



# Practical msticpy Use: *Rainbow Bridge to SIEM for Advanced Threat Hunting*

WITH

Tatsuya Hasegawa

*Technical Security Advisor, GoAhead Inc.*

---

7 - 8 September  
Live Online from Tokyo ☺

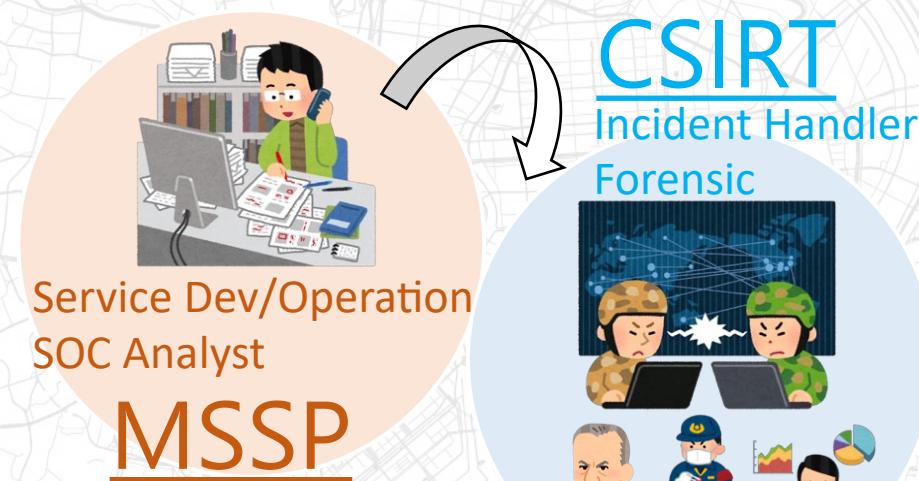
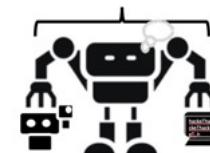


SANS



# \$WHOAMI

- Threat Hunter/App Developer/Threat Researcher
- OSS Contributor
  - msticpy,unprotect,atomic-red-team,cuckoo,capev2..
- Qualifications
  - 7 GIACs
  - CISSP、CISA
- SNS
  - HN: hackeT
  - X: @T\_8ase



**AI Anti-Virus**

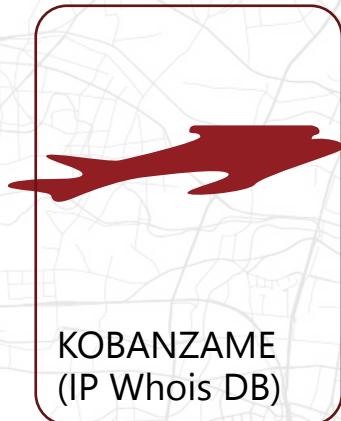
# \$more GoAhead Inc.



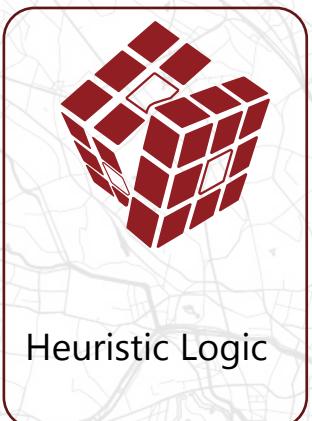
CEO: Mitsuhiro Nakamura  
Splunk.conf 2017@USA  
Splunk Champion

Established in 2017

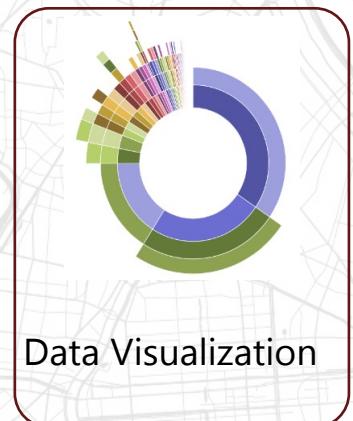
Data Analysis Company  
Splunk is our strength  
for Security Challenges



KOBANZAME  
(IP Whois DB)



Heuristic Logic

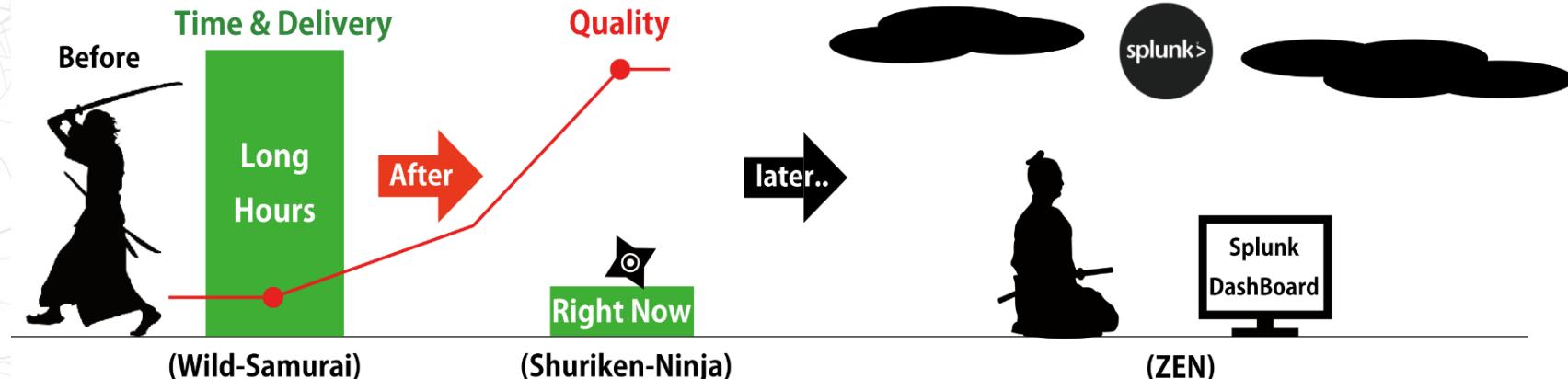


Data Visualization

Free Splunk App/Add-ons by GoAhead  
<https://splunkbase.splunk.com/apps?keyword=goahead>

 <b>GoAhead KobanZame</b> By GoAhead Dev Team Kobanzame is the commercial service provided by GoAhead. GoAhead's access key is needed to utilize this App and onl... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (1) 	 <b>GoAhead D3 DYNAMIC BUBBLE</b> By GoAhead Dev Team "GoAhead D3 DYNAMIC BUBBLE" is a custom visualization App to show the circular packing chart by using "D3.js" ... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (1) 	 <b>GoAhead D3 BUBBLE</b> By GoAhead Dev Team "D3 BUBBLE" is a custom visualization App to show the circular packing chart by using "D3.js". The size of bubble and ... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (1) 
 <b>GoAhead D3 SUNBURST</b> By GoAhead Dev Team "D3 SUNBURST" is a custom viz App to show the sunburst chart by using "D3.js". This sunburst chart has breadcrumbs o... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (1) 	 <b>Defender Advanced Hunting Query App by GoAhead</b> By GoAhead Dev Team API wrapper tool for Microsoft Defender Advanced Hunting. Advanced Hunting uses Kusto Query Language (KQL) and t... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (1) 	 <b>GoAhead Strutis</b> By GoAhead Dev Team Utility commands for string data. BASE64 encode & decode, ROT13, ROT47 convert, coding detection, unescape, ... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (2) 
 <b>Hatching Triage App for Splunk by GoAhead</b> By GoAhead Dev Team Hatching Triage App is an API wrapper tool as the report utility for Hatching Triage sandbox's instance public... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (1) 	 <b>Bing Web Search App by GoAhead</b> By GoAhead Dev Team Bing Web Search App is an API wrapper tool of Microsoft Bing Web Search API (v7). This App requests to... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (2) 	 <b>Spur context API App by GoAhead</b> By GoAhead Dev Team Spur context API App is Spurus context API(v2) wrapper for retrieving the attribute of IP intelligence data like AS,... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (1) 
 <b>GoAhead Mongofetch</b> By GoAhead Dev Team Mongofetch is a pymongo's client only for retrieving db records from MongoDB. This never perform... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (0) 	 <b>TA-fwlog_eventgen</b> By GoAhead Dev Team TA-fwlog_eventgen is a custom LOG generator by [Splunk EventGen] (http://splunk.github.io/eventgen/). Thi... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (0) 	 <b>TA-apache_access_eventgen</b> By GoAhead Dev Team TA-apache_access_eventgen is a custom LOG generator by [Splunk EventGen] (http://splunk.github.io/eventgen/). Thi... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (0) 
 <b>TA-squid_proxy_eventgen</b> By GoAhead Dev Team TA-squid_proxy_eventgen is a custom LOG generator by [Splunk EventGen] (http://splunk.github.io/eventgen/). Thi... PLATFORM: Splunk Enterprise, Splunk Cloud RATING: ★★★★★ (0) 		

Aim for maximum effectiveness with minimum resources





# Agenda

- Invariable Operation with SIEM
- msticpy 101 Overview and Basics
- msticpy 201 Jupyter Notebook and ( pros | cons )
- msticpy 301 Practical use case
- Take Away

