AUTOMATED EMERGING CYBER THREAT IDENTIFICATION AND PROFITING BASED ON NATURAL LANGUAGE PROCESSING

KOPPADI PAVAN KUMAR

23221D5809

ABSTRACT :

The time window between the disclosure of a new cyber vulnerability and its use by cybercriminals has been getting smaller and smaller over time. A recent episode, such as Log4j vulnerability, exemplifies this well. Within hours after the exploit being released, attackers started scanning the internet looking for vulnerable hosts to deploy threats like crypto currency miners and ransomware on vulnerable systems. Thus, it becomes imperative for the cyber security defense strategy to detect threats and their capabilities as early as possible to maximize success of prevention actions. Although crucial, discovering new threats is a challenging activity for security analysts due to the immense volume of data and information sources to be analyzed for signs that a threat is emerging. In this sense, we present a framework for automatic identification and profiling of emerging threats using Twitter messages as a source of events and MITRE ATT&CK as a source of knowledge for threat characterization. The framework comprises three main parts: identification of cyber threats and their names; profiling the identified threat in terms of its intentions or goals by employing two machine learning layers to filter and classify tweets; and alarm generation based on the threat's risk. The main contribution of our work is the approach to characterize or profile the identified threats in terms of its intentions or goals, providing additional context on the threat and avenues for mitigation. In our experiments the profiling stage reached a F1 score of 77% in correctly profiling discovered threats.

INTRODUCTION

Recently there has been an increasing reliance on the Internet for business, government, and social interactions as a result of a trend of hyper-connectivity and hypermobility. While the Internet has become an indispensable infrastructure for businesses, governments, and societies, there is also an increased risk of cyber attacks with different motivations and intentions. Preventing organizations from cyber exploits needs timely intelligence about cyber vulnerabilities and attacks, referred to as threats [1]. Threat intelligence is defined as "evidence-based knowledge, including context, mechanisms, indicators, implications, and actionable advice, about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject's response to that menace or hazard" [2]. Threat intelligence in cyber security domain, or cyber threat intelligence, provides timely and relevant information, such as signatures of the attacks, that can help reduce the uncertainty in identifying potential security vulnerabilities and attacks. Cyber threat intelligence can generally be extracted from informal or formal sources, which officially release threat information in structured data format. Structured threat intelligence adheres to a well-defined data model, with a common format and structure. Structured cyber threat intelligence, therefore, can be easily parsed by security tools to analyze and respond to security threats accordingly. Examples of formal sources of cyber threat intelligence include the Common Vulnerabilities and Exposures (CVE) database 1 and the National Vulnerability Database (NVD) 2 . Cyber threat intelligence is also available on informal sources, such as public blogs, dark webs, forums, and social media platforms. Informal sources allow any person or entity on the Internet to publish, in real-time, the threat information in natural language, or unstructured data format. The unstructured and publicly available threat intelligence is also called Open Source Intelligence (OSINT) [3]. Cyber security-related OSINT are early warning sources for cyber security events such as security vulnerability exploits [4]. To conduct a cyber-attack, malicious actors typically have to 1) identify vulnerabilities, 2) acquire the necessary tools and tradecraft to successfully exploit them, 3) choose a target and recruit participants, 4) create or purchase the infrastructure needed, and 5) plan and execute the attack. Other actors— system administrators, security analysts, and even victims— may discuss vulnerabilities or coordinate a response to attacks [5]. These activities are often conducted online through social media, (open and dark) Web forums, and professional blogs, leaving digital traces behind. Collectively, these digital traces provide valuable

