PROJECT PLAN

Introduction

Security risks are evolving at a rapid pace, prompting security professionals to employ a variety of Natural Language Processing (NLP) approaches to prevent, detect, assess, and potentially mitigate different attacks related to cybersecurity (Devlin et al., 2019). Text memorization, data extraction [], and NER (Named Entity Recognition) are examples of these. To gain a better understanding of the means and consequences of various cyber-attacks, security experts rely on past information, including security alerts or internet studies. However, such reports are frequently unstructured, providing effective data extraction even more difficult.

Project plan

This project is about executing and building python tool that could automate the cybersecurity reports that is related to security information. This project presents a significant problem for security specialists because a tremendous quantity of cyber data is produced on a regular basis, necessitating the use of automated information collection technologies to enable data querying as well as retrieval. As a result, we introduce Open-CyKG, an Open Cyber Threat Intelligence (CTI) Knowledge Graph (KG) architecture built on a recognition-based neural Open Information Extraction (OIE) approach to extracting useful information about cyber threats from unstructured Advanced Persistent Threat (APT) data (Yang et al., 2020). To be more precise, we begin by identifying pertinent entities by creating a NER that assists in identifying association triples created by the OIE model. Following that, the retrieved structured information is canonicalized in order to construct the KG utilizing fusion approaches based on phrase embeddings. As a result, security experts can run queries against the Open-CyKG system to get important data.

Experiment findings show that our recommended elements for Open-CyKG exceed state-of-the-art models.

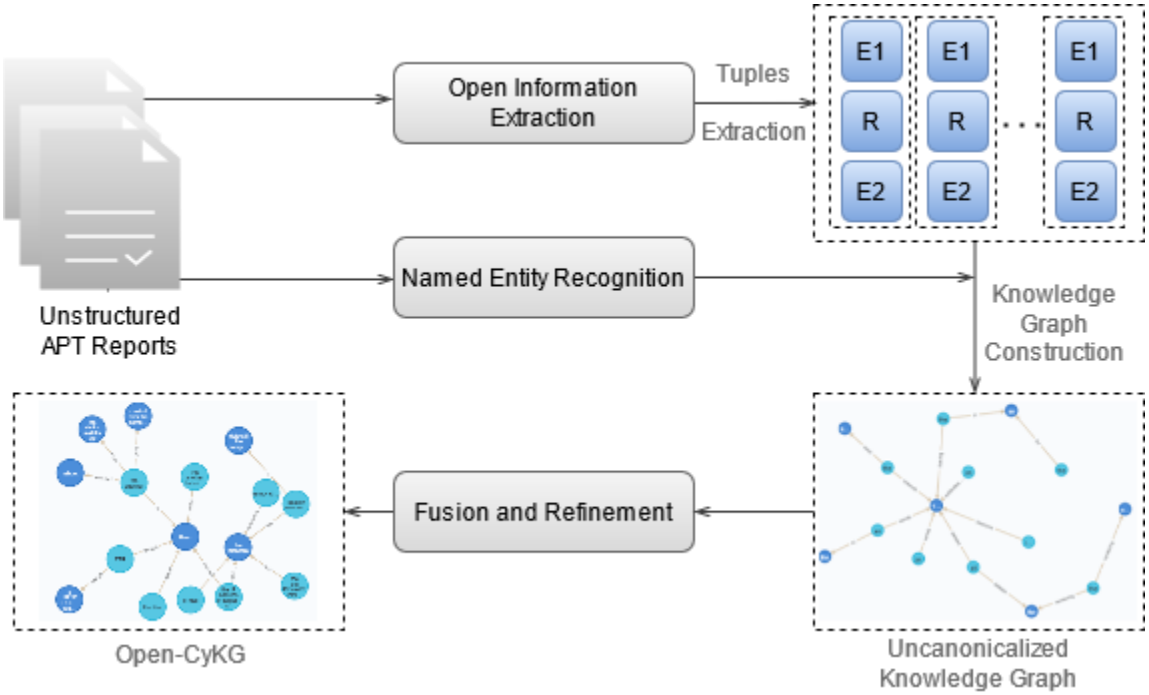


Figure 1: Project Model

Scope:

- We provide the first OIE-based KG within the cybersecurity sector that doesn't limit extraction tasks to a predefined variety of data. Open-CyKG is an efficient open cyber threat assessment KG model that combines OIE plus NER with KG fusion approaches.
- We provide a recognition-based sequential-to-sequence OIE framework that outperforms current network topologies, illustrating its utility in data extraction tasks.
- We create a cybersecurity NER system to categorize important terms in this domain which outperforms many starting points including state-of-the-art models.

