Splunk: Data Manipulation

Introduction

Data processing, parsing, and manipulation in Splunk are crucial for extracting meaningful insights and enabling effective analysis of machine-generated data. From a security perspective, these capabilities are particularly valuable in identifying and responding to security threats, investigating incidents, and monitoring system health.

Learning Objective

Some of the learning objectives that this room will cover are:

- How Events are parsed in Splunk
- Importance of configuration files like inputs.conf, transform.conf, and props.conf
- How to Extract custom fields and apply them to filter
- How to identify the timestamp in the Event logs

Scenario and Lab Instructions

Let's play a scenario where You are John, who is working as a SOC Analyst at CyberT. You have been presented with a scenario where one of our clients needs to ingest some logs from a custom source.

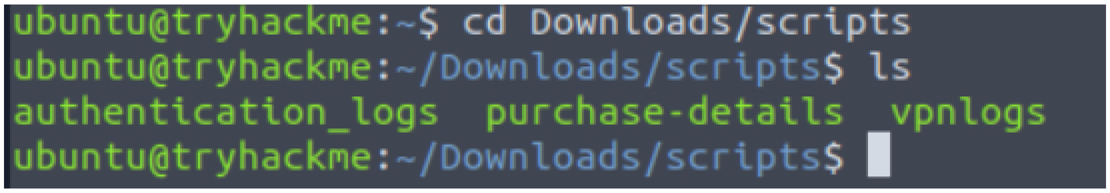
Splunk needs to be properly configured to parse and transform the logs appropriately. Some of the issues being highlighted are:

- Event Breaking: Configure Splunk to break the events properly.
- Multi-line Events: Configure Splunk to configure multi-line events properly.
- Masking: Some logs contain sensitive data. To comply with the PCI DSS (Payment Card Industry Data Security Standard) standard, information like credit card numbers must be masked to avoid any violation.
- Extracting custom fields: In the weblogs, some fields are redundant and need to be removed.

Let's connect to the lab and continue to understand how Splunk data manipulation works at the configuration level.

****

Answer the questions below:
How many Python scripts are present in the ~/Downloads/scripts directory?



```
ubuntu@tryhackme:~$ cd Downloads/scripts
ubuntu@tryhackme:~/Downloads/scripts$ ls
authentication_logs  purchase-details  vpnlogs
ubuntu@tryhackme:~/Downloads/scripts$
```

Answer: 3

Splunk Data Processing: Overview

Splunk is a powerful data analytics platform used for searching, monitoring, and analyzing large amounts of machine-generated data. Data parsing in Splunk involves extracting relevant fields and transforming the data into a structured format for efficient analysis. Here's a step-by-step guide on how data is parsed in Splunk, including the use of props.conf:

Step 1: Understand the Data Format

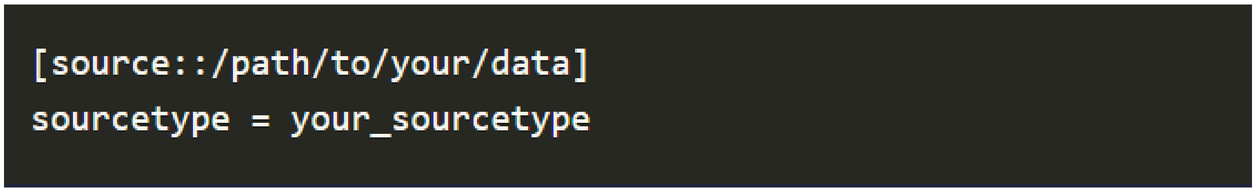
First, you need to understand the data format you want to parse. Splunk supports various data formats, such as CSV, JSON, XML, syslog, and more. Determine the format of your data source and the relevant fields you want to extract.

Step 2: Identify the Sourcetype

In Splunk, the sourcetype represents the format of the data being indexed. It helps Splunk apply the appropriate parsing rules. If your data source does not have a pre-defined sourcetype, you can create a custom one in Splunk.

Step 3: Configure props.conf

The props.conf file defines data parsing settings for specific sourcetypes or data sources. It resides in the $SPLUNK_HOME/etc/system/local directory. Here's an example of how you can configure props.conf:



```
[source::/path/to/your/data]
sourcetype = your_sourcetype
```

In this example, /path/to/your/data is the path to your data source, and your_sourcetype is the name of the sourcetype you want to assign to that data.

Step 4: Define Field Extractions

You can define regular expressions or use pre-built extraction techniques to parse fields from the data. Here's an example of defining field extractions in props.conf:

```
[your_sourcetype]
EXTRACT-fieldname1 = regular_expression1
EXTRACT-fieldname2 = regular_expression2
```

Replace your_sourcetype with the actual sourcetype name you defined. fieldname1 and fieldname2 represent the names of the fields you want to extract, while regular_expression1 and regular_expression2 are the regular expressions used to match and extract the desired values.

Step 5: Save and Restart Splunk
After making changes to props.conf, save the file, and restart Splunk to apply the new configurations. You can do this using the Splunk web interface or by using the command line.

Step 6: Verify and Search the Data
Once Splunk restarts, you can search and verify that the data is being parsed correctly. You can use the extracted fields to filter and analyze the data effectively.

In the next task, we will explore important configuration files.

Exploring Splunk Configuration Files

Splunk uses several configuration files to control various data processing and indexing aspects. Let's explore some of the key configuration files in Splunk, along with examples of their usage:

inputs.conf:

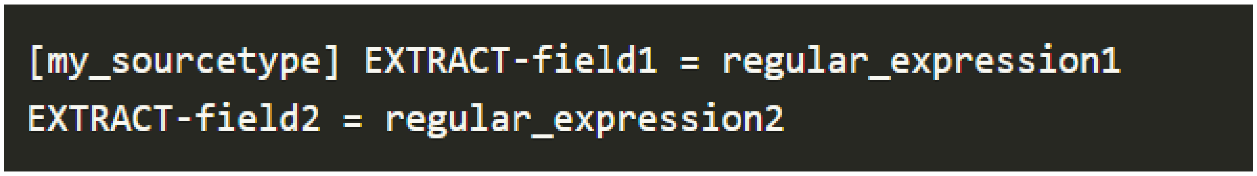
- Purpose: Defines data inputs and how to collect data from different sources.
- Example: Suppose you want to monitor a specific log file. You can configure inputs.conf as follows:

```
[monitor:///path/to/logfile.log]
sourcetype = my_sourcetype
```

props.conf:

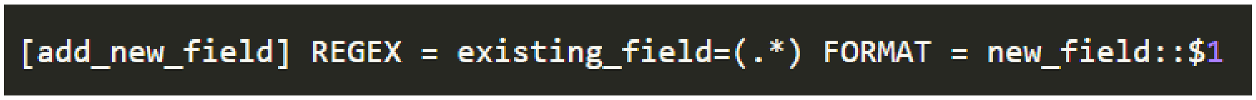
- Purpose: Specifies parsing rules for different sourcetypes to extract fields and define field extractions.

- Example: Suppose you have a custom sourcetype named my_sourcetype and want to extract fields using regular expressions. You can define them in props.conf:



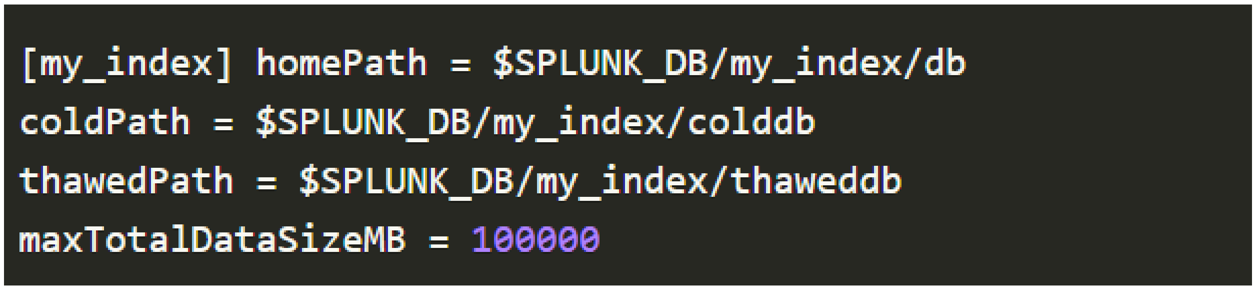
```
[my_sourcetype] EXTRACT-field1 = regular_expression1
EXTRACT-field2 = regular_expression2
```

transforms.conf:
- Purpose: Allows you to define field transformations and enrichments on indexed events.
- Example: Suppose you want to add a new event field based on existing field values. You can use transforms.conf:



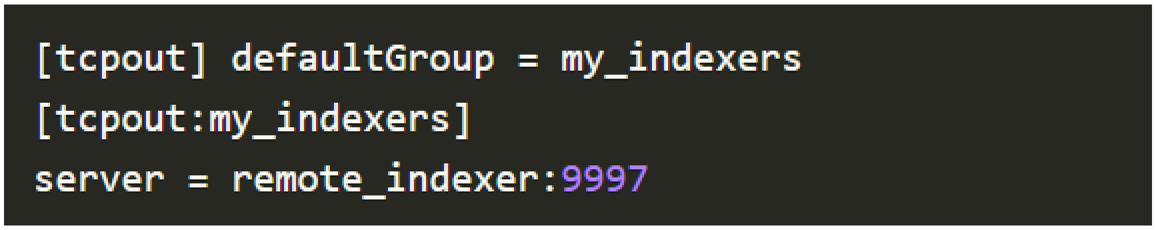
```
[add_new_field] REGEX = existing_field=(.*) FORMAT = new_field::$1
```

indexes.conf:
- Purpose: Manages the configuration of indexes in Splunk, including storage, retention policies, and access control.
- Example: Suppose you want to create a new index named my_index with specific settings. You can configure indexes.conf:



```
[my_index] homePath = $SPLUNK_DB/my_index/db
coldPath = $SPLUNK_DB/my_index/colddb
thawedPath = $SPLUNK_DB/my_index/thaweddb
maxTotalDataSizeMB = 100000
```

outputs.conf:
- Purpose: Specifies the destination and settings for sending indexed data to various outputs, such as remote Splunk instances or third-party systems.
- Example: Suppose you want to forward your indexed data to a remote Splunk indexer. You can configure outputs.conf:



```
[tcpout] defaultGroup = my_indexers
[tcpout:my_indexers]
server = remote_indexer:9997
```

authentication.conf:

- Purpose: Manages authentication settings and user authentication methods.
- Example: Suppose you want to enable LDAP authentication for Splunk users. You can configure authentication.conf:

```
[authentication]
authSettings = LDAP
[authenticationLDAP]
SSLEnabled = true
```

These are just a few examples of the various configuration files used in Splunk. Each file serves a specific purpose and allows you to customize Splunk's behavior based on your data sources, parsing requirements, indexing settings, output destinations, and more.

STANZAS in Splunk Configurations

Splunk configurations contain various stanza configurations that define how data is processed and indexed. These stanzas have a certain purpose, and it's important to understand what these are and how they are used. A brief summary of the common stanzas are explained below:

Stanza	Explanation	Example
[sourcetype]	Specifies the configuration for a specific sourcetype. It allows you to define how the data from that sourcetype should be parsed and indexed.	[apache:access]- Configures parsing and indexing settings for Apache access logs.
TRANSFORMS	Applies field transformations to extracted events. You can reference custom or pre-defined field transformation configurations to modify or create new fields based on the extracted data.	TRANSFORMS-mytransform = myfield1, myfield2 Applies the transformation named "mytransform" to fields myfield1 and myfield2.

REPORT	Defines extraction rules for specific fields using regular expressions. It associates a field name with a regular expression pattern to extract desired values. This stanza helps in parsing and extracting structured fields from unstructured or semi-structured data.	REPORT-field1 = pattern1 - Extracts field1 using pattern1 regular expression.
EXTRACT	Defines extraction rules for fields using regular expressions and assigns them specific names. It is similar to the REPORT stanza, but it allows more flexibility in defining custom field extractions.	EXTRACT-field1 = (?<fieldname>pattern1) Extracts field1 using pattern1 regular expression and assigns it to fieldname
TIME_PREFIX	Specifies the prefix before the timestamp value in events. This stanza is used to identify the position of the timestamp within the event.	TIME_PREFIX = \[timestamp\] - Identifies the prefix [timestamp] before the actual timestamp in events.
TIME_FORMAT	Defines the format of the timestamp present in the events. It allows Splunk to correctly extract and parse timestamps based on the specified format.	TIME_FORMAT = %Y-%m-%d %H:%M:%S - Specifies the timestamp format as YYYY-MM-DD HH:MM:SS.
LINE_BREAKER	Specifies a regular expression pattern that identifies line breaks within events. This stanza is used to split events into multiple lines for proper parsing and indexing.	LINE_BREAKER = ([\r\n]+) - Identifies line breaks using the regular expression [\r\n]+.
SHOULD_LINEMERGE	Determines whether lines should be merged into a single event or treated as	SHOULD_LINEMERGE = false - Disables line merging, treating each line

	separate events. It controls the behavior of line merging based on the specified regular expression pattern in the LINE_BREAKER stanza.	as a separate event.
BREAK_ONLY_BEFORE	Defines a regular expression pattern that marks the beginning of an event. This stanza is used to identify specific patterns in the data that indicate the start of a new event.	BREAK_ONLY_BEFORE = ^\d{4}-\d{2}-\d{2} - Identifies the start of a new event if it begins with a date in the format YYYY-MM-DD.
BREAK_ONLY_AFTER	Specifies a regular expression pattern that marks the end of an event. It is used to identify patterns in the data that indicate the completion of an event.	BREAK_ONLY_AFTER = \[END\] - Marks the end of an event if it contains the pattern [END].
KV_MODE	Specifies the key-value mode used for extracting field-value pairs from events. The available modes are: auto, none, simple, multi, and json. This stanza determines how fields are extracted from the events based on the key-value pairs present in the data. It helps in parsing structured data where fields are represented in a key-value format.	KV_MODE = json - Enables JSON key-value mode for parsing events with JSON formatted fields.