# Invariable Operation with SIEM

**DFIR** デジタルフォレンジックとインシデントレスポンス  
ASIA PACIFIC

SANS



# Background and Issues

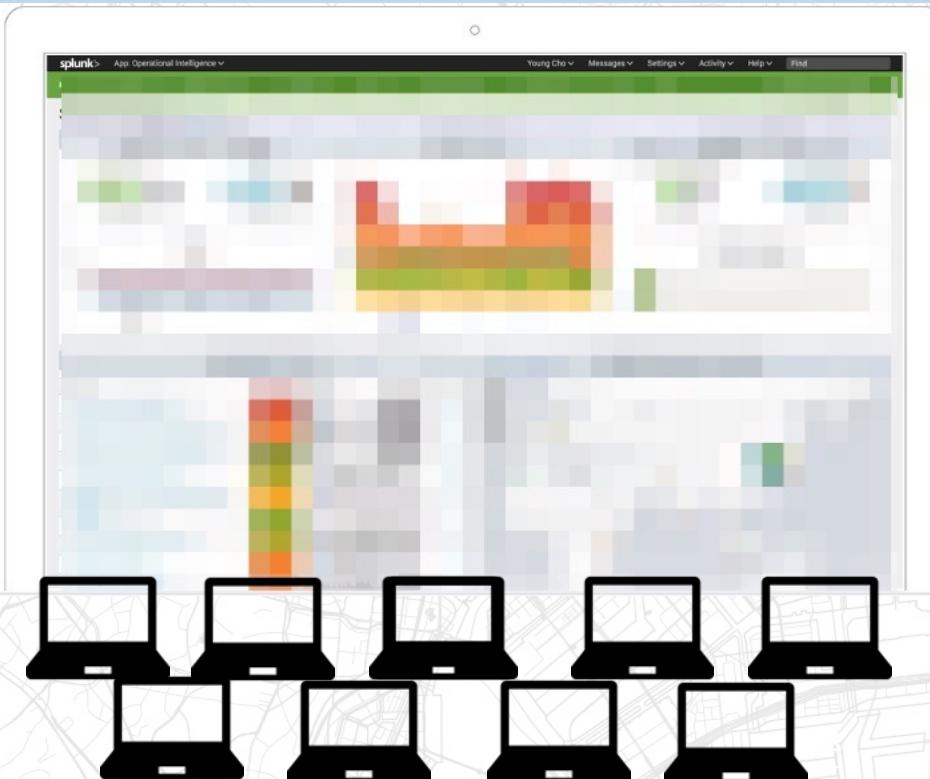
Old fashioned



Analysis : Human-wave tactics  
for raw log

Monitoring : Alert by Email

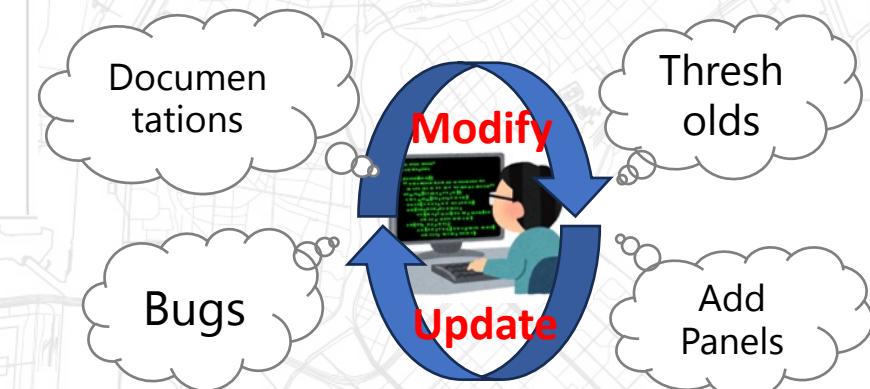
Nowadays



Analysis : Multi-axis search of  
formatted logs  
Monitoring : Visualized Dashboard,  
Alert from SIEM

Never ending Dev & Ope tasks

- Modification and addition of analytical logic to keep up with new threats
- Thresholds tailored to the internal situation as well as the threat situation in the world



SIEM functions and existing dashboards  
Biases sometimes lead to non-free analysis



# Objective

## Advanced Threat Hunting

### Threat Hunting

- Proactive detection and response to signs of malicious activity or threats
- Investigate using threat intelligence, unapplied IOCs, anomaly detection
- Iterations between hypothesis and verification

### Advanced Threat Hunting

- Identifying undetected threats from raw data
  - check raw data too and look for omissions in processing and detection by security product.
- Inherently data analysis with freedom (ad hoc)
  - uniquely conceived analytical logic
  - unrestricted external collaboration,
  - eccentric visualization
  - emphasis that is easy for readers to understand
- Continuous update operation
  - Machine Learning & Deep Learning (ML/DL)
  - Automation



# Security Information and Event Management



## First Generation

Gartner 2005  
Log and Event  
management integration

## Second Generation

Correlation analysis  
with CTI  
Big data processing

## Third Generation

Gartner 2017  
UEBA. SOAR addition

- SIEM Products
  - Splunk/MS Sentinel/IBM Qradar/  
Exabeam/Sumo Logic/Elastic, etc.
  - SIEM by Security vendors
- Can collect/extract/search/analyze/  
visualize/detect/respond
- Have the individual threat hunting function
- Have ML/DL extensions



source: Gartner Inc, 2022 Magic Quadrant



# SIEM's advantage

- Rapid search by indexing and field normalization (CIM, ASIM)
- Statistical calculations are easy with the benefit of its search language
- Can store threat intelligence
- Multiple analyst can see the same data and analysis results
- SIEM vendors also provide a lot of detection logic



# SIEM's breakdown

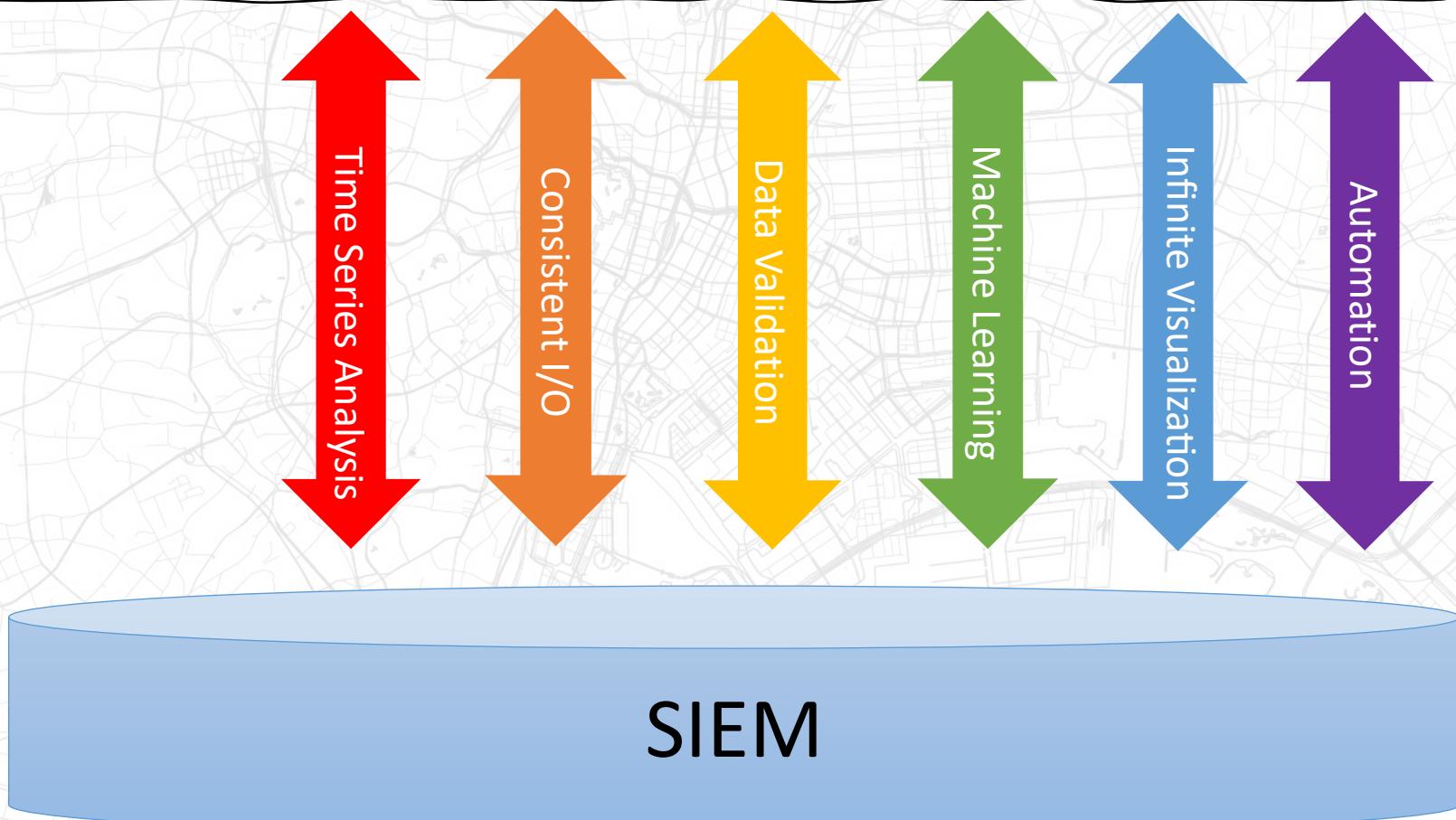
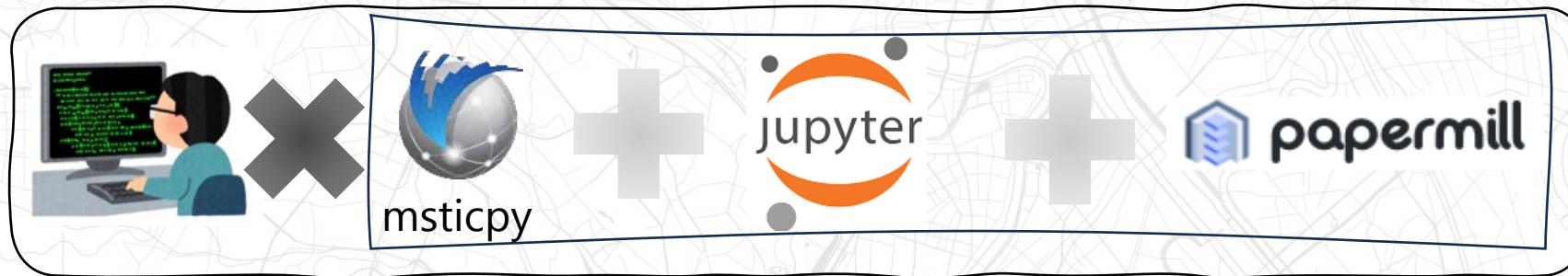
- Rapid search by indexing and field normalization (CIM, ASIM)
  - If extraction fails, it is missing from the search at the beginning or from the analysis along the way.
- Statistical calculations are easy with the benefit of its search language
  - Existing some process which is not good at, and take costs for learning search language
- Can store threat intelligence
  - Most of the intelligence is self-prepared and operational by ourselves.
- Multiple analyst can see the same data and analysis results
  - Various limitations due to shared resources
- SIEM vendors also provide a lot of detection logic
  - Necessary and sufficient ? No!

