Threat Hunting at scale with Spark Notebooks

Ashwin Patil, MSTIC RnD Team
@ashwinpatil

About me

- Senior PM,
- Microsoft Threat Intelligence Center (MSTIC) RnD Team

- Blue Teamer over 10 years in

#SecurityMonitoring #IncidentResponse #DFIR

- Spoken at Conferences –

SecureWorld, JupyterThon ,
Gray Hat 2020 (Blue Team Village), Purple Team Summit

- Published multiple blogs on

#MicrosoftSentinel #ThreatHunting #JupyterNotebook

Agenda

- Introduction to Spark
- Why Spark notebooks for Threat hunting ?
- Spark notebook solutions
- Distributed Processing Example: Broadcast Joins
- Spark use cases in Threat hunting
- Threat Hunting use case: C2 Network beaconing
- Conclusion

Introduction to Spark

- Apache Spark
  - Parallel processing framework
  - Supports in memory processing (faster than disk-based)
  - Supports multiple languages (C#, Scala, PySpark, Spark SQL)
  - Manipulate distributed datasets like local collections

- Typical Use-cases
  - Data Engineering/Data Preparation
  - Machine learning
  - Graph processing
  - Spark streaming

Why Spark Notebooks for Threat Hunting ?

Modern SIEM with powerful query language.

- Complex workflows and data analysis can be done with Jupyter notebooks
- Do not scale well against query, complex processing on large volume of data/ data frames.
- Generally, less retention on historical data available for querying.

Alternative Python libraries for distributed processing of dataframe

- Scaling Pandas : Dask , RAPIDS, Vaex and more
- Supports only Python.
- Great for data exploration, visualization.
- Since 2014- still new, less mature or features, lack of adoption and enterprise support

Apache Spark

- Since 2010, mature and great adoption and enterprise support
- Natively integrate with other Apache projects.
- Multiple language and rich library features natively

Spark Notebook Solutions

Apache Zeppelin

Azure Synapse Analytics

Cloud Dataproc

databricks

Amazon EMR

Distributed Processing Example : Broadcast Join

- Broadcast Join
  - Apache Spark feature that lets us send a read-only copy of a variable to every worker node in the Spark cluster.
  - Use case – Joining Huge dataframe with relatively tiny dataframe.
  - Sends a copy of broadcasted dataframe to every worker node.
  - Faster than Shuffle joins as it avoids reshuffling both dataframes to partition them by join key.

Untitled-1

```
from pyspark.sql.functions import broadcast

huge_dateframe.join(
    broadcast(lookup_data_frame),
    lookup_data_frame.key_column==huge_dateframe.key_column
)
```

Spark Threat Hunting Use cases

Use cases

- ✓ Exploratory analysis on voluminous log data sources (e.g., Network logs, Windows events logs from very large networks)
- ✓ Entity prevalence or rarity based on Historical trend analysis
- ✓ DGA domain, malicious URLs identification with multiple feature extraction which requires historical data.
- ✓ C2 Beaconing activities
- ✓ Detections resulting from machine learning models trained on historical logs.

Threat Hunting use case: C2 Network beaconing

- Network Beaconing
  - Infected/compromised hosts reach out to Command and Control(C2) Server on periodic basis.
    - Time difference is same between each connection
    - Attacker may add jitter/randomness to the pattern.
- Data Source : Network Firewall Connection Logs
  - Identify regular time delta patterns of potential beaconing between connections

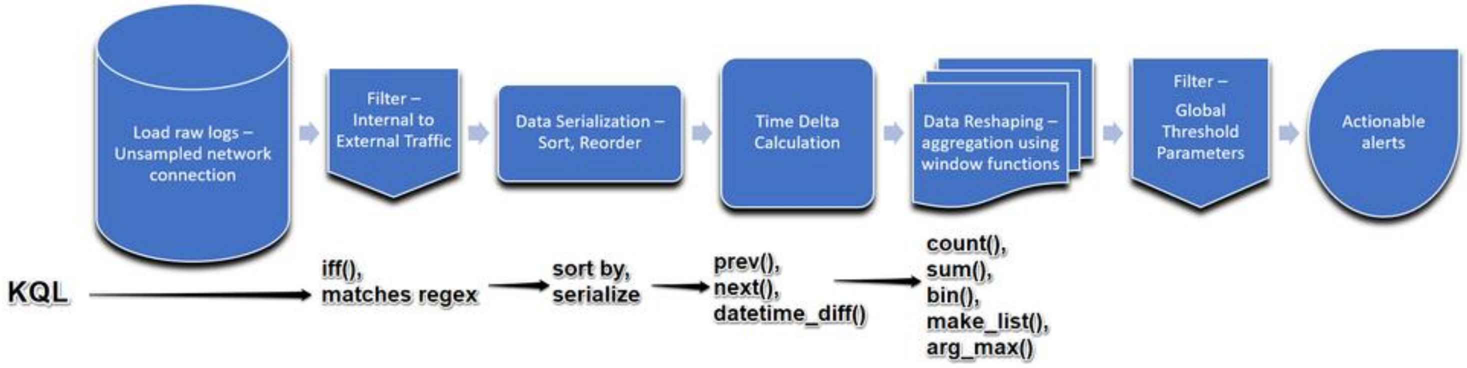
Use Case Workflow – KQL Implementation

- Network Beaconing via Intra-request Time Deltas
- Reference/Previous Work :
  - [Threat hunting Project]
  - [Flare]by [Austin Taylor]