These examples demonstrate the usage of each stanza in props.conf and provide a better understanding of how they can be applied to configure data parsing behavior in Splunk.

In the next task, we will use some of these stanzas explained above to better understand.

****
Answer the questions below:
Which stanza is used in the configuration files to break the events after the provided pattern?
Answer: BREAK_ONLY_AFTER

Which stanza is used to specify the pattern for the line break within events?
Answer: LINE_BREAKER

Which configuration file is used to define transformations and enrichments on indexed fields?
Answer: transforms.conf

Which configuration file is used to define inputs and ways to collect data from different sources?
Answer: inputs.conf

Creating a Simple Splunk App

We have explored the importance and usage of various configuration files and the purpose-based stanzas within those configuration files. We will be using them extensively in the coming tasks. For now, let's create a simple Splunk app using the following steps and generate our first sample event using inputs.conf file.

Start Splunk

Splunk is installed in the /opt/splunk directory. Go to this directory and run the following command bin/splunk start to start the Splunk instance with root privileges. Use the following credentials to log in to the Splunk Interface:

Username: splunk
Password: splunk123

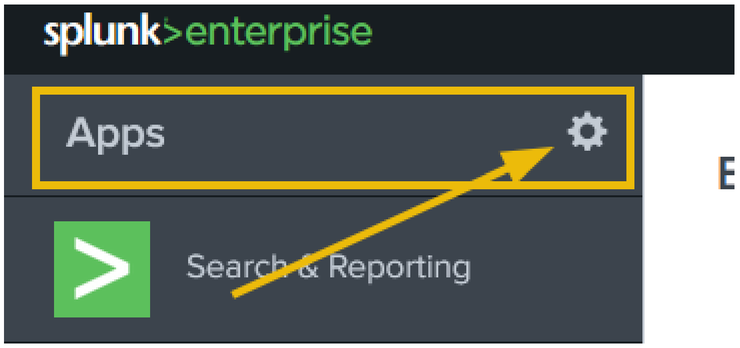
Once it is done, open MACHINE_IP:8000 in the browser.

About Splunk Apps

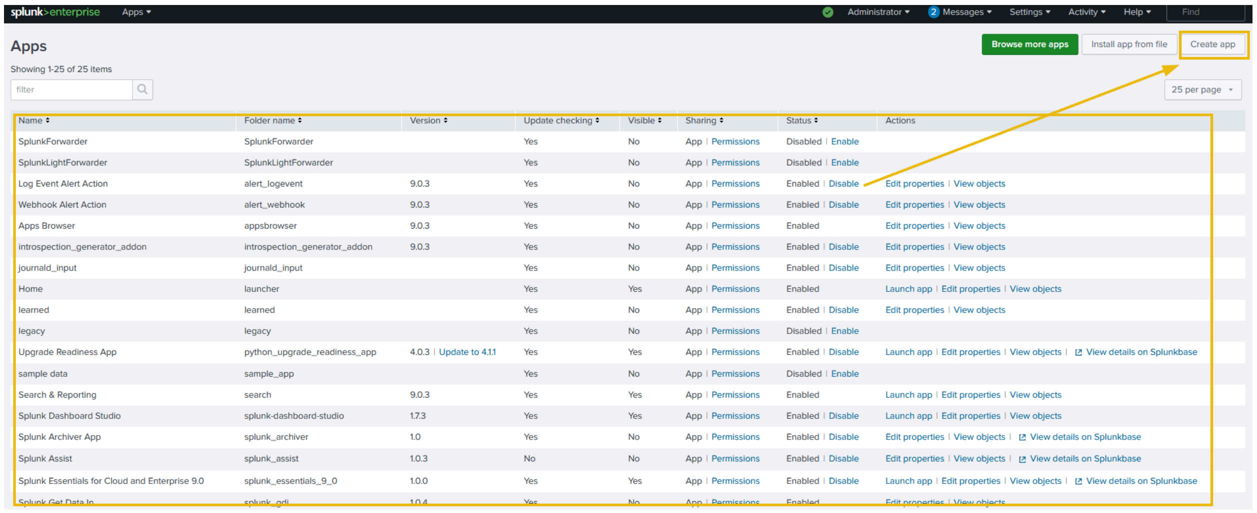
Splunk apps are pre-packaged software modules or extensions that enhance the functionality of the Splunk platform. The purpose of Splunk apps is to provide specific sets of features, visualizations, and configurations tailored to meet the needs of various use cases and industries.

Create a Simple App

Once the Splunk Instance is loaded, click on the Manage App tab as highlighted below:



It will take us to the page that contains all the available apps in Splunk. To create a new app, Click on the Create App tab as shown below:

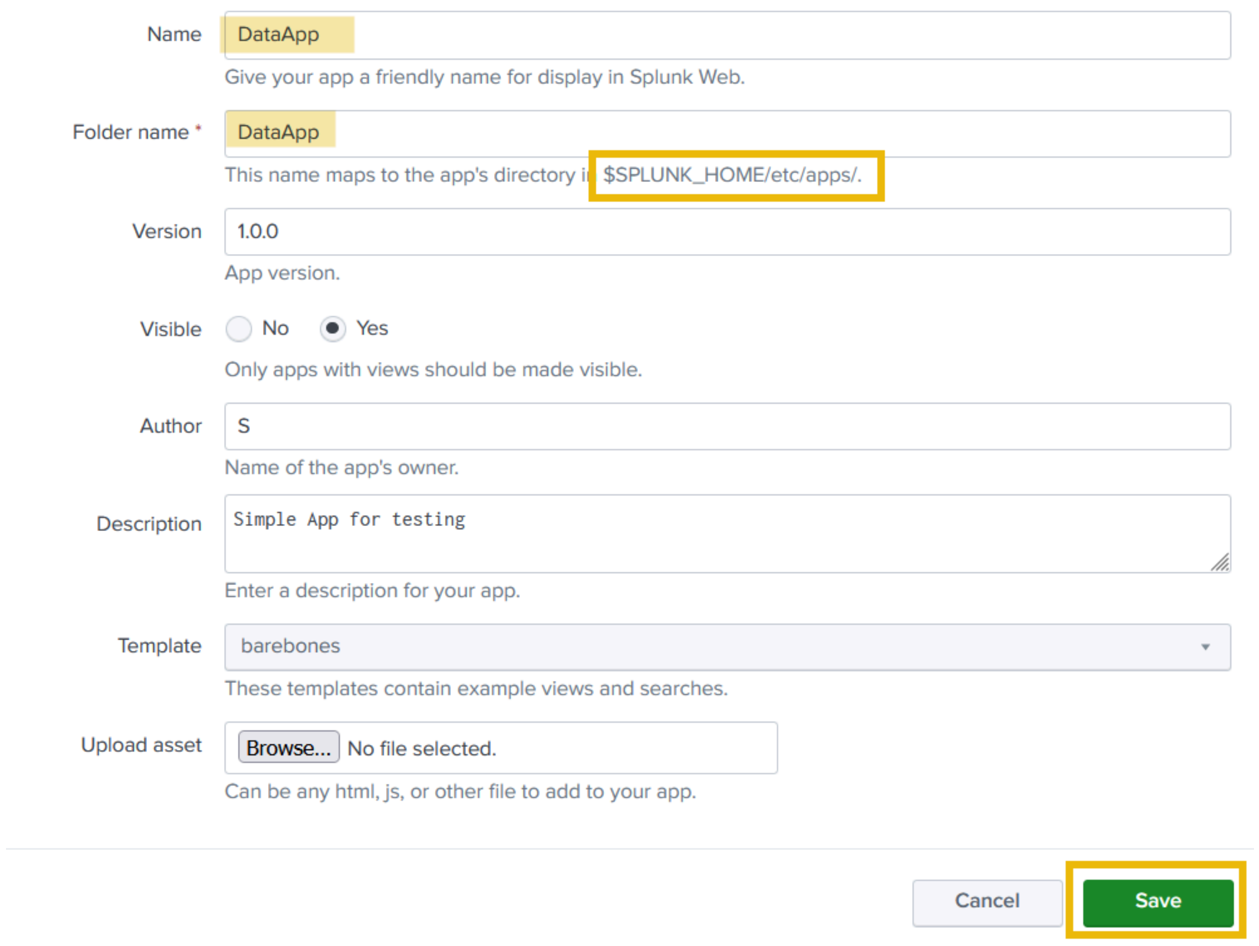


Apps

Showing 1-25 of 25 items

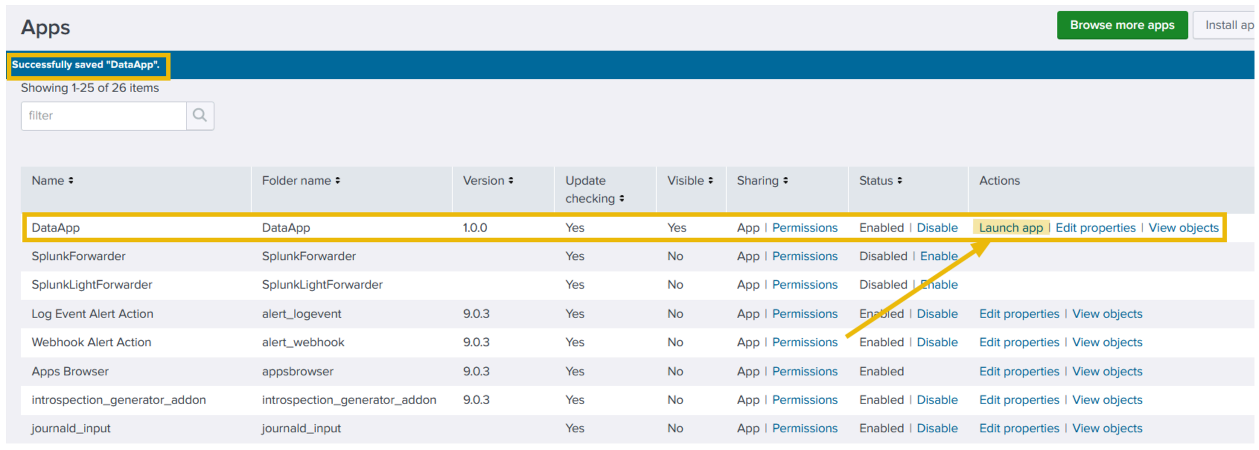
Name	Folder name	Version	Update checking	Visible	Sharing	Status	Actions
SplunkForwarder	SplunkForwarder		Yes	No	App | Permissions	Disabled | Enable	
SplunkLightForwarder	SplunkLightForwarder		Yes	No	App | Permissions	Disabled | Enable	
Log Event Alert Action	alert_logevent	9.0.3	Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects
Webhook Alert Action	alert_webhook	9.0.3	Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects
Apps Browser	appsbrowser	9.0.3	Yes	No	App | Permissions	Enabled	Edit properties | View objects
introspection_generator_addon	introspection_generator_addon	9.0.3	Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects
journald_input	journald_input		Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects
Home	launcher		Yes	Yes	App | Permissions	Enabled	Launch app | Edit properties | View objects
learned	learned		Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects
legacy	legacy		Yes	No	App | Permissions	Disabled | Enable	
Upgrade Readiness App	python_upgrade_readiness_app	4.0.3 | Update to 4.1.1	Yes	Yes	App | Permissions	Enabled | Disable	Launch app | Edit properties | View objects | View details on Splunkbase
sample data	sample_app		Yes	No	App | Permissions	Disabled | Enable	
Search & Reporting	search	9.0.3	Yes	Yes	App | Permissions	Enabled	Launch app | Edit properties | View objects
Splunk Dashboard Studio	splunk-dashboard-studio	1.7.3	Yes	Yes	App | Permissions	Enabled | Disable	Launch app | Edit properties | View objects
Splunk Archiver App	splunk_archiver	1.0	Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects | View details on Splunkbase
Splunk Assist	splunk_assist	1.0.3	No	No	App | Permissions	Enabled | Disable	Edit properties | View objects | View details on Splunkbase
Splunk Essentials for Cloud and Enterprise 9.0	splunk_essentials_9_0	1.0.0	Yes	Yes	App | Permissions	Enabled | Disable	Launch app | Edit properties | View objects | View details on Splunkbase
Splunk Get Data In	splunk_gdi	1.0.4	Yes	No	App | Permissions	Enabled	Edit properties | View objects

Next, fill in the details about the new app that we want to create. The new app will be placed in the /opt/splunk/etc/apps directory as highlighted below:



Name	DataApp
	Give your app a friendly name for display in Splunk Web.
Folder name *	DataApp
	This name maps to the app's directory in $SPLUNK_HOME/etc/apps/.
Version	1.0.0
	App version.
Visible	No [✓ Yes]
	Only apps with views should be made visible.
Author	S
	Name of the app's owner.
Description	Simple App for testing
	Enter a description for your app.
Template	barebones
	These templates contain example views and searches.
Upload asset	Browse... No file selected.
	Can be any html, js, or other file to add to your app.