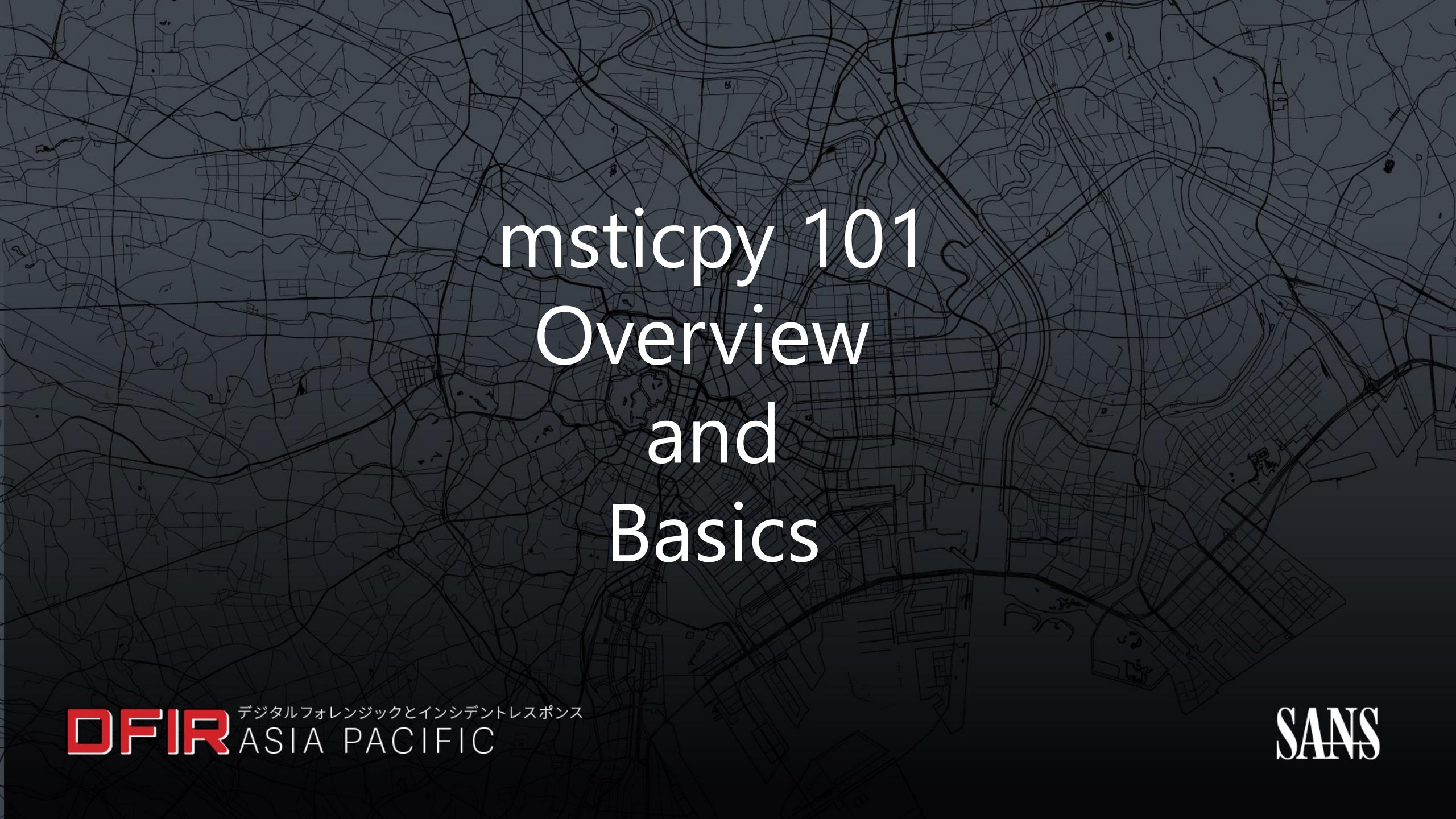
# Not recommend to rely too much on SIEM analysis!



- When a failure occurs, not everyone can be analyzed until recovery.
  - Over-reliance on analysis in SIEM search language only, forgetting how to analyze raw data
  - Who will ensure the integrity of the data and search results in SIEM ?
- Limitations of SIEM
  - Default upper limits for sub search and multi value (truncate)
  - Default upper limit for number of plots on graph (truncate)
- Difficult to notice search omissions due to misconfiguration
- Don't rely solely on the logic provided by SIEM vendor
  - Enterprise SIEMs **Miss 76 Percent** of MITRE ATT&CK Techniques
    - source: CardinalOps, "2023 Report on State of SIEM Detection Risk"

# For Advanced Threat Hunting

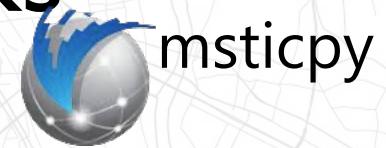




# msticpy 101

## Overview and Basics

# Microsoft Threat Intelligence Center (MSTIC) on Python and Jupyter Notebooks



- MSTICPy: OSS library developed by Microsoft's MSTIC
  - Written in Python, usually used on Jupyter Notebooks
- Extensive functionality for infringement investigation and threat hunting
- March 2019 ~ 200k+ Downloads

<https://github.com/microsoft/msticpy>

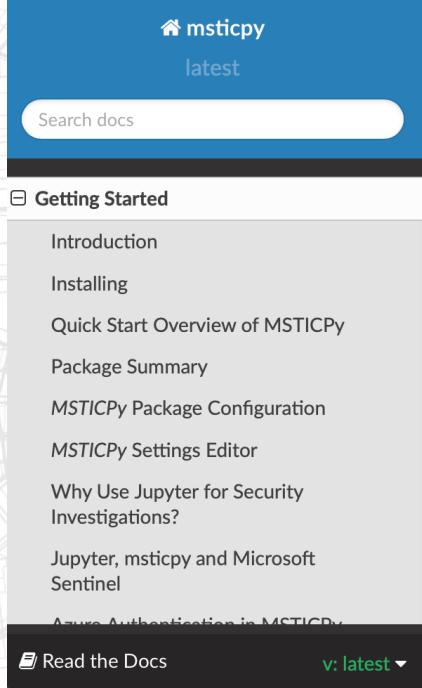
- Presented at BlackHat USA 2020
- Frequent update recently and continues to evolve
  - Still few users and blog article in Asia and Japan
- Fall into the following four process broadly
- Only desired functions can be used piecemeal because of library-based



# msticpy's Documentation & Resource



- MSTICPy  **msticpy** in this presentation
- Official document
  - <https://msticpy.readthedocs.io>
  - Word count 100k+
  - RST files 80+
  - Jupyter Notebook samples 40+
  - Past training resources
    - msticpy-lab, msticpy-training github repo
- Official Blog
  - <https://msticpy.medium.com>



The screenshot shows the 'Getting Started' page of the MSTICPy documentation. At the top, there is a header with the MSTICPy logo and a search bar labeled 'Search docs'. Below the header, a sidebar titled 'Getting Started' lists several topics: Introduction, Installing, Quick Start Overview of MSTICPy, Package Summary, MSTICPy Package Configuration, MSTICPy Settings Editor, Why Use Jupyter for Security Investigations?, and Jupyter, msticpy and Microsoft Sentinel. At the bottom of the sidebar, there is a link 'Read the Docs' and a dropdown menu 'v: latest'. To the right of the sidebar, the main content area displays the 'Introduction' section of the documentation.

[Home](#) / Getting Started [Edit on GitHub](#)

## Getting Started

- [Introduction](#)
  - [Use Cases and Environments](#)
- [Installing](#)
  - [Python 3.8 or Later](#)
  - [Creating a virtual environment](#)
  - [Installation](#)
  - [Selective Installation - using "extras"](#)
- [Quick Start Overview of MSTICPy](#)
  - [Installing](#)
  - [Importing MSTICPy](#)
  - [Searching for a MSTICPy module](#)
  - [Initializing MSTICPy](#)

Time-consuming for learning with the huge resources ...

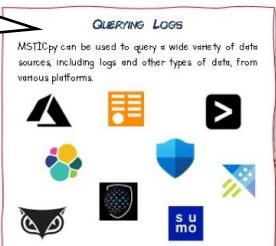


# msticpy Capabilities

## OVERVIEW OF MSTICPY CAPABILITIES

Acquisition

### Querying Logs



#### QUERYING LOGS

MSTICPy can be used to query a wide variety of data sources, including logs and other types of data, from various platforms.

