Weekly reports for IARCS Internship

Wong Chu Feng

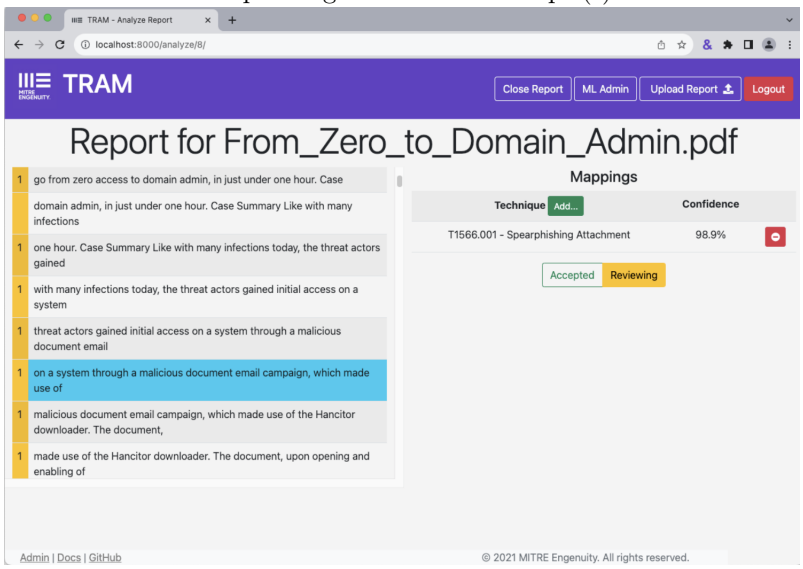
Jan - May 2024

Contents

Week 1: Research on STRIDE and Mitre ATT&CK

Pointers:

- Read up on STRIDE framework, MITRE ATT&CK and OpenAI's API.
- Briefing on internship, phase 1 and 2.
  1. Attack technique classifier
     - Try and evaluate existing ML/NLP techniques to classify ATT&CK techniques into STRIDE categories
     - Evaluate using LLM to classify ATT&CK techniques into STRIDE categories
     - Customize LLM to classify ATT&CK technique using other threat model categories (e.g., RAPIDS)
  2. Architecture diagram analyzer
     - Identify trust zones in architecture diagram based on subnet information
     - Identify unrealistic hand-drawn architecture diagrams
- Studied code from current stride (supervised) model classifier.
- Understood the supervised model's code. Looking at the *raw_capec_data.xlsx* to try clustering methods to create an unsupervised model. The trained supervised model might be useful in feature extraction to train the unsupervised model. What features exactly? Or in transfer learning? Still not too sure.
- Digressing a bit, started researching for similar papers on mapping ATT&CK to STRIDE.
- Discovered TRAM LLM → *Possible to use with the questionaire from GovTech as an input.*
  - identifies keywords, capable of contextual understanding
  - predicts the presence of TTPs (Tactics, Techniques, Procedures) in the text
  - then finds the corresponding ATT&CK technique(s).



- Identification of Attack Paths Using Kill Chain and Attack Graphs → *Needs a separate function to create this graph. Seemingly not helpful.*

TABLE III
ATTACK GRAPH VERTICES

ID	Description
1	External actor.
2	Email address of an employee was published on a website.
3	T1594 – Search Victim-Owned Websites.
4	The attacker knows that the email address exists.
5	Sender reputation analysis was not accomplished.
6	T1566.001 – Spearphishing Attachment.
7	Authentication of sending an email was violated.
8	The employee can click on the attachment.
9	The employee has a user account on a personal computer.
10	Training of users was not accomplished (countermeasure).
11	T1204.002 – User execution of malicious file.
12	Authentication of opening file action was violated.
13	Microsoft Office opens files.
14	Microsoft Office is installed on the personal computer.
15	Microsoft Office 2016 contains CVE-2017-0262 vulnerability.
16	T1203 – Exploitation for Client Execution.
17	The attacker violated the system's authorization (user rights).
18	Microsoft Windows 8.1 is installed on the personal computer.
19	Microsoft Windows 8.1 contains CVE-2017-0263 vulnerability.
20	Software is not regularly updated.
21	T1068 – Exploitation for Privilege Escalation.
22	The attacker violated the system's authorization (admin rights).
23	T1485 – Data Destruction.
24	Data backup was not accomplished (countermeasure).
25	T1486 – Data Encrypted for Impact.
26	Integrity of a sensitive file was violated.

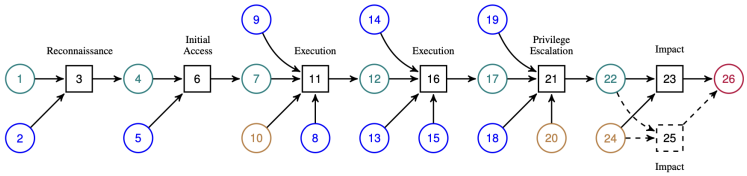


Fig. 3. Example kill chain attack graph for the validation use case. Vertices correspond to attacker's level of control over assets (green), asset properties (blue), countermeasures (brown), attack techniques (black), and an attack goal (dark red). Vertices are described in Table III.

- Thinking of either a standalone unsupervised model or a hybrid one.
- Summary: Make use of the current supervised model that maps Mitre Att&cks using keywords to STRIDE. Then we can spatially obtain the clusters of data each corresponding to a category in STRIDE. We can use this as the baseline to train the unsupervised model. Each similarity of *category from GovTech* will be weighted against STRIDE, thereby allowing the mapping of STRIDE to *the GovTech framework*.
- Consolidate ideas:
  - Use unsupervised model.
  - Use hybrid model. Use the current supervised model to extract features (if any) and pass it to the unsupervised model to train for the *GovTech* framework.
  - Find the similarities between STRIDE and the *GovTech framework*, give each category some keywords and use DL to find its weights. Then use the weights to train the unsupervised model.
- Attempt to change the current supervised model to an unsupervised one.

