Software Log Analysis

By

Dr. Madani

Winter 2024

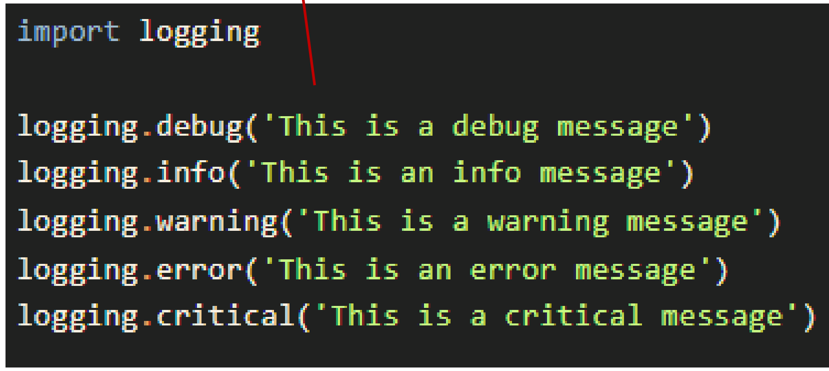
Learning Objectives

Upon completion of this material, you should be able to

- Understand the importance of software logging and log analysis.
- Describe the difference between log mining and log analysis.
- Role of anomaly detection and machine learning in log mining.
- Describe different elements of an automatic log mining system.

Intro

Example of some logging statements.



```
import logging

logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

- What's logging?
  - refers to the practice of recording and storing information about the behavior of a software application or system at runtime. This information, known as logs or log data, can include various types of data such as errors, warnings, debug information, user actions, and system events.
  - Semi-structured text data:
    - Contains some pre-defined format (e.g., timestamp) and some free-text form (e.g., context explanation).

- Why logging?
  - typically used to help diagnose and troubleshoot issues that occur within an application or system, as well as to monitor performance, track usage patterns, and analyze trends. They can also be used for auditing and compliance purposes, as well as for generating reports and other types of analytics.

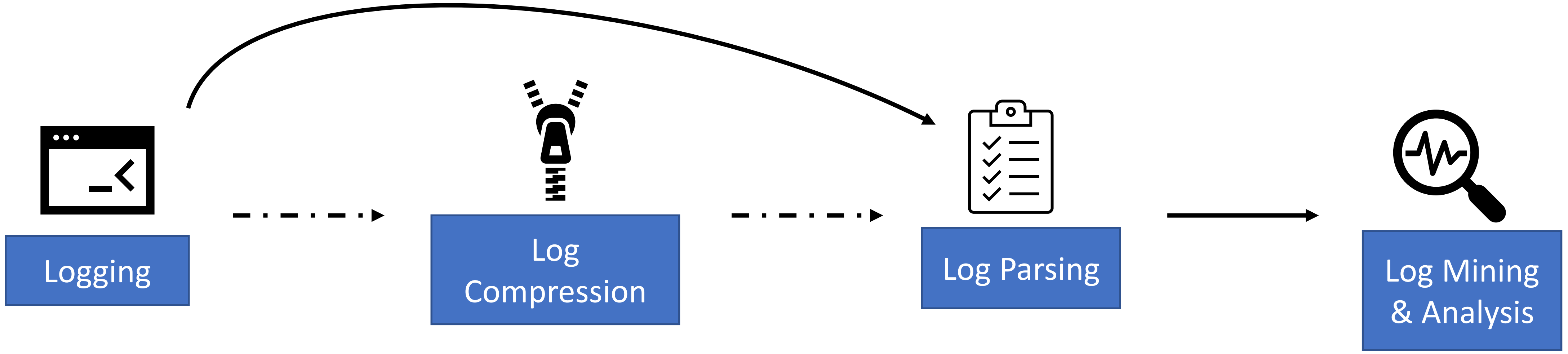
Intro – cont'd

- Log Analysis
  - Is the process of reviewing and interpreting log data generated by an application or system in order to gain insights into its behavior, performance, and usage patterns. This involves collecting, consolidating, and analyzing log data from various sources, such as server logs, application logs, and network logs.

- Automated Log Analysis
  - Log analysis can be performed manually, by reviewing log data and looking for specific patterns or events. However, this approach can be time-consuming and labor-intensive, especially for large-scale applications and systems. Automated log analysis tools and techniques, such as machine learning algorithms and natural language processing, can help to automate this process and identify patterns and anomalies more quickly and efficiently.

Intro – cont'd

- Automated Log Analysis – an overall framework!



Intro - cont’d

S. He et al.

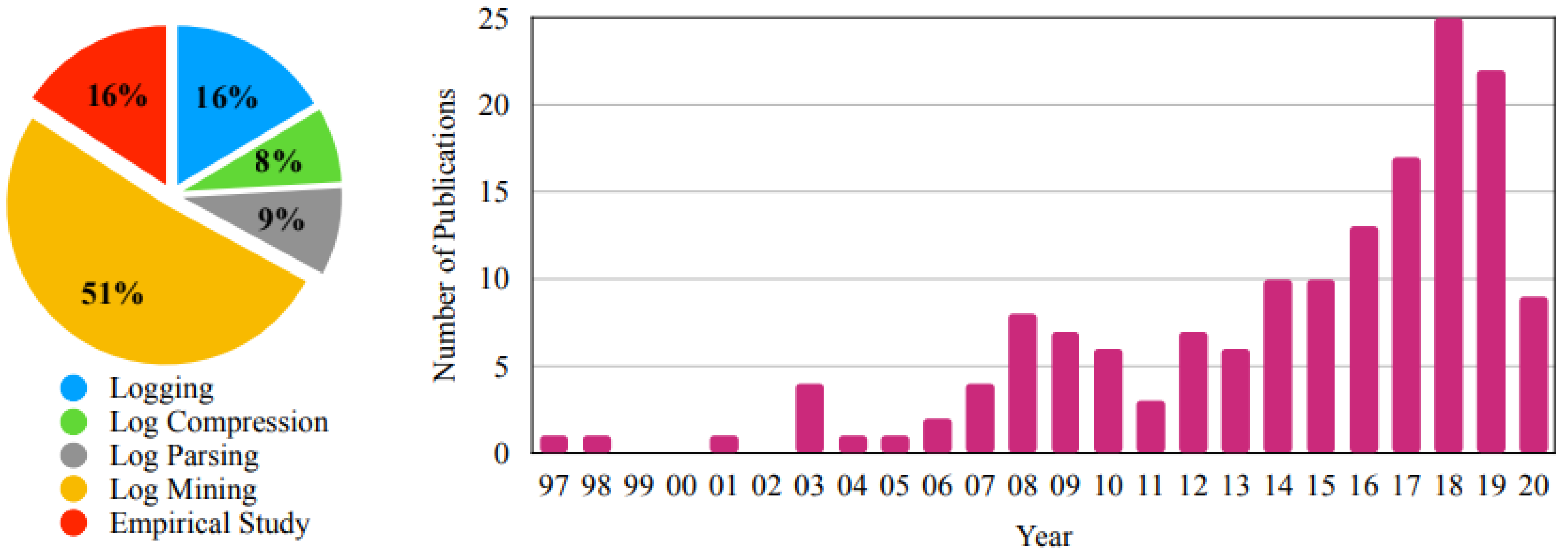


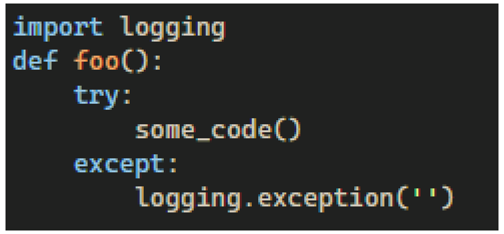
Fig. 3. Paper distribution on each research topic and the associated evolution trend from 1997 to 2020.

https://arxiv.org/pdf/2009.07237.pdf

Logging

- Three main challenges in (automated) logging:
  - Where-to-log:
    - What’s the appropriate locations of logging statements? Use of too many logging statement will result in performance degradation and incurs additional maintenance overhead. [why?]
    - Insufficient logging will also impede logs diagnosability. Incomplete sequence of logs may hinder the reproduction of precise execution paths!
  - What-to-log
    - We must log sufficient and concise information. Logging verbosity refers to the level of detail or amount of information that is included in log messages. Logging verbosity can be set to different levels and can be used to control the amount of log data that is generated and stored.
  - How-to-log
    - Is the “design pattern” and maintenance of logging statements systematically. How do you test for “quality of your logging statements”?

Logging – cont’d