[HTTPS://GITHUB.COM/MICROSOFT/MSTICPY](https://github.com/microsoft/msticpy)  
[HTTPS://MSTICPY.READTHEDOCS.IO](https://msticpy.readthedocs.io)  
[HTTPS://TWITTER.COM/MSTICPY](https://twitter.com/msticpy)  
[HTTPS://MSTICPY.MEDIUM.COM](https://msticpy.medium.com)

Visualization

### Data Visualization

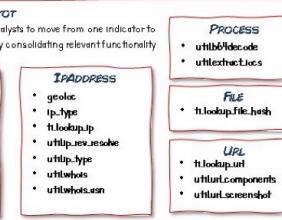
MSTICPy provides visualization capabilities for data analysis. This includes:

- Notebook widgets**: Provides a way to create interactive widgets within Jupyter notebooks to filter and explore data.
- process\_tree**: Allows for the creation of interactive process trees to help understand the relationships between different processes.
- timelines**: Allows for the creation of interactive timelines to help understand the chronology of events.
- columnmap**: Allows the creation of a map using the folium package.
- entity\_graph**: Allows to create a graph for visualizing and tracking links between entities.
- metric\_map**: Allows to show interactions between two sets of items in a grid.
- morphcharts**: This module formats data and configuration files for use with morphcharts.com.

MSTICPy can be used for data transformation. These functions can be used to process and enrich data ingested from various sources.

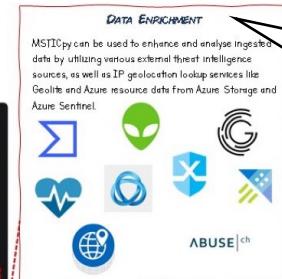
- base64unpack**: Decodes base64 encoded strings.
- toextract**: Extracts indicators of compromise (IOCs) from text.
- emailextract**: Extracts structured data from email logs.
- slog\_utils**: Extracts structured data from syslog messages.
- cmd\_line**: Extracts command line arguments from text.
- domainutils**: Extracts domain names from text.

```
# Install msticpy  
pip install msticpy  
  
# Import the msticpy package and give it  
# the alias "mp"  
import msticpy as mp  
  
# QUERY PROVIDER  
mp.QueryProvider("MSSentinel")  
# list the built-in queries  
query_prov.list_queries()  
  
# DATA PROCESSING AND ENRICHMENT  
ti_lookup = mp.TILookup()  
ti_lookup.lookup.iocs(iocs)  
  
# SECURITY ANALYSIS PACKAGES  
import msticpy.analysis.anomalous_sequence  
import msticpy.analysis.timeseries  
import msticpy.analysis.eventcluster  
import msticpy.analysis.outliers  
  
# VISUALISATION - mp_plot accessor  
df.mp_plot.timeline(group_by="LogonType")  
df.mp_plot.folium_map()  
  
# UTILITY FUNCTION PACKAGES  
import msticpy.transform.auditextract  
import msticpy.transform.syslog_utils  
import msticpy.transform.cmd_line  
  
base64unpack.unpack(input_string=cmdline)  
  
# mp accessor  
df.mp.toextract()  
  
# PIVOT  
IpAddress.util.ip_type("20.72.193.242")  
  
# Initializes the msticpy library for  
# use in the current notebook  
mp.init_notebook()
```



Analysis

### Utility



#### DATA ENRICHMENT

MSTICPy can be used to enhance and analyse ingested data by utilizing various external threat intelligence sources, as well as IP geolocation lookup services like GeoLife and Azure resource data from Azure Storage and Azure Sentinel.

**SECURITY ANALYSIS**  
MSTICPy can be used to perform security analysis on various data sources:

- Anomalous Sequence Detection**: Identifying unusual patterns of events in Office, Active Directory, or other log data.
- Time Series Analysis**: Uncovering abnormal patterns in log data, taking into account normal seasonal variations.
- Event Clustering**: Summarizing large numbers of events into clusters of different patterns to aid in analysis.
- Outlier Identification**: Utilizing Scikit-learn's Isolation Forest, MSTICPy can identify events that deviate from the norm in a single data set.

**msticpyconfig.yaml**  
msticpyconfig.yaml is a configuration file used by the MSTICPy library.  
It contains settings, API key and credentials for data sources, threat intelligence providers, and other services used by the library.  
The file is typically located in the root directory of the MSTICPy package and can be edited manually to configure the library as per your needs.

```
# msticpyconfig.yaml example  
AzureSentinel:  
    Workspaces:  
        Default:  
            WorkspaceId: "d973e3d2-b1ba4"  
            TenantId: "4cdf87a8-f4ac8e61"  
    TIProviders:  
        OTX:  
            Args:  
                AuthKey: "4ea41beb6003a"  
                Primary: True  
                Provider: "OTX"  
        VirusTotal:  
            Args:  
                AuthKey: "4ea41beb6003a"  
                Primary: False  
                Provider: "VirusTotal"
```

@FR0GGER\_  
THOMAS ROCCIA

Analysis

### Pivot

### Data Enrichment

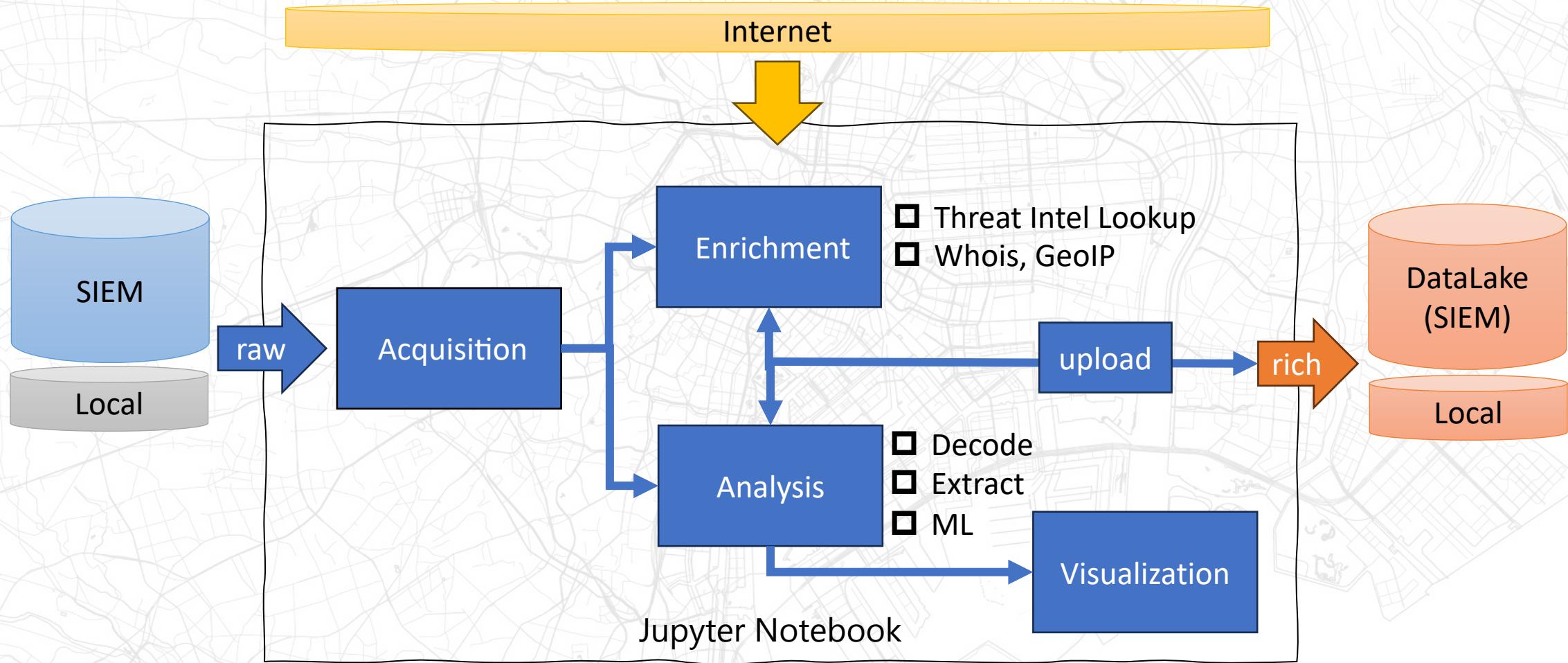
Enrichment

### Security Analysis

Analysis



# msticpy Data Flow Diagram





# msticpy: Data Acquisition (1)

```
import msticpy as mp
mp.QueryProvider.list_data_environments()

✓ 0.0s

['AzureSecurityCenter',
 'Cybereason',
 'Elastic',
 'Kusto',
 'Kusto_New',
 'LocalData',
 'M365D',
 'MDE',
 'MSGraph',
 'MSSentinel',
 'MSSentinel_New',
 'OSQueryLogs',
 'OTRF',
 'ResourceGraph',
 'Splunk',
 'Sumologic',
 'VelociraptorLogs']
```

- Create instance of Query Provider
  - Select from data sources (left picture)

LocalData: connect to .pkl files in ./data dir

```
local_prov = mp.QueryProvider(
    data_environment="LocalData", data_paths=["./data"], query_paths=["./data"]
)
local_prov.connect()

✓ 0.1s

Connected.
```

Splunk: connect to Splunk REST port with msticpyconfig.yaml

```
splunk_prov = mp.QueryProvider("Splunk")
splunk_prov.connect()

✓ 0.1s

connected
```

Communication channel  
is NOT independently encrypted  
by msticpy's uniq func  
=> HTTPS (SSL) is necessary