Week 2: Building the model

Attempted at using *Tokenizer* for feature extraction. Yields 0.469 accuracy. Should be due to the corpus being too small for the model to properly learn the context of the words.

```
tokenizer = Tokenizer(oov_token='<OOV>')
tokenizer.fit_on_texts(df_train['NameDesc'])
X_train = tokenizer.texts_to_sequences(df_train['NameDesc'])
X_test = tokenizer.texts_to_sequences(df_test['NameDesc'])
X_val = tokenizer.texts_to_sequences(df_dev['NameDesc'])

x = [X_train, X_test, X_val]
max_length = 0
for _ in x:
    max_l = max([len(seq) for seq in _])
    max_length = max(max_length, max_l)

X_train_padded = pad_sequences(X_train, maxlen=max_length, padding='post')
X_test_padded = pad_sequences(X_test, maxlen=max_length, padding='post')
X_val_padded = pad_sequences(X_val, maxlen=max_length, padding='post')

y_train = df_train['STRIDE'].values
y_test = df_test['STRIDE'].values
y_val = df_dev['STRIDE'].values
```

Second attempt using *TfidfVectorizer* for feature extraction. This model performs better because TfidfVectorizer works better with a small corpus as it takes into account the frequency of the words appearing which might indicate keywords of a STRIDE category.

```
tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(df_train['NameDesc']).toarray()
X_test_tfidf = tfidf_vectorizer.transform(df_test['NameDesc']).toarray()
X_val_tfidf = tfidf_vectorizer.transform(df_dev['NameDesc']).toarray()

y_train = df_train['STRIDE'].values
y_test = df_test['STRIDE'].values
y_val = df_dev['STRIDE'].values
```

Then I perform hyperparameter tuning on the model.:

```
dropout_rates = [0.2, 0.3, 0.4, 0.5]
activations_list = ['relu', 'leaky_relu', 'elu', 'tanh']
num_neurons = [32, 64, 128, 256]
opt_lr = [1e-2, 1e-3, 1e-4]
L2_lr = [1e-2, 1e-3, 1e-4]
best_params = None
best_val_acc = 0

hyperparam_combi = itertools.product(dropout_rates, num_neurons, activations_list,
    opt_lr, L2_lr)

for dr, nn, al, olr, l2lr in hyperparam_combi:
    modelTest = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(vocab_size,)),
    tf.keras.layers.Dense(nn*2, activation=al),
    tf.keras.layers.Dropout(dr),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(nn, activation=al),
    tf.keras.layers.Dropout(dr),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(nn//2, activation=al),
    tf.keras.layers.Dense(num_classes, kernel_regularizer=tf.keras.regularizers.L2(l2=1e
    -2), activation='softmax')
    ])

    optimizer = tf.keras.optimizers.legacy.Adam(olr)
    modelTest.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy",
    metrics=['accuracy'])
```

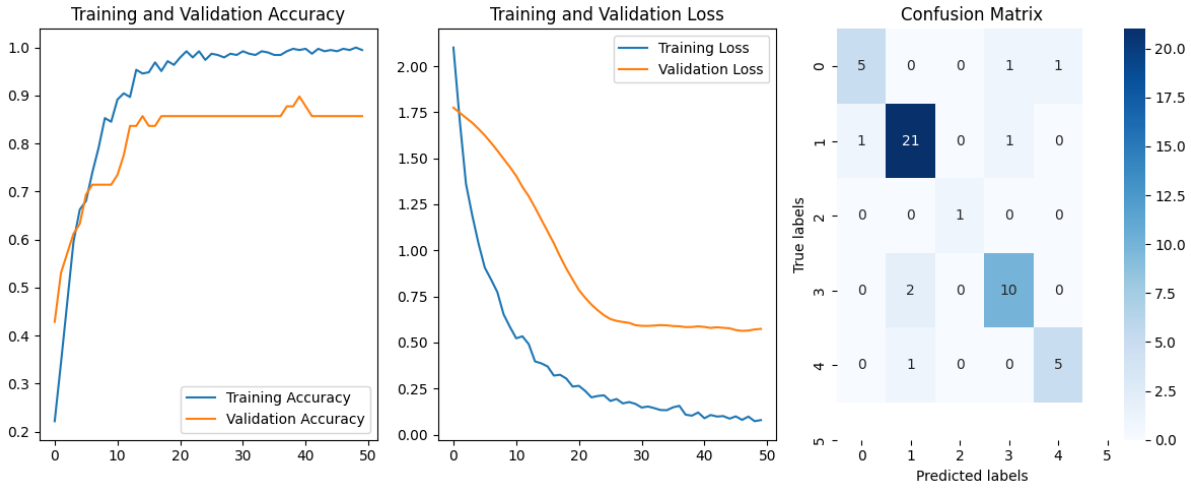
```
26
27      early_stop = EarlyStopping(
28          monitor="val_loss",
29          patience=5,
30          verbose=0,
31          restore_best_weights=True
32      )
33      histTest = modelTest.fit(
34          X_train_tfidf, y_train,
35          batch_size=16,
36          epochs=num_epochs,
37          validation_data=(X_val_tfidf, y_val),
38          verbose=0,
39          callbacks=[early_stop,]
40      )
41
42      val_acc = max(histTest.history['val_accuracy'])
43      # print(f"Dropout: {dr}, Activation: {al}, Hidden Units: {nn}, L2 Reg: {l2lr}, LR: {
        olr}, Best Val Acc: {val_acc}\n==============================")
44      if val_acc > best_val_acc:
45          best_val_acc = val_acc
46          best_params = (dr, nn, al, olr, l2lr)
```

Final Best Hyperparameters:
Dropout: *0.2*,
Activation: *elu*,
Hidden Units: *32*,
L2 Reg: *0.001*,
LR: *0.0001*,
Best Val Acc: 0.*918367326259613*