- In the context of Open-CyKG, we perform an improvement and fusion procedure that uses the resulting NER labels with contextualized embedded words to improve the accuracy of the recovered queries.
- By employing two examples of queries, we demonstrate that after Open-CyKG is built, retrieval of data is able to conduct effectively.

Project Goals

We present Open-CyKG, a CTI and KG developed with OIE triples, in this paper. Open-CyKG is an application that can extract pertinent information from APT documents quickly and encode the recovered data in a KG that enables for quick risk-related data exploration and retrieval. As depicted in Fig. 1. Open-CyKG is made up of two key parts. 1. First, we offer a domain-independent relational triple extraction OIE framework that utilizes consideration from data that is unstructured. Furthermore, a NER model for self-labeling cybersecurity phrases. To be a bit more precise, we start by employing OIE to retrieve structural association tuples from APT reports that are subsequently supplied within the KG employing the NER job.

Problem required to be solve

Data redundancy and ambiguity are important issues encountered during the KG construction process. As a result, we use refinements and canonicalization approaches to combine data within the KG based on their contextualized embeddings of words, followed by hierarchy agglomeration clustering for entity classification. Several empirical investigations were conducted to verify the elements of Open-CyKG, proving that OIE may greatly aid in the construction of knowledge foundations. To solve the aforementioned difficulties, we propose an

innovative and open cybersecurity KG model. We will be analyzing the OIE dataset: Malware DB and the NER dataset in this research.

To meet the above scope, we have used the Python program to conduct codes. First, we import all required libraries shown below:

In [ ]:

```
import tensorflow as tf

print(tf.__version__)
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# running the Stanford POS Tagger from NLTK
import nltk
from nltk import word_tokenize
from nltk import StanfordTagger
import nltk
import pandas as pd
nltk.download("popular")

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

from google.colab import drive
drive.mount('/content/gdrive')

!pip install tensorflow==2.2.0
import tensorflow as tf

print(tf.__version__)
!pip install keras==2.2.4
import keras
keras.__version__
#from tensorflow import keras as k
#import tensorflow as tf
!pip install git+https://www.github.com/keras-team/keras-contrib.git
from keras_contrib.layers import CRF

from subprocess import check_output
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from math import nan
from keras.callbacks import ModelCheckpoint

!pip install git+https://www.github.com/keras-team/keras-contrib.git
from keras_contrib.layers import CRF
from subprocess import check_output
from keras.models import Model, Input
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional, GRU
import keras as k


import pandas as pd
import numpy as np

from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import LSTM, Dense, TimeDistributed, Embedding, Bidirectional
from keras.models import Model, Input
from keras_contrib.layers import CRF
from keras.callbacks import ModelCheckpoint
```

```
import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
!pip install sklearn-crfsuite
!pip install seqeval

from sklearn_crfsuite.metrics import flat_classification_report
from sklearn.metrics import f1_score
from seqeval.metrics import precision_score, recall_score, f1_score, classification_report
from keras.preprocessing.text import text_to_word_sequence
import pickle
!pip install tiny-tokenizer flair
!pip install flair
```

Dframe the datasets:

In [ ]:

```
#dframe = dataset
from sklearn.model_selection import StratifiedKFold, KFold

cvscores = []
cvR=[]
cvF=[]
cvP=[]
i=0
kfold = StratifiedKFold(n_splits=5, shuffle=True)
words = list(dframe['words'].unique())
tags = list(dframe['tag'].unique())
target = dframe.loc[:,'tag']


# This is a class te get sentence. The each sentence will be list of tuples with its tag and pos
class sentence(object):
    def __init__(self, df):
        self.n_sent = 1
        self.df = df
        self.empty = False
        agg = lambda s : [(w, p, t) for w, p, t in zip(s['words'].values.tolist(),
                                                       s['POS'].values.tolist(),
                                                       s['tag'].values.tolist())]
        self.grouped = self.df.groupby("sentence_idx").apply(agg)
        self.sentences = [s for s in self.grouped]

    def get_text(self):
        try:
            s = self.grouped[self.n_sent]
            self.n_sent +=1
            return s
        except:
            return None
getter = sentence(dframe)
sentences = [" ".join([str(s[0]) for s in sent]) for sent in getter.sentences]

sent = getter.get_text()

sentences = getter.sentences
```

LSTM network defining

```
# Number of data points passed in each iteration
batch_size = 50
# Passes through entire dataset
epochs = 30
# Maximum length of review
max_len = 75
# Dimension of embedding vector
embedding = 40
words = list(dframe['words'].unique())
tags = list(dframe['tag'].unique())
word_to_index = {w : i + 2 for i, w in enumerate(words)}
word_to_index["UNK"] = 1
word_to_index["PAD"] = 0

# Dictionary lable:index pair label is key and value is index.
tag_to_index = {t : i + 1 for i, t in enumerate(tags)}
tag_to_index["PAD"] = 0
idx2word = {i: w for w, i in word_to_index.items()}
idx2tag = {i: w for w, i in tag_to_index.items()}
# Converting each sentence into list of index from list of tokens
X = [[word_to_index[w[0]] for w in s] for s in sentences]
# Padding each sequence to have same length  of each word
X = pad_sequences(maxlen = max_len, sequences = X, padding = "post", value = word_to_index["PAD"])
y = [[tag_to_index[w[2]] for w in s] for s in sentences]
y = pad_sequences(maxlen = max_len, sequences = y, padding = "post", value = tag_to_index["PAD"])
num_tag = dframe['tag'].nunique()
y = [to_categorical(i, num_classes = num_tag + 1) for i in y]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

LSTM-CRF Network

In [ ]:
```
import keras
keras.__version__
```

Out[ ]:
```
'2.2.4'
```

In [ ]:
```
num_tags = dframe['tag'].nunique()
# Model architecture
from keras.layers.core import Dense,Dropout,Activation,Flatten,Lambda
input = Input(shape = (max_len,))
model = Embedding(input_dim = len(words) + 2, output_dim = embedding, input_length = max_len, mask_zero = True)(input)
model = Bidirectional(GRU(units = 50, return_sequences=True, recurrent_dropout=0.1))(model)
model = TimeDistributed(Dense(50, activation="relu"))(model)
#crf = Lambda(lambda x:CRF(num_tags+1))  # CRF layer
crf=CRF(num_tags+1)
out = crf(model)

model = Model(input, out)
model.compile(optimizer="rmsprop", loss=crf.loss_function, metrics=[crf.accuracy])
model.summary()

checkpointer = ModelCheckpoint(filepath = 'model.h5',
                       verbose = 0,
                       mode = 'auto',
                       save_best_only = True,
                       monitor='val_loss')
history = model.fit(X_train, np.array(y_train), batch_size=batch_size, epochs=epochs,
                    validation_split=0.1, callbacks=[checkpointer])


