IST-652

Security analytics and recommendations based on machine-learning in distributed-systems

Professor: Debbie Landowski
Student: Sharat Sripada (vssripad)

TABLE OF CONTENTS

Introduction

Securing enterprise/cloud data can be realized via what is commonly known as network firewall policies or rules. In the traditional world, perimeter firewalls can achieve this but with the advent of software defined networks (SDN) an administrator typically calls some APIs on an SDN controller which then pushes the intent down to a distributed data-plane where it is enforced on workloads like virtual-machines, PODs or containers.

Staying on top of constantly varying applications, traffic patterns and new attack vectors at scale can take administrators several repeated iterations to make efficient and this can be particularly daunting at large scale – commonly also known as Massively Scalable Datacenters (MSDC).

The project will attempt, through the knowledge gained in IST-652 to retrieve results of firewall actions written to log files from across several nodes in the distributed plane, analyze patterns and suggest recommendations (through machine learning algorithms) and present it via a security analytics dashboard.

Below is the high-level workflow and architecture:

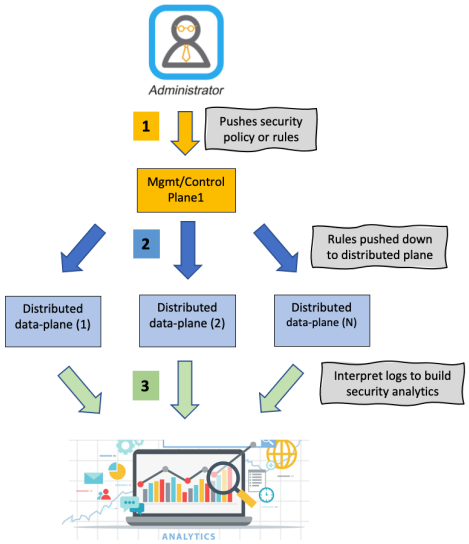


Fig. Workflow, process and Architecture

Data source and description

The source of the data will largely be logs written by a component enforcing network policies or rules. It captures traffic flows or tuples essentially hitting an action (allow/deny).

Here is an excerpt of the log:

```
2017-10-19T22:38:05.586Z 58734 INET match PASS domain-c8/1006 OUT 84 ICMP
172.18.8.121->172.18.8.119 RULE_TAG
2017-10-19T22:38:08.723Z 58734 INET match PASS domain-c8/1006 OUT 60 TCP
172.18.8.121/36485->172.18.8.119/22 S RULE_TAG
```

```
2017-10-19T22:38:18.785Z 58734 INET TERM domain-c8/1006 OUT ICMP 8 0
172.18.8.121->172.18.8.119 2/2 168/168 RULE_TAG
2017-10-19T22:38:20.789Z 58734 INET TERM domain-c8/1006 OUT TCP FIN
172.18.8.121/36484->172.18.8.119/22 44/33 4965/5009 RULE_TAG
```

NOTE
See reference [1] for more details.

Description
A traffic flow or tuple would comprise the following fields:

- src ip-address
- dst ip-address
- src port
- dst port
- TCP/IP Protocol
- Action – Deny/Permit

Viewing this through an example:

```
# tail -f /var/log/dfwpktlogs.log | grep 192.168.110.10

2015-03-10T03:20:31.274Z INET match DROP domain-c27/1002 IN 60 PROTO 1
192.168.110.10->172.16.10.12
2015-03-10T03:20:35.794Z INET match DROP domain-c27/1002 IN 60 PROTO 1
192.168.110.10->172.16.10.12
```

To generate a sufficiently large dataset, the idea is to simulate several traffic flows based on popular applications and therefore cause a corresponding number of actions to be logged.