```
import logging
def foo():
    try:
        some_code()
    except:
        logging.exception('')
```

- Categories of Logging Locations:
  - Try-Catch Block: Exceptions are widely used to capture errors and developers rely on logs for debugging and error diagnostics when exception occur. ~32% logging statements in a software system are in Try-Catch blocks [Li et al ‘20 ACM ASE].
  - Branching Block: logging in blocks associated with decision-making systems, e.g., if/else. ~37% logging statements in a software system are in branching blocks [Li et al ‘20 ACM ASE].
  - Looping Block: ~7% logging statements in a software system are in branching blocks [Li et al ‘20 ACM ASE].
  - The Beginning/End of a Method Declaration Block: these logging statements records start and/or termination execution of a method. Helps significantly with execution path reconstruction. ~15% logging statements in a software system are in method declaration blocks [Li et al ‘20 ACM ASE].

OntarioTech Business & IT

Logging – cont'd

- Automatically Suggesting Logging Locations at the Code Block Level
  - This approach involves using machine learning algorithms and other automated techniques to analyze the source code of an application and identify the most appropriate locations to add logging statements.
  - The goal of this approach is to identify the code blocks that are most likely to generate errors or exceptions, and to add logging statements at those locations.
  - To implement this approach, a machine learning model is trained on a dataset of code snippets and corresponding log data, and is then used to predict the most appropriate locations to add logging statements in new code. The model may take into account factors such as the complexity of the code block, and the likelihood of errors or exceptions occurring in that block.
  - Want to know more? Read https://petertsehsun.github.io/papers/Zhenhao_ASE20.pdf

Logging – cont’d

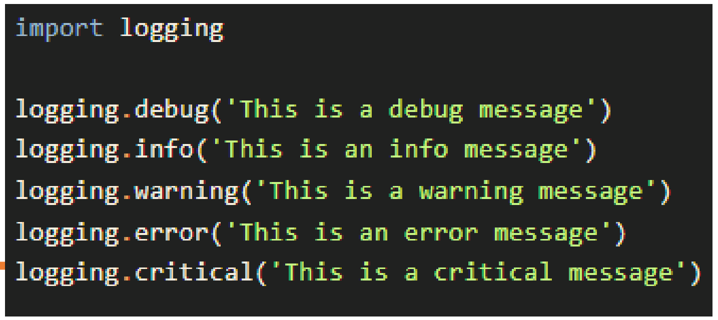
- Notes on Logging Statements’ Maintenance considerations:
  - Logging code updates with time and are often changes are made without consideration for other stakeholders, resulting in sudden failures of log analysis tools and increased maintenance costs!
  - Maintaining logging code along with feature code has proven to be error-prone. Even new programming paradigm that improves the modularity of the logging code is proposed.
  - Intermixing nature of the logging code with feature code usually incurs performance overhead, storage cost, and development efforts!
  - Making sure private-confidential data are not logged!!!!

Log Compression

- is the process of reducing the size of software logs by compressing them using algorithms such as gzip or bzip2.
    - helps to save disk space and reduce the cost of storing and managing log data.
    - makes it easier to transfer log files over a network or to a remote storage location.
    - compressed log files can be searched and analyzed more quickly, since they take up less space on disk and can be read more efficiently.

- Another straightforward way is to reduce the amount of logging statements in the source code or to set a less verbose log level (e.g., change from INFO to ERROR) at runtime.

Log Compression – cont’d