# msticpy: Data Acquisition (2)



- Return: Pandas DataFrame
- Ad hoc query function
  - exec\_query(): arbitrary query
- Built-in query function
  - select from the list varies by data source

```
splunk_prov.SplunkGeneral.get_events_parameterized('print',
    index="botsv2",
    source="WinEventLog:Microsoft-Windows-Sysmon/Operational",
    timeformat='"%Y-%m-%d %H:%M:%S"',
    start="2017-08-25 00:00:00",
    end="2017-08-25 10:00:00",
    add_query_items='',
    count=0
)
0.0s
| table TimeCreated, host, EventID, EventDescription, User, process,
```

```
splunk_query = ''
search index="_internal" earliest=-1h latest=now
| table *
...
df = splunk_prov.exec_query(splunk_query)
df.head()
```

```
splunk_prov.list_queries()
✓ 0.0s
['Alerts.list_alerts',
 'Alerts.list_alerts_for_dest_ip',
 'Alerts.list_alerts_for_src_ip',
 'Alerts.list_alerts_for_user',
 'Alerts.list_all_alerts',
 'Authentication.list_logon_failures',
 'Authentication.list_logons_for_account',
 'Authentication.list_logons_for_host',
 'Authentication.list_logons_for_source_ip',
 'SplunkGeneral.get_events_parameterized',
 'SplunkGeneral.list_all_datatypes',
 'SplunkGeneral.list_all_savedsearches',
 'audittrail.list_all_audittrail']
```



# msticpy: Enrichment

- Threat Intel Lookup
  - Pivot TI function (Only on Jupyter Notebook)

```
pivot_ti_df = IpAddress.tilookup_ip(ip_list)
```

- TIlookup class (Available on also python program)

```
ti_lookup = mp.TILookup(providers=["VirusTotal","XForce","GreyNoise"])
ti_df = ti_lookup.lookup_iocs(df, ioc_col="IPAddress", default_providers=["VirusTotal"])
```

- GeoIP (MaxMind GeoLite2, IPStack)
- IPWhois (Cymru, RADB, RDAP)

```
mp.TILookup.list_available_providers()

✓ 0.0s

OTX
AzSTI
GreyNoise
XForce
IntSights
OPR
Tor
VirusTotal
RiskIQ
Pulsedive
```



# msticpy: Analysis (Utility)

- Base64 Decode

```
# Filter the records with powershell base 64 encoded data
process_enc_logs = process_logs[process_logs['CommandLine'].str.contains("-enc")]
process_enc_logs
# specify the data and column parameters
dec_df = mp.transform.base64unpack.unpack_df(data=process_enc_logs, column='CommandLine')
# display dataframe
display(dec_df.full_decoded_string)
```

✓ 0.0s

```
YWN0aXZlIC1Ob3Byb2ZpbGUgLUNvbW1hbowershell -enc <decoded type='string' name='[None]' index='1' depth='1'>-Noninteractive -N
```

- IoC Extract

```
# extract iocs from decoded string
ioc_df = dec_df.mp.ioc_extract(columns='full_decoded_string')
if len(ioc_df):
    display(HTML("<h3>IoC patterns found in process tree.</h3>"))
    display(ioc_df)
```

✓ 0.0s

Python

IoC patterns found in process tree.

IoCType	Observable	SourceIndex	Input
0 dns	wh401k.org	0	.\\powershell -enc <decoded type='string' name='[None]' index='1' depth='1'>-Noninteractive -Nop...
1 url	http://wh401k.org/getps	0	.\\powershell -enc <decoded type='string' name='[None]' index='1' depth='1'>-Noninteractive -Nop...



# msticpy: Analysis (Pivot)

- Pivot Functions *being loaded by "init\_notebook()" is required basically*

- Wrap msticpy functions and classes for ease of discovery and use
- Standardization of function parameters, syntax, and output format
- ".mp\_pivot." can be piped in multiple stages

```
# Pivot
pivot.browse()

✓ 0.0s
```

Entities
entity
Account
Dns
File
Host
<b>IpAddress</b>
Process
Url

Selected entity pivot Function

```
pivot functi... VT.communicatin... VT.historical_ssl... VT.historical_who... VT.referrer_files... VT.resolutions... VT.subdomains... geoloc ip_type ti.lookup_ip tilookup_ip util.geoloc util.geoloc_ips util.ip_rev_resolv util.ip_type util.whois util.whois_asn whois whois_asn
```

```
suspicious_ips
# Lookup IPs at VT
.mp_pivot.run(IPAddress.ti.lookup_ip, column="IPAddress")
# Filter on high severity
.query("Severity == 'high'")
# lookup whois info for IPs
.mp_pivot.run(IPAddress.whois, column="Ioc", join="left")
# display sample of intermediate results
.mp_pivot.display(title="TI High Severity IPs", cols=["Ioc", "Provider", "Reference"], head=5)
.mp_pivot.tee(var_name="ti_whois_df")
# Query IPs that have login attempts
.mp_pivot.run(IPAddress.AzureSentinel.list_aad_signins_for_ip, ip_address_list="Ioc")
# Send the output of this to a plot
.mp_plot.timeline(
    title="High Severity IPs with Logon attempts",
    source_columns=["UserPrincipalName", "IPAddress", "ResultType", "ClientAppUsed", "UserAgent",
    group_by="UserPrincipalName"]
```



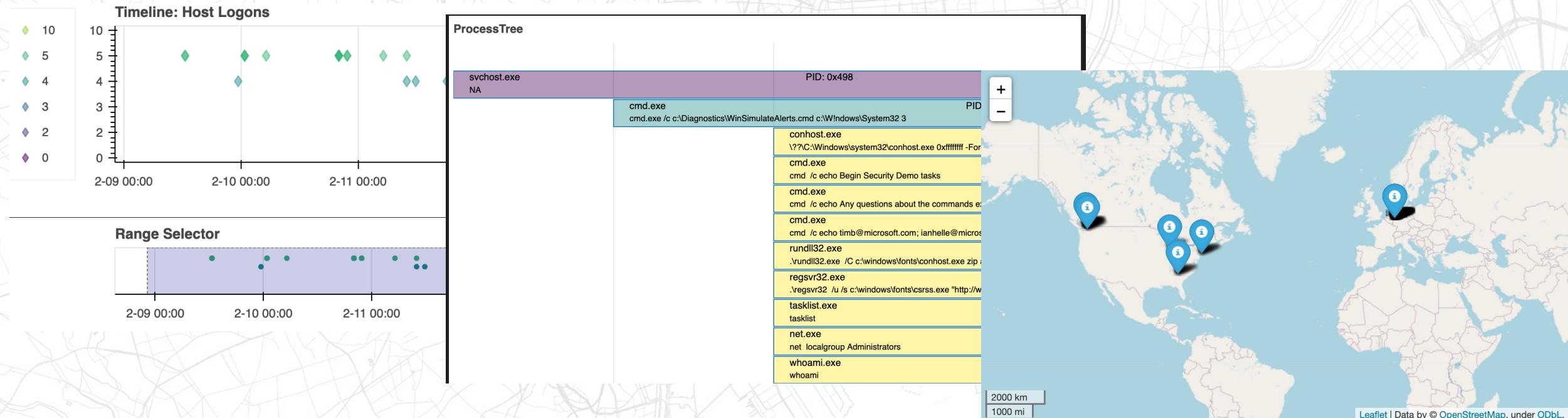
# msticpy: Analysis (Security)

- Event Clustering
  - Classification of “process and logon events” on the host machine
- Time Series Analysis
  - Anomaly detection in time series data considering seasonal variations
- Outlier Identification
  - Outlier detection using decision trees
- Anomalous Session
  - Unusual pattern detection of rare event sequences with low likelihood
  - Use of the event’s command name, its parameter names and values