Research Questions

Following are some of the questions we seek to answer:

1. What are the traffic patterns or flows across the datacenter?
   - Analyze and visualize traffic patterns – get counts of flows across TCP, UDP and ICMP protocols
2. Provide rule metrics across all network policies
3. Plot a time-series graphs:
   - For high hitting flows (allow/deny) identify spikes, saw-tooth behavior or sustained/repeated patterns etc.
4. Finally, based on traffic patterns make recommendations to add/remove/update network policies or rules. Explore machine-learning algorithms to achieve this.

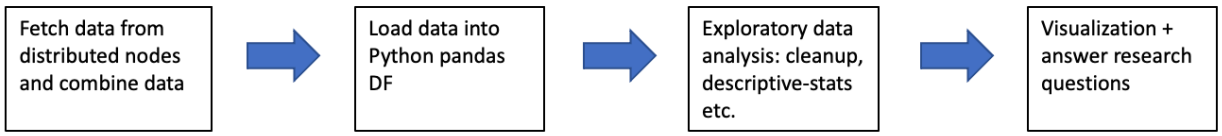


Fig. Workflow

Data Analysis

Gathering data

As a first step, 1,000 flow records were collected across 128x distributed plane nodes and written to .pkl file using the Python pickle utility. The data is stored in a Python dictionary in the following format:

{
Host-1: [<flow-1>, <flow-2>, <flow-3>, …<flow-1000>],
Host-2: [<flow-1>, <flow-2>, <flow-3>, …<flow-1000>],
Host-3: [<flow-1>, <flow-2>, <flow-3>, …<flow-1000>],
.
.
Host-128: [<flow-1>, <flow-2>, <flow-3>, …<flow-1000>]
}

where:

- *Host*-<>: Is ip-address of host with virtual-machines reporting flows
- *flow*-<>: Packet tuple comprising various details of source, destination, protocol etc.

  Example flows:

  Flow-1: 2020-12-05T20:51:02.220Z 073596a1 INET match DROP 2026 IN 36 PROTO 2 0.0.0.0->224.0.0.1
  Flow-2: 2020-12-05T20:51:02.428Z 073596a1 INET match DROP 2026 OUT 92 UDP 192.2.0.2/45325->20.20.0.1/53
  Flow-3: 2020-12-05T20:18:52.003Z b5de0e44 INET TERM 3106 IN TCP FIN 100.64.48.39/4302->192.2.0.118/80 5/5 401/593
  Flow-4: 2020-12-05T09:11:30.739Z 073596a1 INET match DROP 2026 OUT 60 TCP 192.2.0.2/46164->192.2.0.4/9887 S
  .
  Flow-N: 2020-12-05T20:19:39.367Z cf6e2066 INET match PASS 1075 IN 88 ICMP 192.1.0.209->192.1.0.213

Summary of records loaded from .pkl file:

```
pkl_file = open('mypkl.pkl', 'rb')
dfw_dict = p.load(pkl_file)

# Examine a few key/values, get size of data
flow_count = 0
active_sources = []
for ip in dfw_dict:
    if dfw_dict[ip]:
        flow_count += len(dfw_dict[ip])
        active_sources.append(ip)
print('Active sources: %d, Active flows: %d' %(len(active_sources),
                                               flow_count))
```

```
Active sources: 80, Active flows: 77423
```

Fig. Using pickle to unpack data

NOTE
To state this explicitly, the size of data would be 77,423 records (or rows in a data-frame).

Mongo-DB to store flow-records

Flow-records were then needed to be translated into the following format to be stored in Mongo-DB:

Time	Host	Reason	Action	Rule	Dir	Len	Proto	Src-IP	S-port	Dst-IP	D-port
<>	<>	match	DROP	2026	IN	36	2	0.0.0.0		224.0.0.1	
<>	<>	match	DROP	2026	OUT	92	UDP	192.2.0.2	45325	20.20.0.1	53
<>	<>	0	TERM	3106	IN	0	TCP	100.64.48.39	4302	192.2.0.118	9887
<>	<>	match	DROP	2026	OUT	60	TCP	192.2.0.2	46164	192.2.0.4	9887
<>	<>	match	PASS	1075	IN	88	ICMP	192.1.0.209		192.1.0.213	