Final model used:

```
1  hidden_units = 32
2  num_classes = 6
3  batch_size = 16
4  num_epochs = 50
5  classes = [0,1,2,3,4,5]
6  vocab_size = X_train_tfidf.shape[1]
7  optimizer = tf.keras.optimizers.legacy.Adam(1e-4)
8
9  model3 = tf.keras.Sequential([
10     tf.keras.layers.Input(shape=(vocab_size,)),
11     tf.keras.layers.Dense(hidden_units*2, activation='elu'),
12     tf.keras.layers.Dropout(.2),
13     tf.keras.layers.BatchNormalization(),
14     tf.keras.layers.Dense(hidden_units, activation='elu'),
15     tf.keras.layers.Dropout(.2),
16     tf.keras.layers.BatchNormalization(),
17     tf.keras.layers.Dense(hidden_units//2, activation='elu'),
18     tf.keras.layers.Dense(num_classes, kernel_regularizer=tf.keras.regularizers.L2(l2=1e
       -3), activation='softmax')
19 ])
20
21 model3.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=['
       accuracy'])
22 model3.summary()
23 # plot_model(model3, show_shapes=True, show_layer_names=True)
```

Results: Plotted a confusion matrix to evaluate the model. It yields an accuracy of 0.857 where the diagonal entries are the true labels predicted.



Summary:

- Try fuzzy clustering on the dataset since GMM and DBSCAN proves to be not too helpful.
- Attempt pulling out more specific keywords and predict the STRIDE category
- Examine using one model for each of STRIDE category

Week 3: Finding better models

Note: I discovered an error in the code for the previous model shown in week 2 where the promising results were ultimately spurious.

This is a more aggressive text preprocessing on the corpus. I first conduct a preliminary analysis of the word frequencies and trivially remove unimportant words.

```
def basic_processing(df):
    words_to_remove = ["e.g.", "code", "may", "attack", "system", "adversary", "
    Adversaries"]
    for word in words_to_remove:
        df['NameDesc'] = df['NameDesc'].apply(lambda x: x.replace(word, ''))
    for word in words_to_remove:
        df['NameDesc'] = df['NameDesc'].apply(lambda x: re.sub(r'\b' + re.escape(word) +
     r'\b', '', x))

    # df['NameDesc'] = df['NameDesc'].str.replace(r"\b(" + "|".join(words_to_remove) + r
    ")\b", "", regex=True)
    df['NameDesc'] = df['NameDesc'].str.replace("<br><br>", "", regex=True)
    df['NameDesc'] = df['NameDesc'].str.replace("\(Citation:.*?\)", "", regex=True)
    df['NameDesc'] = df['NameDesc'].str.replace("http\S+", "", regex=True)
    df['NameDesc'] = df['NameDesc'].str.replace("--+", "-", regex=True)
    df['NameDesc'] = df['NameDesc'].str.replace("[^A-Za-z]", "-", regex=True)
    return df

def rm_stopwords(df):
    stop_words = set(stopwords.words('english'))
    df['NameDesc'] = df['NameDesc'].apply(lambda x: [word for word in x if word not in
    stop_words])
    print(f"Removed stopwords:\n{df.head(3).NameDesc}\n")
    return df

def lemmatize(df):
    lemmatizer = WordNetLemmatizer()
    def lemmatize_tokens(tokens):
        def get_wordnet_pos(word):
            tag = nltk.pos_tag([word])[0][1][0].upper()
            tag_dict = {"J": wordnet.ADJ,
                        "N": wordnet.NOUN,
                        "V": wordnet.VERB,
                        "R": wordnet.ADV}
            return tag_dict.get(tag, wordnet.NOUN)
        lemmas = [lemmatizer.lemmatize(token, get_wordnet_pos(token)) for token in
    tokens]
        return lemmas
    df['NameDesc'] = df['NameDesc'].apply(lambda x: lemmatize_tokens(x))
    print(f"Lemmatized words:\n{df.head(3).NameDesc}")
    return df

def text_preprocessing(df):
    basic_processing(df)
    df['NameDesc'] = df['NameDesc'].apply(lambda x: word_tokenize(x))
    rm_stopwords(df)
    lemmatize(df)
    print("==========")
    return df
```

Table 1: Extracting keywords

Dataset	Original text	Stemmed & Lemmatized text
df_train[0]	Exfiltration to Cloud Storage Adversaries may exfiltrate data to a cloud storage service rather than over their primary command and control channel. Cloud storage services allow for the storage, edit, and retrieval of data from a remote cloud storage server over the Internet. Examples of cloud storage services include Dropbox and Google Docs. Exfiltration to these cloud storage services can provide a significant amount of cover to the adversary if hosts within the network are already communicating with the service. Analyze network data for uncommon data flows (e.g., a client sending significantly more data than it receives from a server) to known cloud storage services. Processes utilizing the network that do not normally have network communication or have never been seen before are suspicious. User behavior monitoring may help to detect abnormal patterns of activity.	[Exfiltration, Cloud, Storage, exfiltrate, data, cloud, storage, service, rather, primary, command, control, channel, Cloud, storage, service, allow, storage, edit, retrieval, data, remote, cloud, storage, server, Internet, Examples, cloud, storage, service, include, Dropbox, Google, Docs, Exfiltration, cloud, storage, service, provide, significant, amount, cover, host, within, network, already, communicate, service, Analyze, network, data, uncommon, data, flow, client, send, significantly, data, receives, server, know, cloud, storage, service, Processes, utilize, network, normally, network, communication, never, see, suspicious, User, behavior, monitoring, help, detect, abnormal, pattern, activity]
df_train[2]	Runtime Data Manipulation Adversaries may modify systems in order to manipulate the data as it is accessed and displayed to an end user, thus threatening the integrity of the data.(Citation: FireEye APT38 Oct 2018)(Citation: DOJ Lazarus Sony 2018) By manipulating runtime data, adversaries may attempt to affect a business process, organizational understanding, and decision making.¡br¿¡br¿Adversaries may alter application binaries used to display data in order to cause runtime manipulations. Adversaries may also conduct [Change Default File Association](https://attack.mitre.org/techniques/T1546/001) and [Masquerading](https://attack.mitre.org/techniques/T1036) to cause a similar effect. The type of modification and the impact it will have depends on the target application and process as well as the goals and objectives of the adversary. For complex systems, an adversary would likely need special expertise and possibly access to specialized software related to the system that would typically be gained through a prolonged information gathering campaign in order to have the desired impact. Inspect important application binary file hashes, locations, and modifications for suspicious/unexpected values.	[Runtime, Data, Manipulation, modify, order, manipulate, data, access, displayed, end, user, thus, threaten, integrity, data, By, manipulate, runtime, data, adversary, attempt, affect, business, process, organizational, understand, decision, make, alter, application, binary, use, display, data, order, cause, runtime, manipulation, also, conduct, Change, Default, File, Association, Masquerading, cause, similar, effect, The, type, modification, impact, depends, target, application, process, well, goal, objective, For, complex, would, likely, need, special, expertise, possibly, access, specialized, software, related, would, typically, gain, prolong, information, gathering, campaign, order, desire, impact, Inspect, important, application, binary, file, hash, location, modification, suspicious, unexpected, value]

