ARIEL UNIVERSITY

DNS detection project

Gal Braymok 315695981
Eden Dahary 312274244
Netanel Indik 311300784
David Hodefi 205474026

Introduction

Malicious Domain Name System(DNS) detection is a rapidly evolving field that is concerned with identifying a malicious or non-malicious domain based on their pattern. It has become an increasingly important area of study in recent years. The main objective of such research is to develop algorithms and techniques that can accurately and reliably identify on what domain is malicious or benign. This involves analyzing the query of the domain including the name of the domain, response time, its type and the answer we got. Domains can be broadly classified into four categories: BENIGN, MALWARE, PHISHING, Command and Control (CNC).

Introduction

Detecting malicious DNS can pose several challenges due to the ever-evolving nature of cyber threats.
One major problem is that malicious domains creators are developing new techniques to bypass detection systems. They may use tactics such as DGA, fast-flux networks, or domain cloaking to make their malicious domains appear legitimate or constantly change their domain names to evade detection. Which makes it hard to detect the new malicious domains.

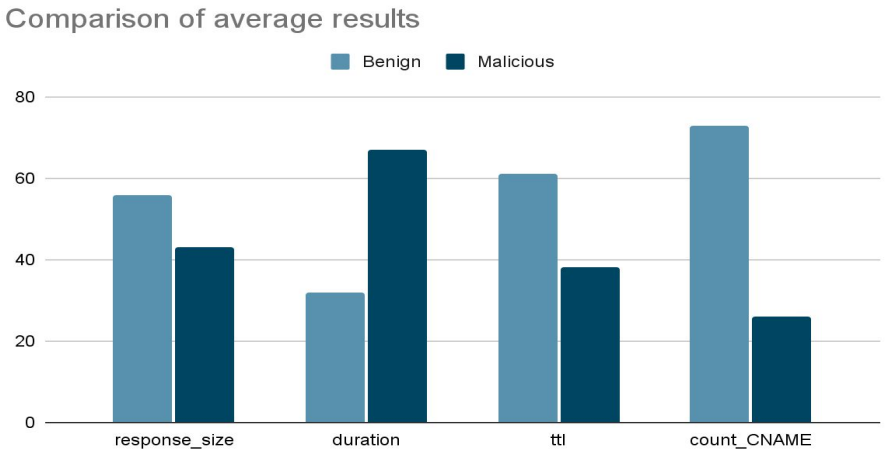
Dataset

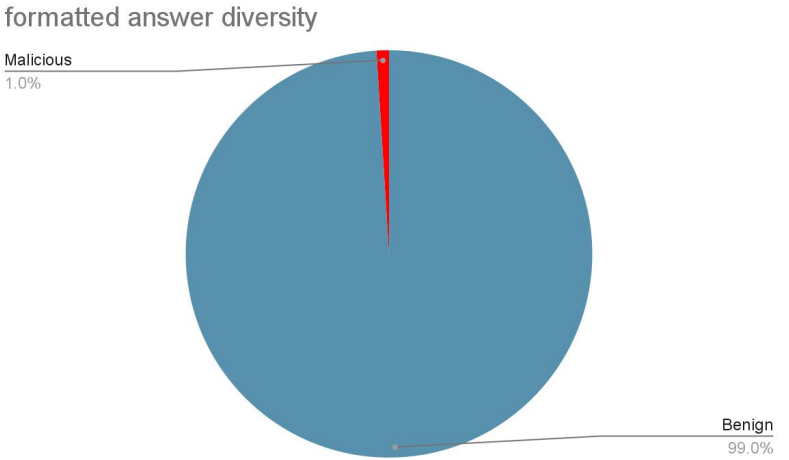
The origin of the dataset used in this research is Akamai company. The data was collected by one of Akamai's products; a DNS protection system. The data was obtained passively by observing the communications between end hosts and their DNS resolvers. The dataset contains info about DNS requests and their responses as well as the verdict of the DNS security system; whether the request is malicious or benign. The system relies on blacklists to determine that.

Dataset

Example of how our data look like:

request_ts	82816516
response_ts	82816518
client_token_dec	52149
qname	www.google.com
qtype	1
answer	[{class: IN,name :www.google.com, ttl: 525, type: 1, array ([98, 100,...)], value_str :www.google.co.il}...]
nx_domain	0
l1_cache_hit	0
response_size	150
is_malicious	0





Current Approach

We offer an hybrid approach that utilizes both finding patterns of malicious domains in the answer part of the response as well as semantic and syntactic analysis of the domain name to intercept domain names generated by DGA. We will use BertForSequenceClassification with the pretrained "bert-base-uncased" model for the task. BertForSequenceClassification is a Bert Model transformer with a sequence classification/regression head on top (a linear layer on top of the pooled output).