Great. A new Splunk app has been created successfully and it can be shown on the Apps page. Click on the Launch App to see if there is any activity logged yet.



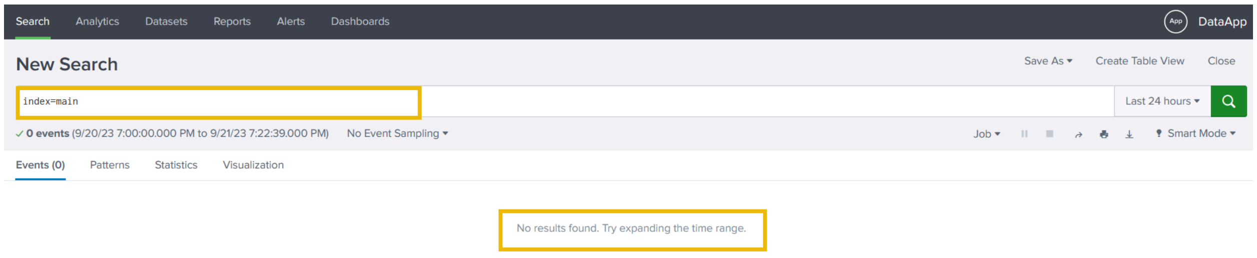
Apps

Successfully saved "DataApp".

Showing 1-25 of 26 items

Name	Folder name	Version	Update checking	Visible	Sharing	Status	Actions
DataApp	DataApp	1.0.0	Yes	Yes	App | Permissions	Enabled | Disable	Launch app | Edit properties | View objects
SplunkForwarder	SplunkForwarder		Yes	No	App | Permissions	Disabled | Enable	
SplunkLightForwarder	SplunkLightForwarder		Yes	No	App | Permissions	Disabled | Enable	
Log Event Alert Action	alert_logevent	9.0.3	Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects
Webhook Alert Action	alert_webhook	9.0.3	Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects
Apps Browser	appsbrowser	9.0.3	Yes	No	App | Permissions	Enabled	Edit properties | View objects
introspection_generator_addon	introspection_generator_addon	9.0.3	Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects
journald_input	journald_input		Yes	No	App | Permissions	Enabled | Disable	Edit properties | View objects

As it is evident, no activity has been logged yet. Follow the next steps to generate sample logs.



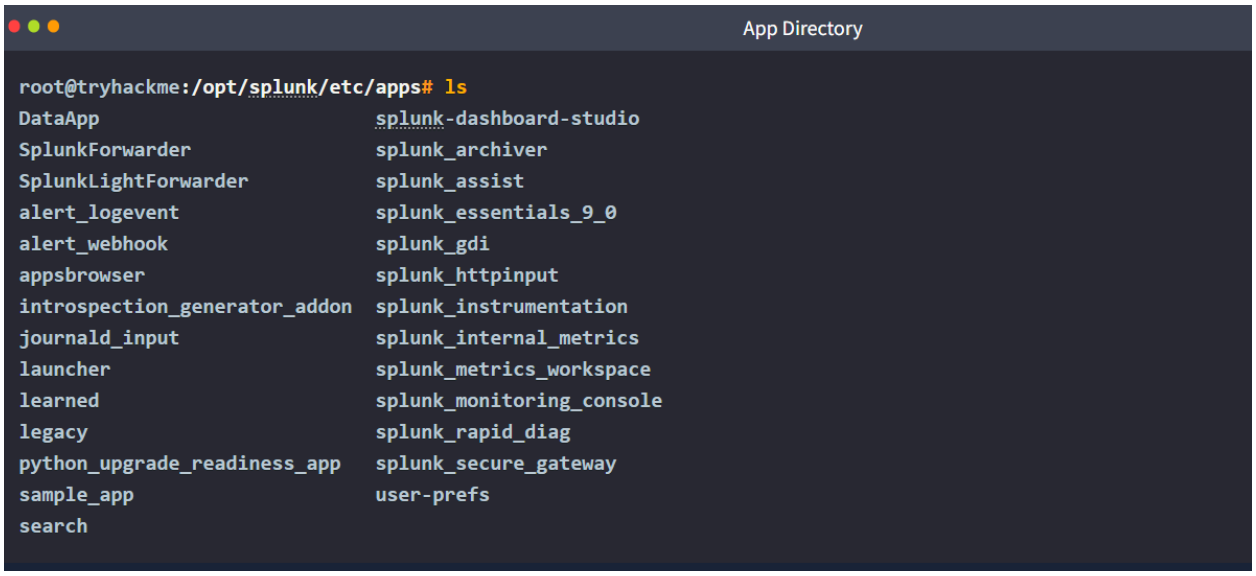
```
index=main
```

✓ 0 events (9/20/23 7:00:00.000 PM to 9/21/23 7:22:39.000 PM) No Event Sampling

No results found. Try expanding the time range.

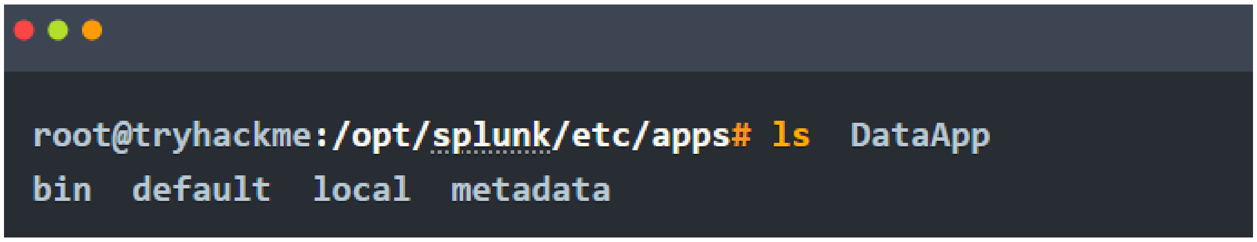
Understand the App Directory

Go to the app directory /opt/splunk/etc/apps , where we can locate our newly created app DataApp, as shown below



```
root@tryhackme:/opt/splunk/etc/apps# ls
DataApp                         splunk-dashboard-studio
SplunkForwarder                 splunk_archiver
SplunkLightForwarder            splunk_assist
alert_logevent                  splunk_essentials_9_0
alert_webhook                   splunk_gdi
appsbrowser                     splunk_httpinput
introspection_generator_addon   splunk_instrumentation
journald_input                  splunk_internal_metrics
launcher                        splunk_metrics_workspace
learned                         splunk_monitoring_console
legacy                          splunk_rapid_diag
python_upgrade_readiness_app    splunk_secure_gateway
sample_app                      user-prefs
search
```

Content Within the App Directory



```
root@tryhackme:/opt/splunk/etc/apps# ls  DataApp
bin  default  local  metadata
```

Splunk App Directory

File/Directory	Description

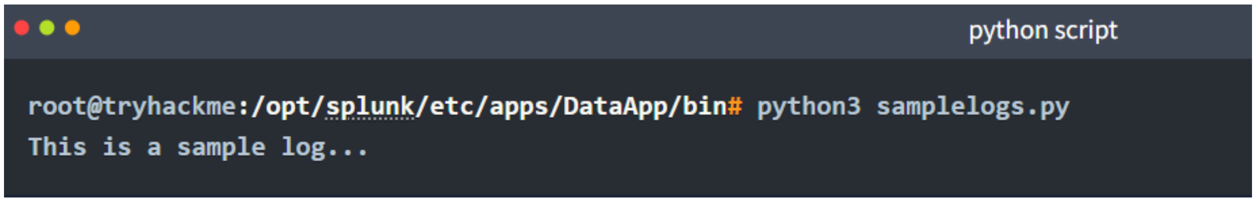
app.conf	Metadata file defining the app’s name, version, and more.
bin(directory)	Holds custom scripts or binaries required by the app.
default(directory)	Contains XML files defining app dashboards and views.
local(directory)	Optionally used for overriding default UI configurations.

Create a Python Script to Generate Sample Logs

As we learned that the bin directory contains the scripts required by the app, let's go to the bin directory and create a simple Python script using the command nano samplelogs.py, copy the following line in the file, and save.

```
print("This is a sample log...")
```

Let’s use python3 to run the file as shown below and see what output we get:



```
root@tryhackme:/opt/splunk/etc/apps/DataApp/bin# python3 samplelogs.py
This is a sample log...
```

It seems the script is ready. Note down the full path of the script file, that is /opt/splunk/etc/apps/DataApp/bin/samplelogs.py, which we will need later.

Creating inputs.conf

In the default directory, we will create all necessary configuration files like inputs.conf, transform.conf, etc. For now, let’s create an inputs.conf using the command nano inputs.conf add the following content into the file and save.

```
[script:///opt/splunk/etc/apps/DataApp/bin/samplelogs.py]
index = main
source = test_log
sourcetype = testing
interval = 5
```

The above configuration picks the output from the script samplelogs.py and sends it to Splunk with the index main every 5 seconds.

Restart Splunk using the command /opt/splunk/bin/splunk restart.

Summary
So far, we have created a simple Splunk app, used the bin directory to create a simple Python script, and then created inputs.conf file to pick the output of the script and throw the output into Splunk in the main index every 5 seconds. In the coming tasks, we will work on the scripts that will generate some events that will have visible parsing issues and then we will work with different configuration files to fix those parsing issues.

****

Answer the questions below:
If you create an App on Splunk named THM, what will be its full path on this machine?
Answer: /opt/splunk/etc/apps/THM

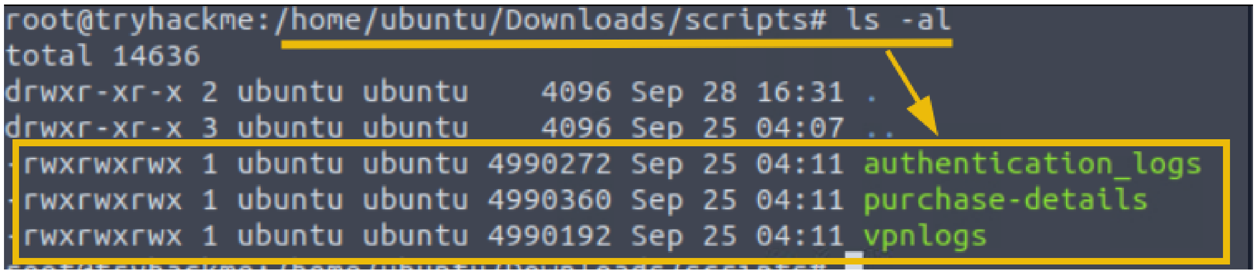
Event Boundaries - Understanding the problem

Event breaking in Splunk refers to breaking raw data into individual events based on specified boundaries. Splunk uses event-breaking rules to identify where one event ends, and the next begins. Let's walk through an example using a sample log to understand how event breaking works in Splunk.

Understanding the Events

In this room, we will be working on the DataApp created in the previous task and is placed at /opt/splunk/etc/apps/DataApp/.

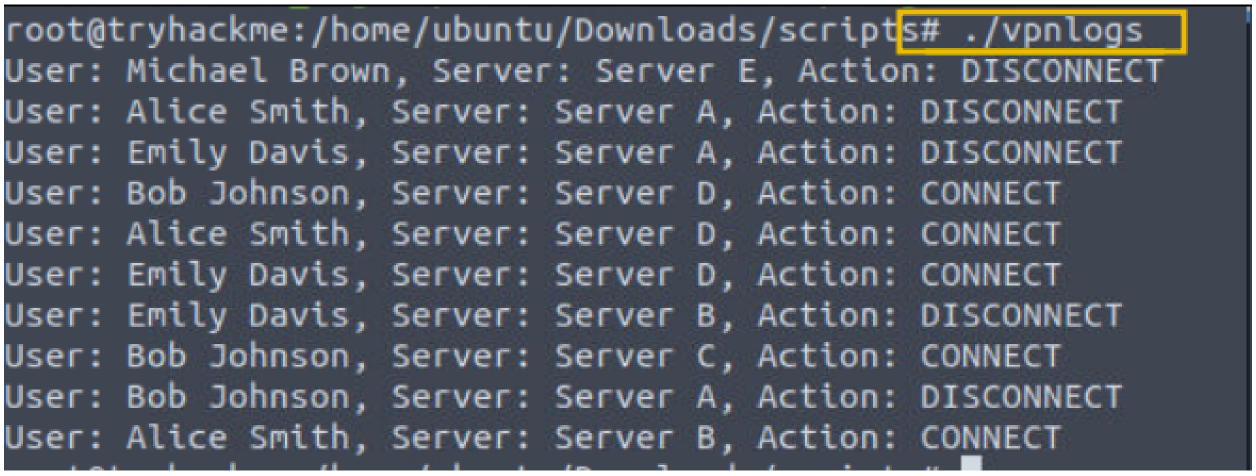
For this task, we will use the Python script vpnlogs from the ~/Downloads/scripts directory, as shown below:



```
root@tryhackme:/home/ubuntu/Downloads/scripts# ls -al
total 14636
drwxr-xr-x 2 ubuntu ubuntu    4096 Sep 28 16:31 .
drwxr-xr-x 3 ubuntu ubuntu    4096 Sep 25 04:07 ..
-rwxrwxrwx 1 ubuntu ubuntu 4990272 Sep 25 04:11 authentication_logs
-rwxrwxrwx 1 ubuntu ubuntu 4990360 Sep 25 04:11 purchase-details
-rwxrwxrwx 1 ubuntu ubuntu 4990192 Sep 25 04:11 vpnlogs
```

This directory contains various scripts, which we will explore later in this room. For now, let's focus on the vpnlogs script.