Visualising the word frequencies in the corpus:
Using TfidfVectorizer and after hyperparameter tuning, the model to use is as follows.

```
hidden_units = 32
batch_size = 16
num_epochs = 50
num_classes = 6
classes = [0,1,2,3,4,5]
vocab_size = X_train_tfidf.shape[1]
optimizer = tf.keras.optimizers.legacy.Adam(1e-4)

model4 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(vocab_size,)),
    tf.keras.layers.Dense(hidden_units*2, activation='elu'),
    tf.keras.layers.Dropout(.5),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(hidden_units, activation='elu'),
    tf.keras.layers.Dropout(.5),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(hidden_units//2, activation='elu'),
    tf.keras.layers.Dense(num_classes, kernel_regularizer=tf.keras.regularizers.L2(l2=1e-2), activation='softmax')
])
r
model4.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=['accuracy'])
model4.summary()
# plot_model(model4, show_shapes=True, show_layer_names=True)
```

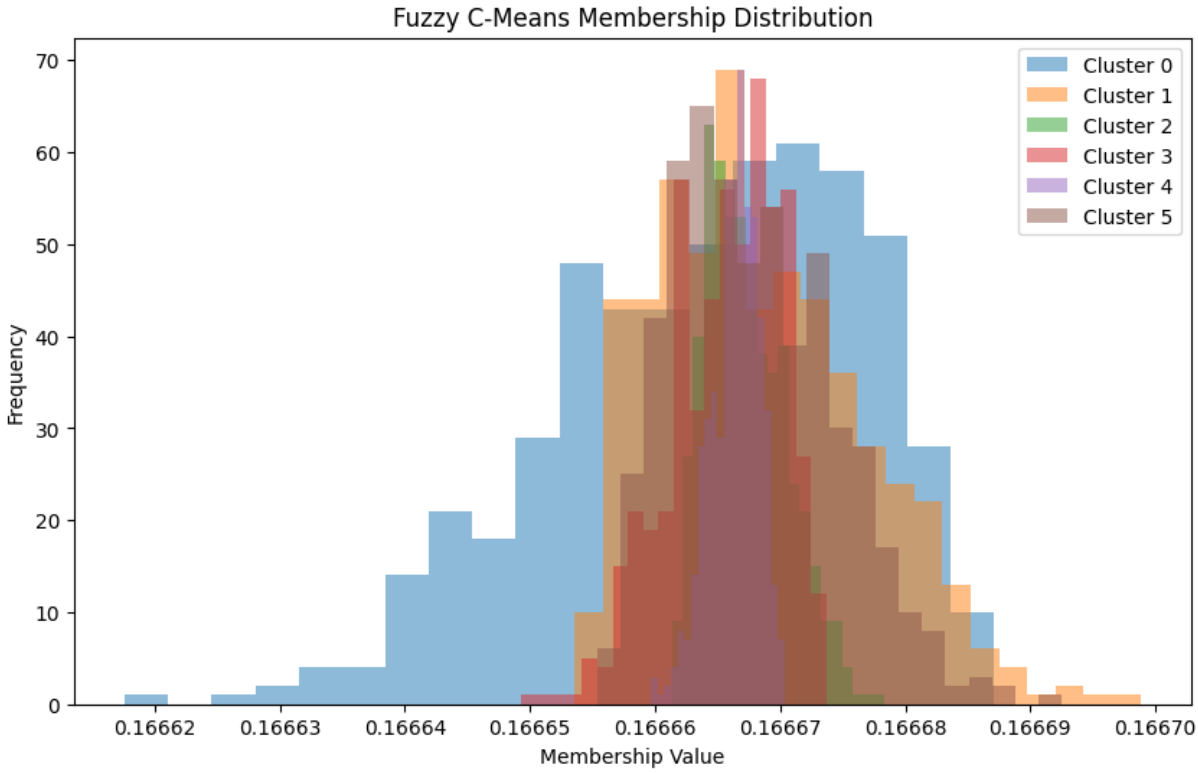
This model predicts with accuracy of 0.79.

Attempt on Fuzzy Clustering

Table 2: Extracting keywords

Cluster	Unique STRIDE values
0	['000001' '010000' '000100' '000010' '110000' '010001' '100000' '101000' '100100' '011000']
1	['000010' '010000' '000001' '000100' '100000' '010010' '011000' '100001' '100100' '101000']
2	['000001' '010000' '000100']
3	['011000']
4	['000001' '010000' '010100' '000100' '100000']
5	['010001' '000001' '010000' '011000' '000010' '000100' '100100' '100000']

Visualising the clusters:



Moving forward, there are several improvements to be made.