```
import logging

logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.error('This is an error message')
logging.critical('This is a critical message')
```

- Logging Verbosity: refers to the level of detail or amount of information that is included in log messages generated by an application or system. Logging verbosity can be set to different levels, depending on the specific needs of the application or system, and can be used to control the amount of log data that is generated and stored.
  - Error logging: This level of logging captures only error messages, warnings, and other critical information that may indicate a problem or issue with the application or system.
  - Warning logging: This level of logging captures warning messages, which may indicate potential problems or issues that could affect the performance or reliability of the application or system.
  - Informational logging: This level of logging captures general information about the behavior of the application or system, such as startup and shutdown messages, as well as other non-critical events.
  - Debug logging: This level of logging captures detailed information about the internal state of the application or system, including variables, data structures, and other debugging information that can be used to diagnose and troubleshoot issues.
  - Trace logging: This level of logging captures the most detailed information about the behavior of the application or system, including detailed stack traces and other low-level information that is typically only used for advanced debugging and troubleshooting.

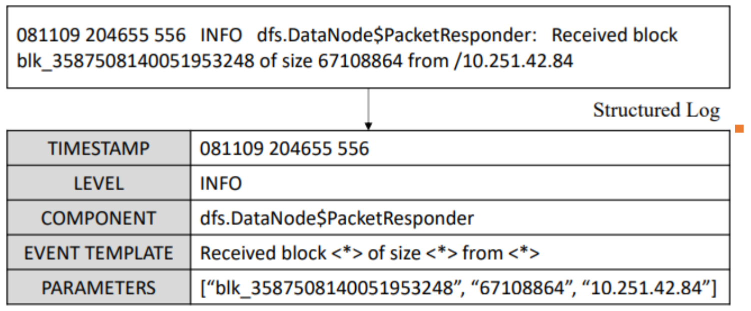
Log Parsing

- Log parsing is the process of analyzing and extracting useful information from log files generated by software systems.
  - Log files contain a vast amount of data, including event timestamps, error messages, system activity, user interactions, and more.
  - Log parsing involves searching for specific patterns or keywords in the log data and extracting relevant information.
    - Automatic analyzers and extractors require "structured data"!
    - Parsing, in nutshell, is transformation of semi-structured data to structured data

- Log parsing can be done manually by searching through log files, but it is often automated using specialized tools or scripts. Automated log parsing can save time and reduce the risk of missing important information.

Log Parsing – cont’d

- An event template is a structured representation of log data that captures the key features of specific events or sequences of events. These templates can be used to classify log entries, normalize log data, and facilitate log analysis.



```
081109 204655 556   INFO   dfs.DataNode$PacketResponder:   Received block
blk_3587508140051953248 of size 67108864 from /10.251.42.84
```

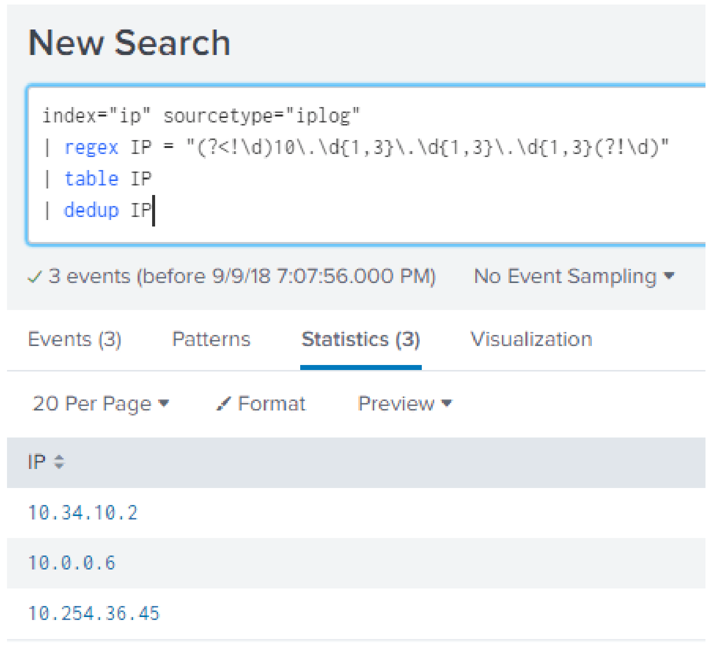
Structured Log

TIMESTAMP	081109 204655 556
LEVEL	INFO
COMPONENT	dfs.DataNode$PacketResponder
EVENT TEMPLATE	Received block <*> of size <*> from <*>
PARAMETERS	[“blk_3587508140051953248”, “67108864”, “10.251.42.84”]

Fig. 4. A log parsing example.

https://arxiv.org/pdf/2009.07237.pdf

- Log Parsing Approaches
  - Ad-hoc regular expression (heuristics): can be used to define patterns that match specific text strings or characters in the log file. For example, a regular expression pattern can be defined to match a specific error message or event in the log data. Once the pattern is defined, it can be used to extract the relevant information from the log file.
    - Heuristic example: “word=value” template!
    - What’s the limitation? [In class discussion]



New Search