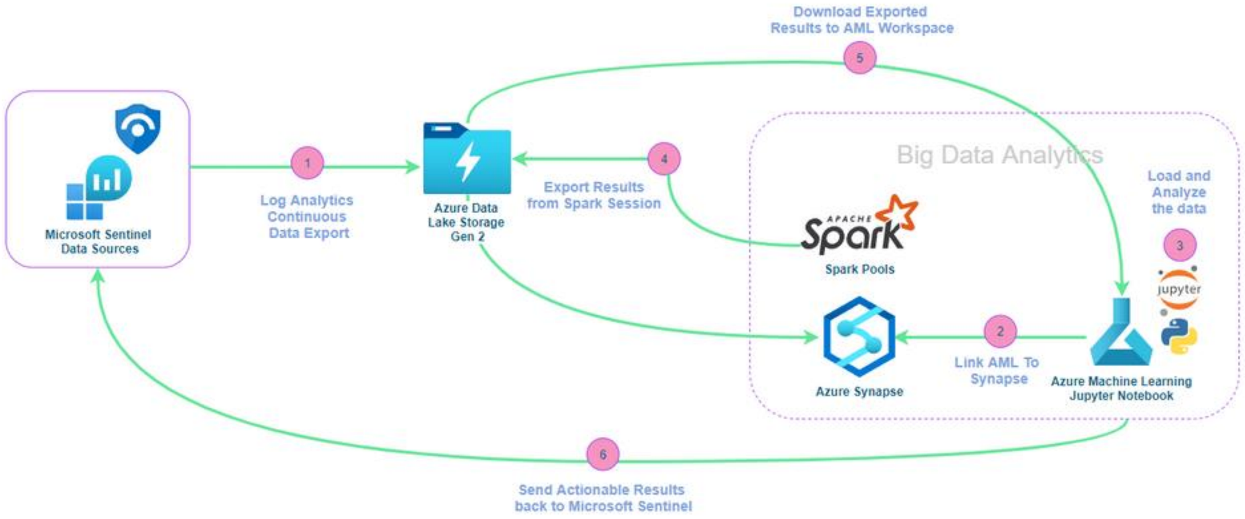
- Blog:
  Detect Network beaconing via Intra-Request time delta patterns in Microsoft Sentinel - Microsoft Tech Community

KQL Detection :

- **No Jitter:** Azure-Sentinel/PaloAlto-NetworkBeaconing.yaml at master · Azure/Azure-Sentinel (github.com)
- **With Jitter :** Azure-Sentinel/Fortinet-NetworkBeaconPattern.yaml at master · Azure/Azure-Sentinel (github.com)



Data Source



- CommonSecurityLog :
  - Firewall Vendors: Palo Alto , Fortinet, Checkpoint etc
- Calculate thresholds based on initial exploratory analysis:
  - TotalEventsThreshold - 30
  - DegreeofSourceIps - 25
  - TimeDeltaThreshold - 60
  - PercentBeaconThreshold - 75
  - BinWindow - 3

TimeGenerated	SourceIP	SourcePort	DestinationIP	DestinationPort	ReceivedBytes	SentBytes	DeviceVendor
2020-05-23T08:00:11	192.168.10.10	50423	67.217.69.224	80	433	390	Palo Alto Networks
2020-05-23T08:00:41	192.168.10.10	50423	67.217.69.224	80	433	390	Palo Alto Networks
2020-05-23T08:01:11	192.168.10.10	50423	67.217.69.224	80	433	390	Palo Alto Networks
2020-05-23T08:01:41	192.168.10.10	50423	67.217.69.224	80	433	390	Palo Alto Networks
2020-05-23T08:02:11	192.168.10.10	50423	67.217.69.224	80	433	390	Palo Alto Networks
2020-05-23T08:02:41	192.168.10.10	50423	67.217.69.224	80	433	390	Palo Alto Networks
2020-05-23T08:03:11	192.168.10.10	50423	67.217.69.224	80	433	390	Palo Alto Networks
2020-05-23T08:03:41	192.168.10.10	50423	67.217.69.224	80	433	390	Palo Alto Networks

PySpark Implementation

- **Load Data :** Raw Traffic Connection Logs

- **Filter Data:**
  - Filter Internal to External traffic.
  - Apply event-based thresholds and remove potential benign Source Ips.
    - E.g. Source Ips with at least 10 events in 24 hours (remove high cardinality – not eligible for data analysis)
  - Find benign destinations based on DegreeOfSourceIPs in historical dataset. (2-4 weeks avg)
    - E.g. Destination Ips with Less than 25 Source Ips connected in last 24 hours/ avg in last 2-4 weeks.
    - (Removes common FPs - Windows update, common agent beaconing behavior etc)

- **Data Wrangling**
  - **Windowing - Data Serialization**
    - **Calculate Time Delta : Datetime difference between Previous and Next Timestamp**
    - **Same SourceIP, DestinationIP , DestinationPort**
  - TimeDeltaThreshold : Assumption beaconing under 10 seconds/thresholds is too noisy for attacker so not eligible for analysis
  - BinWindow : Cumulative sum for all Time Delta within Window (SourceIP, DestinationIP, DestinationPort) bin specified
  - PercentBeaconThreshold : Cumulative Sum/ Total Events * 100. Anything above 75 can indicate potential beaconing.

Conclusion

- Provide new detection ideas which are not scalable with your SIEM to transform into PySpark notebooks and feed results with rich context back to SIEM.
- Spark provides capability to hunt on historical data with complex analysis for analysts to get more insightful and contextual results.

Microsoft

Resources:

- Is Spark still relevant – PyData NYC 2019
- Notebook : Detect potential network beaconing using Apache Spark via Azure Synapse
- Blog: Hunting for potential network beaconing patterns using Apache Spark via Azure Synapse – Part 1
- Apache PySpark SQL : https://spark.apache.org/docs/3.1.1/api/python/reference/pyspark.sql.html
- Introducing Window Functions in Spark SQL - The Databricks Blog

Thank you