Detecting DNS Tunneling

David Purtseladze
Information Security Office
University of Texas at Austin

Domain Name System
01

DNS Tunneling
02

Malicious queries
03

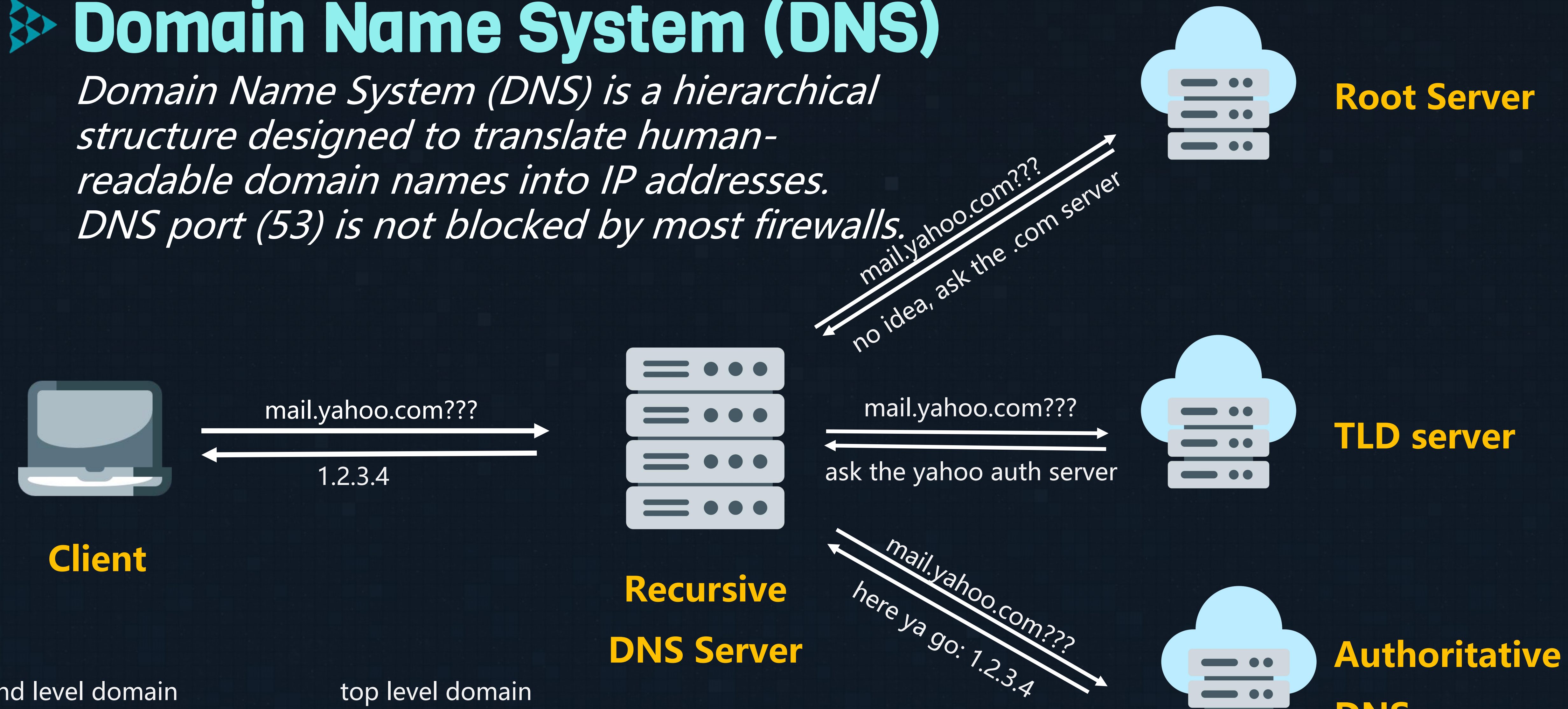
Feature Engineering
04

LSTM Model
05

Training and Deployment
06

Domain Name System (DNS)

Domain Name System (DNS) is a hierarchical structure designed to translate human-readable domain names into IP addresses.
DNS port (53) is not blocked by most firewalls.