```
index="ip" sourcetype="iplog"
| regex IP = "(?<!\d)10\.\d{1,3}\.\d{1,3}\.\d{1,3}(?!\d)"
| table IP
| dedup IP
```

✓ 3 events (before 9/9/18 7:07:56.000 PM)

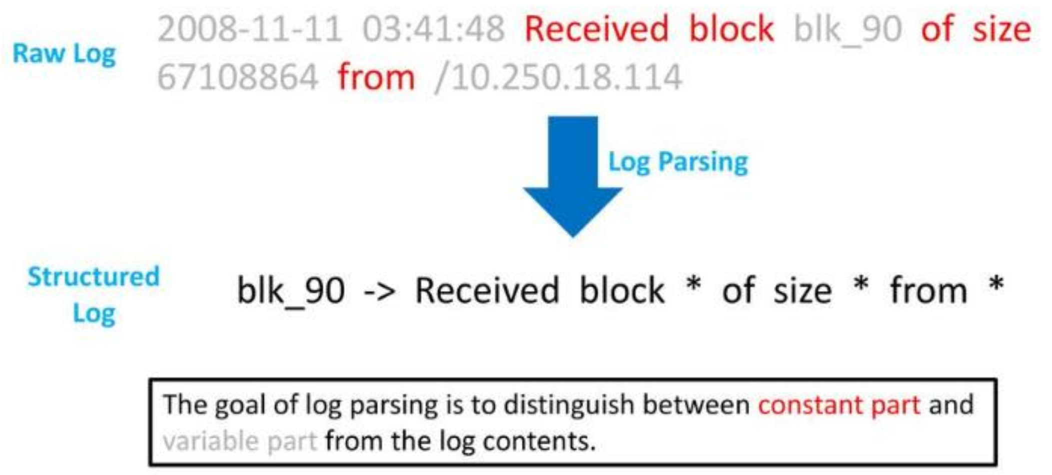
IP
10.34.10.2
10.0.0.6
10.254.36.45

https://splunkonbigdata.com/usage-of-splunk-commands-regex/

OntarioTech Business & IT

Log Parsing – cont’d

- Log Parsing Approaches (cont’d)
  - Frequent Pattern Mining: studying the content of files for frequently used words/tokens. Frequent words/tokens will be considered as constants and will be used in creation of event templates (serve as attributes in the template).
    - Once the event templates have been defined, they can be used to parse new log data and extract relevant information.



```
2008-11-11 03:41:48 Received block blk_90 of size
67108864 from /10.250.18.114
```

```
blk_90 -> Received block * of size * from *
```

The goal of log parsing is to distinguish between constant part and variable part from the log contents.

http://ristov.github.io/publications/slct-ipom03-web.pdf

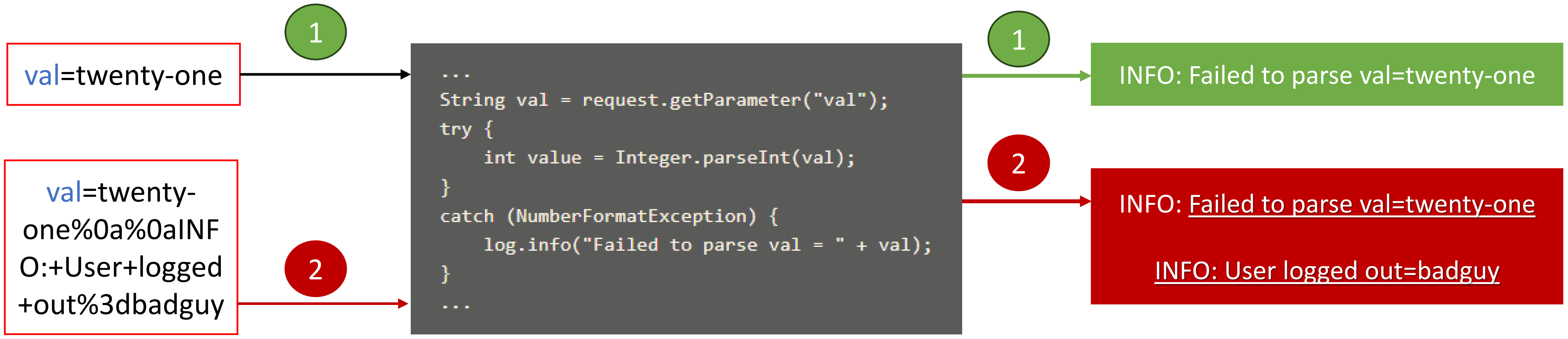
Log Parsing – cont'd

- Log Parsing Approaches (cont'd)
  - Clustering: clustering is a technique used in log parsing to group log entries based on their similarity. Clustering algorithms analyze log data to identify patterns and similarities between log entries, and group them into clusters or categories. This technique can be used to reduce the complexity of log data, and identify pattern.
  - It is often useful when non-constant parameters have variable length. E.g.,
    - "Downloaded Whats App and install" – "Downloaded Facebook and install". They both have the same template event of "Downloaded <*> and install" whil; the length of the parameter is flexible.
  - *There are other approaches such as Evolutionary algorithms and iterative partitioning approaches that are out of scope of this course ☹*

INFO: User logged out=badguy

Log Parsing – cont’d

- Th input that gets logged contains characters like newline or carriage return, an attacker can craft messages that, when logged, break the log file's integrity.
  - This can result in forged log entries that appear legitimate but are actually maliciously inserted.



val=twenty-one

val=twenty-one%0a%0aINFO:+User+logged+out%3dbadguy

