reality isn’t a ton of information but that is the point of a “catch-all”. The idea is to catch it, identify it as a genuine log so we can then build and subject it to more complex regex that can parse out more information.  A general understanding of the syslog catch-all helps because it is quite simple. If we apply that concept to a more complex catch-all such as the one used in the MS Event XML – System Policy; we can see how additonal information can be parsed out of a log. See the below log. This is an example of 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>  **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 syslog catch all seen in many other policies. 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  The understanding of the 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. . . we refer to this as “parsing”.  In screenshot 4 on the next page you will see the same MS Event XML – System Policy log sample from screenshots 2 & 3. The difference is that I have intentionally modified the regex for the catch all rule to show you what it looks like when a parsing rule is a “defect”.  **PART II : I understanding parsing but how will that manifest in my day at LogRhythm?**  Here at LogRhythm you will run into four main categories of parsing challenges and cases for troubleshooting. They are: parsing defect, parsing improvement, feature request, and host inference token. It will be important to note that feature requests and host inference tokens will NOT be covered by this article. Feature requests are straight forward directions from the confluence page linked below. Host Inference Tokens are a beast in and of themselves and have an appropriate KA for troubleshooting. See links below:  [**https://confluence.logrhythm.com/display/GS/MDI+Case+Handling#MDICaseHandling-NewDeviceRequest**](https://confluence.logrhythm.com/display/GS/MDI+Case+Handling#MDICaseHandling-NewDeviceRequest)  [**https://logrhythm.my.salesforce.com/lightning/articles/Knowledge/AI-Engine-Host-Inference-Token-Corruption-Troubleshooting**](https://logrhythm.my.salesforce.com/lightning/articles/Knowledge/AI-Engine-Host-Inference-Token-Corruption-Troubleshooting)  The contents of this section will be split in section II-A, II-B, and II-C. II-A will explain a parsing defect while sections II-B/C will model the different types of parsing “improvement” cases. The “improvement” is placed in quotes so as to highlight the factual difference between these types of cases which can often be left up to the imagination to define. I’ll give you some guidelines below.  **PART II-A: Parsing Defects.**  Whether you understood the basics of regex or read above there will be a major action-item in your role as a TSE: parsing defects. It should be noted that a Parsing “Defect” is very narrowly defined by our MDI Team here at LogRhythm. A good solid one sentence question to ask yourself for the defect/improvement question is: ***“do I have one or more field parsed into the INCORRECT field”?*** If the answer is “yes” you have a defect. If the answer is “no” you more than likely have an improvement. See below.  The rationale for this narrow definition is because it needs to be sorted out if a parsing rule is defective or if the regex wasn’t originally written to parse that field. This often leads to confusion because without an understanding of regex syntax it can be challenging to determine if the rule is in fact defective. If you can *identify that the regex syntax SHOULD be parsing out a field that it is missing* this would likewise be a defect. If you don’t have a deep-enough knowledge of regex syntax your safest bet will be to elevate as a Parsing Improvement. A member of the MDI team will be able to identify it as a defect. Short version…***you will see more improvements than defects.***  Definitions are great but what does an actual parsing defect look like? Screenshot 4 can be found on the following page. You will notice that it is the same Catch All : Level 2 from screenshots 2 and 3 in Part I. The difference in screenshot 4 is that I intentionally re-wrote part of the regex to simulate a defect. In other words, I intentionally broke it.  ***In screenshot 4 you can notice that I switched dname and session in the regex***. This is good to re-highlight that the regex isn’t magic. In its correct form it says “the information for *dname* will be here” and “the information for *session* will be there”. The system simply parses the information at those locations into those categories. When I switched them you can see that I was able to parse glsp01.southcentralbank.biz into the session category. This is clearly incorrect and ***defective.***  **Screenshot 4**  Graphical user interface, text  Description automatically generated  A parsing defect is what happens when for a variety of reasons, the regex in a particular LogRhythm MPE Policy is defective. This could be because a vendor changed their logs which would make our regex outdated. It could be an oversight when the rule was originally written. It could even be a customer not using the default policy. The short of what makes a defect over improvement is asking the question: is something “wrong” or does something need “improvement”? Below you will see an example of how this can manifest in your troubleshooting.  