2nd level domain top level domain

account.protonvpn.com

1st level domain

What is DNS Tunneling?

The process of encapsulation of various internet protocols over DNS. Used for exfiltrating data, creating covert communication channels, bypassing firewalls.

Encapsulation of protocols through DNS:

- **TCP over DNS - iodine, dns2tcp**
- **IP over DNS - dnscat**
- **DNS beacons - cobaltstrike**

Encoded payload

Data Exfiltration

DNS Data

DNS logs are generated by Zeek. Queries and corresponding responses are parsed and stored together in a single entry.

```
dest_ip: 199.7.91.13
dest_port: 53
proto: udp
query: test.domain.name.zzz
query_length: 20
rcode_name: NXDOMAIN
record_type: A
src_ip: ***.***.***.***
src_port: 58817
ts: 2024-02-02T23:51:41.466837Z
——-----truncated——-----
```

199.7.91.13 is one of the Root name servers, hosted by University of Maryland

NXDOMAIN - non existent domain, name servers could not resolve it to an IP address

record_type: A - the query was made for an IPv4 address

Malicious DNS Queries

These queries were captured in a lab environment after setting up DNS tunneling tools (dns2tcp and cobaltstrike).

dns2tcp (base64 encoded):

```
724w5jd+cmohljqw90ecgx9km4hs/qrpfgt8dg91+cuun0ndjwkhb+lo1/wqtib.r
d27+r9rcnzidhsv55tp9y/obz7svhvzirwd3bneura/rgv/mlxuqijrdfqy9rw.yimiixsh
1zz1kiv9yr9rwuahbeqfxphmlkfqn2dvhrmkqiazo3ziqp.
```

cobaltstrike (hexadecimal encoding):

```
post.1243ae0915408fd6c0acdf56c.24cec631c.3d2da7be.beacon.pacattack.xyz
```

Intuition

Legitimate queries	Potentially malicious queries
• Human readable • Easy to remember • Short, meaningful • Domain name from majestic million • High tier top level domain: .com, .net, .ai	• Long queries • Encoded (base64, hex16, other encoding) • Large number of subdomains • NXDOMAIN response • Cheap throwaway top level domain: .xyz, .top, .pw

tld-list.com

Feature Engineering

The process of extracting and transforming data using domain knowledge to enhance ML model training and performance.

DNS query length	length (L)
A Large number of different characters	unique_characters (U)
High string entropy	L*log(U)
Capitalization (domain names are case insensitive)	uppercase_mask
Mix of letters and digits	alphanumeric_mask
Encoded payload	
A large number of subdomains	
Suspicious top level domain	
Newly registered domain	
Querying TXT Records	

Masks

Using uppercase and alphanumeric mask to helps the model learn the relationship between categorical variables (lowercase chars, uppercase chars, digits).
Models are capable of learning it on their own using embedded layers, but this process requires a lot of training data.

uppercase mask

"g83ng02bg3GO0++0" → "0110011001001001"

alphanumeric mask

"g83ng02bg3GO0++0" → "0110011001111111"