- Instead of selecting keywords from the corpus, it can be constructed by hand using the definition of each categories of STRIDE, and subsequently of RAPIDS.
- Filtering of keywords can include more manual methods to further reduce the keyword corpus.
- Visualise and examine the frequencies for some keywords and manually remove some, then we will be left with lesser keywords that the model will be used to train on, potentially increasing the accuracy of classification.

Week 4: Refining the corpus and model

Some minor improvements on keyword filtering is done like removing duplicate words. It can be found in Appendix 1

Creating a set of keywords using Microsoft's definition of STRIDE

S: Involves illegally accessing and then using another user's authentication information, such as username and password

T: Involves the malicious modification of data. Examples include unauthorized changes made to persistent data, such as that held in a database, and the alteration of data as it flows between two computers over an open network, such as the Internet

R: Associated with users who deny performing an action without other parties having any way to prove otherwise—for example, a user performs an illegal operation in a system that lacks the ability to trace the prohibited operations. Non-Repudiation refers to the ability of a system to counter repudiation threats. For example, a user who purchases an item might have to sign for the item upon receipt. The vendor can then use the signed receipt as evidence that the user did receive the package

I: Involves the exposure of information to individuals who are not supposed to have access to it—for example, the ability of users to read a file that they were not granted access to, or the ability of an intruder to read data in transit between two computers

D: Denial of service (DoS) attacks deny service to valid users—for example, by making a Web server temporarily unavailable or unusable. You must protect against certain types of DoS threats simply to improve system availability and reliability

E: An unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system. Elevation of privilege threats include those situations in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself, a dangerous situation indeed

Create keywords from the definitions

S: ['authenticate', 'username', 'password', 'access']

T: ['modify', 'persistent', 'database', 'alter', 'open', 'network', 'internet']

R: ['deny', 'action', 'prove', 'non-repudiation', 'item', 'sign', 'receipt', 'receive', 'evidence', 'package', 'untrace',]

I: ['exposure', 'individual', 'access', 'file', 'granted', 'intruder', 'transit']

D: ['denial', 'service', 'dos', 'web', 'server', 'unavailable', 'unusable', 'system', 'available', 'reliable']

E: ['unprivileged', 'privileged', 'access', 'compromise', 'entire', 'system', 'elevation', 'penetrate', 'defenses', 'untrusted', 'trusted']

Keywords obtained from manually filtering the corpus

S_keep: ['information', 'detection', 'take', 'include', 'malicious', 'control', 'network', 'search', 'name',

'access', 'infrastructure', 'traffic', 'data', 'suspicious', 'trust', 'reconnaissance', 'email', 'phishing', 'resource', 'initial', 'visibility', 'monitor', 'server', 'form', 'open', 'potentially', 'websites', 'address', 'process', 'detect', 'credential', 'file', 'certificate', 'internet', 'install', 'key', 'online', 'link', 'source']

T_keep: ['malicious', 'file', 'activity', 'process', 'execute', 'access', 'information', 'control', 'software', 'modify', 'network', 'data', 'abuse', 'exe', 'manipulate', 'bypass', 'malware', 'functionality', 'integrity', 'dll', 'anomaly', 'install']

R_keep: ['user', 'application', 'api', 'activity', 'audit', 'source', 'system', 'native', 'hide', 'error', 'intrusion', 'function', 'record', 'clear', 'gcp', 'permission', 'analysis', 'collection', 'updatesink', 'indicate', 'detection', 'data', 'collect', 'environment', 'call', 'limit', 'cloudtrail', 'loss', 'conduct', 'prior', 'delete', 'cloud', 'configservicev', 'cloudwatch', 'diagnostic', 'capability', 'sufficient', 'insight', 'avoid']

I_keep: ['data', 'network', 'activity', 'access', 'behavior', 'environment', 'process', 'detection', 'remote', 'base', 'target', 'tool', 'file', 'api', 'traffic', 'acquire', 'application', 'host', 'infrastructure', 'device']

D_keep: ['service', 'target', 'tool', 'command', 'cause', 'server', 'network', 'outside', 'denial', 'dos', 'availability', 'high', 'destruction', 'infrastructure']

E_keep: ['process', 'access', 'file', 'execute', 'activity', 'execution', 'network', 'behavior', 'create', 'control', 'log', 'privilege', 'application', 'service', 'within', 'event', 'account', 'modify', 'run', 'abuse', 'monitoring', 'environment', 'binary', 'credential', 'enable', 'api', 'exe', 'function', 'payload', 'target', 'method', 'services', 'launch', 'root', 'os', 'many"accounts']

→ Now, merge both lists together by group to get for example *S_final*.

Summary: Next week, to compare the semantic similarity of keywords to choose the final set of keywords to use for training the model. If it does not work, I will look into the Large Language Model Meta AI *LLaMA* language model.

Week 5: Solution for selecting final set of keywords

Continuing from last week: With respect to each STRIDE cateogry, when referencing the list keywords we just obtained, eg. *S_final*, and the initial list of keywords we obtained, there exists an underlying problem where we are unsure of which words should be deleted or kept for training the model.
The solution I implemented using *word2vec* to compare both lists and compare the similarities between words and set a minimum cosine value threshold. Any words with similarities below the threshold are deemed 'not similar enough' and will be dropped.
Through experimentation, the threshold value selected is 0.28. An example can be found in Appendix 2