insights into evolving cyber threats and can signal a pending or developing attack well before the malicious activity is noted on a target system. For example, exploits are discussed on Twitter before they are publicly disclosed [4] and on darkweb forums even before they are discussed on social media [6]. Open Source Intelligence (OSINT) is intelligence gathered from public-available sources such as social network sites, forums, wikis, blogs, and so on [7]. Malicious actors, system administrators, security analysts, and victims of cyber attacks usually use such platforms to discuss vulnerabilities, and exploits or to coordinate a response to attacks. Although more difficult to consume due to the volume and unstructured format of the content, data obtained from OSINT sources can complement intelligence obtained from structured intelligence sources, which usually provide malicious IP addresses and hashes, for example, as indicators of compromise (IOCs) that must be monitored or blocked by security platforms. Among OSINT sources available, we choose Twitter due to its ability to act as a natural aggregator of multiple sources [8] and its big data characteristics: a large volume of data, a highly diverse pool of users, high accessibility, and, mainly, timely production of new content [9]. The popularity of this medium in the cybersecurity community provides an environment for both offensive and defensive practitioners to discuss, report, and advertise timely indicators of vulnerabilities, attacks, malware, and other types of cyber events that are of interest to security analysts. In the past decade, Twitter has become an important source of intelligence. The real-time nature of information on Twitter has allowed researchers to use the microblog to extract intelligence about different areas such as terrorist attacks [10], earthquakes [11], forest fires [12] and so on. The value of Twitter with regards to security is well-demonstrated by the numerous initial reports of cyber events, examples of which include disclosures of multiple 0-day, user reports on DDoS attacks, and exposure of ransomware campaigns. For example, in June 2017, the global ransomware outbreak of ’Petya/NotPetya’ was discussed widely via Twitter before being reported by mainstream media [13]. Another more recent example of cyber threat initially discussed in Twitter was Log4Shell. Log4Shell was the name given to a 0-day exploit to a vulnerability in Log4j2 (CVE2021-44228), a popular Java logging library. The Log4j2 vulnerability along with a link to the exploit code, which means the code able to take advantage of a vulnerability in an easy way, was disclosed by the profile @P0rZ9 on December 9th, 2021, on Twitter. Following this post, hundreds of Twitter profiles, including independent researchers and journalists specialized in cyber security, started to post about the vulnerability. Given this strong and constant presence of the cyber security community in Twitter,

over the recent years, the research on Twitter-based OSINT collection has led to the proposal of multiple frameworks [14], [15], [7], [16], [17] for detection and analysis of threat indicators in the Twitter stream. The shortness of tweets, which nowadays is a text of 280 maximum characters, is considered one of the main challenges when classifying tweets using machine learning algorithms [18]. In contrast with large document corpora, analyzing short documents such as tweets presents some specific semantic challenges towards extracting terms, relationships, patterns, and actionable insights in general [19].

LITERATURE SURVEY

Title: Data Warehousing Process Modeling from Classical Approaches to New Trends: Main Features and Comparisons

Author: Asma Dhaouadi , Khadija Bousselmi , Mohamed Mohsen Gammoudi, Sébastien Monnet, Slimane Hammoudi

The extract, transform, and load (ETL) process is at the core of data warehousing architectures. As such, the success of data warehouse (DW) projects is essentially based on the proper modeling of the ETL process. As there is no standard model for the representation and design of this process, several researchers have made efforts to propose modeling methods based on different formalisms, such as unified modeling language (UML), ontology, model-driven architecture (MDA), model-driven development (MDD), and graphical flow, which includes business process model notation (BPMN), colored Petri nets (CPN), Yet Another Workflow Language (YAWL), CommonCube, entity modeling diagram (EMD), and so on. With the emergence of Big Data, despite the multitude of relevant approaches proposed for modeling the ETL process in classical environments, part of the community has been motivated to provide new data warehousing methods that support Big Data specifications. In this paper, we present a summary of relevant works related to the modeling of data warehousing approaches, from classical ETL processes to ELT design approaches. A systematic literature review is conducted and a detailed set of comparison criteria are defined in order to allow the reader to better understand the evolution of these processes. Our study paints a complete picture of ETL modeling approaches, from their advent to the era of Big Data, while comparing their main characteristics. This study allows for the identification of the main challenges and issues related to the design of Big Data warehousing systems, mainly involving the lack of a generic design model for data collection, storage, processing, querying, and analysis. The ETL process is used to extract data from different sources; transform them to meet specific analytical needs; and, finally, load the processed data into a dedicated storage system to support them, called a data warehouse. As the success of the project and the ease of its maintenance are strongly linked to the modeling stage, all DW development projects should rely on the well-designed modeling of the data warehousing process, as there is no standard model for the representation and design of this process at present. In the early 2000s, the research community