history.history.keys()
```

Expected Outcomes

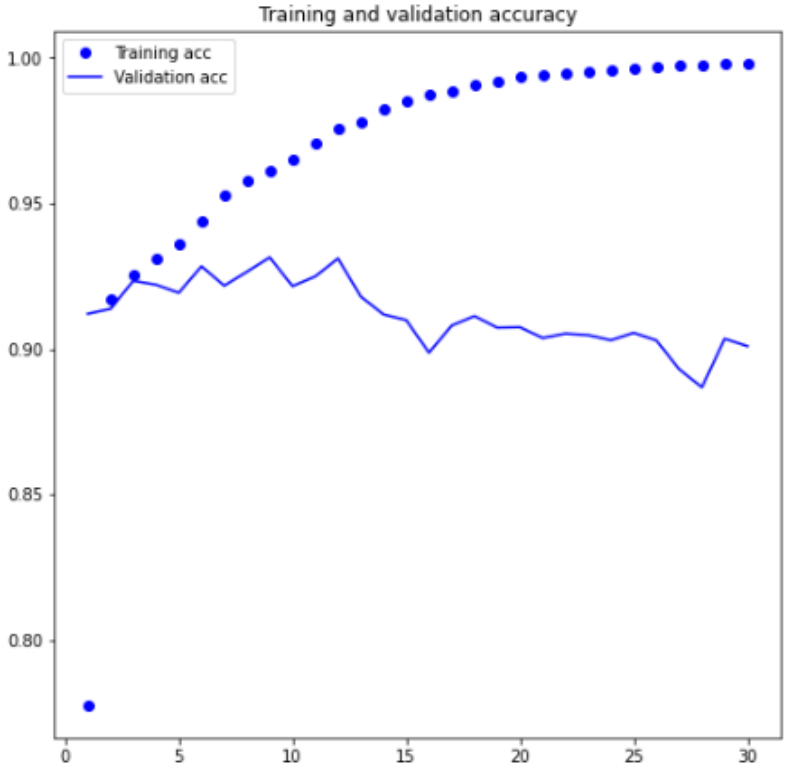
Table 1: Timeline

NO.	Phase	Task Name	Start (Date)	End (Date)	Duration (Days)
1	Project Planning				
		1.1 Brainstorming	4/17/2023	4/18//2023	1
		1.2 Model Discussion	4/17/2023	4/18//2023	1
		1.3 Plan Documentation	4/17/2023	4/18//2023	1
2	Modelling	2.1 Neural OIE model	4/17/2023	4/18//2023	1
		2.2 Cybersecurity-NER	4/18/2023	4/19/2023	1
		2.3 KG construction	4/18/2023	4/19/2023	1
		2.4 KG Canonicalized	4/18/2023	4/19/2023	1
3	Results and evaluation	3.1 KG validation	4/18/2023	4/19/2023	1
	Documentation	5.1. Document all the analysis	4/18/2023	4/20/2023	1

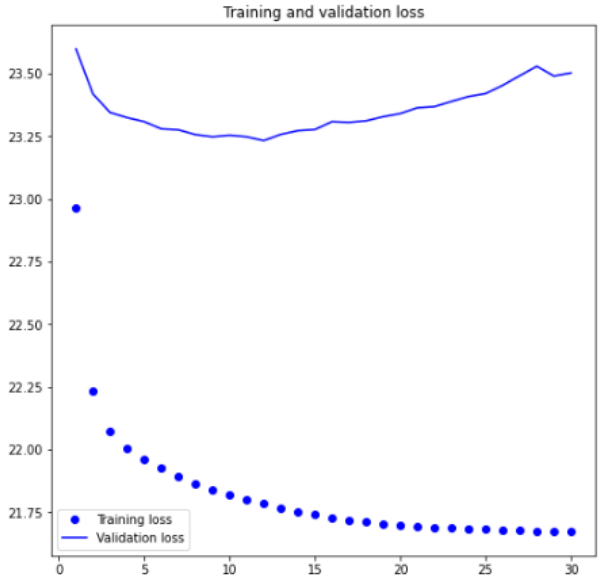
The excepted outcomes will be presented on April 20, 2023.

In [ ]:
```
acc = history.history['crf_viterbi_accuracy']
val_acc = history.history['val_crf_viterbi_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.figure(figsize = (8, 8))
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f2744162390>
```



```
In [ ]:
plt.figure(figsize = (8, 8))
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```



The predicted outcomes give an innovative structure for generating a knowledge network from sophisticated persistent danger alerts that incorporates data from multiple parts as well as fusion approaches employing contextualized word embedding. The suggested framework will consist of two main components: a recognition-based OIE plus a cybersecurity NER system. By testing every part independently, the expected results would confirm the excellence of the created KG. On two separate datasets, we compared our cybersecurity NER system toward numerous baselines and cutting-edge models. Our approach worked to provide the greatest results.

References

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North, 1. https://doi.org/10.18653/v1/n19-1423

Yang, S., Yu, X., & Zhou, Y. (2020, June 1). LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example. IEEE Xplore. https://doi.org/10.1109/IWECAI50956.2020.00027