```
...
String val = request.getParameter("val");
try {
    int value = Integer.parseInt(val);
}
catch (NumberFormatException) {
    log.info("Failed to parse val = " + val);
}
...
```

```
INFO: Failed to parse val=twenty-one
```

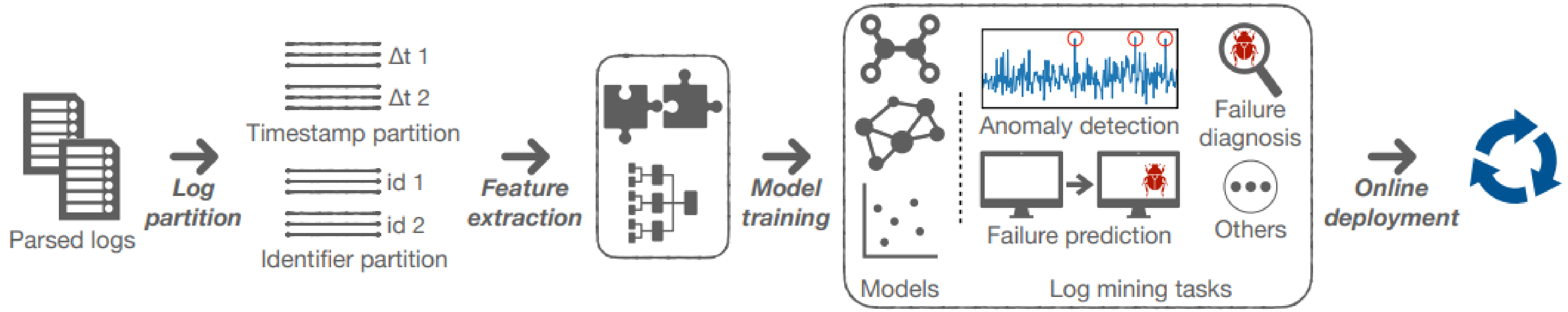
```
INFO: Failed to parse val=twenty-one