Let's say our client has a custom VPN application that generates VPN logs that contain information about the user, the VPN server, and the action performed on the connection, as shown in the output below when we run the command ./vpnlogs:



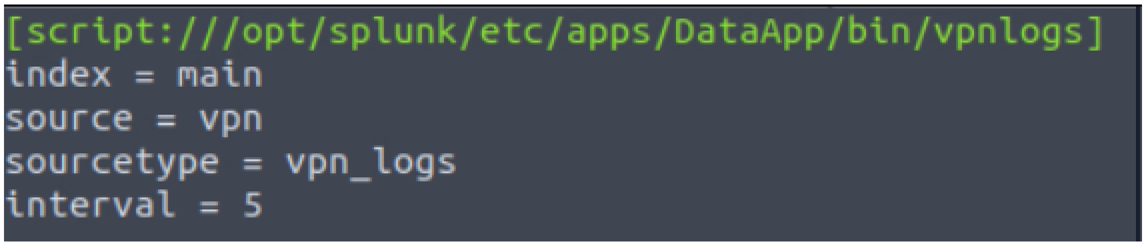
```
root@tryhackme:/home/ubuntu/Downloads/scripts# ./vpnlogs
User: Michael Brown, Server: Server E, Action: DISCONNECT
User: Alice Smith, Server: Server A, Action: DISCONNECT
User: Emily Davis, Server: Server A, Action: DISCONNECT
User: Bob Johnson, Server: Server D, Action: CONNECT
User: Alice Smith, Server: Server D, Action: CONNECT
User: Emily Davis, Server: Server D, Action: CONNECT
User: Emily Davis, Server: Server B, Action: DISCONNECT
User: Bob Johnson, Server: Server C, Action: CONNECT
User: Bob Johnson, Server: Server A, Action: DISCONNECT
User: Alice Smith, Server: Server B, Action: CONNECT
```

Generating Events

Our first task is to configure Splunk to ingest these VPN logs. Copy the vpnlogs script into the bin directory, open the inputs.conf , and write these lines:

```
[script:///opt/splunk/etc/apps/DataApp/bin/vpnlogs]
index = main
source = vpn
sourcetype = vpn_logs
interval = 5
```

The above lines tell Splunk to run the script vpnlogs every 5 seconds and send the output to the main index with sourcetype vpn_logs and host value as vpn_server. The inputs.conf file looks like this:



```
[script:///opt/splunk/etc/apps/DataApp/bin/vpnlogs]
index = main
source = vpn
sourcetype = vpn_logs
interval = 5
```

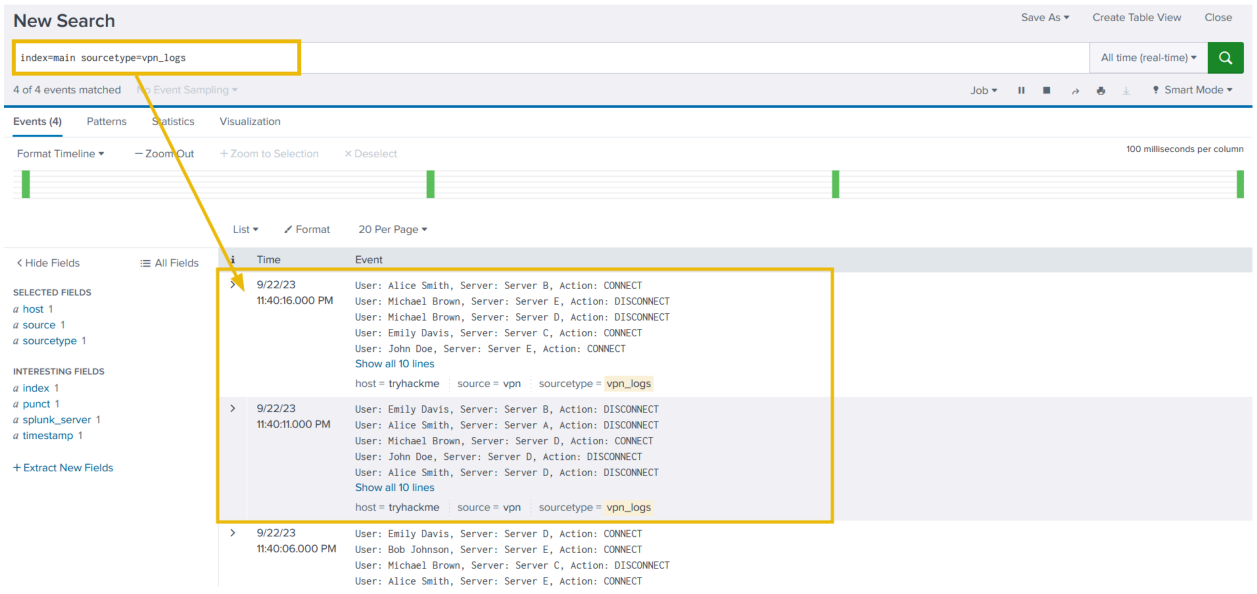
Restart Splunk

Save the file and restart Splunk using the command /opt/splunk/bin/splunk restart. Open the Splunk instance at MACHINE_IP:8000 and navigate to the search head.

Search Head

Select the time range All time (Real-time) and use the following search query to see if we are getting the logs.

Search Query: index=main sourcetype=vpn_logs



```
index=main sourcetype=vpn_logs
```

```
Time                 Event
9/22/23              User: Alice Smith, Server: Server B, Action: CONNECT
11:40:16.000 PM      User: Michael Brown, Server: Server E, Action: DISCONNECT
                     User: Michael Brown, Server: Server D, Action: DISCONNECT
                     User: Emily Davis, Server: Server C, Action: CONNECT
                     User: John Doe, Server: Server E, Action: CONNECT
                     Show all 10 lines
                     host = tryhackme   source = vpn   sourcetype = vpn_logs
9/22/23              User: Emily Davis, Server: Server B, Action: DISCONNECT
11:40:11.000 PM      User: Alice Smith, Server: Server A, Action: DISCONNECT
                     User: Michael Brown, Server: Server D, Action: CONNECT
                     User: John Doe, Server: Server D, Action: DISCONNECT
                     User: Alice Smith, Server: Server D, Action: DISCONNECT
                     Show all 10 lines
                     host = tryhackme   source = vpn   sourcetype = vpn_logs
9/22/23              User: Emily Davis, Server: Server D, Action: CONNECT
11:40:06.000 PM      User: Bob Johnson, Server: Server E, Action: CONNECT
                     User: Michael Brown, Server: Server C, Action: DISCONNECT
                     User: Alice Smith, Server: Server E, Action: CONNECT
```

Identifying the Problem

Excellent, we are getting the VPN logs after every 5 seconds. But can you observe the problem? It's evident that Splunk cannot determine the boundaries of each event and considers multiple events as a single event. By default, Splunk breaks the event after carriage return.

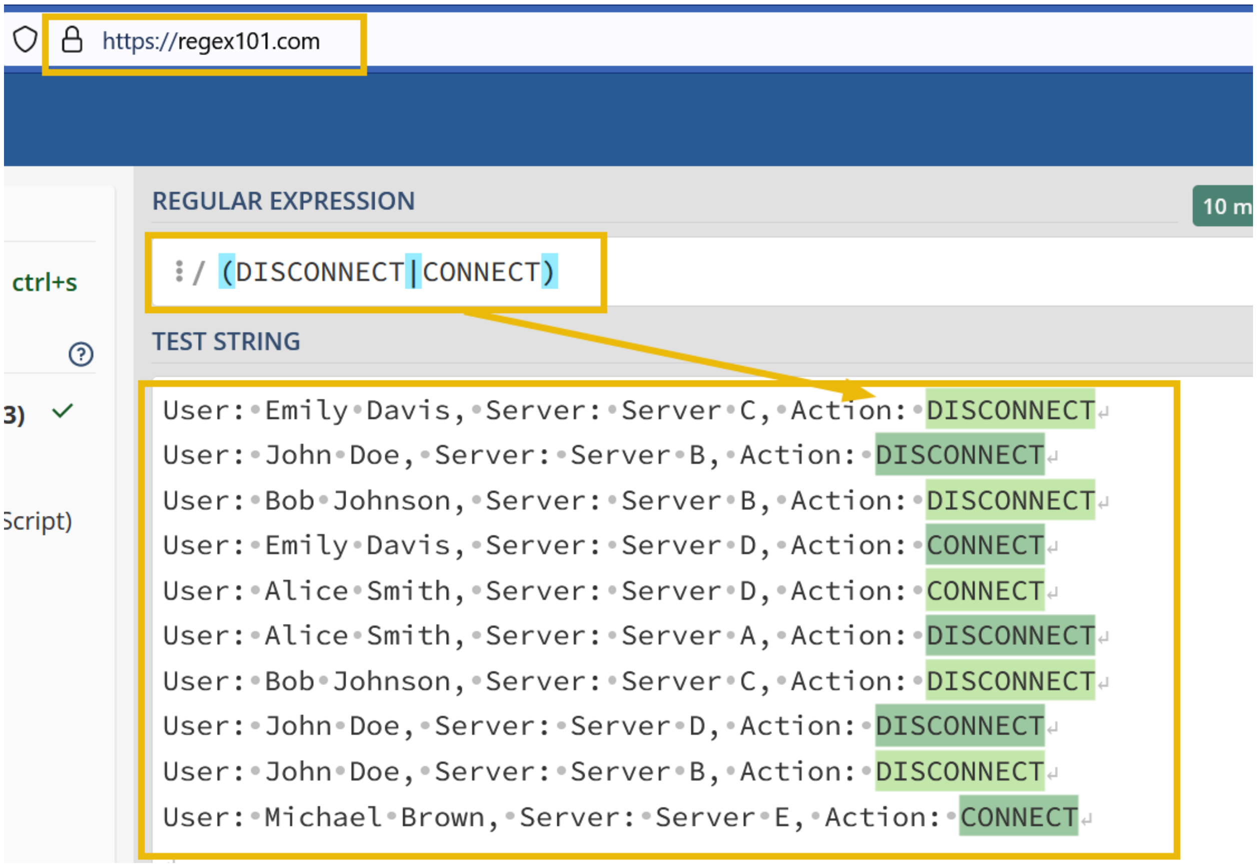
Fixing the Event Boundary

We need to fix the event boundary. To configure Splunk to break the events in this case, we have to make some changes to the props.conf file. First, we will create a regex to determine the end of the event. The sample events are shown below:

Sample Events

```
User: Emily Davis, Server: Server C, Action: DISCONNECT
User: John Doe, Server: Server B, Action: DISCONNECT
User: Bob Johnson, Server: Server B, Action: DISCONNECT
User: Emily Davis, Server: Server D, Action: CONNECT
User: Alice Smith, Server: Server D, Action: CONNECT
User: Alice Smith, Server: Server A, Action: DISCONNECT
User: Bob Johnson, Server: Server C, Action: DISCONNECT
User: John Doe, Server: Server D, Action: DISCONNECT
User: John Doe, Server: Server B, Action: DISCONNECT
User: Michael Brown, Server: Server E, Action: CONNECT
```

We will use reg101.com to create a regex pattern. If we look closely, all events end with the terms DISCONNECT or CONNECT. We can use this information to create a regex pattern (DISCONNECT|CONNECT) , as shown below:



```
/ (DISCONNECT|CONNECT)
```

```
User: Emily Davis, Server: Server C, Action: DISCONNECT
User: John Doe, Server: Server B, Action: DISCONNECT
User: Bob Johnson, Server: Server B, Action: DISCONNECT
User: Emily Davis, Server: Server D, Action: CONNECT
User: Alice Smith, Server: Server D, Action: CONNECT
User: Alice Smith, Server: Server A, Action: DISCONNECT
User: Bob Johnson, Server: Server C, Action: DISCONNECT
User: John Doe, Server: Server D, Action: DISCONNECT
User: John Doe, Server: Server B, Action: DISCONNECT
User: Michael Brown, Server: Server E, Action: CONNECT
```

Now, let's create a props.conf in the default directory within the DataApp and add the following lines:

```
[vpn_logs]
SHOULD_LINEMERGE = true
MUST_BREAK_AFTER = (DISCONNECT|CONNECT)
```

This configuration tells Splunk to take the sourcetype to merge all lines and it must break the events when you see the pattern matched in the mentioned regex.

Restart Splunk

Save the file and restart Splunk using the command /opt/bin/splunk restart. Open the Splunk instance at MACHINE_IP:8000 and navigate to the search head.