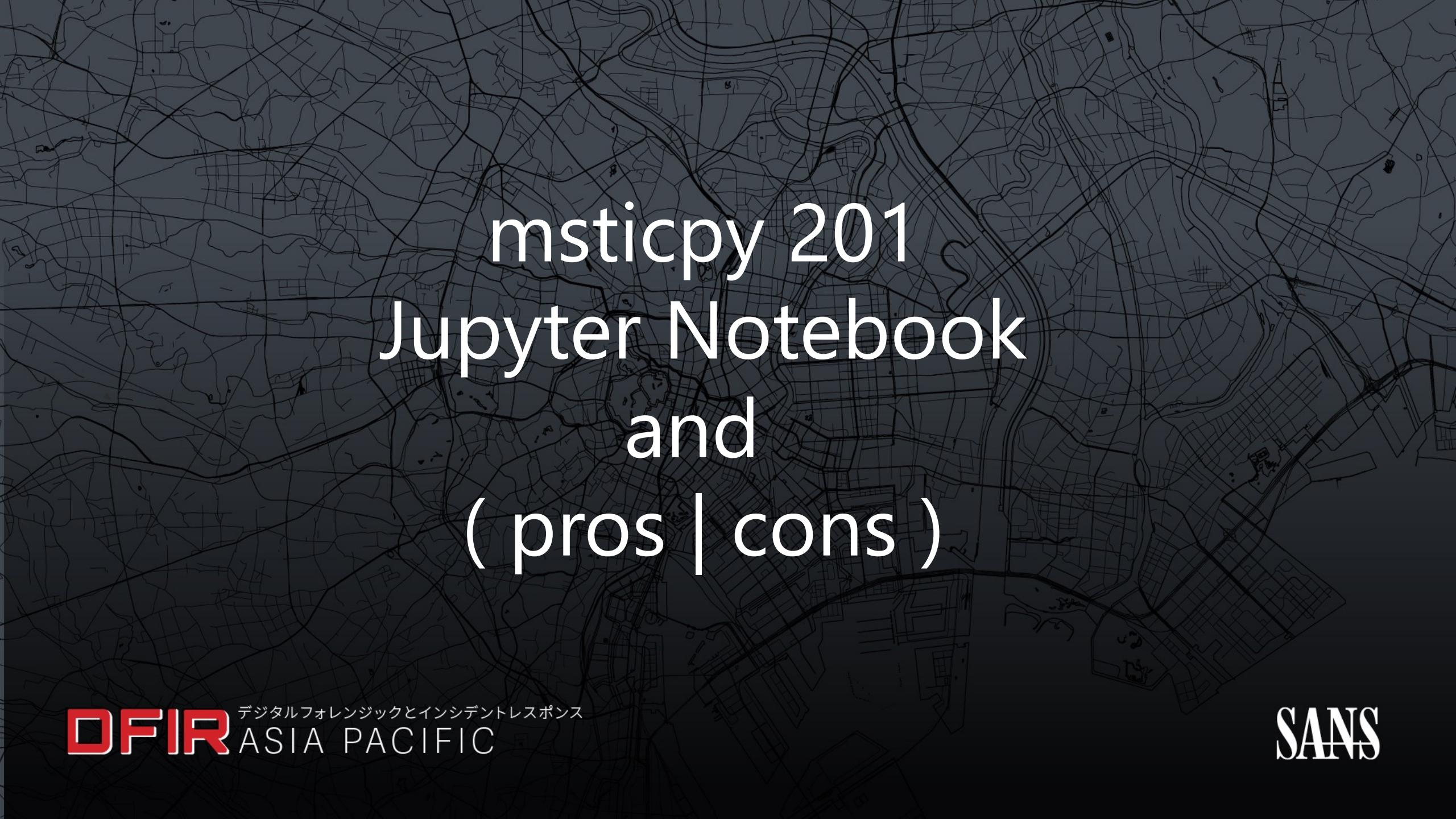


# msticpy: Visualization

- Implemented with BokehJS
- Viz charts implemented in msticpy
  - Timeline, ProcessTree, Folium Map, Matrix Plot, Entity/Network Graph , etc.



- Can create additional charts with MorphCharts



# msticpy 201 Jupyter Notebook and (pros | cons)

# Benefits of Analyzing with Jupyter Notebook

- Reproducibility of data, it can output of intermediate results
- Easy combination/integration with external sources
- Easy use of ML/DL frameworks
- Extensive visualization library at your disposal
- Gain applied skills as a data scientist

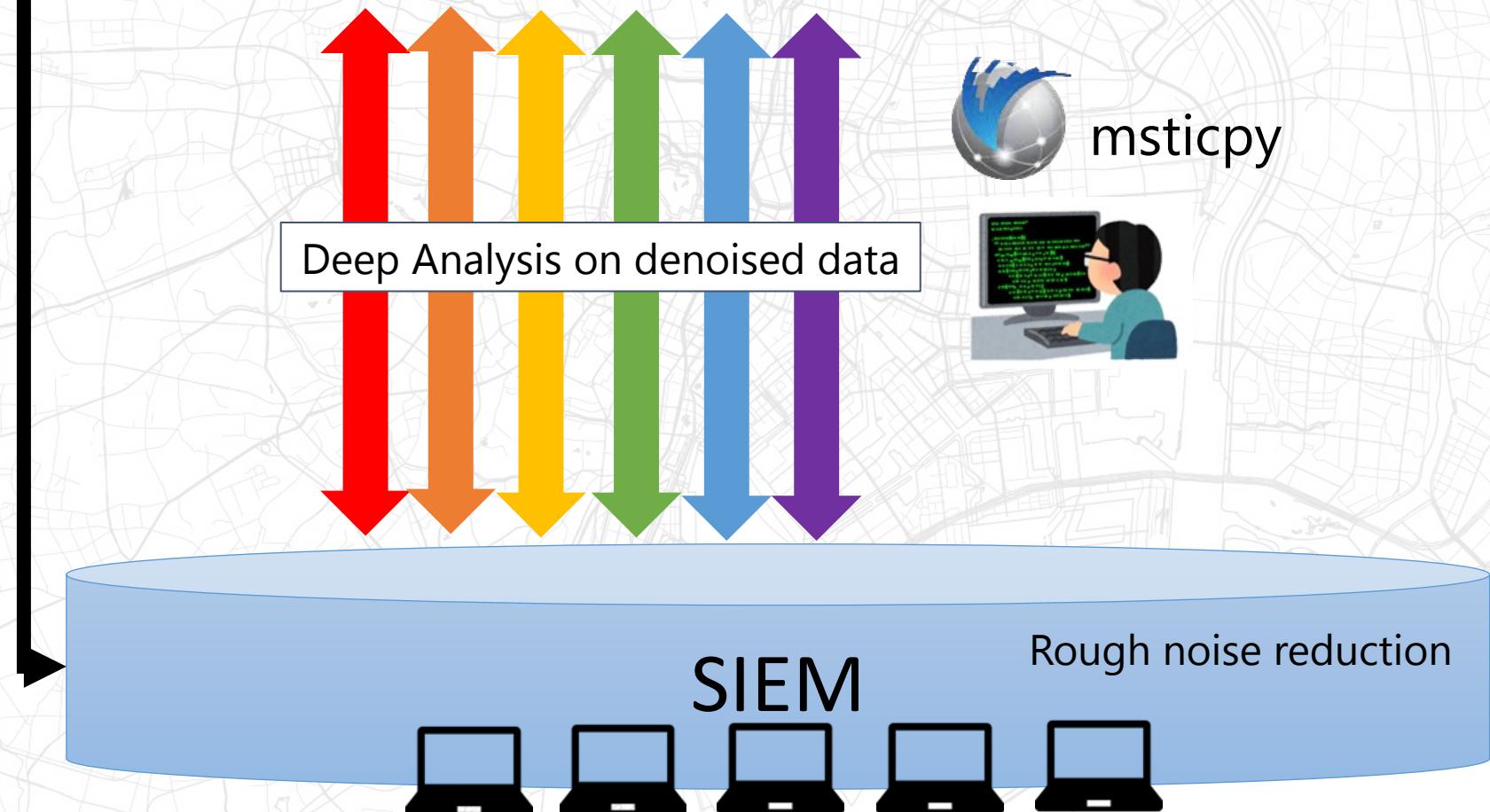




# Ideal Relationship between Jupyter Notebook and SIEM

## Advanced Threat Hunting

Intelligence  
Knowledge





# msticpy's pros: Seasonal-Trend decomposition using LOESS

```
from msticpy.analysis.timeseries import timeseries_anomalies_stl  
df_count = df_action[['date_clock','count']]  
df_count = df_count.set_index('date_clock')  
output = timeseries_anomalies_stl(df_count)
```

output[output.anomalies == 1]										...	
[148]	✓ 0.0s	...	date_clock	count	residual	trend	seasonal	weights	baseline	score	anomalies
			94 2022-09-05 16	52	15	33	3	1	36	3.119998	1





# msticpy's pros: Consistent I/O

- Sending by Data Uploader function (Transfer)
  - Only Azure Sentinel and Splunk are supported as of Aug 2023
  - Can upload Data Frame, File, Folder



```
from msticpy.data.uploaders.splunk_uploader import SplunkUploader
spup = SplunkUploader(username=USERNAME, host=HOST, password=PASSWORD)

spup.upload_df(data=DATAFRAME, table_name=TABLE_NAME, index_name=INDEX_NAME)
```



# Jupyter & msticpy's pros: Data Validation

- Check the DataFrame result sequentially
  - Save for accidental overwriting by copy() func
- Value type conversion and strip null values
  - Easy to validate char codes
  - GUI for time ranges ➡
- Pre-confirming actual Queries via Query Provider by "print" option

```
splunk_prov.SplunkGeneral.get_events_parameterized('print',
    index="botsv2",
    source="WinEventLog:Microsoft-Windows-Sysmon/Operational",
    timeformat='"%Y-%m-%d %H:%M:%S"',
    start="2017-08-25 00:00:00",
    end="2017-08-25 10:00:00"
)
```

Query to be searched

Python

```
' search index=botsv2 source=WinEventLog:Microsoft-Windows-Sysmon/Operational timeformat="%Y-%m-%d %H:%M:%S"
```

splunk\_prov.query\_time  
✓ 0.0s

Set query time boundaries

Origin Date 2023 / 08 / 03 Time (24h)

Time Range

Query start time (UTC): 2023-07-15 02:51:03.533692

Query end time (UTC) : 2023-08-21 02:51:03.533692



# Jupyter's pros: Use of much ML/DL

- Only a few ML models have built-in msticpy
  - Event Clustering
  - Time Series Analysis and Anomalies
  - Outlier Identification
  - less parameter tuning is required since they are specialized for commonly used threat hunting applications
- Flexibility to use Python's rich ML/DL library