When you see this in the wild your steps will be to access the directions on Confluence. Pro-Tip: bookmark the page but access Confluence each time. DO NOT screenshot those instructions. They change more frequently than the forecast in Colorado. Reading from Confluence will ensure you are doing it correctly. Below I have laid out step by step how I would work this case if it were in my queue.   1. Inform customer I am working on the case and ensure that I have LLX, CSV, SCMPE, and LPS Detail. 2. Open Rule Builder in Super-User Mode (ask a teammate about Super-User if confused). 3. Run the LLX against the entire policy to identify which rule in the policy these logs align with if that information is not already know. 4. Run the LLX against that specific rule. Analyze the results in your rule builder against those in the customer’s CSV. (\*CSVs will show how a log parsed on the customer side…LLX is a secure transmission of log raw data that can be parsed and re-parsed). 5. In this case I would see glsp01.southcentralbank.biz parsed to “session” on my rule builder as well as on the CSV. This would conclude as a “parsing defect”.   The next step will be to complete the template provided on Confluence for Parsing Defect. The template at the time of authorship on 24 Jan is shown below. You will notice that the template covers classification and common event. You can add what you need to. **REMEMBER** to always check Confluence to make sure no changes have been made to this template in an elevation analysis:  **Parsing Defect - Misclassified/Wrong Common Event : It seems that a user is being parsed into the session category incorrectly.**  Existing customer or proof of concept? Existing Customer  Complete log source type name: MS Windows Event Logging XML – System  Is the customer on the latest KB? If not, why not? Yes, Version 7.1.630  Does the logging format adhere to our standards (e.g., the correct IIS fields are enabled, etc.)? Yes  Which base-rule is affected by the wrong classification? Catch All: Level 2  MPERuleRegexID: 1008164  Actual classification: Classification is fine….user being parsed to session.  Actual common event: Common event is fine…user being parsed to session.  Desired classification: parse this user correctly to Dname.  Desired common event: parse this user correctly to Dname.  \*You can and I normally add some prose at the bottom such as: Hey MDI Team. I can see on this case that the dName is Session seemed to be mixed up. I looked at the regex and my best thought is that the regex in fact looks defective because it is showing that the computer should parse into session which doesn’t look correct. I tested the logs with Rule Builder on my end and they match the customer’s CSV. Elevating to you for further troubleshooting.  At this point you would complete the process by informing the customer and elevating the case as the instruction on Confluence indicate.  **PART II-A: Parsing Improvement: Unprocessed Log Queue Growing**  When it comes to MDI/Parsing cases you will see a variety of causes. On this topic you might here a customer concern along the lines of “help, my unprocessed log queue is growing”. It will first be important to double check that the mediator is functioning. Ensure that the service is on and that the spooling is occuring into S:\Program Files\LogRhythm\LogRhythm Mediator Server\state\UnprocessedLogs directory. That might live on another drive but if the logs are not spooling here then you have a different problem on your hands.  A great example of how to work this case was 432574 by Grant Hogan. You can see that on a call he confirmed logs were spooling to that directory and then he went back to get a historical view of when unprocessed logs started to climb. Take a look at screenshot 5 from his case.  **Screenshot 5**  Chart  Description automatically generated  It should be pretty simple to see that starting on 15 December there was a massive jump in unprocessed logs. Less clear but very much in the picture is the height of the bars themselves. It’s always important to take log volume into consideration. In this example log volume is relatively unchanged so we can rule that out as part of the analysis/solution. The next step will be to figure out which of these logs are exactly “unidentified”. See screenshot 6 below  **Screenshot 6**  **Text, timeline  Description automatically generated**  The above screenshot is how you would configure an investigation to find unidentified logs. One thing to keep in mind about processing and mediator flows is that what you are seeing are logs that have been indexed. That is to say they have been indexed without a Classification/Common Event. This occurs after the 2052 time out error is created to alert us of challenges with processing. If you have spooled logs in the state directory listed above it will be important to note that those ARE NOT represented in this data. Those have yet to error out and parse as blank classifications or common events. You can see exactly how they parse in screenshot 7 which is from Grant’s case. He would have gotten this by doing the above investigation, going to log viewer, and double clicking on a log.  **Screenshot 7**  Graphical user interface, text, application, Word  Description automatically generated  Notice how there is a full log message, but classification and common event are missing. The next step for this type of scenario is to replicate it on your VM. Before you get off the call with the customer make sure that you have done both “export all logs” and “send all logs”. One will create a CSV and one will create an LLX. You may want both of them in your troubleshooting. If you haven’t gathered the scmpe.log or LPS\_Detail.log then you will want to do that as well.  You will want to open client console in SuperUser mode. Open MPE Rule-Builder and use the yellow folder to select the proper policy. In this case you should be selecting Syslog-Fortinet Fortigate v6.0. The important part here will be to select the “test all” button. Don’t worry, the screen will click but otherwise will look as if you made no changes. This is what happens while test a whole policy.  **Screenshot 8**  Graphical user interface, application, table  Description automatically generated  Once you have loaded the policy you will want to load in your llx. See screenshots 9 + 10 below. If you did it correctly it should look like that.  **Screenshot 9**  Graphical user interface, text, application  Description automatically generated  **Screenshot 10**  Graphical user interface, application, table  Description automatically generated  Next step is to actually test it. Click the button that says, “test all”. You will notice MPE Rule Builder processing for a minute (or several) to which it will produce a screenshot like the one from Grant’s case.    **Screenshot 11**  Graphical user interface, application, table, Excel  Description automatically generated  The most important thing to observe in the results above is how the llx sample did not hit on any of the parsing rules except for the catch-all level 3. You can see this in the red box. We would have expected these logs to hit on the second to last rule called “UTM : SSL Message”. The regex of that rule clearly did not catch them and connecting to the lesson of catch-alls above we can understand that this log will be identified but in need of improvement.  Once you have identifed that the logs are all falling to the catch-all your next step will be to determine which rule they should have belong to. Keep in mind that this is going above and beyond but it is a best practice because it will teach you more about MPE and help you better understand LogRhythm as a whole. You can identify logs by studying them and asking the customer. An example of the log in Grant’s case looks like this:  12 16 2021 06:02:55 10.21.7.6 <LOC7:NOTE> date=2021-12-16 time=06:02:54 devname="s5652a" devid="FGT6HD3916804855" eventtime=1639634575050932732 tz="+0000" logid="1700062302" type="utm" subtype="ssl" eventtype="ssl-anomalies" level="notice" vd="s5654a" action="resign-as-untrusted" policyid=1034 sessionid=1062819319 service="HTTPS" user="U115969" group="Domain-Users" profile="SSL\_Inspection\_HTTPS" srcip=172.27.69.211 srcport=58744 dstip=20.54.89.106 dstport=443 srcintf="port12" srcintfrole="undefined" dstintf="port13" dstintfrole="undefined" proto=6 eventsubtype="certificate-anomaly" msg="Server certificate is re-signed as untrusted, certificate-status: untrusted." hostname="slscr.update.microsoft.com"  The log above has a key part highlighted in yellow. There are only so many rules where that can be match. SSL messages would be an appropriate hypothesis. Often you can simply ask the customer and system admins will know their logs better than you will know. They’ll tell you what it is.  You can double check the parsing of the llx against the singular rule directly. This is a great thing to check because it can help determine if there is something wrong with the policy or the rule. This is all information that is helpful to the MDI team. You do this by loading the individual rule in and click test. You’ll get something like screenshot 12 below:  **Screenshot 12**  Graphical user interface, text, application  Description automatically generated  Notice how in screenshot 12 that none of the fields parse. This screenshot also gives MDI an idea of timing for this log processing. With this information your next step will be to complete the template for MDI-Elevation. Grant’s completed template can be found below. Template is in **purple** while Grant’s responses are in **green**.  **Parsing Improvement**    **Existing customer or proof of concept?** Customer    **Current Log Processing Policy Version: (Required)**  LogRhythm Default  **Use case or justification for the request (Required)**   The customer has been seeing numerous regex timeouts for the Syslog – Fortinet Fortigate v6 log source.  Upon investigating this we could see that there was a number of unidentified logs coming into the deployment.  The issue started on 16/12 which was just a day after the 7.1.628 KB was released so it appears the KB update has changed how this was parsed.  