That's it. We can see that with a few changes in the props.conf file, we changed how Splunk broke these VPN logs generated by the custom vpn_server.

In the next task, we will look at a different case study.

****
Answer the questions below:
Which configuration file is used to specify parsing rules?
Answer: props.conf

What regex is used in the above case to break the Events?
Answer: (DISCONNECT|CONNECT)

Which stanza is used in the configuration to force Splunk to break the event after the specified pattern?
Answer: MUST_BREAK_AFTER

If we want to disable line merging, what will be the value of the stanza SHOULD_LINEMERGE?
Answer: false

Parsing Multi-Line Events

As we know, different log sources have their own ways of generating logs. What if a log source generates event logs that comprise multi-lines? One such example is Windows Event logs. In order to understand how multi-line events can be handled in Splunk, we will use the event logs generated from the script authentication_logs. The sample event log is shown below:

```
[Authentication]:A login attempt was observed from the user Michael Brown and machine MAC_01
at: Mon Jul 17 08:10:12 2023 which belongs to the Custom department. The login attempt looks suspicious.
```

As it is clearly shown, the event contains multiple lines. Let’s update the inputs.conf file to include this script and see if Splunk is able to break the event as intended.

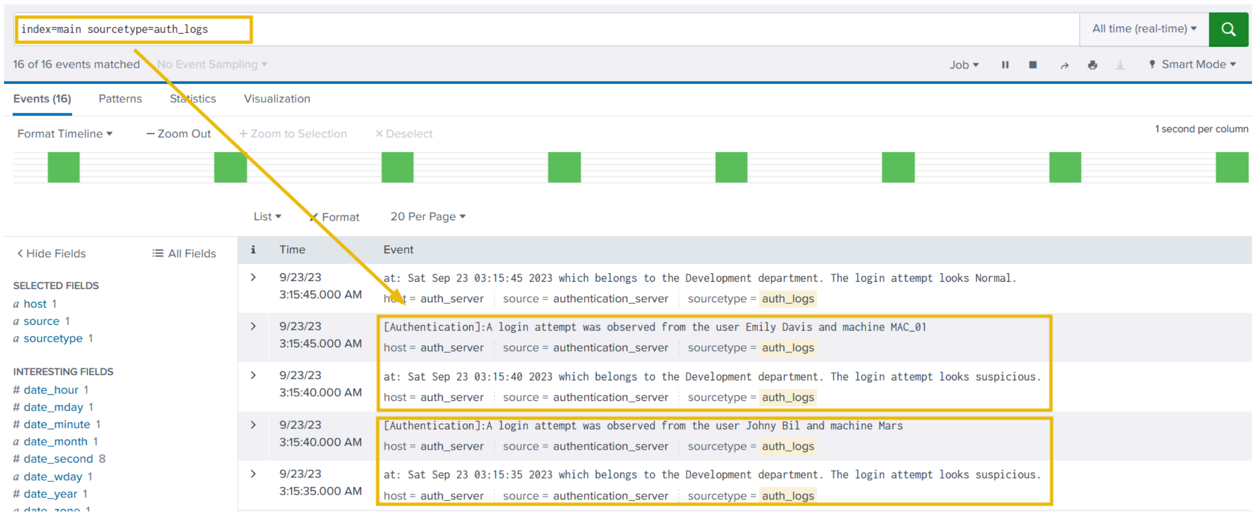
Copy the authentication_logs script from the ~/Downloads/scripts directory into the bin folder of the DataApp and add the following lines in inputs.conf, save the file, and restart Splunk:

```
[script:///opt/splunk/etc/apps/DataApp/bin/authentication_logs]
interval = 5
index = main
sourcetype= auth_logs
host = auth_server
```

Search Head

Let’s look at the Splunk Search head to see how these logs are reflected.

Search Query: index=main sourcetype = auth_logs



i	Time	Event
>	9/23/23 3:15:45.000 AM	at: Sat Sep 23 03:15:45 2023 which belongs to the Development department. The login attempt looks Normal. host = auth_server source = authentication_server sourcetype = auth_logs
>	9/23/23 3:15:45.000 AM	[Authentication]:A login attempt was observed from the user Emily Davis and machine MAC_01 host = auth_server source = authentication_server sourcetype = auth_logs
>	9/23/23 3:15:40.000 AM	at: Sat Sep 23 03:15:40 2023 which belongs to the Development department. The login attempt looks suspicious. host = auth_server source = authentication_server sourcetype = auth_logs
>	9/23/23 3:15:40.000 AM	[Authentication]:A login attempt was observed from the user Johny Bil and machine Mars host = auth_server source = authentication_server sourcetype = auth_logs
>	9/23/23 3:15:35.000 AM	at: Sat Sep 23 03:15:35 2023 which belongs to the Development department. The login attempt looks suspicious. host = auth_server source = authentication_server sourcetype = auth_logs

Identifying the Problem

If we observe the events, we will see that Splunk is breaking the 2-line Event into 2 different events and is unable to determine the boundaries.

Fixing the Event Boundary

In order to fix this issue, we can use different stanzas in the props.conf file. If we run the script a few times to observe the output, we can see that each event starts with the term [Authentication], indicating the start of the event. We can use this as the regex pattern with the stanza BREAK_ONLY_BEFORE and see if it could fix this problem. Copy the following lines in props.conf file, save the file, and then restart Splunk to apply changes.

```
[auth_logs]
SHOULD_LINEMERGE = true
BREAK_ONLY_BEFORE = \[Authentication\]
```

****

Answer the questions below:

Which stanza is used to break the event boundary before a pattern is specified in the above case?

Answer: BREAK_ONLY_BEFORE

Which regex pattern is used to identify the event boundaries in the above case?
Answer: \[Authentication\]

Masking Sensitive Data

Masking sensitive fields, such as credit card numbers, is essential for maintaining compliance with standards like PCI DSS (Payment Card Industry Data Security Standard) and HIPAA (Health Insurance Portability and Accountability Act). Splunk provides features like field masking and anonymization to protect sensitive data. Here's an example of credit card numbers being populated in the Event logs generated by the script purchase-details present in the ~/Downloads/scripts directory.

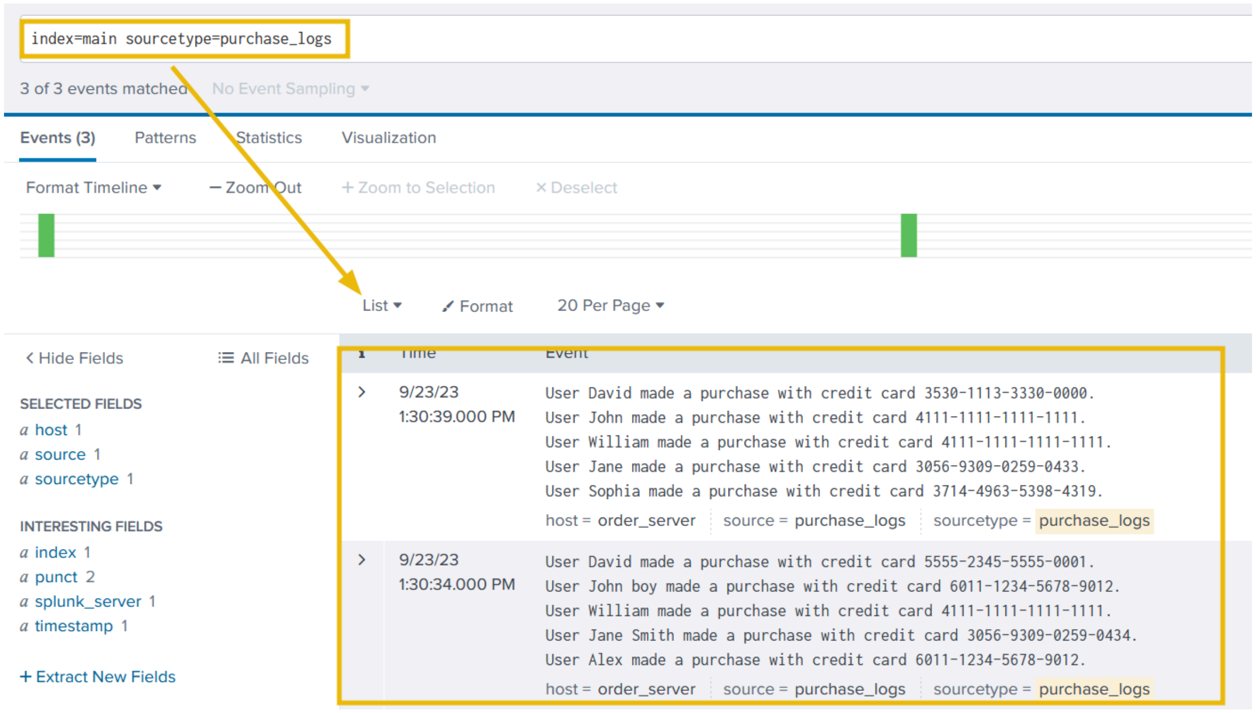
Sample Output

```
User William made a purchase with credit card 3714-4963-5398-4313.
User John Boy made a purchase with credit card 3530-1113-3330-0000.
User Alice Johnson made a purchase with credit card 6011-1234-5678-9012.
User David made a purchase with credit card 3530-1113-3330-0000.
User Bob Williams made a purchase with credit card 9876-5432-1098-7654.
```

Copy this script file into the bin folder of the DataApp and configure the inputs.conf file to ingest these logs into Splunk. To do so, add the following lines in the inputs.conf file.

```
[script:///opt/splunk/etc/apps/DataApp/bin/purchase-details]
interval = 5
index = main
source = purchase_logs
sourcetype= purchase_logs
host = order_server
```

This configuration tells Splunk to get the output from the purchase-details script, and index into the main index every 5 seconds, with sourcetype purchase_logs and host as order_server. Now, save the file and restart Splunk. Log on to Splunk and apply the following search query: Search Query: index=main sourcetype=purchase_logs



```
index=main sourcetype=purchase_logs
```

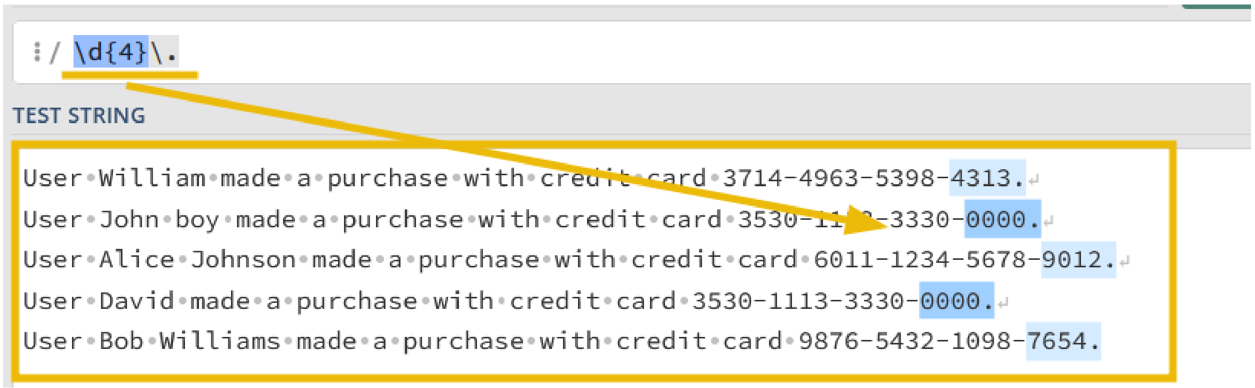
3 of 3 events matched

i	Time	Event
>	9/23/23 1:30:39.000 PM	User David made a purchase with credit card 3530-1113-3330-0000. User John made a purchase with credit card 4111-1111-1111-1111. User William made a purchase with credit card 4111-1111-1111-1111. User Jane made a purchase with credit card 3056-9309-0259-0433. User Sophia made a purchase with credit card 3714-4963-5398-4319. host = order_server source = purchase_logs sourcetype = purchase_logs
>	9/23/23 1:30:34.000 PM	User David made a purchase with credit card 5555-2345-5555-0001. User John boy made a purchase with credit card 6011-1234-5678-9012. User William made a purchase with credit card 4111-1111-1111-1111. User Jane Smith made a purchase with credit card 3056-9309-0259-0434. User Alex made a purchase with credit card 6011-1234-5678-9012. host = order_server source = purchase_logs sourcetype = purchase_logs

It looks like we have two problems to address. We need to hide the credit card information that is being added to each event and also need to fix the event boundaries.