*** for simplicity non letter characters are considered uppercase**

Model selection: LSTM

Long Short Term Memory

Performs well on sequential data

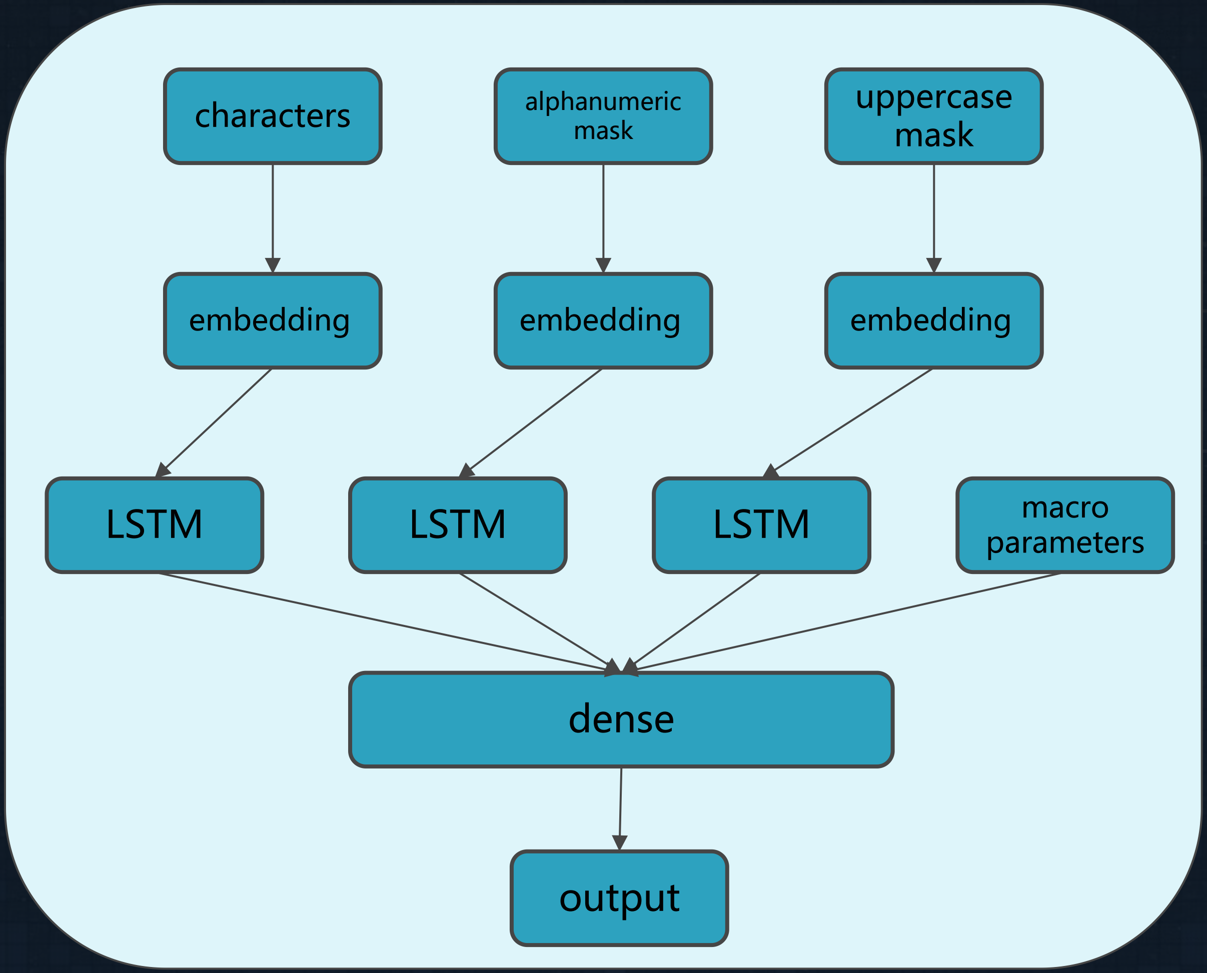
Captures long range dependencies

Can handle variable length inputs

Avoids Exploding/Vanishing gradients

post.1243ae0915408fd6c0acdf56c.24cec631c.3d2da7be.beacon.pacattack.xyz

High Level Neural Network Diagram

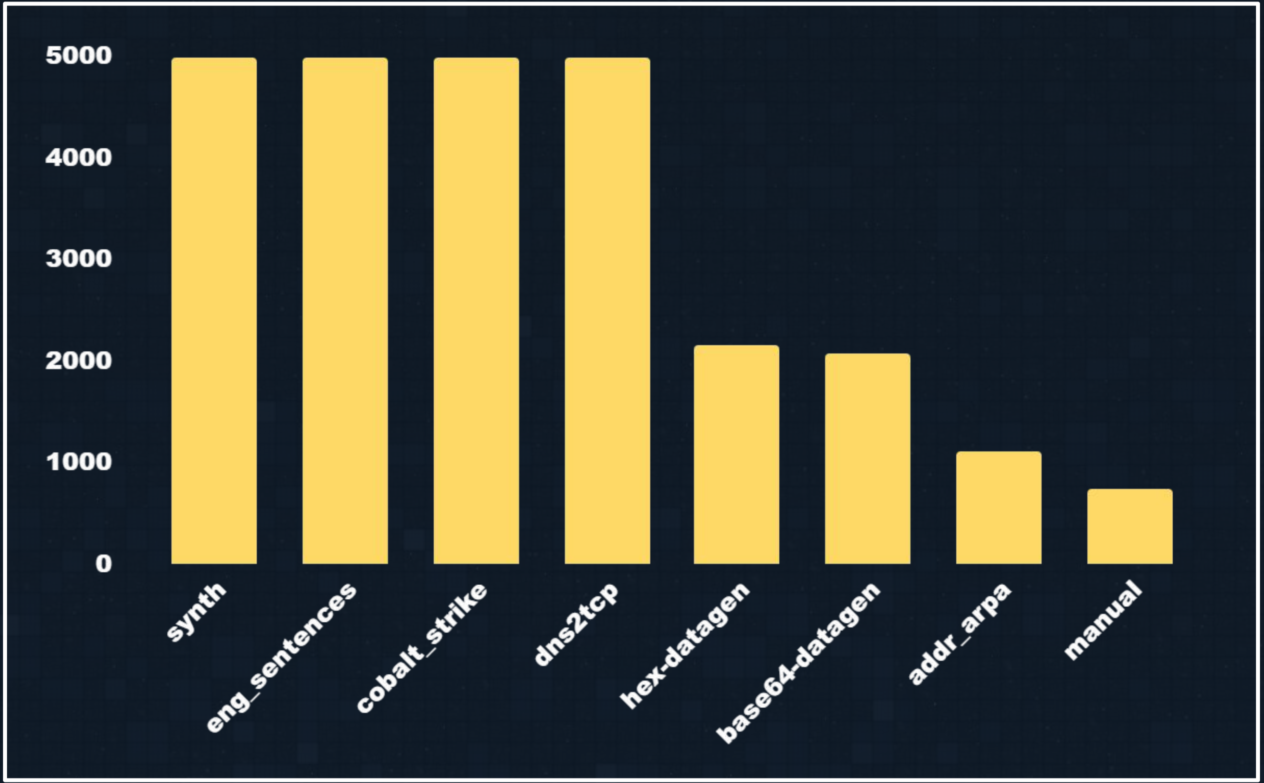


- Character sequence and masks are fed into an embedding layer
- The output of the embedding layer is fed into an LSTM layer
- The output of LSTM layer is concatenated with macro parameters and is fed into a series of dense layers
- The output represents probability of a query being malicious

Training Data

DNS tunneling data is not readily available. We used DNS payloads captured in a lab environment, hex16 and base64 encoded strings, english sentences and synthesized data created by concatenating string segments from other sources.

- **base64, hex16 encoding training**
- **n-gram training (EAT vs QHG)**
- **malicious+bening = malicious**
- **manual data can be relabeled**
- **26108 samples**