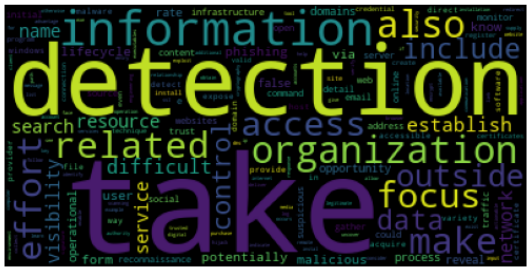
```
def merge_lists(l1, min_cosine_value, w2v, ref_list):
final_list = []

for word in l1:
    if word in w2v:
        similarities = []
        for ref_word in ref_list:
            if ref_word in w2v:
                sim = w2v.similarity(word, ref_word)
                similarities.append(sim)
        if any(sim >= min_cosine_value for sim in similarities):
            final_list.append(word)
if 'use' in final_list: final_list.remove('use')
elif 'also' in final_list: final_list.remove('also')
return final_list
```

Table 3: Comparing keywords (arbitary example)

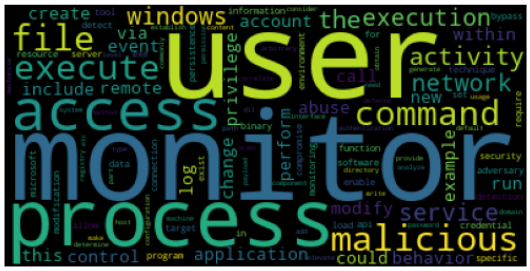
Original set	**Manually filtered set**	**Resultant set + words removed by semantic analysis**
domain, activity, high, direct, include, path, relationship, network, gather, visibility, software, potentially, difficult, make, trusted, supply, gathering, manage, access, form, via, compromise, place, accounts, business, information, focus, elicitation, lifecycle, drive, defender, second, effort, set, provider, take, victim, social, owned, outside, phishing, operational, positive, reconnaissance, stage, domains, party, associate, way, false, variety, chain, well, connect, service, hardware, opportunity, various, online, organization, detection, search, accessible, this, media, target, use, elevate, detail, initial, data, adversary, related, reveal, occurrence, establish, third, etc, expose, relationships, resource, shipment, contractor, ex, open, rate, websites, also	authenticate, username, password, access, information, detection, take, include, malicious, control, network, search, name, access, infrastructure, traffic, data, suspicious, trust, reconnaissance, email, phishing, resource, initial, visibility, monitor, server, form, open, potentially, websites, address, process, detect, credential, file, certificate, internet, install, key, online, link, source	domain, include, relationship, network, gather, visibility, software, potentially, make, trusted, manage, access, form, via, information, lifecycle, provider, take, phishing, operational, reconnaissance, domains, connect, service, hardware, online, organization, detection, search, accessible, media, use, initial, data, reveal, establish, expose, relationships, resource, open, websites, ~~source, file, malicious, certificate, monitor, authenticate, suspicious, email, install, server, link, detect, control, trust, address, credential, internet, process, password, traffic, key, username, name, infrastructure~~

A word cloud for each STRIDE category is generated to visualise the keywords.