Fig: Flow-records translated from raw-format

Correspondingly, this flow data was translated into a *key:value* store or map and inserted into Mongo-DB (**DB-name:** flowdb, **Collection-name:** flow_collection) using the pymongo *insert()* utility:

```
doc = {'time': _time, 'source': _src, 'reason': _reason, 'action': _action,
        'rule': _ruleid, 'dir': _dir, 'pktlen': _pktlen,
        'proto': _proto, 'sip': _sip, 'dip': _dip,
        'sport': _sport, 'dport': _dport}
```

Fetch and examine documents and confirm count of documents/records:

```
# Fetch + examine a few entries from Mongo-DB
doclist = flow_collection.find()
for doc in doclist[:10]:
    print(doc)
print('Total records in the DB = {0}'.format(doclist.count()))
```

```
{'_id': ObjectId('5fcca3b7a7d4d6819b6e9eb1'), 'time': '2020-12-05T20:51:02.220Z', 'source': '20.20.177.78', 'reason
': 'match', 'action': 'DROP', 'rule': '2026', 'dir': 'IN', 'pktlen': '76', 'proto': 'ICMP', 'sip': 'fe80::ffff:ffff
:ffff:ffff', 'dip': 'ff02::1', 'sport': 0, 'dport': 0}
{'_id': ObjectId('5fcca3b7a7d4d6819b6e9eb2'), 'time': '2020-12-05T20:51:02.428Z', 'source': '20.20.177.78', 'reason
': 'match', 'action': 'DROP', 'rule': '2026', 'dir': 'OUT', 'pktlen': '92', 'proto': 'UDP', 'sip': '192.2.0.2', 'di
p': '20.20.0.1', 'sport': '45325', 'dport': '53'}
{'_id': ObjectId('5fcca3b7a7d4d6819b6e9eb3'), 'time': '2020-12-05T20:51:03.175Z', 'source': '20.20.177.78', 'reason
': 'match', 'action': 'DROP', 'rule': '2026', 'dir': 'OUT', 'pktlen': '92', 'proto': 'UDP', 'sip': '192.2.0.2', 'di
p': '20.20.0.1', 'sport': '50584', 'dport': '53'}
{'_id': ObjectId('5fcca3b7a7d4d6819b6e9eb4'), 'time': '2020-12-05T20:51:07.671Z', 'source': '20.20.177.78', 'reason
': 'match', 'action': 'DROP', 'rule': '2026', 'dir': 'OUT', 'pktlen': '92', 'proto': 'UDP', 'sip': '192.2.0.2', 'di
p': '20.20.0.1', 'sport': '45325', 'dport': '53'}
{'_id': ObjectId('5fcca3b7a7d4d6819b6e9eb5'), 'time': '2020-12-05T20:51:08.421Z', 'source': '20.20.177.78', 'reason
': 'match', 'action': 'DROP', 'rule': '2026', 'dir': 'OUT', 'pktlen': '92', 'proto': 'UDP', 'sip': '192.2.0.2', 'di
p': '20.20.0.1', 'sport': '50584', 'dport': '53'}
{'_id': ObjectId('5fcca3b7a7d4d6819b6e9eb6'), 'time': '2020-12-05T20:51:09.221Z', 'source': '20.20.177.78', 'reason
': 'match', 'action': 'DROP', 'rule': '2026', 'dir': 'IN', 'pktlen': '36', 'proto': '2', 'sip': '0.0.0.0', 'dip': '
224.0.0.1', 'sport': 0, 'dport': 0}
Total records in the DB = 77423
```

Fig. Sample documents in a Mongo-DB collection

Pre-processing, cleaning and munging data

Data from Mongo-DB was then loaded into a Pandas data-frame and examined for missing values, as-types, if normalizations were required etc.