When replicating this issue in my lab I could see I was on KB version 7.1.625 so I tested it with version and could see the logs were being matched on a MPE rule called ‘Catch All : Level 3’.   I then updated my KB to 7.1.628 to match the customers deployment and could see that the logs didn’t match on any of the MPE rules. I could also see that the ‘Catch All : Level 3’ from KB version 7.1.625 was no longer present.  **MPERuleRegexID to be improved?**   The logs are not being hit by any MPE rule and I no longer have the regex ID from the catch all rule from the 7.1.625 KB.  **The customer MUST be on the latest version before we can elevate to MDI.**  **Ensure the KB Version field is populated.** 7.1.628    **Does the logging format adhere to our standards (e.g., the correct IIS fields are enabled, etc.)?** Yes    **Actual parsing behavior:** Screenshots shown above.    **Desired parsing behavior:** For logs to be parsed and not come through as unidentified.     * Log samples are required * Ensure log samples only contain logs that need the parsing improvement (in investigator, filter for non-matched logs > right click > send selected logs)   Log samples are attached in the case    HAVE YOU ENTERED THE LOGSOURCE TYPE INTO THE SUPPORT CASE? Yes  **You would then follow the SOP set at this link which would have you move it into the appropriate MDI queue.**  <https://confluence.logrhythm.com/display/GS/MDI+Case+Handling#MDICaseHandling-ParsingImprovementRequest>  **PART II-B: Rule Not Parsing Field Improvement**  Part II-A outlined one of the more common “help me I’m in trouble scenarios”. You will see that more often because it’s rather difficult to miss when your mediator is spooling. However, that is not the ONLY way that parsing can take a nosedive for the worse. Another common type of elevation is when a rule works but a particular field is left blank. This can also blend with the parsing defect example above where perhaps the field is over or under-parsing a particular field. These cases will usually come from astute customers who are very observant of their traffic and/or sales engineers here at LogRhythm. In either case it is critical to stress that these types of cases you will have the benefit of the customer end being able to explain the challenge in depth. Get on a call to diagnose.  Case 435794 submitted by Sales Engineer Brandon Pace here at LogRhythm is a fantastic example of this type of case. In this particular case there was no call needed because, to the point above, Brandon explained concisely in the ticket. See what Brandon wrote in the ticket below:  Case 435794: *I noticed during a demo today that "Flat File - Cisco Umbrella DNS" no longer parses out the URL/Domain info into the URL field (or any field). When was this change done and was there are reason for this, or was it a mistake? Looks like the "Group" field could use some attention as well:*  *Group example*  *Gregory Pallet,PSB-WIN-ECHO,Default Site*  Based on the above there are up to four steps to completing this case successfully.   1. Check that llx (or csv…but llx preferred) is attached to case. 2. Engage with customer for additional information (skipped here because all info was present) 3. Using the same methods as earlier sections you will want to test the llx against the appropriate policy as a whole. 4. Once the rule you need to test is identifed you will want to confirm/refute what the customer is claiming by testing against that rule specifically.   ***\* Should you test the log sample and receive different results than the customer you might want to check mediator configurations, LPS Detail (other log sources messing up), custom policies, or KB version to name a few things.***  Here are two log samples from the LLX presented in case 435794. Notice the URLs and Groups as pointed out by Brandon in the case summary.  "2018-04-02 17:23:15","Jake Reynolds","Jake Reynolds,PSB-WIN-MGMT,Default Site,PM Sandbox,Pub NAT","192.168.51.127","65.127.112.131","Allowed","1 (A)","NOERROR","s3.us-west-2.amazonaws.com.",""  "2018-04-02 15:07:13","Manish Bhatnagar","Manish Bhatnagar,PSB-STL-AS1,Default Site","192.168.135.221","54.183.86.198","Allowed","1 (A)","NOERROR","lse.ac.uk.","Educational Institutions"  **Screenshot 13**  Graphical user interface, application, table, Excel  Description automatically generated  **Screenshot 14**  Graphical user interface, application  Description automatically generated  Screenshot 13 is the embodiment of steps 1-3 above. We had the llx on the case and the case summary had enough details to understand the identified problem (steps 1-2). We then logged in as a Super User, opened MPE Rule Builder, and selected the Umbrella Flat File Policy with the “test-all” function. We loaded the llx and when we tested the logs where we can see that 100% of these logs identified as Umbrella DNS Requests.  Side Question: How does it know it is Umbrella DNS?  Side Answer: This is what differentiates parsing improvement type in II-A from type II-B. It’s not the policy that needs improvement but rather a particular rule. The regex within a rule can often work to identify a log based on its syntax. It is the difference between “identifying” and “identifying well”. It should also be noted again that II-A or II-B; these are names for this article. In the field it is all simply “parsing improvement”.  