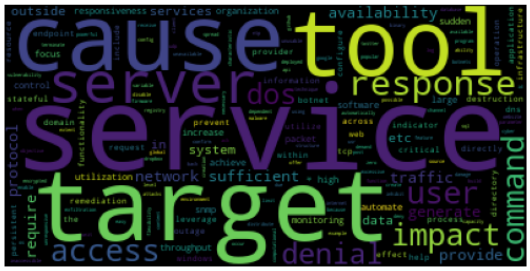
Word Cloud for STRIDE '0': S



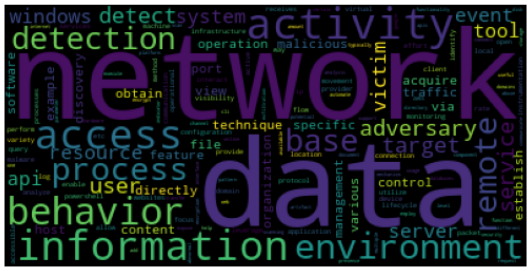
Word Cloud for STRIDE '1': E



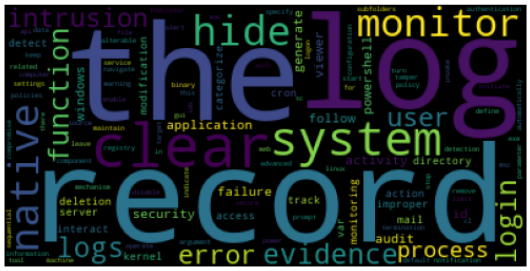
Word Cloud for STRIDE '2': D



Word Cloud for STRIDE '3': I



Word Cloud for STRIDE '4': R

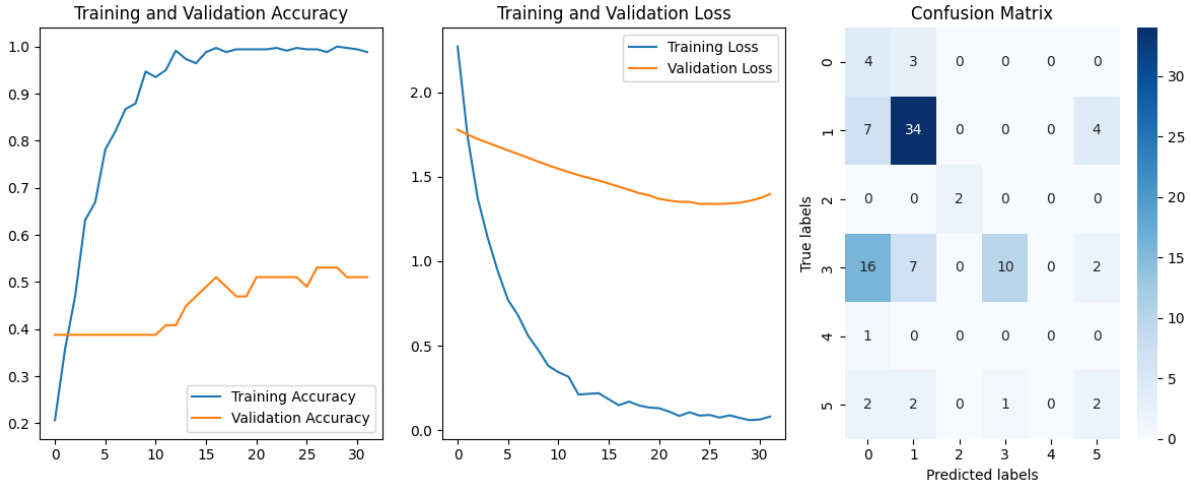


Word Cloud for STRIDE '5': T

Moving on, using the right most column of Table 5, I tokenize them and pass it into a multi-layer perceptron model where it is fine-tuned by hyperparamter testing (Appendix 3).

```
hidden_units = 128
batch_size = 16
num_epochs = 50
num_classes = 6
classes = [0,1,2,3,4,5]
vocab_size = X_train_tfidf.shape[1]
optimizer = tf.keras.optimizers.legacy.Adam(1e-4)

model5 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(vocab_size,)),
    tf.keras.layers.Dense(hidden_units*2, activation='leaky_relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(hidden_units, activation='leaky_relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(hidden_units//2, activation='leaky_relu'),
    tf.keras.layers.Dense(num_classes, kernel_regularizer=tf.keras.regularizers.L2(l2=1e-3), activation='softmax')
])

model5.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=['accuracy'])
model5.summary()
```

The results of this model is as follows:



This model seems to perform terribly on 'T' and 'E' categories.

Some improvements to be done:

- Create a separate model to classify 'T' and 'E' as they yield low accuracies. Then try to combine these models with *model5*
- Evaluate the usefulness of *LLaMA* model

Week 6: Setting up Llama 2

There was a lot of attempt in trying to setup new technologies that I was unfamiliar with this week. The summary is as follows:

- **6.1** Setup Llama 2 7B-chat model on my personal machine
- **6.2** Tried using Google Colab and AWS instances to run the LLM
- **6.3** Attempted at setting up a docker container on the company server to run the Llama 2 model
- **6.4** Found a Massive Text Embedding Benchmark (MTEB) model to try

6.1 Setting up Llama 2 7B-chat model

Findings:

- Llama 2 comes with some variants, {7B, 13B, 70B} models. I was only able to run the 7B model locally due to system requirements. The 13B model works perfectly on the company's server.
- There is still much work to do to fine tune this model, i.e. it still needs to learn how to map MITRE ATT&CKS to STRIDE.

6.2 Google Colab and AWS instance

- Colab is limited by the free storage of 15GB. The smallest 7B model is already 13.48GB and after quantizing it, there is an additional binary file of 3.83GB bringing total storage required to ¿18GB. Hence, insufficient memory.
- AWS instance is doable, but storage costs $0.023 per GB. The hassle comes when I have to delete all files after each time I done working with the server to avoid incurring a charge.

6.3 Setting up a docker container

There was no success to report and am in the process of learning how to create docker container to run a Python code.

Week 7: Experimenting with Llama 2 & different training techniques

The main goals achieved this week are:

- **7.1** Retraining model5 from the previous week using a smaller training set and larger test set
- **7.2** Found a Massive Text Embedding Benchmark (MTEB) model to try
- **7.3** One-vs-the-Rest (OvR) multiclass strategy
- **7.4** Prompt engineering results on Llama 2
- *misc.* Reformatted the original Jupyter files (.ipynb) into Python files (.py) for easier integration with Docker containers

7.1 Training model5 on smaller training set $\Longrightarrow$ model6

I did some research and found out that training models on a smaller training set and larger test set might be useful when training resources are limited. Sure enough, the attempts were not all futile.

I noticed a striking pattern between 'Information disclosure' (I) and 'Elevation of privilege' (E). Most of the time the model either classfies most of the data correctly as the former or latter. This means that if a lot of 'I' is classfied correctly, 'E' is classified wrongly as 'I', vice versa.

Strangely, this could suggest the keywords for both categories are very similar. The following result was the best that I could achieve with the same architecture as model5.

I believe that the model architecture might be too straightforward. As such, I have implemented a more complex model in the next section.

7.2 More complex model architecture

This model is based on the Massive Text Embedding Benchmark (MTEB) model. The model is a transformer-based model. The paper suggests that transformers might help in injecting context awareness into the language model via self-attention.

7.3 One-vs-the-Rest (OvR) multiclass strategy

This OvR strategy is a way to break down a mutliclass classification problem into mutliple binary classification problems, effectively "training a separate models for each category and combining them into a single model".