Following seemed necessary:

(1) proto column had values or integers (2) which were converted to a string – 'IGMP' (so as to be uniform with other protocol values ICMP, TCP, UDP in that column)

(2) sport and dport columns of type *object* were converted to as-type *int* which was required when loading the data-frame into scikit encoders

Here is a snapshot of the data using the *head()* and *tail()* utility after above steps:

	time	reason	source	action	rule	dir	pktlen	proto	sip	dip	sport	dport
0	2020-12-05T20:51:02.220Z	match	20.20.177.78	DROP	2026	IN	36	IGMP	0.0.0.0	224.0.0.1	0	0
1	2020-12-05T20:51:02.220Z	match	20.20.177.78	DROP	2026	IN	36	IGMP	0.0.0.0	224.0.0.1	0	0
2	2020-12-05T20:51:02.220Z	match	20.20.177.78	DROP	2026	IN	76	ICMP	fe80::ffff:ffff:ffff:ffff	ff02::1	0	0
3	2020-12-05T20:51:02.220Z	match	20.20.177.78	DROP	2026	IN	36	IGMP	0.0.0.0	224.0.0.1	0	0
4	2020-12-05T20:51:02.220Z	match	20.20.177.78	DROP	2026	IN	76	ICMP	fe80::ffff:ffff:ffff:ffff	ff02::1	0	0

	time	reason	source	action	rule	dir	pktlen	proto	sip	dip	sport	dport
77418	2020-12-05T21:01:26.152Z	0	20.20.177.139	TERM	1018	TCP	0	TCP	192.1.0.4	192.1.0.3	38802	9887
77419	2020-12-05T21:02:16.306Z	match	20.20.177.139	PASS	1018	OUT	81	UDP	192.1.0.3	20.20.0.1	59158	53
77420	2020-12-05T21:02:16.306Z	match	20.20.177.139	PASS	1018	OUT	81	UDP	192.1.0.3	20.20.0.1	51603	53
77421	2020-12-05T21:02:17.320Z	match	20.20.177.139	PASS	1018	OUT	81	UDP	192.1.0.3	20.20.0.1	38484	53
77422	2020-12-05T21:02:17.320Z	match	20.20.177.139	PASS	1018	OUT	81	UDP	192.1.0.3	20.20.0.1	40096	53

Fig: Cleaned data in a data-frame

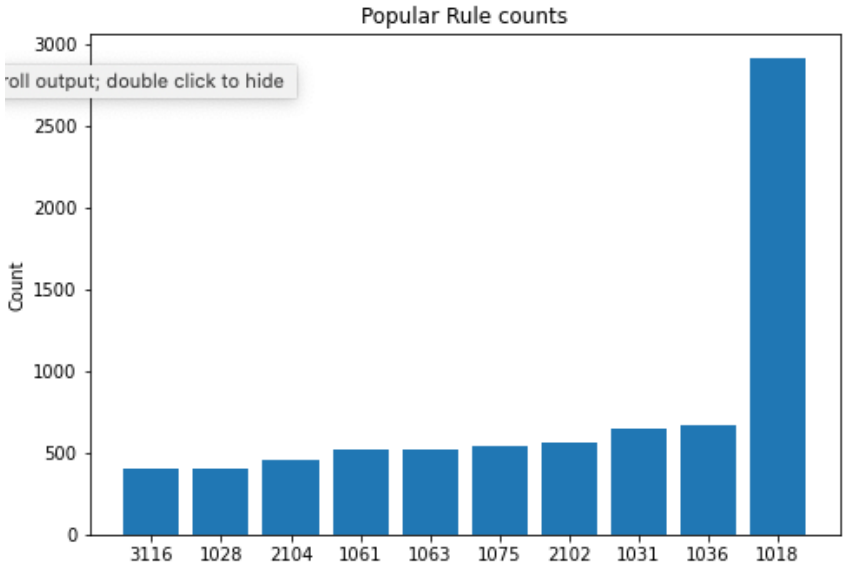
Analytics and security dashboard

This section will capture and present insights into security policies or firewall rules that pervasively work in a datacenter comprising distributed nodes and workloads. As noted earlier, see reference [1] about how logging packet outcomes can help here.