Screenshot 14 models the step 4 above and follows a similar procedure except in this round we test against a single rule in Rule Builder we can see how it actually parses. You can visibly identify the difference because while testing whole policies you will see rule builder stay fairly blank (minus the snap) and a singular rule you can see the regex populate. The idea here is to match what the customer describes OR confirm that the policy is fine and that the customer has something else going on (see \* above).  In screenshot 14 you can see that the issue Brandon identified also appears on my VM here that is running KB 7.1.631. This tells us that it is NOT the customer environment but rather the regex inside the rule that could do with some improvement.  We would again use the parsing improvement template which I have complete an example of here:  **Parsing Improvement**    **Existing customer or proof of concept? Proof of Concept.**    **Current Log Processing Policy Version: (Required)**  **LogRhythm Default**    **Use case or justification for the request (Required)**  **Sales was modelling this log parsing policy and one of the key points is to have URLs parsed out to (theoretically the object field…but parsed in general). It doesn't look good when we say look at this and then it doesn't work.**    **MPERuleRegexID to be improved? 1009127**    **The customer MUST be on the latest version before we can elevate to MDI.**  **7.1.631**    **Does the logging format adhere to our standards (e.g., the correct IIS fields are enabled, etc.)?**    **Yes**  **Actual parsing behavior: URL does not parse….Group category looks like it occasionally parses too much (secondary concern)**    **Desired parsing behavior: URL to be parsed into object and perhaps a clean up of how group is parsed.**     * Log samples are required - Included   HAVE YOU ENTERED THE LOGSOURCE TYPE INTO THE SUPPORT CASE? Yes  **You would then follow the SOP set at this link which would have you move it into the appropriate MDI queue.**  <https://confluence.logrhythm.com/display/GS/MDI+Case+Handling#MDICaseHandling-ParsingImprovementRequest>  **PART II – C Parsing Improvement: Parsing Improvement Rule Time Performance**  Part II-C is dedicated to understanding 2052 rule time outs and what to do for them during a support case. We will case 427933 handled by TSE Dominic Mejia as a model of how to do this type of case and insert important information that he would have had just from experience while resolving.  To start this work you will need:  - scmpe.log found in:  -lps\_detail.log found in:  -log samples of timed out logs in llx format  -log samples of timed out logs in csv format  \*Note – The “need” for .csv is relative. It is a nice thing to have and if you can include it please do. If unavailable or if it turns into “for the sake of collection” do not worry about it. LLX can be made to work.  The deep dive covers the history and functionality of timeouts here:  <https://confluence.logrhythm.com/display/GS/DD+-+Data+Processor+-+MPE>  Because the deep dive covers what 2052 timeouts are and their history in the product we will not focus this knowledge article on “what they are” as much as we will discuss “what to do about them”. A one sentence summary of what you can find in the deep dive would look like this: “2052 timeout errors occur when a particular log has been passed by a specific MPERuleID for more than the default 100ms and as such did not parse correctly”. With that knowledge let’s jump into case 427933:  It should be noted that an email went out from Principal Technical Support Engineer Sean Matthews some time ago regarding Support’s policy regarding the increase of timeout in policy. See below:  *Sending this wide and far as this setting is being increased globally.*  *The MPERuleTimeout value is set to 100MS per log processing policy by default, this is a very generous value. You should not be increasing this unless you are extremely sure it needs to be increased in order to parse critical logs.*  *Typically a good performing MPE rule will take <40MS to parse logs against it, normally less than this. So 100MS is a very generous amount of time.*  *Increasing this timeout value is going to have a severe impact on the deployment. Do Not go and increase it just because you see timeouts in the scmpe.log, timeouts are good, that is what the setting is there for, to speed up the processing speed. By increasing this timeout you are basically giving it longer to timeout each individual MPE rule which in turn makes performance of the DP much worse.*  *Should you feel the need to increase the MPERuleTimeout setting for a Log Processing Policy, you should be raising a case with MDI to improve the performance of the MPE rules, with the intention to turn it back to the default 100MS.*  *I am currently seeing many deployments with this set to 400MS or even up to 1000MS for log processing policies, this is criminal, you risk slowing down processing performance on the Data Processors, which in turn creates an unprocessed logs queue, which in turn means a delay in processing and potential risk of logs not being observed against AIE use cases.