INFO: User logged out=badguy
```

Log Mining

- is the process of analyzing log data generated by software applications to extract valuable information and insights. Common use cases
  - Troubleshooting errors in software applications. By analyzing log data, developers can identify the root cause of errors and make changes to the software to prevent similar errors from occurring in the future.
  - System administrators can identify bottlenecks and areas of inefficiency in the system and make changes to improve performance.
  - Detect anomalous behaviors/state in an information system.
  - Failure Prediction.

Log Mining – cont’d

- Log **Analysis** Approaches:
  - Rule-based systems: traditional keyword search (e.g., “error”, “exception”, and “failed”) to mine suspicious logs that might be associated with software problems.
    - Very labour intensive and do not scale [why?] well given today’s variety and velocity of logs produced by moder software systems.
  - Log Mining

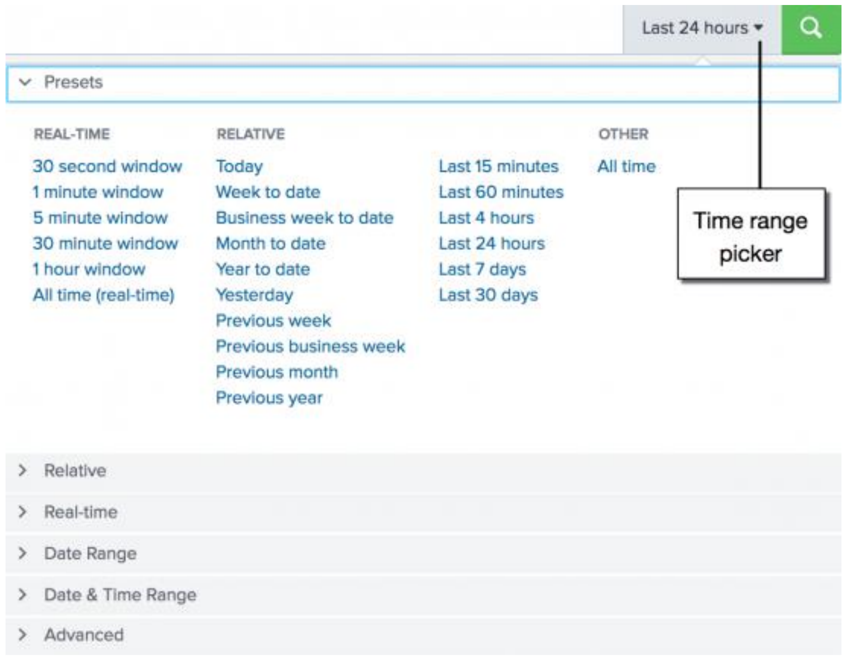


https://arxiv.org/pdf/2009.07237.pdf

Log Mining – cont'd

- Log Partitioning
  - to divide log data into smaller, more manageable subsets. Involves partitioning log data based on specific criteria, such as time of occurrence, source, or type of log entry.
  - can be useful for security and compliance purposes. access to sensitive log data can be restricted to authorized users, while other users can be granted access to less sensitive log data.

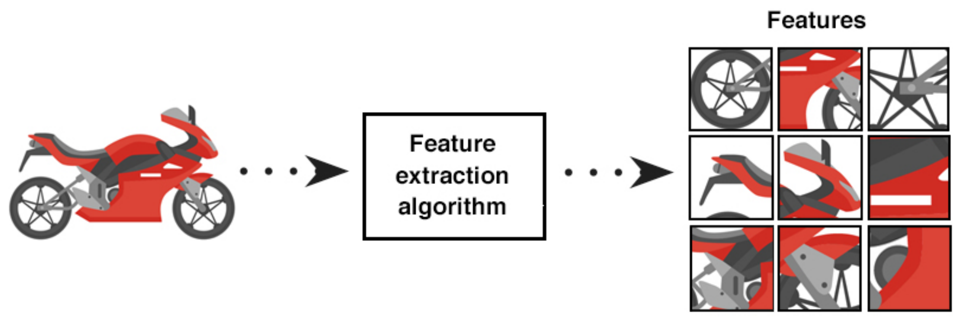
  - Timestamp partitioning is very common (e.g., sliding time window):
    - By partitioning log data into smaller time intervals, trends and patterns in system behavior can be more easily identified and analyzed. This can help to identify potential issues and opportunities for performance improvement.



https://docs.splunk.com/Documentation/Splunk/9.0.4/Search/Selecttimerangestoapply

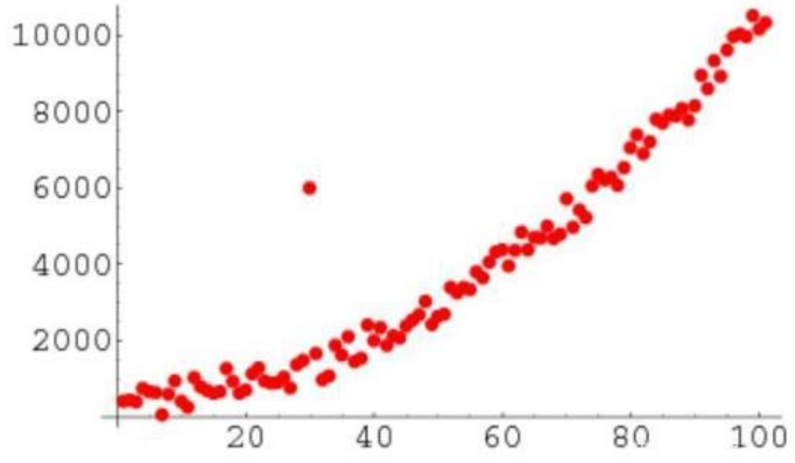
Log Mining – cont’d

- Feature Extraction
  - Feature extraction is a technique used in machine learning and data analysis to identify and extract relevant features or attributes from raw data that can be used to build models for failure prediction, anomaly detection, etc.
  - The importance of feature extraction lies in the fact that many machine learning algorithms require a relatively small number of informative features to build accurate models.



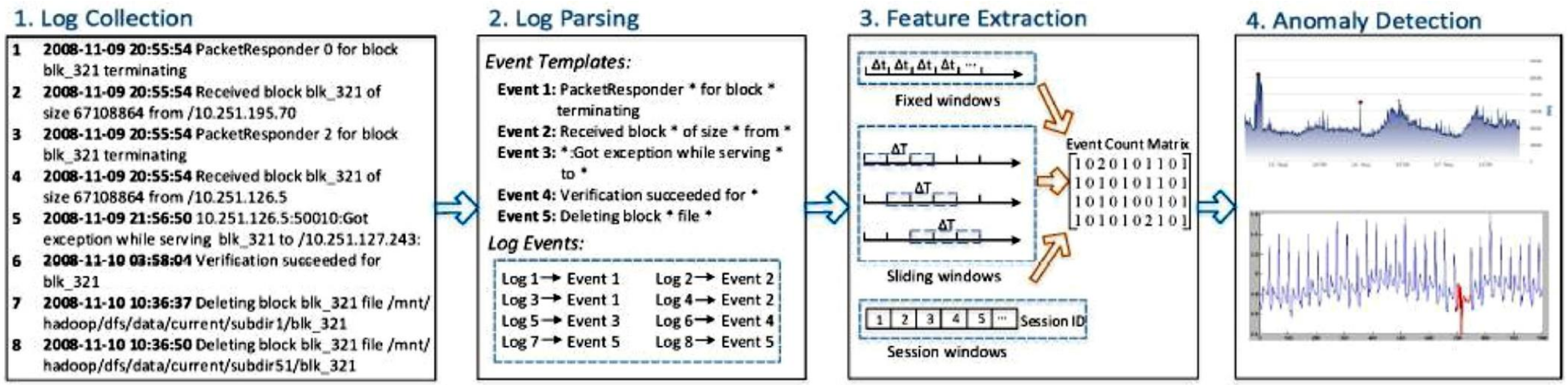
Log Mining – cont’d

- Log Mining Tasks – E.g., Anomaly Detection
  - Anomaly detection in software logs is the process of identifying unusual or unexpected events or patterns in log data that may indicate a potential problem or security threat. Anomalies can be caused by a variety of factors, such as hardware or software failures, cyberattacks, or user errors, and can have a significant impact on system performance and reliability.