Popular Rule Counts

This metric or visualization helps identify popular rule count:

- Thousands of security policies or firewall rules may be enforced in a datacenter and often there is scope for these rules to run redundantly therefore consuming memory and other critical resources on hosts.
- It also helps see traffic patterns and negate apparent or obvious anomalies



Summary note:
The bar-plot captures the top-10 security policies or rules by popularity. Rule **#1018** is popular by a far margin and security admins would navigate to the Management Plane and investigate into its posture or why it is frequently hit

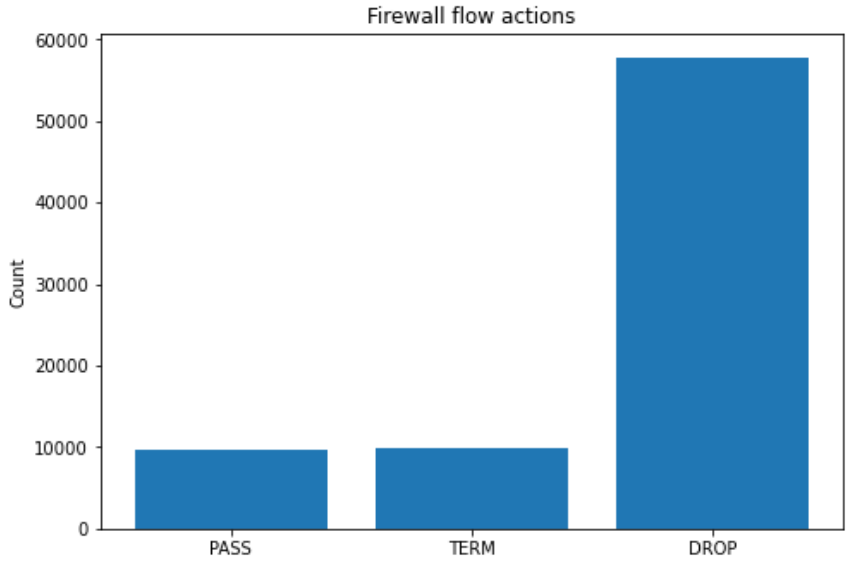
Firewall Action

Firewalls make important decisions based on what they have been told to enforce. Here are a few common or popular actions:

- PASS: Accept the packet.
- DROP: Drop the packet.
- NAT: SNAT rule.
- PUNT: Send the packet to a service VM running on the same hypervisor of the current VM.
- REDIRECT: Send the packet to network service running out of the hypervisor of the current VM.
- REJECT: Reject the packet.

See reference [1] for all possible actions.

In this section, flows would be analyzed to bring out where Firewalls are spending CPU cycles.



Summary note:
It may be worthwhile to understand why such a large proportion of flows are being ***DROPPED*** and possibly mitigate the threat by quarantining the corresponding virtual-machines or ports

Drop packet analysis by Destination and Ports

The next interesting piece would be to identify the source, destination and protocols that are being DROPPED by the firewall.