*  *I cannot stress this enough, do not increase this value unless you are 100% sure you absolutely need to and you have a plan to bring it back down to 100MS.*  **Action item from that email:** if you find a need to increase the timeout beyond 100ms we should be ultimately asking why. This may include an analysis of customer logs and/or elevating the case to MDI to improve the regex if the use-case can be recreated in a lab.  You can see a great example of this type of case work with case 427933 worked by TSE Dominic Mejia. You can see that this case was a follow up to an original case 403079. The idea here is that the customer was continuing to see 2052 Timeout Warnings despite MDI informing them that the fix had been published on 25 June 2021 with project P-0012294.  Dominic started this case review by confirming the customer concern and that there were indeed timeouts in the scmpe.log. There are two ways to do this. You can open the scmpe.log in Notepad ++ and read through the 2052 warning OR you can use the LogAnalyzer tool to gain further insight. You can accomplish the mission with either of these tools but a best practice in my experience is to use them both in conjunction. This example will start with the actual log analysis of the scmpe.log and continue into how the LogAnalyzer tool can be combined to further solve the case. See an example 2052 from Dominic’s case below:  12/29/2021 03:09:20.182634 [CBCLOGRHYTHM-S1] \*\*\*WARNING\*\*\* EVID=2052. MESG=Regex rule match timed out. MsgSourceId = 5372, MPERuleRegexID = 1008299, Base MPERuleID = 1409614, Rule Name = Network Traffic, KB Version = 7.1.628.0, Timeout = 700ms, MsgSourceTypeID = 1000546, PolicyID = -1000546.  While it might be simple reading it can help to have the anatomy of this log defined. The first line explains itself and we arrive at MsgSourceID= 5372. This should correlate to the LogSourceID on the Log Sources tab to identify which log source had a log that timed out. You can next see the MPERuleRegexID =1008299 as well as the BaseMPERuleID =1409614. These numbers are crucial to know because this is how we identify which rule within the policy is timing out. It won’t be relevant to this case but an additional detail to include regarding the difference between MPERuleRegexID and BaseMPERuleID. At a certain point you will learn to identify when sub-rules are the challenge. MPERuleRegexID can point out a RegexID pattern whereas the BaseMPERuleID will help you in determining if a sub-rule is at play. That said, we digress to the rest of the log.  The other parts of this log are general information which may help troubleshooting. It states the rule’s name in plain text as well as the KB Version. The KB version is critical because MDI releases new KBs every Monday. If a customer is on an older version you might be literally re-inventing the wheel for a problem that has already been fixed. Within the parsing policy itself this is the actual rule that experienced the timeout.  In addition to that information you can find how long the timeout is configured for, MsgSourceTypeID, and PolicyID. The timeout is helpful because it defines whether you are working with a default or tuned rule. The MsgSourceTypeID and PolicyID numbers actually align with Message Sources and Policy IDs inside the EMDB in SQL. It is unlikely you will need that information but it is certainly fun to know if you’re trying to learn how LogRhythm functions on a deeper level.  As discussed above you can also use the LogAnalyzer tool that will give you a perspective of timeouts on a LogRhythm Deployment. A brief explanation of the LogAnalyzer tool will be provided here although there is a separate KA for using this tool. See screenshot 15 and the following explanation to see how it fits into Dominic’s case:  **Screenshot 15**  Table  Description automatically generated with medium confidence  The LogAnalyzer helps by telling you how many 2052 time outs exist and on which rules they exist. On Dominic’s case we can see that the top four MPERuleRegexIDs that are experiencing timeouts were 1010769, 1010505, 1007793, and 1008299. For this case and in this article we will be focusing on MPERuleRegexID 1008299 as it is associated with the Zscaler nano policy. The other rules are not part of this particular case but if you are interested in learning how to read this output in its entirety check out it’s cousin Knowledge Article [Using LogAnalyzer For SCMPE and MDI].  MPERuleRegexID 1008299 is in the 4th position with 13 timeouts. Rule 1008299 is the Network Traffic Rule on the Nano Zscaler policy. Now that we have a specific rule that we are looking at we will want determine how impactful that 13 really is. If you remember back in the scmpe.log’s actual text that the timeout configuration was set to 700? This means that even with 700ms to process there are still 13 timeouts in the log for that time-period. That is bad. In fact, that is horrible. It should be noted that increasing the timeout a little bit can be helpful tuning. If you do this you should note the email from Sean Matthews (former Principal Engineer/Current SRE) regarding MDI elevations. Short version is that if you change the configuration for timeouts beyond 100 then you should be consulting with MPE experts on your team whether the case merits going to MDI. Changing it and calling it resolved is applying a band aid and not helping LogRhythm maintain a leading edge.  