NLP

**MeCab**

 **GINZA**

ML

 **scikit-learn**

 **XGBoost**

 **stumpy**

 **SM StatsModels  
Statistics in Python**

DL

 **PYTORCH**

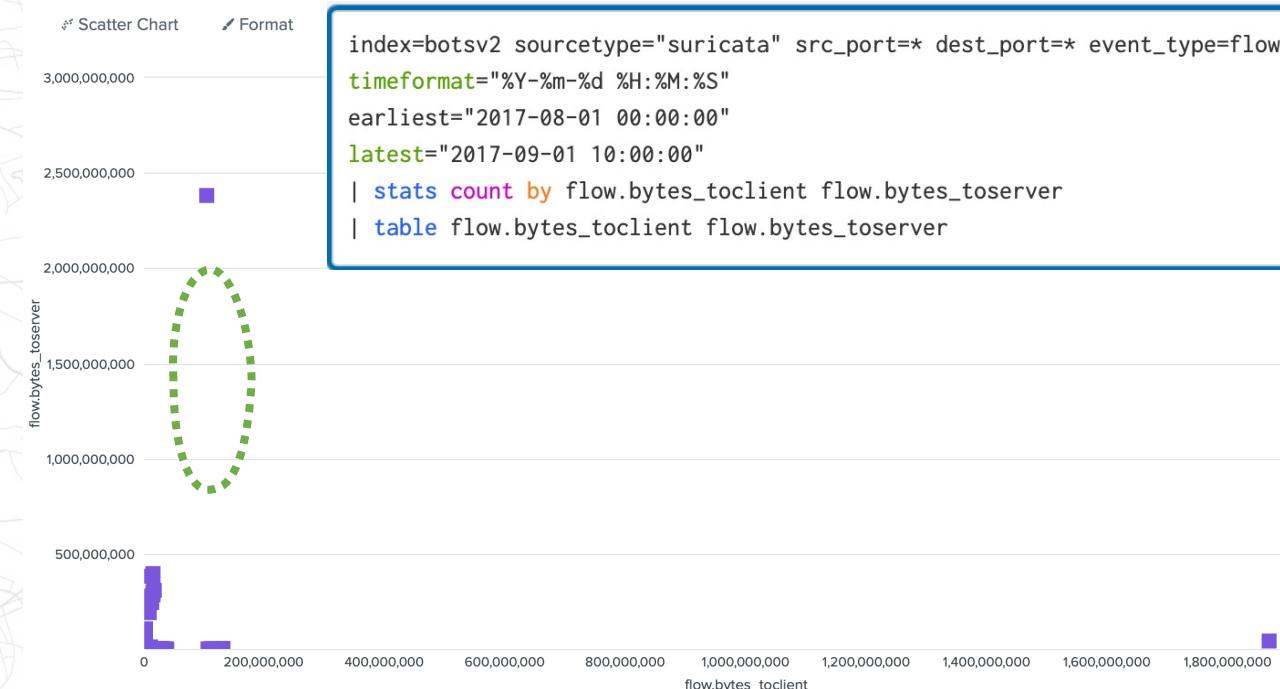
 **TensorFlow**

 **Keras**

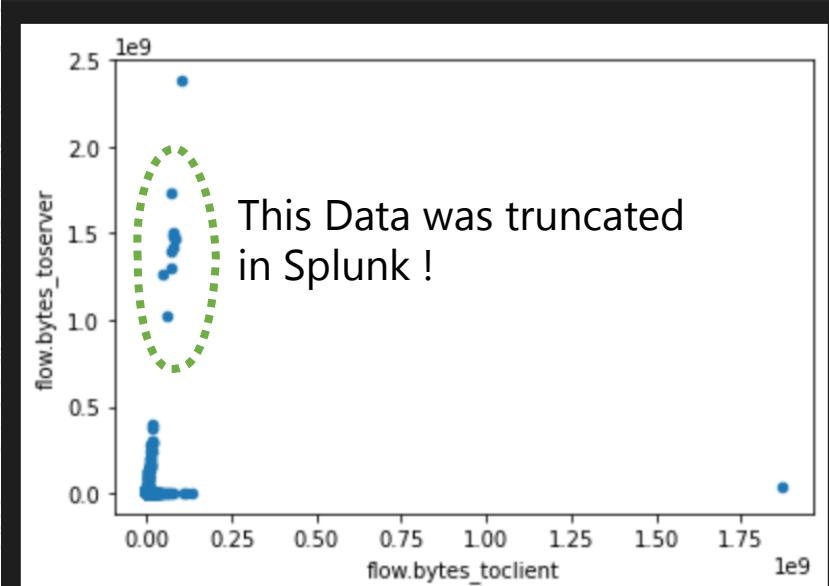
 **Chainer**



# Jupyter's pros: Infinite Visualization



```
splunk_query = ''  
search index=botsv2 sourcetype="suricata" src_port==* dest_port==* event_type=flow  
timeformat="%Y-%m-%d %H:%M:%S"  
earliest="2017-08-01 00:00:00"  
latest="2017-09-01 10:00:00"  
| stats count by flow.bytes_toclient flow.bytes_toserver  
| table flow.bytes_toclient flow.bytes_toserver  
''  
suricata_df = splunk_prov.exec_query(splunk_query,timeout=300)  
print(len(suricata_df))  
suricata_df.head()  
  
# important type cast  
suricata_df['flow.bytes_toclient'] = suricata_df['flow.bytes_toclient'].astype(int)  
suricata_df['flow.bytes_toserver'] = suricata_df['flow.bytes_toserver'].astype(int)  
suricata_df.plot.scatter(x='flow.bytes_toclient', y='flow.bytes_toserver')
```



Maximum number of data plots (by default)

Splunk	MS Sentinel	Jupyter
10,000	10,000	$\infty$ (Infinity)

# [FYI] Change the upper limit in the dashboard options



- We can change the limit with the dashboard option "charting.data.count" in Splunk, but...

Search   Analytics   Datasets   Reports   Alerts   Dashboards   > Search & Reporting

suricata\_viz\_all

View all plot

flow.bytes\_toserver

flow.bytes\_toclient

Edit   Export   ...

i No validation issues

```
<dashboard version="1.1">
  <label>suricata_viz_all</label>
  <row>
    <panel>
      <title>View all plot</title>
      <chart>
        <search>
          <query>index=botsv2 sourcetype="suricata" src_port=* de</query>
          <timeformat>%Y-%m-%d %H:%M:%S</timeformat>
          <earliest>2017-08-01 00:00:00</earliest>
          <latest>2017-09-01 10:00:00</latest>
        </search>
        <option name="charting.chart">scatter</option>
        <option name="charting.data.count">100000</option> // Line 19
        <option name="charting.drilldown">none</option>
        <option name="refresh.display">progressbar</option>
      </chart>
    </panel>
  </row>
</dashboard>
```



# Jupyter's pros: Automation with papermill



- Python library
- Batch execution of Notebook files with different parameters
- Introduced in the "Put it into Operation" section at the end of msticpy's training materials

CUI

```
!papermill input.ipynb output.ipynb -p start_strtime "2017-08-25 00:00:00" -p end_strtime "2017-08-26 00:00:00"
✓ 13.0s

Input Notebook: input.ipynb
Output Notebook: output.ipynb
Executing: 0% | 0/11 [00:00<?, ?cell/s] Executing notebook with kernel: pyth
Executing: 100% | 11/11 [00:11<00:00, 1.07s/cell]
```

Python

```
import papermill as pm
pm.execute_notebook(
    'input.ipynb',
    'output_pyapi.ipynb',
    parameters=dict(
        start_strtime="2017-08-25 00:00:00",
        end_strtime="2017-08-26 00:00:00"
    )
)
```

Parameters are overwritten in the output notebook

```
start_strtime = "2017-08-24 00:00:00"
end_strtime = "2017-08-25 10:00:00"
× parameters + Tag

# Parameters
start_strtime = "2017-08-25 00:00:00"
end_strtime = "2017-08-26 00:00:00"

× injected-parameters + Tag
```

# Jupyter's cons: Security Concerns about Data Transfer



- Possibility to transfer sensitive data in SIEM to external Jupyter
  - Handling it with SIEM's ACL may be the only way.
- Eavesdropping/MITM Attack during data transfer to the Jupyter
  - SSL security dependencies on the SIEM side
- More complicated security design
- Transferring Threat Intelligence data to SIEM is relatively clear.





# msticpy 301

## Practical use case

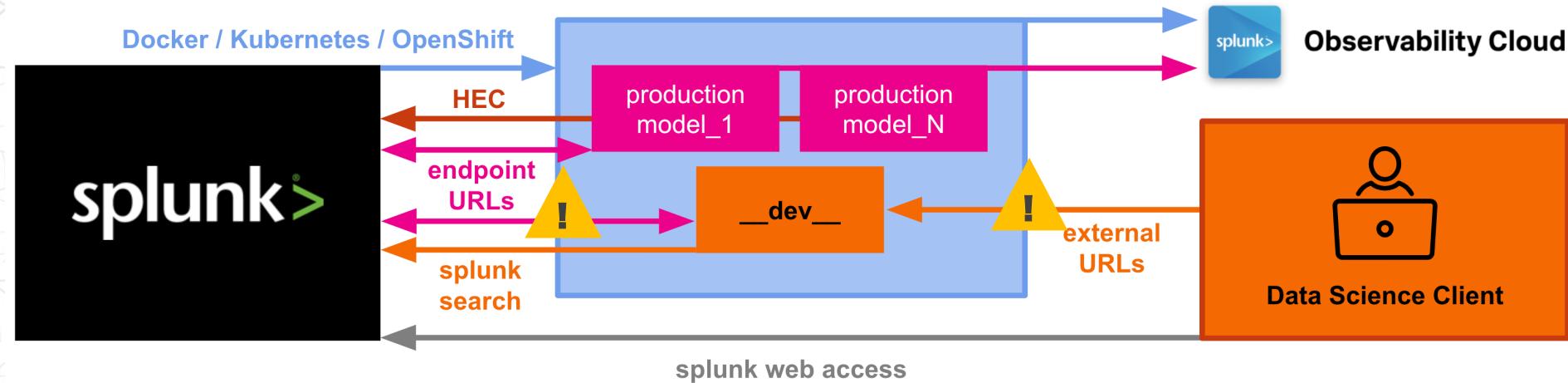


# Toward Practical msticpy Use

- Push direction is fine
  - Intelligence collected from external sources, analyzed and processed, and transferred to SIEM
- Pull direction has the **security concern of data transferring.**
  - Planning a new security design from scratch for msticpy alone is a hurdle.
  - SIEM vendor's advanced analytical tricks with Jupyter
    - MS Sentinel ↗ 「Microsoft Azure Machine Learning Workspace」
      - Completed within Azure
    - Splunk ↗ 「Splunk App for Data Science and Deep Learning (DSDL)」
      - Preparing machine resources such as Docker containers externally
      - Data exchange between containers and Splunk
      - Installing msticpy in container side
  - + Store the credential strings in "Azure Key Vault" and load them from there