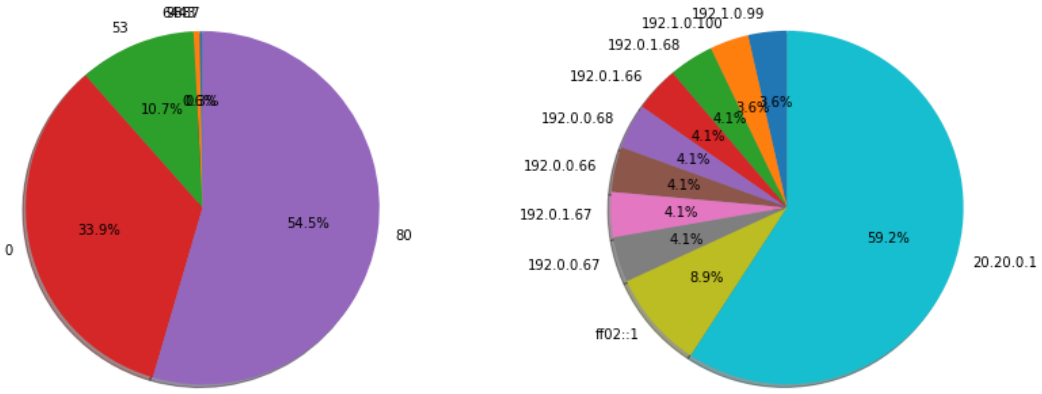


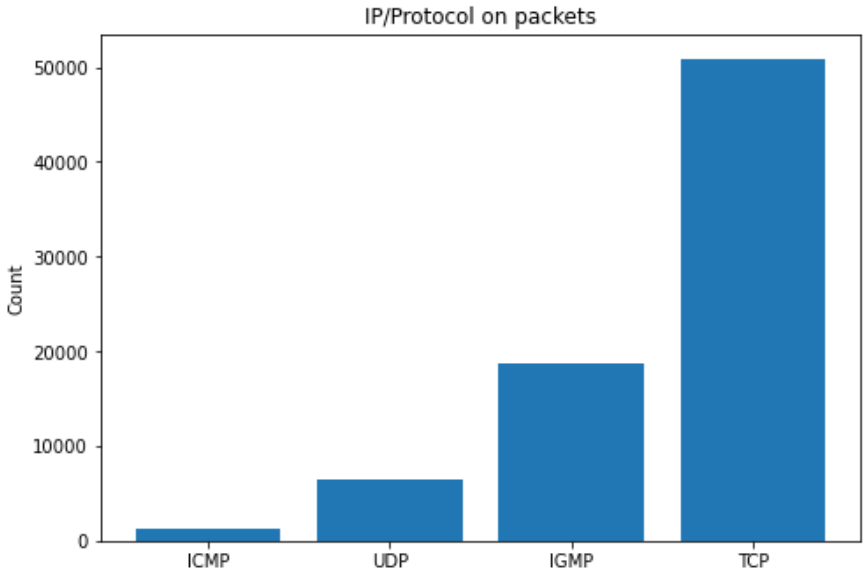
Fig. Pie-charts that drill into flows being DROPPED by Firewall

Summary note:
Most sources are attempting to communicate with destination ip-address *20.20.0.1* and port *80 (HTTP protocol)* and are being rejected by the firewall

Flows based on protocol

Finally, it helps to understand the nature of applications in this datacenter.

The commonly known protocols applications are built on are TCP and UDP which are connection-oriented and connectionless/best-effort IP protocols respectively.



Summary note:
As expected, a majority of the applications have been implemented on TCP or connection-oriented protocols

NOTE
IGMP is a multicast based L2 protocol and security admins may want to understand the nature of these applications (multicast is rare among applications)

Conclusion

This is merely a start or proof-of-concept, but rich Security Analytics Dashboards can be built crunching and presenting data at near real-time.

Recommendations and Predictions

The final section presents a view into making recommendations and predictions using Machine-learning algorithms.

For the purpose of this project, ***Decision-Tree*** Classification and prediction techniques have been used in conjunction with ***Gini Index*** Attribute Selection Measure. Further, sklearn libraries for pre-processing/encoding data have been used:

- LabelEncoder - Encode *Target variables*
- OneHotEncoder - Encode *Feature variables*