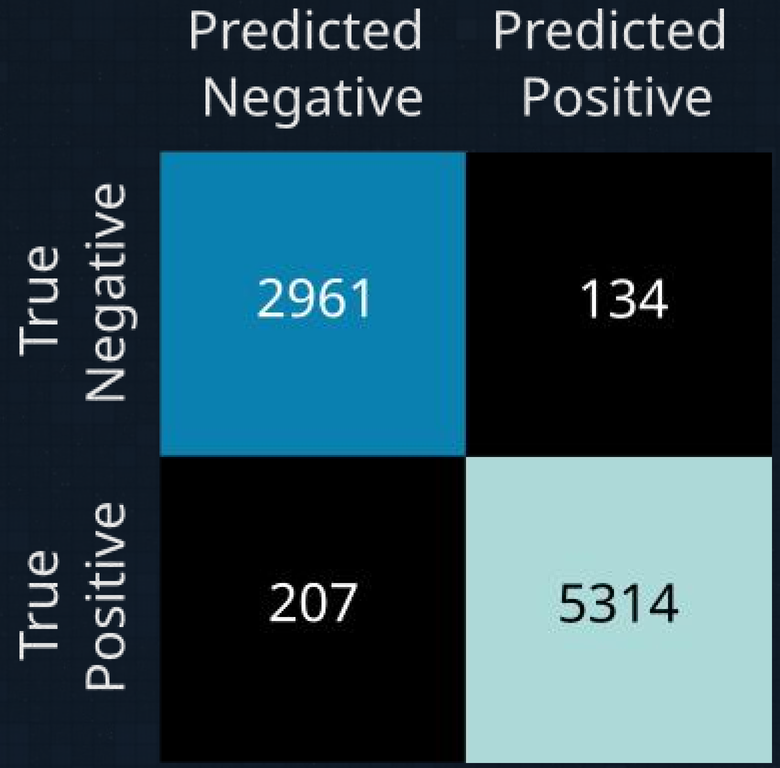
Sample Training Data

English sentences	dns2tcp payloads	Synthesized	cobaltstrike beacons
The tour did get the group the best pre...	724w5jd+cmohljqw90ecgx9km4hs/qrpfgt8dg9...	nd people fall into three groups: those...	www.180.034376ed1.683d1dbe.beacon.pacat..
In order to do this students must under...	7246izqjba	make a reasonably goba1a445d172d5c9bb5...	wpad.localdomain
This, plus the fact that the lower conf...	1xmmzwzkba	1xmcgwkyba. over 20 years experience, h...	post.1243ae0915408fd6c0acdf56c.24cec631...
I think that sounds just right.	dp8uixshba	t Turocy of Doculabs commented, "This i...	post.121cea0c6.31c775120.3d2da7be.beaco...
Have you any idea how much the tyres fr...	zvua/weucfcgcs/ga0xakyv06zdpumk8r8hkkqn...	ad equalised almost imme725643r7cfba4xi...	683d1dbe.beacon.pacattack.xyz

hex16-generated	base64-generated	Reverse lookup	Manually labeled
56D70207472617368657320466175636920616E...	cA0qDXxxKMavcGBJLa5keiAqSjOQApFfN88Rr8Q...	239.155.96.156.in-addr.arpa	szeloba.nask.waw.pl...
26563746F72206F6620746865204E6174696F6E...	bcbvzmzlcib5b3ugysb2zxj5ihbvb3igc...	131.47.96.156.in-addr.arpa	ns3.inwx.de...
3656E617465204865616C74682C204564756361...	wvjagfuaxntlibodw1lcm91cyb0...	197.44.96.156.in-addr.arpa	113.ip-54-37-154.eu
6573706f6e736520746f2074686520636f726f6...	cmUgYXZhaWxhYmxlIHRvI...	165.44.96.156.in-addr.arpa	4tmwwtwje5gxaur5ojzntxjkpem6bhz5._domainkey.co
6e2d506f6f6c2f476574747920496d616765732...	kga2v5libuagugzgf0ysbpcybqdxn0igluigfub...	236.151.96.156.in-addr.arpa	25.ip-51-75-28.eu

Model Performance

67% of stratified samples were used for training and 33% for validation.
The training was finished in 5 epochs.



	Predicted Negative	Predicted Positive
True Negative	2961	134
True Positive	207	5314

Sensitivity	**0.9754**	TPR = TP / (TP + FN)
Precision	0.9625	PPV = TP / (TP + FP)
False Positive Rate	0.0653	FPR = FP / (FP + TN)
False Negative Rate	**0.0246**	FNR = FN / (FN + TP)
Accuracy	0.9604	ACC = (TP + TN) / (P + N)

Re-tuning and Deployment

Samples labeled as suspicious are available on internal network. There is a web interface that allows incident handlers to view the queries, label false positives either on individual basis or regex rules. Samples labeled as false positives will be used to retune the model. The model is retuned every 2 weeks.

We are dealing with 2-3 billion DNS queries on a daily basis. Processing that much information requires a lot of resources. Several evaluation processes are running in parallel, each working on an individual CSV file exported from Clickhouse. A wrapper script is responsible for launching more processes if currently fewer than 12 are running and the CPU usage is less than 60%.

Now what?