  - Anomaly detection algorithms can be classified into two main categories: supervised and unsupervised. In supervised anomaly detection, a model is trained on a labeled dataset to identify known anomalies. This approach is useful when there is prior knowledge of the types of anomalies that may occur in the log data. In unsupervised anomaly detection, no prior knowledge is assumed, and the algorithm identifies anomalies based on deviations from expected patterns in the log data.



Log Mining – cont’d

- Log Mining Tasks – E.g., Anomaly Detection
  - Important: in supervised anomaly detection, the constructed models can only detect “anomaly” that have previously been known (and present in the training data) and detected.
    - Unsupervised anomaly detection, the constructed models may be able to detect “novel” anomalies that never seen before!



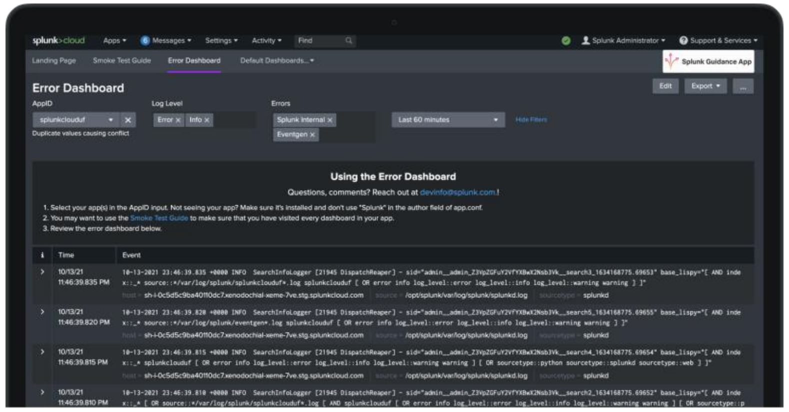
```
1  2008-11-09 20:55:54 PacketResponder 0 for block
   blk_321 terminating
2  2008-11-09 20:55:54 Received block blk_321 of
   size 67108864 from /10.251.195.70
3  2008-11-09 20:55:54 PacketResponder 2 for block
   blk_321 terminating
4  2008-11-09 20:55:54 Received block blk_321 of
   size 67108864 from /10.251.126.5
5  2008-11-09 21:56:50 10.251.126.5:50010:Got
   exception while serving blk_321 to /10.251.127.243:
6  2008-11-10 03:58:04 Verification succeeded for
   blk_321
7  2008-11-10 10:36:37 Deleting block blk_321 file /mnt/
   hadoop/dfs/data/current/subdir1/blk_321
8  2008-11-10 10:36:50 Deleting block blk_321 file /mnt/
   hadoop/dfs/data/current/subdir51/blk_321
```

Event 1: PacketResponder * for block * terminating
Event 2: Received block * of size * from *
Event 3: * :Got exception while serving * to *
Event 4: Verification succeeded for *
Event 5: Deleting block * file *

Log 1 → Event 1	Log 2 → Event 2
Log 3 → Event 1	Log 4 → Event 2
Log 5 → Event 3	Log 6 → Event 4
Log 7 → Event 5	Log 8 → Event 5

$$\begin{bmatrix} 1&0&2&0&1&0&1&1&0&1 \\ 1&0&1&0&1&0&1&1&0&1 \\ 1&0&1&0&1&0&0&1&0&1 \\ 1&0&1&0&1&0&2&1&0&1 \end{bmatrix}$$

What’s out there!



Error Dashboard

AppID: splunkclouduf
Duplicate values causing conflict

Log Level: Error, Info

Errors: Splunk Internal, Eventgen

Last 60 minutes

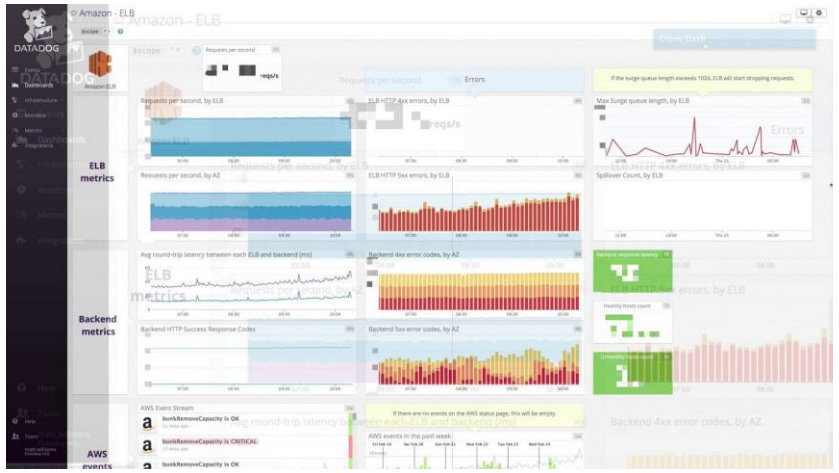
Using the Error Dashboard

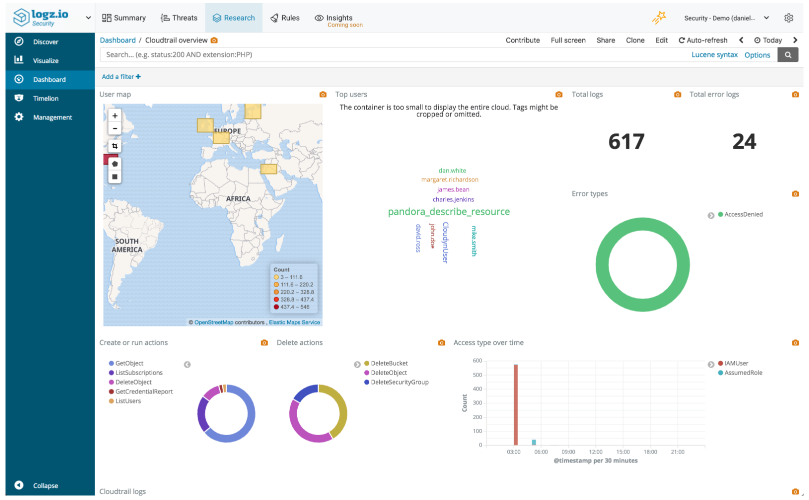
Questions, comments? Reach out at devinfo@splunk.com!

1. Select your app(s) in the AppID input. Not seeing your app? Make sure it's installed and don't use "Splunk" in the author field of app.conf.
2. You may want to use the Smoke Test Guide to make sure that you have visited every dashboard in your app.
3. Review the error dashboard below.