Fixing Event Boundaries

We will use regex101.com to create a regex pattern to identify the end boundary of each event, as shown below:



```
/ \d{4}\.
```

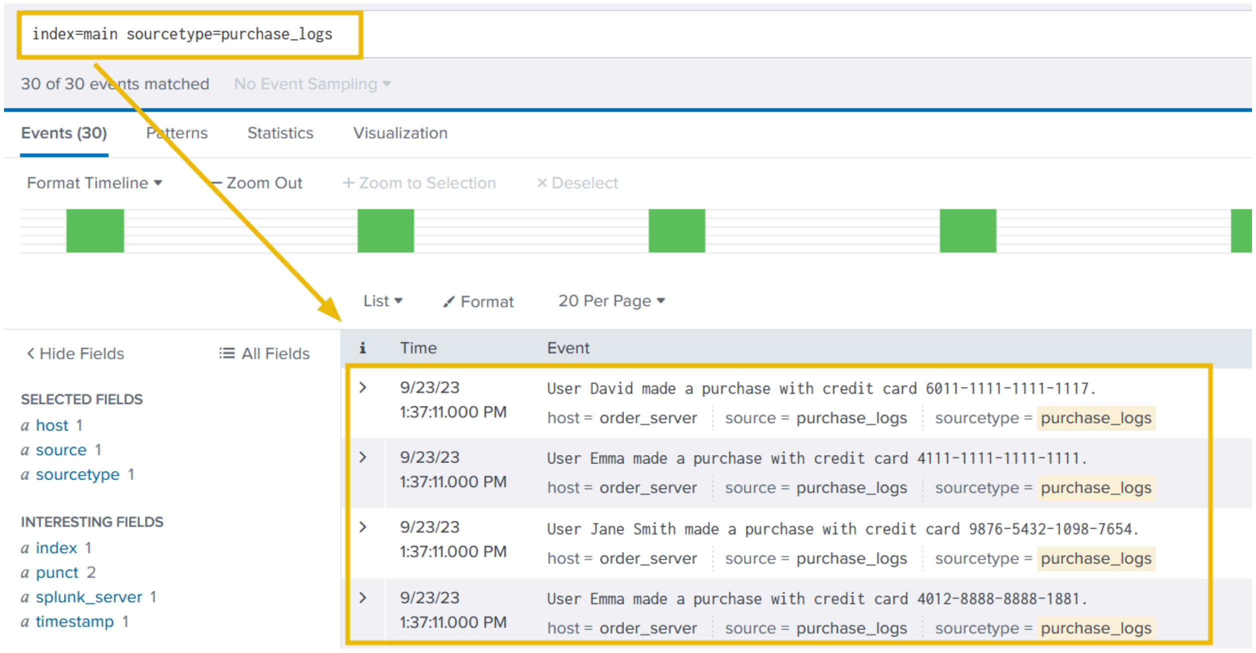
TEST STRING

```
User William made a purchase with credit card 3714-4963-5398-4313.
User John boy made a purchase with credit card 3530-1113-3330-0000.
User Alice Johnson made a purchase with credit card 6011-1234-5678-9012.
User David made a purchase with credit card 3530-1113-3330-0000.
User Bob Williams made a purchase with credit card 9876-5432-1098-7654.
```

Let's update the props.conf, as shown below:

```
[purchase_logs]
SHOULD_LINEMERGE = true
MUST_BREAK_AFTER = \d{4}\.
```

Save the file, and restart Splunk. If everything goes well, the event should be propagating correctly, as shown below:



```
index=main sourcetype=purchase_logs
```

30 of 30 events matched

SELECTED FIELDS
- a host 1
- a source 1
- a sourcetype 1

INTERESTING FIELDS
- a index 1
- a punct 2
- a splunk_server 1
- a timestamp 1

i	Time	Event
>	9/23/23 1:37:11.000 PM	User David made a purchase with credit card 6011-1111-1111-1117. host = order_server source = purchase_logs sourcetype = purchase_logs
>	9/23/23 1:37:11.000 PM	User Emma made a purchase with credit card 4111-1111-1111-1111. host = order_server source = purchase_logs sourcetype = purchase_logs
>	9/23/23 1:37:11.000 PM	User Jane Smith made a purchase with credit card 9876-5432-1098-7654. host = order_server source = purchase_logs sourcetype = purchase_logs
>	9/23/23 1:37:11.000 PM	User Emma made a purchase with credit card 4012-8888-8888-1881. host = order_server source = purchase_logs sourcetype = purchase_logs

Now that we have fixed the event boundary issue. It's time to mask the sensitive information from the events.

Introducing SEDCMD

In Splunk, the sedcmd configuration setting is used in the props.conf file to modify or transform data during indexing. It allows us to apply regular expression-based substitutions on the incoming data before indexing it. The sedcmd setting uses the syntax and functionality of the Unix sed command.

Here's a brief explanation of how the sedcmd works in props.conf:

1. Open the props.conf file in your Splunk configuration directory.
2. Locate or create a stanza for the data source you want to modify.
3. Add the sedcmd setting under the stanza.
4. Specify the regular expression pattern and the replacement string using the s/ syntax similar to the sed command.

Here's an example of using sedcmd in props.conf to modify a field called myField:

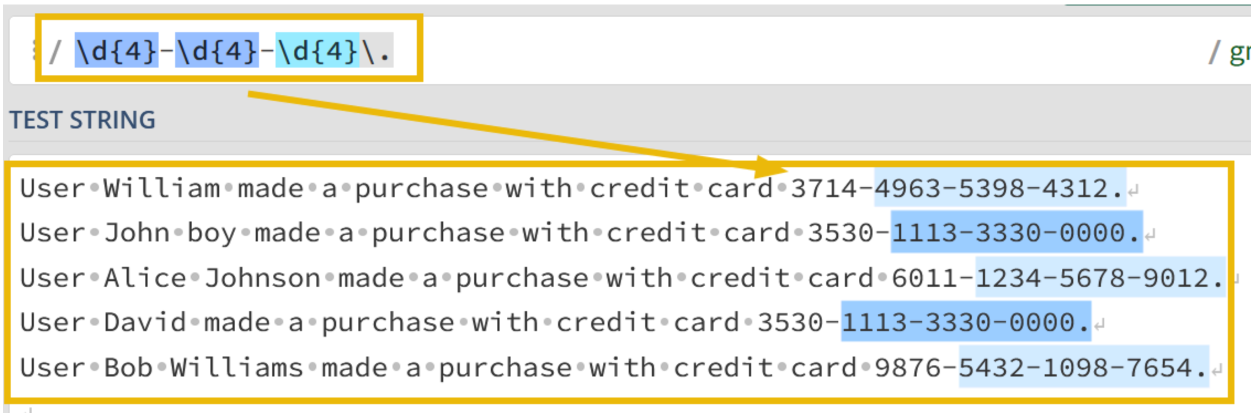
```
[source::/path/to/your/data]
SEDCMD-myField = s/oldValue/newValue/g
```

In this example, the sedcmd setting is applied to the data from a specific source path. It uses the regular expression pattern oldValue and replaces it globally with newValue using the g flag in the myField field. This transformation occurs before Splunk indexes the data.

It is important to note that, this sedcmd is just one of the configuration settings props.conf used for data transformation. There are other options available, such as REGEX, TRANSFORMS, etc.

Masking CC Information

Let's now use the above knowledge gain to create a regex that replaces the credit card number with something like this -> 6011-XXXX-XXXX-XXXX., as shown below:



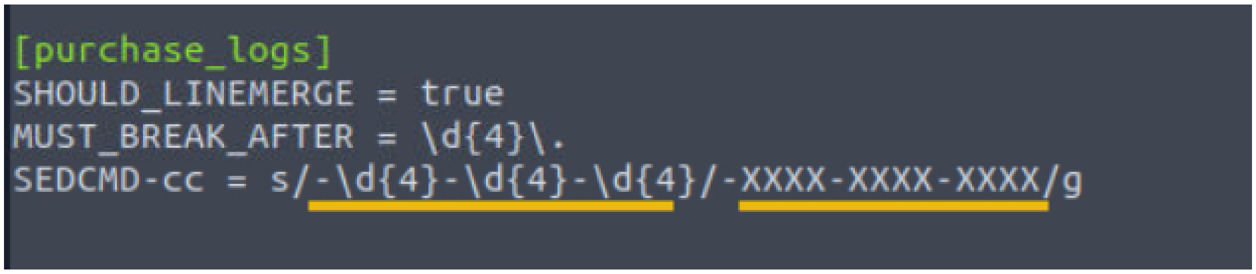
```
/ \d{4}-\d{4}-\d{4}\.                                                  / gi
```

TEST STRING

```
User William made a purchase with credit card 3714-4963-5398-4312.
User John boy made a purchase with credit card 3530-1113-3330-0000.
User Alice Johnson made a purchase with credit card 6011-1234-5678-9012.
User David made a purchase with credit card 3530-1113-3330-0000.
User Bob Williams made a purchase with credit card 9876-5432-1098-7654.
```

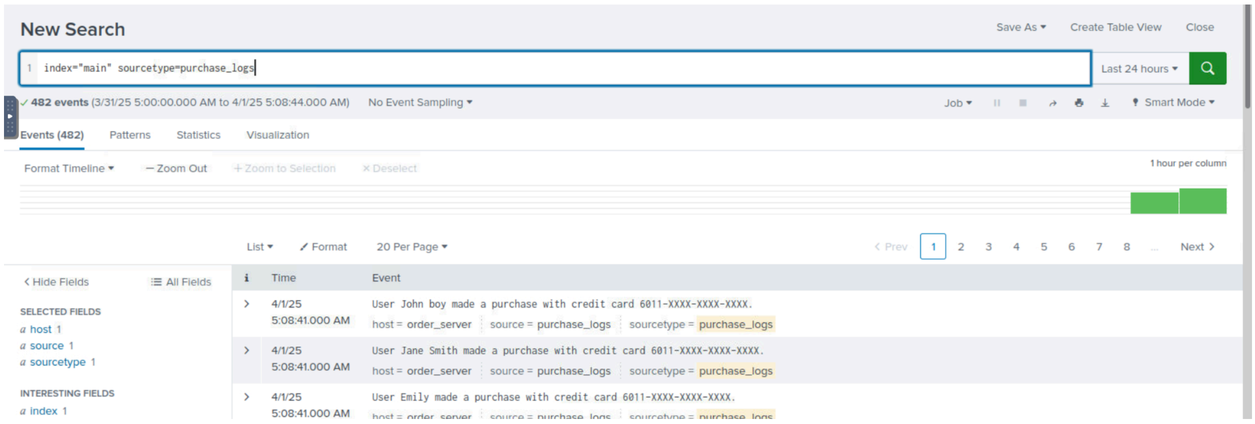
Now, our task is to use this s/OLD_VALUE>/<NEW_VALUE>/g regex in sedcmd to replace the credit card numbers with XXXX-XXXX-XXXX. The final sedcmd value will become s/-\d{4}-\d{4}-\d{4}/-XXXX-XXXX-XXXX/g

Our configuration in the props.conf would look like this:



```
[purchase_logs]
SHOULD_LINEMERGE = true
MUST_BREAK_AFTER = \d{4}\.
SEDCMD-cc = s/-\d{4}-\d{4}-\d{4}/-XXXX-XXXX-XXXX/g
```

Restart Splunk and check Splunk Instance to see how our changes are reflected in the logs.



```
index="main" sourcetype=purchase_logs
```

✓ 482 events (3/31/25 5:00:00.000 AM to 4/1/25 5:08:44.000 AM)

Time	Event
4/1/25 5:08:41.000 AM	User John boy made a purchase with credit card 6011-XXXX-XXXX-XXXX. host = order_server source = purchase_logs sourcetype = purchase_logs
4/1/25 5:08:41.000 AM	User Jane Smith made a purchase with credit card 6011-XXXX-XXXX-XXXX. host = order_server source = purchase_logs sourcetype = purchase_logs
4/1/25 5:08:41.000 AM	User Emily made a purchase with credit card 6011-XXXX-XXXX-XXXX. host = order_server source = purchase_logs sourcetype = purchase_logs

Great. With some changes in the configurations, we were able to mask the sensitive information. As a SOC analyst, it is important to understand the criticality of masking sensitive information before being logged in order to comply with standards like HIPAA, PCI-DSS, etc.

****

Answer the questions below:

Which stanza is used to break the event after the specified regex pattern?

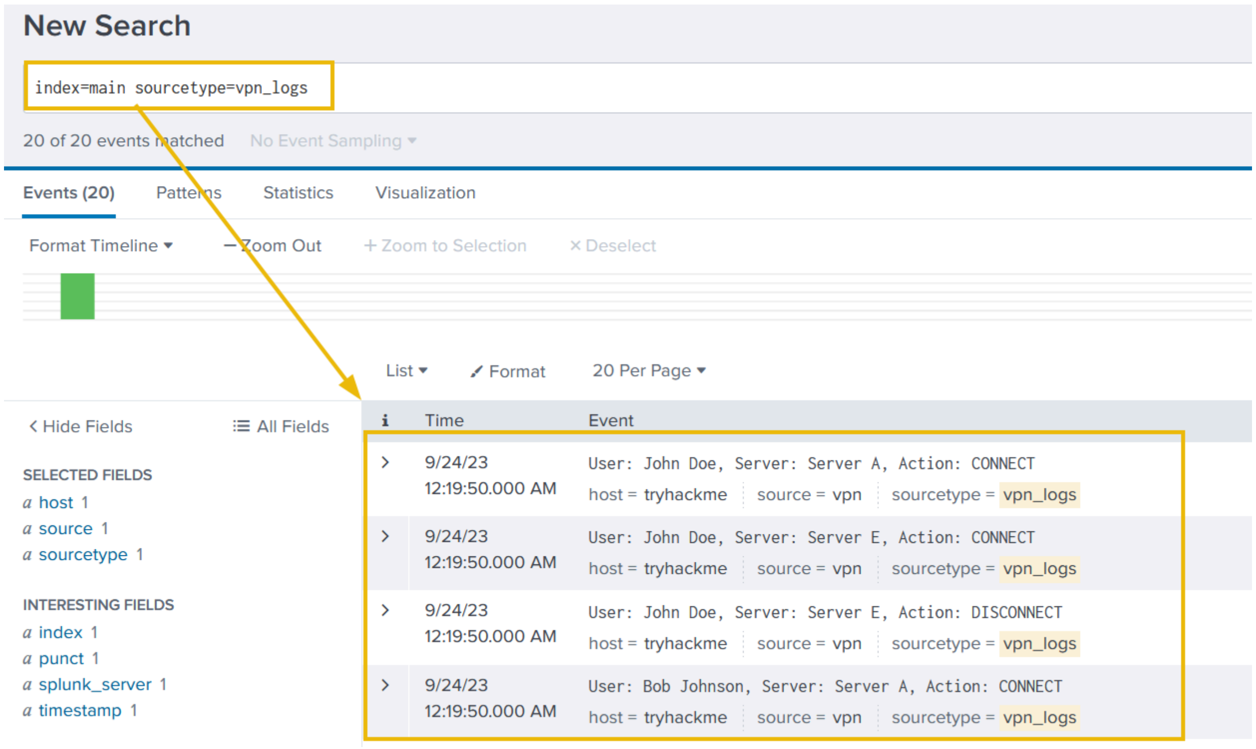
Answer: MUST_BREAK_AFTER

What is the pattern of using SEDCMD in the props.conf to mask or replace the sensitive fields?