worked towards proposing different methods for conceptual, logical, and physical modeling for the ETL process. As a result, many studies have been published in this field, where each proposed contribution has its specific advantages and suffers from limitations. However, with the emergence of Big Data, the community has been faced with new challenges. Hence, considering the importance of this topic, our main objective in this paper was to review relevant research conducted from the introduction of ETLs to the present day. In this paper, we defined a set of comparison criteria to simplify understanding ETL/ELT process characteristics. We categorized the proposed research works into six major classes, UML, ontology, MDA and MDD, graphical flow, ad hoc formalisms, and, finally, contributions in the context of Big Data. Then, a comparative study of the different contributions was presented and discussed. Our synthetic study browsed the related review papers in this field and we discussed other findings from our survey, thus proving the usefulness of our literature review. We proposed some general recommendations and a case study using the comparative study. Finally, we found that, to date, no synthetic study in the field of ETL process modeling considering the characteristics of Big Data has been carried out. Consequently, ETL process modeling, in its different phases, must evolve to support the new generation of technologies that have emerged in the era of Big Data, particularly in terms of data collection, storage, processing, querying, and analysis.

Title: Integration of Data Warehouse and Unstructured Business Documents

Author: Ahmad Abdullah Alqarni; Eric Pardede

The profusion of unstructured data forced organizations to manage and take advantage of such data especially in the decision making process. The feasibility of integrating or mapping unstructured data to a data warehouse is becoming significant to bridge this gap and take the full potential of these data. In this paper, we propose a multi-layer schema for mapping structured data stored in a data warehouse and unstructured data in business-related documents. The multi-layer schema facilitates the mapping between the two different data. Linguistically correlated data is identified using Word Net to enable the integration between both data sources. We also propose a generic XML schema for business-related unstructured documents to assist the mapping. The use Word Net to identify the matching result is promising in the absence of schema-instance and without the need to domain specific knowledge. The recent development of analytical information systems shows that the necessary integration of structured and unstructured data sources in data warehousing is possible. The usage of the market information system shows that the database

improves the analytical power of decision makers, in order to recognize tendencies in the energy market promptly. Nevertheless the respective model and the system must grant high flexibility to adjust them to changing conditions in the energy market. Furthermore the activities on the energy market and the work of the analysts will enhance the system. Market information systems have to be optimized by better evaluation of external information and automatization of process integration. Only documents of decision relevance should be delivered to the management. The ROI of data warehouse projects can be increased if event-based and accepted information improves the decision quality significantly. The information flow alignment in MAIS is equivalent to a classification problem. We assure this by using role profiles and embedded recommendation systems with a document trigger mechanism. Furthermore the use of a simulation method is tightly linked to this process by matching simulation variables to trigger conditions. The integration of metadata from a data warehouse, personalized search patterns and simulation variables give a powerful repository for active data warehousing. The theoretical approach and the benefit of creating interfaces for the meta models are part of further research. Nevertheless, decision makers gain individualized decision support and early insight into future developments.

The quality of classification algorithms must be examined in regular time intervals to guarantee best results. Therefore it is necessary to optimize the structure of the test environment which has to support intersubjective and intertemporal comparability of the test results. Classification evaluations are often accomplished; however these results are only important in the context of the selected data set and evaluation environment. In order to acquire concrete statements for MAIS, such an evaluation environment and the results are described in this paper. In order to find the perfect search terms, the most relevant documents are to be found so that not just the classification itself has to be optimized, but the Internet retrieval as well.

Title: The History, Present, and Future of ETL Technology

Author: Alkis Simitsis, Spiros Skiadopoulos, Panos Vassiliadis

There is an abundance of data, but a large volume of it is unusable. Data may be noisy, unstructured, stored in incompatible for direct analysis medium or format, and often expensive to access. In most practical cases, the data needs to be processed before it can be used to extract valuable business insights. We refer to the nontrivial, end-to-end operation of extracting intelligence from raw data as an ETL process. In this paper, we review how the ETL technology

has been evolved in the last 25 years, from a rather neglected engineering challenge to a first-class citizen in analytics and data processing. We present a brief historical overview of ETL, discuss its various applications and incarnations in modern data processing environments, and argue about exciting, feasible or wishful, potential future directions. The ETL technology and data integration in general has been the cornerstone of business intelligence, decision making, and data analytics for over 25 years. ETL thrives while at the same time it evolves along with shifting business needs and data technology advancements. As researchers and practitioners alike are exploring ways to extract value from large collections of raw data, ETL is the connecting glue to make this happen. In this paper, we presented a brief overview of the ETL history, described recent trends in the end-to-end data stack, and discussed some interesting, in our opinion, future directions that will most likely impact the next generation of ETL and data integration technology. The past 20+ years have been educating, enjoyable, and productive in devising and realizing efficient and effective ways to tame data intricacies and peculiarities blending a multiplicity of technologies and applying them in the real world. We look forward to the next 20 that will be even more exciting and fruitful.