# \$more Splunk App for DSDL



- single-instance | **side-by-side**
- Implemented data security features
  - Use of proprietary SSL certificates
  - Custom password settings for Jupyter
  - Fine-grained ACL design with Splunk access tokens
- Splunk MLTK commands can interact with containers
  - `| fit` ( Training to create a model )
  - `| apply` ( Apply the trained model to the data for identification )

# Use Case: Powershell process command line(1)



Search in Splunk

powershell  
-enc

Decode  
base64

Delete null  
byte (\x00)

Extract IoC

Enrichment  
IoC

Return to  
Splunk

| fit

Required the first time for model creation

| apply

Originally, this mechanism is prepared for ML/DL algorithms, so I developed a **custom model incorporating msticpy**.

```
def fit(model,df,param):  
    # model.fit()  
    info = {"message": "passthru model creat  
    return info
```

By executing the fit command,  
one .py file is created in app/model directory,  
the file is consisting of export functions from .ipynb

```
def apply(model,df,param):  
    mp.init_notebook()  
  
    # decode base64  
    dec_df = mp.transform.base64unpack.unpack  
    # remove nullbyte \x00  
    i=0  
    while i < len(dec_df["decoded_string"]):  
        dec_df.at[dec_df.index[i], 'decode_v  
        i+=1  
    # extract ioc  
    ioc_df = dec_df.mp.ioc_extract(columns=[  
        ioc_types=['ipv4','ipv6','dn  
    # format ioc to dict  
    ioc_input = {}  
    for _, row in ioc_df.iterrows():  
        ioc_type = row['ioc_type']  
        if ioc_type == 'ip':  
            ioc_value = row['ioc_value']  
            if ioc_value not in ioc_input:  
                ioc_input[ioc_value] = {  
                    'ioc_type': ioc_type  
                }  
            else:  
                ioc_input[ioc_value].append({  
                    'ioc_type': ioc_type  
                })  
        elif ioc_type == 'domain':  
            ioc_value = row['ioc_value']  
            if ioc_value not in ioc_input:  
                ioc_input[ioc_value] = {  
                    'ioc_type': ioc_type  
                }  
            else:  
                ioc_input[ioc_value].append({  
                    'ioc_type': ioc_type  
                })  
        elif ioc_type == 'file':  
            ioc_value = row['ioc_value']  
            if ioc_value not in ioc_input:  
                ioc_input[ioc_value] = {  
                    'ioc_type': ioc_type  
                }  
            else:  
                ioc_input[ioc_value].append({  
                    'ioc_type': ioc_type  
                })  
        elif ioc_type == 'url':  
            ioc_value = row['ioc_value']  
            if ioc_value not in ioc_input:  
                ioc_input[ioc_value] = {  
                    'ioc_type': ioc_type  
                }  
            else:  
                ioc_input[ioc_value].append({  
                    'ioc_type': ioc_type  
                })  
    return ioc_input
```

# Use Case: Powershell process command line(2)



```
index=botsv2 "powershell" "-enc" source="WinEventLog*Microsoft-Windows-Sysmon*Operational*"
| where LIKE(ParentCommandLine,"%powershell%-enc%") | stats count by ParentCommandLine
| fit MLTKContainer algo=msticpy_powershell_ioc ParentCommandLine into app:process_b64_iocs_enrich
```

fit

※Example of Splunk botsv2 dataset

✓ 510 events (before 8/7/23 7:18:35.0

Events Patterns Statistics (3)

20 Per Page ▾ Format Preview

```
index=botsv2 "powershell" "-enc" source="WinEventLog*Microsoft-Windows-Sysmon*Operational*"
| where LIKE(ParentCommandLine,"%powershell%-enc%") | stats count by ParentCommandLine
| apply process_b64_iocs_enrich
| rename predicted_* as *, asn_country_code as asn_cc, asn_description as asn_desc
| table decode_validated ioc_ipv4 ioc_url asn asn_cc asn_desc ParentCommandLine|
```

✓ 510 events (before 8/7/23 7:20:26.000 PM) No Event Sampling ▾

Events Patterns Statistics (3) Visualization

20 Per Page ▾ Format Preview ▾

predicted\_ParentCommandLine ▾

"C:\Windows\System32\WindowsPowerShell\v1.0\msticpy.ps1" WwBSAGUAZgBdAC4AQQBzAFMARQBtAGIAT

msticpy results

```
SYstem.NET.SERVICEPOInTManagER]::EXPEct100ContInue=0;$wC=NEW-ObjEcT
ValidationCallback = {$true};$wC.HeAdErs.AdD('User-Agent', $u);$WC.Proxy=::ASCII.GetByTeS('389288edd78e8ea2f54946d3209b16b8');$R=[$]="$S[$H],$S[$I];$_-
```

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell -noP -sta -w 1 -enc WwBSAGUARgBdAC4AQQBTAHMARQBNAGIAT
```

```
[SYstEM.Net.SERviCePoINTManAGer]::EXpEct100ContInuE=0;$Wc=NeW-Object
ValidationCallback = {$true};$WC.HEADers.Add('User-Agent', $u);$WC.Proxy=::ASCII.GetBytes('389288edd78e8ea2f54946d3209b16b8');$R=[$]="$S[$H],$S[$I];$_-
```

```
powershell -noP -sta -w 1 -enc WwBSAEUARgBdAC4AQQBTAFMARQBtAGIAb
```

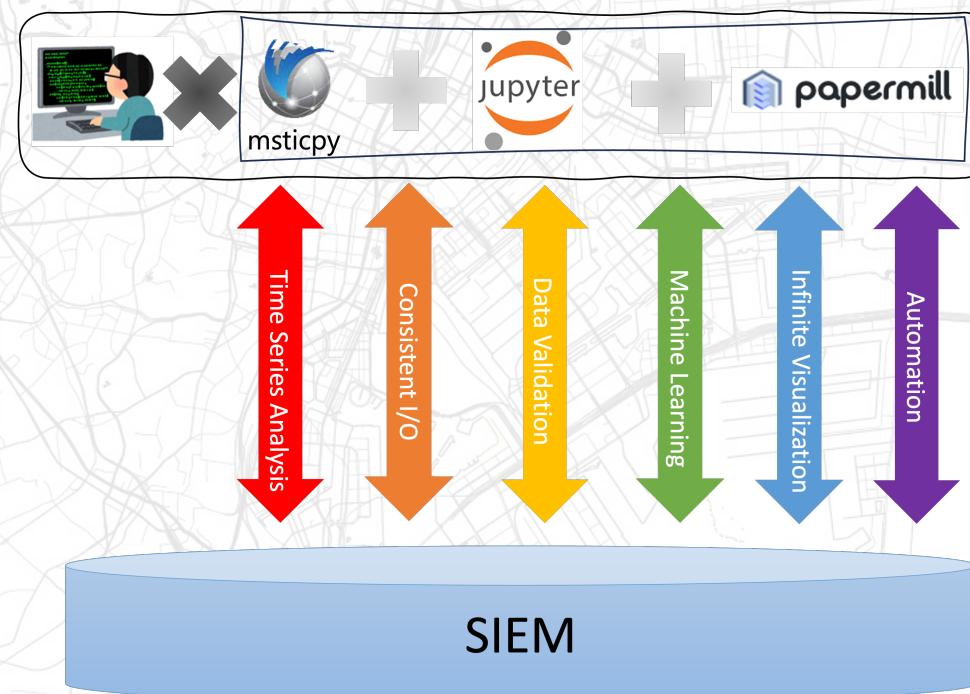
```
SySTEM.Net.SerVIcEPoInTManAGEr]::EXPEcT100CONTinue=0;$wC=NEw-OBJEcT
ValidationCallback = {$true};$wC.HeAdErs.Add('User-Agent', $u);$wC.PROxy=::ASCII.GetBYTeS('389288edd78e8ea2f54946d3209b16b8');$R=[$]="$S[$H],$S[$I];$_-
```

apply



# Take Away

- Not recommend to rely too much on SIEM analysis!
- msticpy's missionary work: happy to see more APAC users
- Let's analyze and code on Jupyter Notebook to hone your skills!
- Let's get on existing mechanisms for data security concerns!
- Let's become a contributor of your favorite OSS. **Happy msticpying!**



# Quotations & References



- msticpy docs <https://msticpy.readthedocs.io/en/latest/>
- msticpy-training <https://github.com/microsoft/msticpy-training>
- msticpy-lab <https://github.com/microsoft/msticpy-lab>
- Splunk DSDL docs <https://docs.splunk.com/Documentation/DSDL/5.1.0/User/IntroDSDL>
- Splunk botsv2 dataset <https://github.com/splunk/botsv2>
- Microsoft Sentinel Notebook and msticpy <https://learn.microsoft.com/en-us/azure/sentinel/notebook-get-started>
- papermill docs <https://papermill.readthedocs.io/en/latest/>
- macnica SIEM introduction by exabeam  
[https://www.macnica.co.jp/business/security/manufacturers/exabeam/feature\\_07.html](https://www.macnica.co.jp/business/security/manufacturers/exabeam/feature_07.html)
- My Qiita blog about msticpy <https://qiita.com/hackeT>
- Machine Learning for Security Engineers <https://www.oreilly.co.jp/books/9784873119076/>
- awesome detection engineering <https://github.com/infosecB/awesome-detection-engineering>
- CardinalOps's 2023 report <https://cardinalops.com/whitepapers/2023-report-on-state-of-siem-detection-risk/>



Thank you !