Individual queries are not useful for data exfiltration and covert communication channels. Aggregation based on source ip and domain must be done to eliminate false positives.

- **Aggregate on domain names**
- **Aggregate on IP addresses**
- **Look for anomalous DNS behavior of internal hosts**
- **Cross reference with other suspicious activity**
- **Look for DNS traffic surges**
- **Use Open Source Intelligence: shodan.io, abuseipdb.com, whois.com**
- **Understand how DNS payload encoding might be used in a non-malicious way - unsubscribe links, ad networks**
- **Be on a lookout for new DNS tunneling tools, train your models to detect them**

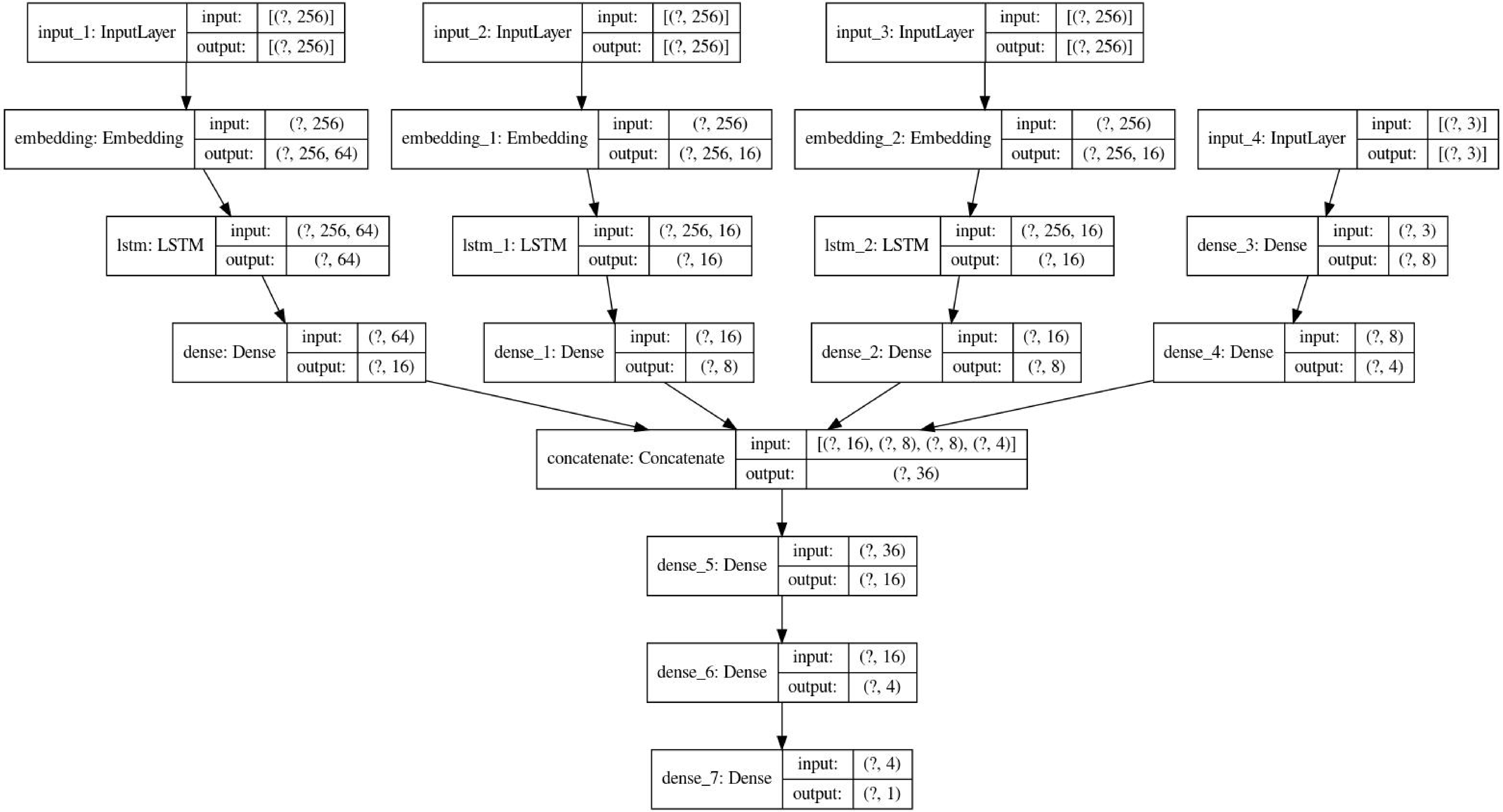
Quetions?