```
i  Time                  Event
>  10/13/21              10-13-2021 23:46:39.835 +0000 INFO  SearchInfoLogger [21945 DispatchReaper] - sid="admin__admin_Z3VpZGFuY2VfYXBwX2Nsb3Vk__search3_1634168775.69653" base_lispy="[ AND inde
   11:46:39.835 PM       x::_* source::*/var/log/splunk/splunkclouduf*.log splunkclouduf [ OR error info log_level::error log_level::info log_level::warning warning ] ]"
                         host = sh-i-0c5d5c9ba40110dc7.xenodochial-xeme-7ve.stg.splunkcloud.com   source = /opt/splunk/var/log/splunk/splunkd.log   sourcetype = splunkd
>  10/13/21              10-13-2021 23:46:39.820 +0000 INFO  SearchInfoLogger [21945 DispatchReaper] - sid="admin__admin_Z3VpZGFuY2VfYXBwX2Nsb3Vk__search5_1634168775.69655" base_lispy="[ AND inde
   11:46:39.820 PM       x::_* source::*/var/log/splunk/eventgen*.log splunkclouduf [ OR error info log_level::error log_level::info log_level::warning warning ] ]"
                         host = sh-i-0c5d5c9ba40110dc7.xenodochial-xeme-7ve.stg.splunkcloud.com   source = /opt/splunk/var/log/splunk/splunkd.log   sourcetype = splunkd
>  10/13/21              10-13-2021 23:46:39.815 +0000 INFO  SearchInfoLogger [21945 DispatchReaper] - sid="admin__admin_Z3VpZGFuY2VfYXBwX2Nsb3Vk__search4_1634168775.69654" base_lispy="[ AND inde
   11:46:39.815 PM       x::_* splunkclouduf [ OR error info log_level::error log_level::info log_level::warning warning ] [ OR sourcetype::python sourcetype::splunkd sourcetype::web ] ]"
                         host = sh-i-0c5d5c9ba40110dc7.xenodochial-xeme-7ve.stg.splunkcloud.com   source = /opt/splunk/var/log/splunk/splunkd.log   sourcetype = splunkd
>  10/13/21              10-13-2021 23:46:39.810 +0000 INFO  SearchInfoLogger [21945 DispatchReaper] - sid="admin__admin_Z3VpZGFuY2VfYXBwX2Nsb3Vk__search2_1634168775.69652" base_lispy="[ AND inde
   11:46:39.810 PM       x::_* [ OR source::*/var/log/splunk/splunkclouduf*.log [ AND splunkclouduf [ OR error info log_level::error log_level::info log_level::warning warning ] [ OR sourcetype::p
```

elasticsearch + logstash + kibana





Azure Log Analytics

Log Injection

- It occurs when
    1. Data enters into logs from untrusted source,
    2. The malicious data is written into an application or system log file.

- And can cause
    1. Injection of new/bogus log events,
    2. Injection of XSS attacks, assuming log events are viewed in a vulnerable web application,
    3. Injection of commands that parsers could execute,
    4. Poison Machine Learning (ML) models that use log files for model training.