Answer: s/oldValue/newValue/g

Extracting Custom Fields

From a SOC analyst's point of view, we would often encounter logs either custom log sources, where not all fields are extracted by the SIEM automatically, or we are required to extract custom fields to improve the analysis. In that case, we need a way to extract custom fields from the logs. To demonstrate this with an example, let's go back to our vpn_logs case. The output we are getting in Splunk is, as shown below:


```
index=main sourcetype=vpn_logs
```

20 of 20 events matched

Time	Event
9/24/23 12:19:50.000 AM	User: John Doe, Server: Server A, Action: CONNECT host = tryhackme source = vpn sourcetype = vpn_logs
9/24/23 12:19:50.000 AM	User: John Doe, Server: Server E, Action: CONNECT host = tryhackme source = vpn sourcetype = vpn_logs
9/24/23 12:19:50.000 AM	User: John Doe, Server: Server E, Action: DISCONNECT host = tryhackme source = vpn sourcetype = vpn_logs
9/24/23 12:19:50.000 AM	User: Bob Johnson, Server: Server A, Action: CONNECT host = tryhackme source = vpn sourcetype = vpn_logs

It's clear that none of the fields are extracted automatically, and we can not perform any analysis on these events until fields like username, server, and action are extracted.

Extracting Username

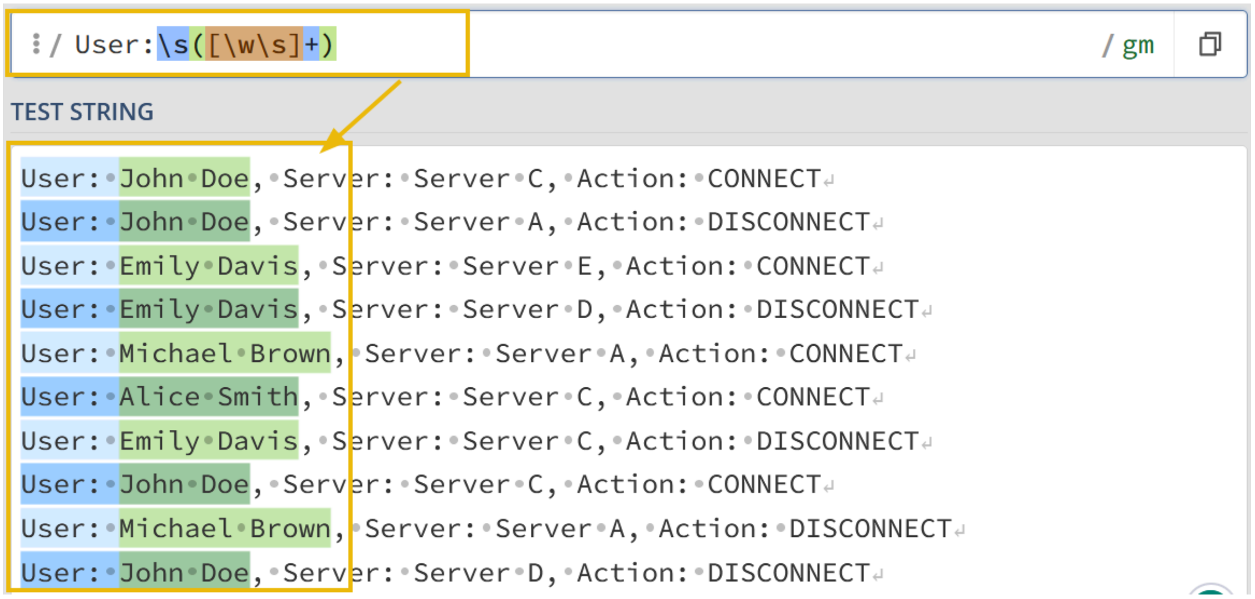
Let’s first go through the process of extracting the usernames and putting them under the field as Username, and then we can follow the same steps to extract other fields as well.

Creating RegEx Pattern

Our first task would be to create a regex pattern to capture the username values we are trying to capture. Sample event logs look like this:

```
User: John Doe, Server: Server C, Action: CONNECT
User: John Doe, Server: Server A, Action: DISCONNECT
User: Emily Davis, Server: Server E, Action: CONNECT
User: Emily Davis, Server: Server D, Action: DISCONNECT
User: Michael Brown, Server: Server A, Action: CONNECT
User: Alice Smith, Server: Server C, Action: CONNECT
User: Emily Davis, Server: Server C, Action: DISCONNECT
User: John Doe, Server: Server C, Action: CONNECT
User: Michael Brown, Server: Server A, Action: DISCONNECT
User: John Doe, Server: Server D, Action: DISCONNECT
```

By creating a regex pattern as: User:\s([\w\s]+) and creating a capturing group, we have successfully captured all the usernames that we want to extract.



```
/ User:\s([\w\s]+) / gm
```

TEST STRING

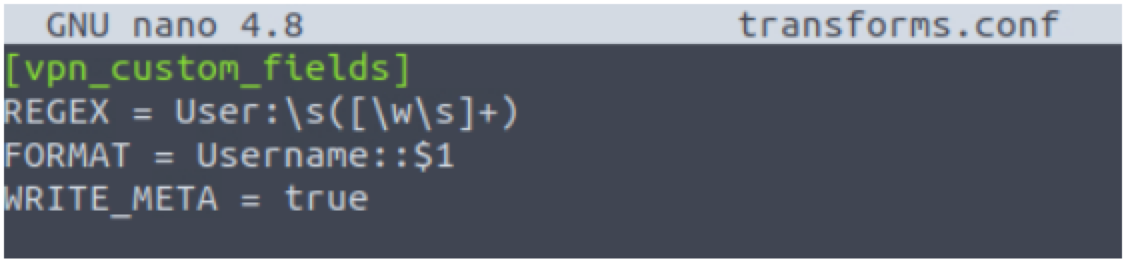
```
User: John Doe, Server: Server C, Action: CONNECT
User: John Doe, Server: Server A, Action: DISCONNECT
User: Emily Davis, Server: Server E, Action: CONNECT
User: Emily Davis, Server: Server D, Action: DISCONNECT
User: Michael Brown, Server: Server A, Action: CONNECT
User: Alice Smith, Server: Server C, Action: CONNECT
User: Emily Davis, Server: Server C, Action: DISCONNECT
User: John Doe, Server: Server C, Action: CONNECT
User: Michael Brown, Server: Server A, Action: DISCONNECT
User: John Doe, Server: Server D, Action: DISCONNECT
```

Creating and Updating transforms.conf

Now, let's create a transforms.conf in the default folder of the DataApp directory, and put the following configurations in it as it is.

```
[vpn_custom_fields]
REGEX = User:\s([\w\s]+)
FORMAT = Username::$1
WRITE_META = true
```

The transforms.conf would look like this:

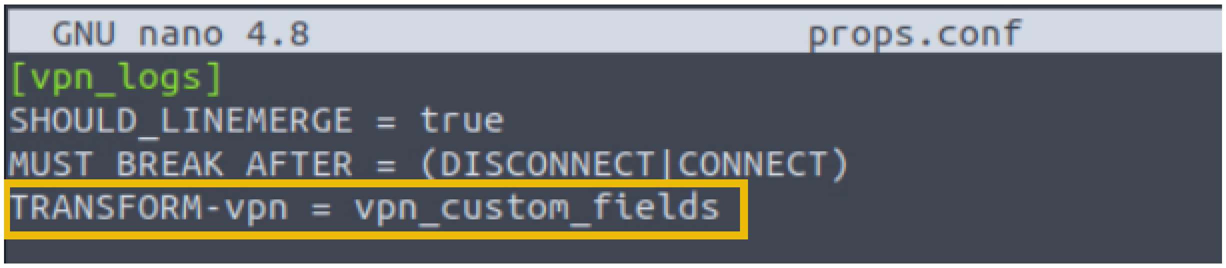


```
  GNU nano 4.8                        transforms.conf
[vpn_custom_fields]
REGEX = User:\s([\w\s]+)
FORMAT = Username::$1
WRITE_META = true
```

Explanation: We have created a custom identifier vpn_custom_fields, used the regex pattern to pull the usernames from the logs, mentioned the field name as Username, and asked to capture the first group by referring to it as $1. Save the configuration and move to the next step.

Updating props.conf

We need to update the props.conf to mention the recent updates we did in transforms.conf. Here, we are appending the configuration for sourcetype vpn_logs with the line TRANSFORM-vpn = vpn_custom_fields, as shown below:



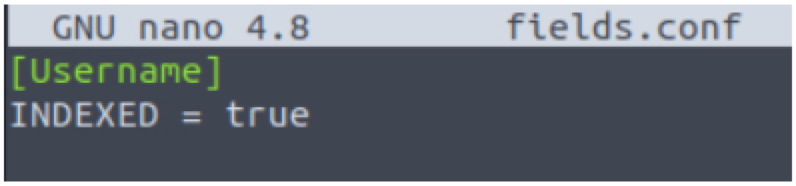
```
  GNU nano 4.8                           props.conf
[vpn_logs]
SHOULD_LINEMERGE = true
MUST_BREAK_AFTER = (DISCONNECT|CONNECT)
TRANSFORM-vpn = vpn_custom_fields
```

Creating and Updating fields.conf

The next step would be to create fields.conf and mention the field we are going to extract from the logs, which is Username. INDEXED = true means we are telling Splunk to extract this field at the indexed time.

```
[Username]
INDEXED = true
```

Fields.conf file would look like this:



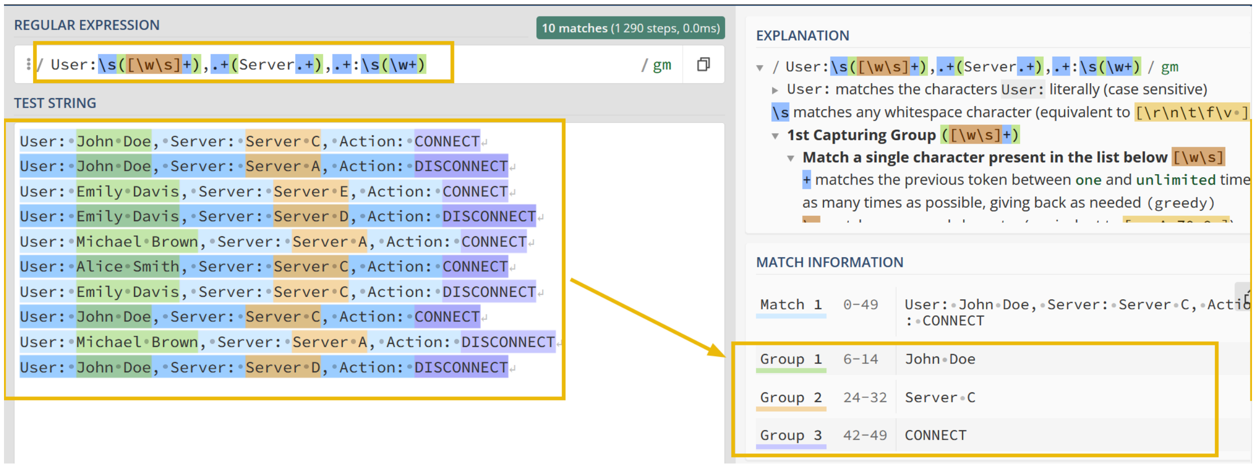
```
  GNU nano 4.8              fields.conf
[Username]
INDEXED = true
```

Restart Splunk

That's all we need in order to extract the custom fields. Now, restart the Splunk instance so that the changes we have made are committed. Go to the Splunk instance and use the search query index=main sourcetype=vpn_logs
This is it. With some changes to the configuration files, we were able to extract a custom field from the logs.

Let's use the same process and extract the remaining two fields as well.