Improvements - Accuracy & Performance

The experimental evaluation of our DNS classification models yielded promising results, although our performance did not reach the accuracy of prior research. Our classification model, which utilizes BERT embeddings as input, demonstrated a significant improvement in the field because it is robust and can be implemented in any language.
Moreover our studies in its essence does not count on the number of appearance as we saw in other article. The prior article by Akamai and Ariel Uni count on the times each DNS qname appear because of that if they will get unreasonable amount of malicious their model will need to adapt, were our model will recognize the patterns itself in the formatted answer and we will get the right classification

Results

Benign

Precision = 99.962 percent

Recall = 99.983 percent

F1 Score = 99.972 percent

Malicious

Precision = 77.5 percent

Recall = 88.571 percent

F1 Score = 82.666 percent

overall Accuracy = 99.946 percent

Macro-average Precision = 88.731 percent

Macro-average Recall = 94.277 percent

Macro-average F1 score = 94.2715 percent

True Class / Predicted Class	Malicious	Benign
Malicious	24116	9
Benign	4	31

Code Questions

```
types_dict = {
    1: "A",
    2: "NS",
    5: "CNAME",
    28: "AAAA",
    65: "HTTPS"
}

def type_i_to_s(t):
  if t in [1,2,5,28]:
    return types_dict[t]
  return str(t)

def simplify_answer(ans):
  simple = ""
  if ans.size > 0:
    l = len(ans)
    for i, line in enumerate(ans):
      simple += f"{line['name'][:-1]} {line['ttl']} {type_i_to_s(line['type_'])} {line['value_str']}"
      if i < l-1:
        simple += " "
  return simple
```

```
def get_filter_out_calculate(df):
    # filter out
    dnsq = df.loc[df.qname != 'deathstar.iamaka.net.']
    dnsq = dnsq.loc[dnsq.l1_cache_hit == False]
    dnsq = dnsq[dnsq['answer'].apply(lambda arr: len(arr) > 0)]

    # calculate
    dnsq['duration'] = (dnsq['response_ts'] - dnsq['request_ts'])
    dnsq['cnt_ans_parts'] = dnsq['answer'].apply(lambda arr: len(arr))
    dnsq['mean_ttl'] = dnsq['answer'].apply(lambda arr: np.mean([d.get('ttl') for d in arr]))
    dnsq['cnt_ans_A'] = dnsq['answer'].apply(lambda arr: sum(d.get('type_') == 1 for d in arr))
    dnsq['cnt_ans_NS'] = dnsq['answer'].apply(lambda arr: sum(d.get('type_') == 2 for d in arr))
    dnsq['cnt_ans_CNAME'] = dnsq['answer'].apply(lambda arr: sum(d.get('type_') == 5 for d in arr))
    dnsq['cnt_ans_AAAA'] = dnsq['answer'].apply(lambda arr: sum(d.get('type_') == 28 for d in arr))
    dnsq['formatted_ans'] = dnsq["answer"].apply(simplify_answer)
    dnsq.drop_duplicates(subset=['formatted_ans'], inplace=True)

    return dnsq
```

```
labels = [int(label) for label in df_all["is_malicious"].to_list()]  # List of labels
texts, labels = shuffle_lists(texts, labels)

# Tokenize the texts and convert them to input tensors
input_ids = []
attention_masks = []

for text in texts:
    encoded_text = tokenizer.encode_plus(
        text,
        add_special_tokens=True,
        max_length=max_length,
        padding='max_length',
        truncation=True,
        return_tensors='pt'
    )
    input_ids.append(encoded_text['input_ids'])
    attention_masks.append(encoded_text['attention_mask'])

input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)



# Define the split ratio
train_ratio = 0.8
train_size = int(train_ratio * len(texts))

# Split the dataset
train_inputs, val_inputs = input_ids[:train_size], input_ids[train_size:]
train_masks, val_masks = attention_masks[:train_size], attention_masks[train_size:]
train_labels, val_labels = labels[:train_size], labels[train_size:]

# Create PyTorch DataLoader for training and validation sets
train_dataset = TensorDataset(train_inputs, train_masks, train_labels)
val_dataset = TensorDataset(val_inputs, val_masks, val_labels)

batch_size = 32 #changed from 16 -> 8
train_dataloader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)


# Set the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device = torch.device('cpu')

# Move the model to the device
model.to(device)

# Define the optimizer and learning rate
optimizer = AdamW(model.parameters(), lr=2e-5)

# Training loop
num_epochs = 3

for epoch in range(num_epochs):
    model.train()
    total_loss = 0

    for batch in train_dataloader:
        input_ids = batch[0].to(device)
        attention_masks = batch[1].to(device)
        labels = batch[2].to(device)

        optimizer.zero_grad()

        outputs = model(input_ids, attention_mask=attention_masks, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()

        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()
```

ARIEL UNIVERSITY

Discussion – Algorithms

While the BERT classification model demonstrates proficiency in identifying benign DNS queries, further efforts should be directed towards enhancing its capability to detect and classify malicious DNS queries accurately.
This could involve fine-tuning the model, augmenting the training data, or exploring other techniques to address the class imbalance issue. By continuously refining the model's performance, we can enhance its effectiveness in real-world applications where the detection of malicious DNS queries is of utmost importance.

Discussion – Open Questions & Future Work

For future work, we recommend the following directions:

• Fine-tuning the model to further improve its performance in detecting and classifying malicious DNS queries.

• Augmenting the training data to increase its size and diversity, including data from multiple sources, organizations, and time periods.

• Exploring advanced techniques in NLP and machine learning, such as transformer-based models like GPT or XLNet.

• Integrating unsupervised learning method

• Integrating NLP models with ML models that leverage statistical analysis of DNS queries to enhance the detection and classification capabilities.

open questions we yet understand the the specific format of each DNS attacks, if we have enough DNS queries of each attacks and we figure out the difference between them we may need to rethink our model and classified in other ways

ARIEL UNIVERSITY

Special thanks to Pr. Amit Dvir and Dr. Enidjar Or Chaim for their valuable input and assistance throughout the project.

We thanks Akamai Technologies, for trusting us with their valuable data, and let us explore and study the world of DNS, we appreciate it.