The encoders help transform and classify discrete datatypes into data that can be understood and interpreted by the *DecisionTreeClassifier()*

```
# Using Decision-Trees make predictions
# Since Information Gain(IG) methods have known to have some biases
# used the GINI method (which is also the default Attribute Selection Measure)

# Choose feature and target variables
feature_cols = ['proto', 'sip', 'dip', 'sport', 'dport']
x = df[feature_cols]
y = df['action']

# Data-preprocessing using scikitlearn packages:
#  - LabelEncoder - Encode target variables (here 'action')
#  - oneHotEncoder - Encode feature variables

# Step-1:
# Using the LabelEncoder transform the predictor variable
le = preprocessing.LabelEncoder()

# Fit LabelEncoder
le.fit(y)

# Check/print the classes - expect to see TERM, PASS, DROP
print('LabelEncoder classes for Actions = {0}'.format(le.classes_))

# Transform/encode values based on LabelEncoder
y_transform = le.transform(y)

# Step-2:
# Using the OneHotEncoder scheme to transform feature variables
enc = OneHotEncoder(handle_unknown='ignore')

# Fit OneHotEncoder
enc.fit(x)

# Transform/encode feature columns
x_transform = enc.transform(x)
```

```
LabelEncoder classes for Actions = ['DROP' 'PASS' 'TERM']
```

Fig. Shows the encoding techniques on x & y variables

Also, see references [2] and [3] for usage.

Finally, split the data - 80% for training and 20% for testing before running predictions and determine the accuracy.

```
# Step-5:
# Check the accuracy
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

# Conclusion:
# Using Decision-Trees (GINI method), given certain packet/flow attributes we
# predict with upto 78% accuracy if Firewall action is DROP, PASS or TERM
```

```
Accuracy: 0.7804326767839845
```

Summary note:
As seen, the Decision-Tree is predicting outcomes at 78% accuracy

Conclusion

The Project has illustrated use of techniques and knowledge gained through IST-652 and Prof. Debbie Landowski.

The course has greatly helped expand scope in using Python as the programming language for data-analysis while understanding the nuances of the types of data, means to store it and restore it into data-frames when presenting analysis. It greatly helped:

- Realize the sought pipeline

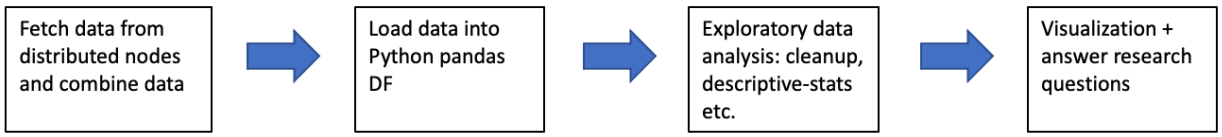


Fig. Workflow

- Analyze, visualize and research on recommendation techniques in a very niche space within the networking paradigm of security, commonly also known as Micro-segmentation.

Future Work

There is tremendous scope for future work, especially when one thinks about existential methods of creating, maintaining, updating and deleting (also known as C.R.U.D) security policies or firewall rules today. Migrating to methods or techniques backed by data-analysis and machine-learning could greatly help adapt to constantly changing attack vectors and datacenter ecosystems.

As next steps:

(1) Improve accuracy of prediction with usage of Random Forests (ensemble methods) or other kernel SVM techniques
(2) Explore application of deep-learning methods
    - Research if database signatures of commonly known threat/attack vectors can be maintained in an MNIST-like datastore
(3) Build a simple utility – given a packet tuple (s.ip, d.ip, ports, protocol) predict Firewall action based on flows extracted

References

[1] NSX Firewall logs - https://docs.vmware.com/en/VMware-NSX-Data-Center-for-vSphere/6.4/com.vmware.nsx.admin.doc/GUID-6F9DC53E-222D-464B-8613-AB2D517CE5E3.html
[2] LabelEncoder usage - https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.LabelEncoder.html
[3] OneHotEncoder usage - https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.OneHotEncoder.html#sklearn.preprocessing.OneHotEncoder
[4] Usage of Decision-Tree in Python - https://www.datacamp.com/community/tutorials/decision-tree-classification-python