If you have made it this far then you are most likely evaluating a case for elevation and trying to work out the exact details to send to MDI. This is where you will want to start analyzing the lps\_detail.log. Below you can find where to start with this task.  The first step with the lps\_detail.log is Notepadd ++. Once it is open you will want to scroll all the way to the bottom. Click “search” and check “backwards direction” and type in the appropriate name for the log source. In this case Dominic would have typed in “zscaler”. It is important to note this because the lps\_detail runs in cycles. You will find many entries for zscaler and by doing this you direct your attention to the most recent. You could also simply use the log called lps\_detail snapshot and avoid that entire process.  Once you have done the above you should be looking at something that resembles screenshot 16.  **Screenshot 16**  Table  Description automatically generated  Screenshots 16 shows us a lot of information. The first thing we want to look at per the deep dive in an LPS\_Detail is the Total Compares and the LPS-Policy-Total. Read the deep dive for depth but a great line in there was : *A general rule of thumb is to look for a LPS-Policy-Total that is below 100 for logs that are greater than 10,000 compares.* Another way of looking at that same math is 1% or .01.  We can see that autosorting is not a problem with this Zscaler policy as the “A” in the third column has moved the MPERegexID 1008299 to the very top. This in turn though has created a bottleneck in some ways. Bottlenecks can appear anywhere in the policy and our role in support is to identify where they exist; MDI will fix them. In this case we can identify the difference between “bottle neck” and “properly sorted autosort rule being at the top” because it is sorted at the top and the ratio of total compares and LPS\_total is not within that 100:10,000 provided by engineering. Keep in mind that 1% of 971,311 is 9,713. Our LPS-Policy-Total is currently at 1,962. This math should be considered as the deep dive suggests “a rule of thumb”. It is not completely accurate. If you are getting numbers here in the 6,7, or 8,000 range then it’d probably be a volume thing. In this case though we can say that 1,962 is for sure quite low.  ***This is also supported by the scmpe.log which was showing timeouts with the policy set to 700ms time out.***  The next steps for this particular case are set: an elevation to MDI was needed and that is what Dominic pursued. It might also be helpful to harvest some more fun facts out of screenshot 16. You can see that after the initial 971,311 logs parsed that there were 6 logs left that did not match that first rule. These then fall to the next rule and the next until they match. Ou can see that some matched on the catch all level 4 and the rest on the catch all level 1. If logs make it to the bottom and have not been sorted that is where we get unclassified logs and a return to the other types of parsing cases in this KA series.  **NEXT STEPS AND ACTION ITEMS WHEN YOU ARE IN THIS POSITION**  **Parsing Improvement**    **Existing customer or proof of concept? Existing Customer**    **Current Log Processing Policy Version: (Required)**  **LogRhythm Default**    **Use case or justification for the request (Required)**  **Customer is seeing consistent timeouts on Syslog – Zscaler Nano Streaming Service. Time Out is currently configured to 700 MS and timeouts continue. In addition LPS-Policy-Total and Compares are about ¼ of what the should be according to the deep dive.**    **MPERuleRegexID to be improved? 1008299**    **The customer MUST be on the latest version before we can elevate to MDI.**  **7.1.621**  **Does the logging format adhere to our standards (e.g., the correct IIS fields are enabled, etc.)?**    **Yes**  **Actual parsing behavior: Parsing is slow enough to create timeouts.**    **Desired parsing behavior: Faster parsing to improve timeouts.**     * Log samples are required - Included   HAVE YOU ENTERED THE LOGSOURCE TYPE INTO THE SUPPORT CASE? Yes  **You would then follow the SOP set at this link which would have you move it into the appropriate MDI queue.**  <https://confluence.logrhythm.com/display/GS/MDI+Case+Handling#MDICaseHandling-ParsingImprovementRequest>  -------  Side Note After Case Was Elevated  This would be out of the realm of support but as a piece of interest as to the question: what happens when it leaves my queue?  You can follow the case progress and you can see that MDI first tackled this with project (P-0012679) and then again with P-0012731. The second of those projects seemed to be a permanent fix according to the customer. |
| Remediation Process |
| Root Cause |
| Additonal Resources  Make sure to add some common regex stuffs in here like what common symbols there are  Add regex 1010  Add a tutorial about windows event logging |
| Tags |