Thank you!

Feel free to email me if you have any quetions: data@utexas.edu

```
# Build a keras model. The model has 4 inputs.
# S - character sequence (case insensitive)
# U - uppercase mask
# A - alpha mask
# M - macro parameters: number of unique characters, length and
information.
from tensorflow import keras
from tensorflow.keras import layers

inputS = keras.Input(shape = (256, ))
inputU = keras.Input(shape = (256, ))
inputA = keras.Input(shape = (256, ))
inputM = keras.Input(shape = (3, ))

# Lowercase character sequence
s = layers.Embedding(input_dim = vocab_size, output_dim=64,
input_length = max_seq_len)(inputS)
s = layers.LSTM(64, activation = 'tanh')(s)
s = layers.Dense(16, activation = 'relu')(s)
s = keras.Model(inputs=inputS, outputs=s)

# Uppercase sequence
u = layers.Embedding(input_dim = 3, output_dim=16, input_length =
max_seq_len)(inputU)
u = layers.LSTM(16, activation = 'tanh')(u)
u = layers.Dense(8, activation = 'relu')(u)
u = keras.Model(inputs=inputU, outputs=u)
```

```
# Alphanumeric sequence
a = layers.Embedding(input_dim = 3, output_dim=16, input_length =
max_seq_len)(inputA)
a = layers.LSTM(16, activation = 'tanh')(a)
a = layers.Dense(8, activation = 'relu')(a)
a = keras.Model(inputs=inputA, outputs=a)

# Macro features
m = layers.Dense(8, activation = 'relu')(inputM)
m = layers.Dense(4, activation = 'relu')(m)
m = keras.Model(inputs=inputM, outputs = m)

combined = keras.layers.concatenate([s.output, u.output, a.output,
m.output])

z = layers.Dense(16, activation="sigmoid")(combined)
z = layers.Dense(4, activation="sigmoid")(z)
z = layers.Dense(1, activation="sigmoid")(z)

model = keras.Model(inputs=[s.input, u.input, a.input, m.input],
outputs=z)

# Binary cross entropy loss for binary classification
model.compile(optimizer = 'adam', loss='binary_crossentropy', metrics =
['accuracy'])
```


input_1: InputLayer
input: [(?, 256)]
output: [(?, 256)]
input_2: InputLayer
input: [(?, 256)]
output: [(?, 256)]
input_3: InputLayer
input: [(?, 256)]
output: [(?, 256)]
embedding: Embedding
input: (?, 256)
output: (?, 256, 64)
embedding_1: Embedding
input: (?, 256)
output: (?, 256, 16)
embedding_2: Embedding
input: (?, 256)
output: (?, 256, 16)
input_4: InputLayer
input: [(?, 3)]
output: [(?, 3)]
lstm: LSTM
input: (?, 256, 64)
output: (?, 64)
lstm_1: LSTM
input: (?, 256, 16)
output: (?, 16)
lstm_2: LSTM
input: (?, 256, 16)
output: (?, 16)
dense_3: Dense
input: (?, 3)
output: (?, 8)
dense: Dense
input: (?, 64)
output: (?, 16)
dense_1: Dense
input: (?, 16)
output: (?, 8)
dense_2: Dense
input: (?, 16)
output: (?, 8)
dense_4: Dense
input: (?, 8)
output: (?, 4)
concatenate: Concatenate
input: [(?, 16), (?, 8), (?, 8), (?, 4)]
output: (?, 36)
dense_5: Dense
input: (?, 36)
output: (?, 16)
dense_6: Dense
input: (?, 16)
output: (?, 4)
dense_7: Dense
input: (?, 4)
output: (?, 1)