Creating RegEx Pattern
This regex pattern User:\s([\w\s]+),.+(Server.+),.+:\s(\w+) captures all the three fields and places them into the groups, as shown below:



REGULAR EXPRESSION   10 matches (1 290 steps, 0.0ms)

```
/ User:\s([\w\s]+),.+(Server.+),.+:\s(\w+)    / gm
```

TEST STRING

```
User: John Doe, Server: Server C, Action: CONNECT
User: John Doe, Server: Server A, Action: DISCONNECT
User: Emily Davis, Server: Server E, Action: CONNECT
User: Emily Davis, Server: Server D, Action: DISCONNECT
User: Michael Brown, Server: Server A, Action: CONNECT
User: Alice Smith, Server: Server C, Action: CONNECT
User: Emily Davis, Server: Server C, Action: DISCONNECT
User: John Doe, Server: Server C, Action: CONNECT
User: Michael Brown, Server: Server A, Action: DISCONNECT
User: John Doe, Server: Server D, Action: DISCONNECT
```

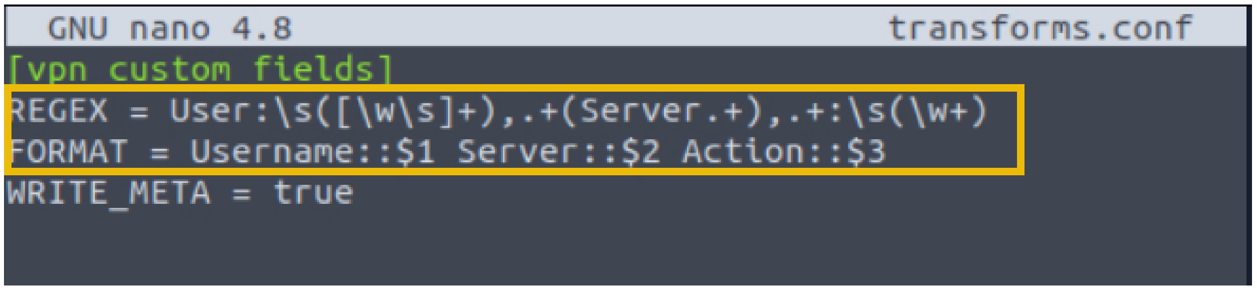
EXPLANATION

/ User:\s([\w\s]+),.+(Server.+),.+:\s(\w+) / gm
User: matches the characters User: literally (case sensitive)
\s matches any whitespace character (equivalent to [\r\n\t\f\v]
1st Capturing Group ([\w\s]+)
Match a single character present in the list below [\w\s]
+ matches the previous token between one and unlimited time
as many times as possible, giving back as needed (greedy)

MATCH INFORMATION

Match 1	0-49	User: John Doe, Server: Server C, Acti : CONNECT
Group 1	6-14	John Doe
Group 2	24-32	Server C
Group 3	42-49	CONNECT

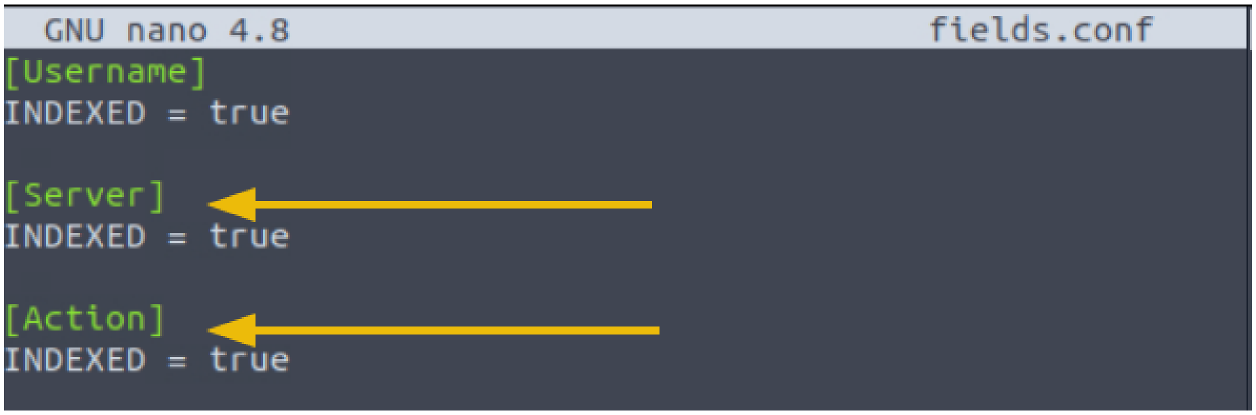
Updating transforms.conf
Now that we have captured the fields that we want to extract, let's update the transforms.conf file, as shown below:



```
  GNU nano 4.8                                  transforms.conf
[vpn_custom_fields]
REGEX = User:\s([\w\s]+),.+(Server.+),.+:\s(\w+)
FORMAT = Username::$1 Server::$2 Action::$3
WRITE_META = true
```

In the configuration file, we have updated the REGEX pattern and the FORMAT, where we have specified different fields separating with a space.

Updating fields.conf
Now it's time to update the fields.conf with the field names that we want Splunk to extract at index time.



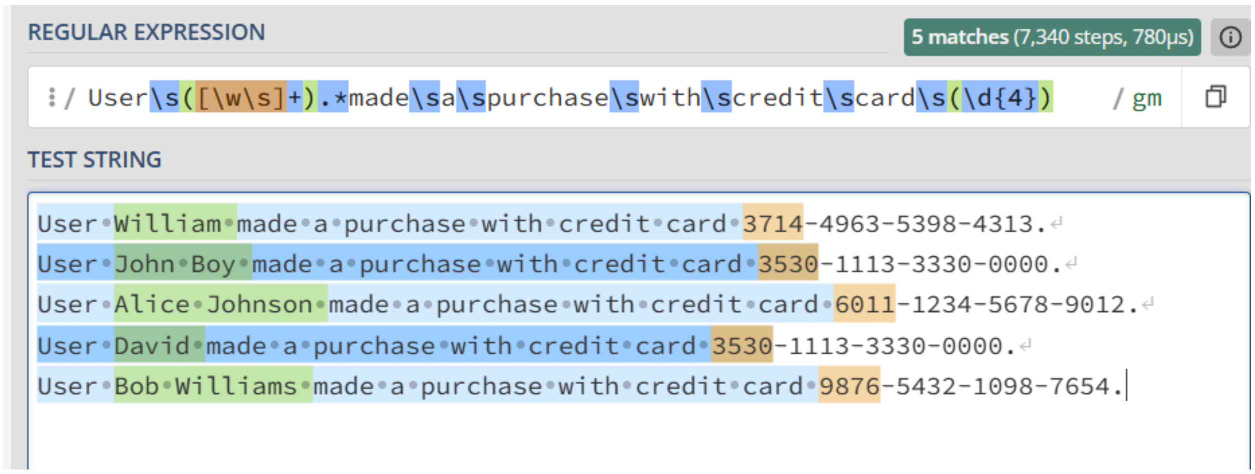
```
GNU nano 4.8                                   fields.conf
[Username]
INDEXED = true

[Server]
INDEXED = true

[Action]
INDEXED = true
```

****

Answer the questions below:

Create a regex pattern to extract all three fields from the VPN logs.

Answer: No answer needed



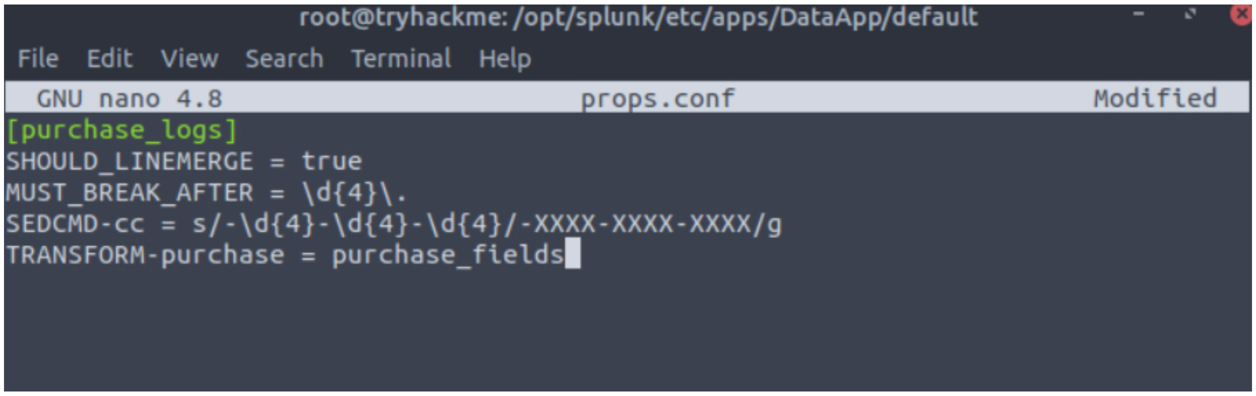
```
/ User\s([\w\s]+).*made\sa\spurchase\swith\scredit\scard\s(\d{4})   / gm
```

```
User William made a purchase with credit card 3714-4963-5398-4313.
User John Boy made a purchase with credit card 3530-1113-3330-0000.
User Alice Johnson made a purchase with credit card 6011-1234-5678-9012.
User David made a purchase with credit card 3530-1113-3330-0000.
User Bob Williams made a purchase with credit card 9876-5432-1098-7654.
```

Extract the Username field from the sourcetype purchase_logs we worked on earlier. How many Users were returned in the Username field after extraction?

Answer: 14

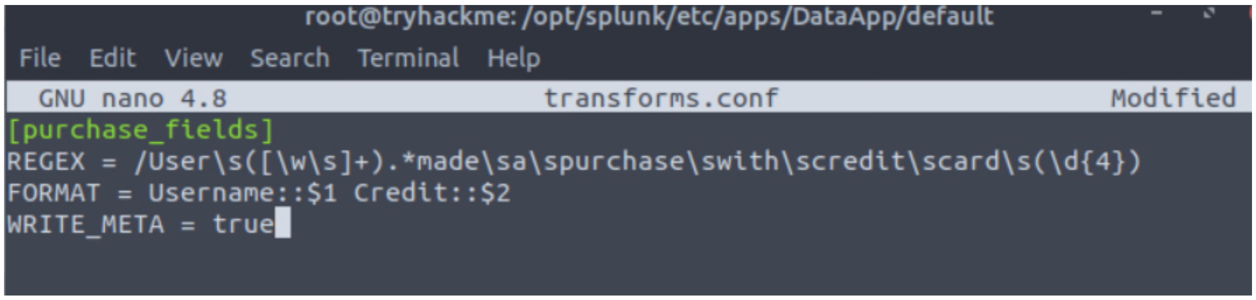
New props.conf for the extraction



root@tryhackme: /opt/splunk/etc/apps/DataApp/default
File Edit View Search Terminal Help

```
GNU nano 4.8                    props.conf                    Modified
[purchase_logs]
SHOULD_LINEMERGE = true
MUST_BREAK_AFTER = \d{4}\.
SEDCMD-cc = s/-\d{4}-\d{4}-\d{4}/-XXXX-XXXX-XXXX/g
TRANSFORM-purchase = purchase_fields
```

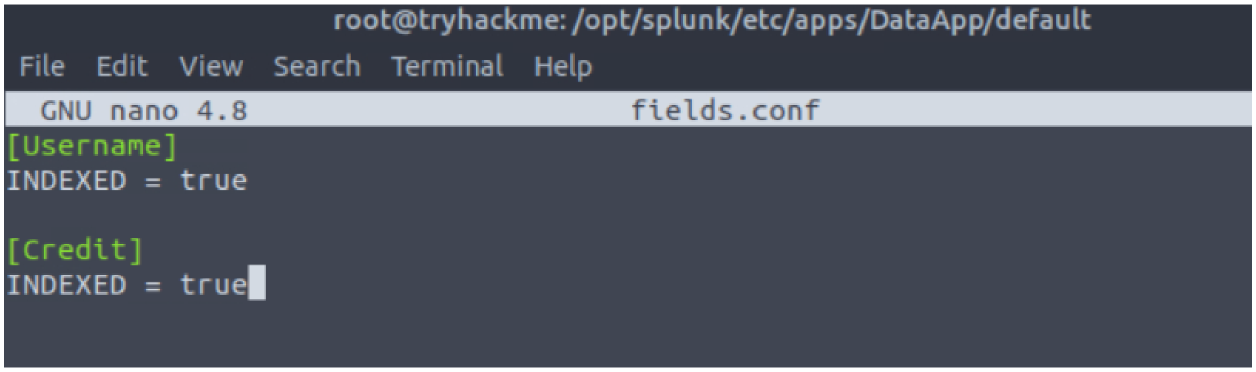
Transforms.conf



root@tryhackme: /opt/splunk/etc/apps/DataApp/default

```
GNU nano 4.8                    transforms.conf                    Modified
[purchase_fields]
REGEX = /User\s([\w\s]+).*made\sa\spurchase\swith\scredit\scard\s(\d{4})
FORMAT = Username::$1 Credit::$2
WRITE_META = true
```

Fields.conf



root@tryhackme: /opt/splunk/etc/apps/DataApp/default

```
GNU nano 4.8                    fields.conf
[Username]
INDEXED = true

[Credit]
INDEXED = true
```

Extract Credit-Card values from the sourcetype purchase_logs, how many unique credit card numbers are returned against the Credit-Card field?

Answer: 16