Appendix

A1. Keyword filtering v2.0

Table 4: Extracting keywords v2.0

Dataset	Original text	Processed text
df_train[0]	Business Relationships Adversaries may gather information about the victim's business relationships that can be used during targeting. Information about an organization's business relationships may include a variety of details, including second or third-party organizations/domains (ex: managed service providers, contractors, etc.) that have connected (and potentially elevated) network access. This information may also reveal supply chains and shipment paths for the victim's hardware and software resources. Adversaries may gather this information in various ways, such as direct elicitation via [Phishing for Information](https://attack.mitre.org/techniques/T1598). Information about business relationships may also be exposed to adversaries via online or other accessible data sets (ex: [Social Media](https://attack.mitre.org/techniques/T1593/001) or [Search Victim-Owned Websites](https://attack.mitre.org/techniques/T1594)).(Citation: ThreatPost Broadvoice Leak) Gathering this information may reveal opportunities for other forms of reconnaissance (ex: [Phishing for Information](https://attack.mitre.org/techniques/T1598) or [Search Open Websites/Domains](https://attack.mitre.org/techniques/T1593)), establishing operational resources (ex: [Establish Accounts](https://attack.mitre.org/techniques/T1585) or [Compromise Accounts](https://attack.mitre.org/techniques/T1586)), and/or initial access (ex: [Supply Chain Compromise](https://attack.mitre.org/techniques/T1195), [Drive-by Compromise](https://attack.mitre.org/techniques/T1189), or [Trusted Relationship](https://attack.mitre.org/techniques/T1199)). Much of this activity may have a very high occurrence and associated false positive rate, as well as potentially taking place outside the visibility of the target organization, making detection difficult for defenders. Detection efforts may be focused on related stages of the adversary lifecycle, such as during Initial Access.	['websites', 'associate', 'contractor', 'supply', 'path', 'online', 'variety', 'operational', 'this', 'phishing', 'gathering', 'via', 'search', 'use', 'adversary', 'open', 'take', 'initial', 'drive', 'victim', 'include', 'etc', 'manage', 'potentially', 'trusted', 'access', 'connect', 'target', 'resource', 'elicitation', 'hardware', 'organization', 'direct', 'place', 'rate', 'elevate', 'network', 'service', 'well', 'domains', 'second', 'activity', 'gather', 'way', 'establish', 'stage', 'provider', 'compromise', 'data', 'defender', 'relationship', 'false', 'chain', 'reconnaissance', 'accounts', 'media', 'social', 'relationships', 'outside', 'information', 'effort', 'domain', 'detail', 'related', 'business', 'software', 'various', 'also', 'opportunity', 'lifecycle', 'ex', 'set', 'owned', 'positive', 'detection', 'third', 'accessible', 'party', 'high', 'difficult', 'occurrence', 'make', 'shipment', 'visibility', 'focus', 'reveal', 'expose', 'form']

Table 5: Extracting keywords v2.0

df_train[2]	Symmetric Cryptography Adversaries may employ a known symmetric encryption algorithm to conceal command and control traffic rather than relying on any inherent protections provided by a communication protocol. Symmetric encryption algorithms use the same key for plaintext encryption and ciphertext decryption. Common symmetric encryption algorithms include AES, DES, 3DES, Blowfish, and RC4. With symmetric encryption, it may be possible to obtain the algorithm and key from samples and use them to decode network traffic to detect malware communications signatures. In general, analyze network data for uncommon data flows (e.g., a client sending significantly more data than it receives from a server). Processes utilizing the network that do not normally have network communication or have never been seen before are suspicious. Analyze packet contents to detect communications that do not follow the expected protocol behavior for the port that is being used.(Citation: University of Birmingham C2)	['content', 'sample', 'packet', 'know', 'obtain', 'data', 'flow', 'provide', 'with', 'uncommon', 'control', 'utilize', 'traffic', 'port', 'key', 'general', 'detect', 'de', 'algorithm', 'see', 'send', 'rc', 'follow', 'decryption', 'suspicious', 'employ', 'protection', 'processes', 'plaintext', 'cryptography', 'malware', 'ciphertext', 'inherent', 'analyze', 'never', 'conceal', 'behavior', 'use', 'in', 'signature', 'symmetric', 'command', 'common', 'possible', 'network', 'include', 'aes', 'encryption', 'protocol', 'server', 'rather', 'rely', 'blowfish', 'receives', 'des', 'expect', 'communication', 'significantly', 'normally', 'client']

A2. Word2Vec semantic analysis

```
min_cosine_value = 0.28
list1 = ['apples', 'oranges', 'samsung', 'today']
reference_words = ['fruit', 'fruits']
```

With reference to the code snippet above, I want to filter out the words that are associated with 'fruits' as in *reference_words*. I adjust the *min_cosine_value* (threshold) such that the words *merge_lists()* outputs are fruits.

Similarly in the main project, the reference words are the manually obtained keywords for each STRIDE category, while *list1* is the original set of keywords that requrie additional filtering.

A3. Hyperparameter tuning for MLP

```
num_epochs = 50
num_classes = 6
vocab_size = X_train_tfidf.shape[1]
dropout_rates = [0.2, 0.3, 0.4, 0.5]
activations_list = ['relu', 'leaky_relu', 'elu', 'tanh']
num_neurons = [32, 64, 128, 256]
opt_lr = [1e-2, 1e-3, 1e-4]
L2_lr = [1e-2, 1e-3, 1e-4]
best_params = None
best_val_acc = 0

hyperparam_combi = itertools.product(dropout_rates, num_neurons, activations_list, opt_lr, L2_lr)

for dr, nn, al, olr, l2lr in hyperparam_combi:
    modelTest = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(vocab_size,)),
    tf.keras.layers.Dense(nn*2, activation=al),
    tf.keras.layers.Dropout(dr),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(nn, activation=al),
    tf.keras.layers.Dropout(dr),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(nn//2, activation=al),
    tf.keras.layers.Dense(num_classes, kernel_regularizer=tf.keras.regularizers.L2(l2=1e-2), activation='softmax')
    ])

    optimizer = tf.keras.optimizers.legacy.Adam(olr)
    modelTest.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=['accuracy'])

    early_stop = EarlyStopping(
        monitor="val_loss",
        patience=5,
        verbose=0,
        restore_best_weights=True
    )
    histTest = modelTest.fit(
        X_train_tfidf, y_train,
        batch_size=16,
        epochs=num_epochs,
        validation_data=(X_val_tfidf, y_val),
        verbose=0,
        callbacks=[early_stop,]
    )

    val_acc = max(histTest.history['val_accuracy'])
    # print(f"Dropout: {dr}, Activation: {al}, Hidden Units: {nn}, L2 Reg: {l2lr}, LR: {olr}, Best Val Acc: {val_acc}\n==========================")
    if val_acc > best_val_acc:
        best_val_acc = val_acc
        best_params = (dr, nn, al, olr, l2lr)
print(f"Final Best Hyperparameters: Dropout: {best_params[0]},\nActivation: {best_params[2]},\nHidden Units: {best_params[1]},\nL2 Reg: {best_params[4]},\nLR: {best_params[3]},\nBest Val Acc: {best_val_acc}")
```