Title: An Overview of Data Warehouse and Data Lake in Modern Enterprise Data Management

Author: Athira Nambiar * and Divyansh Mundra

Data is the lifeblood of any organization. In today's world, organizations recognize the vital role of data in modern business intelligence systems for making meaningful decisions and staying competitive in the field. Efficient and optimal data analytics provides a competitive edge to its performance and services. Major organizations generate, collect and process vast amounts of data, falling under the category of big data. Managing and analyzing the sheer volume and variety of big data is a cumbersome process. At the same time, proper utilization of the vast collection of an organization's information can generate meaningful insights into business tactics. In this regard, two of the popular data management systems in the area of big data analytics (i.e., data warehouse and data lake) act as platforms to accumulate the big data generated and used by organizations. Although seemingly similar, both of them differ in terms of their characteristics and applications. This article presents a detailed overview of the roles of data warehouses and data lakes in modern enterprise data management. We detail the definitions, characteristics and related works for the respective data management frameworks. Furthermore, we explain the architecture and design considerations of the current state of the art. Finally, we provide a perspective on the challenges and promising research directions for the future. Enterprises and business organizations exploit a

huge volume of data to understand their customers and to make informed business decisions to stay competitive in the field. However, big data come in a variety of formats and types (e.g., structured, semi-structured and unstructured data), making it difficult for businesses to manage and use them effectively. Based on the structure of the data, typically, two types of data storage are utilized in enterprise data management: the data warehouse (DW) and data lake (DL). Although being used as interchangeable terms, they are two distinct storage forms with unique characteristics that serve different purposes. In this review, a comparative analysis of data warehouses and data lakes by highlighting the key differences between the two data management approaches was envisaged. In particular, the definitions of the data warehouse and data lake, highlighting their characteristics and key differences, were detailed. Furthermore, the architecture and design aspects of both DWs and DLs are clearly discussed. In addition, a detailed overview of the popular DW and DL tools and services was also provided. The key challenges of big data analytics in general, as well as the challenges of implementation of DWs and DLs, were also critically analyzed in this survey. Finally, the opportunities and future research directions were contemplated. We hope that the thorough comparison of existing data warehouses vs. data lakes and the discussion of open research challenges in this survey will motivate the future development of enterprise data management and benefit the research community significantly.

Title: An Efficient Spark-Based Hybrid Frequent Itemset Mining Algorithm for Big Data

Author: Mohamed Reda Al-Bana, Marwa Salah Farhan and Nermin Abdelhakim Othman

Frequent itemset mining (FIM) is a common approach for discovering hidden frequent patterns from transactional databases used in prediction, association rules, classification, etc. Apriori is an FIM elementary algorithm with iterative nature used to find the frequent itemsets. Apriori is used to scan the dataset multiple times to generate big frequent itemsets with different cardinalities. Apriori performance descends when data gets bigger due to the multiple dataset scan to extract the frequent itemsets. Eclat is a scalable version of the Apriori algorithm that utilizes a vertical layout. The vertical layout has many advantages; it helps to solve the problem of multiple datasets scanning and has information that helps to find each itemset support. In a vertical layout, itemset support can be achieved by intersecting transaction ids (tidset/tids) and pruning irrelevant itemsets. However, when tids become too big for memory, it affects algorithms efficiency. In this paper, we introduce SHFIM (spark-based hybrid frequent itemset mining), which is a three-phase algorithm that utilizes both horizontal and vertical layout diffset instead of tidset to keep track of the

differences between transaction ids rather than the intersections. Moreover, some improvements are developed to decrease the number of candidate itemsets. SHFIM is implemented and tested over the Spark framework, which utilizes the RDD (resilient distributed datasets) concept and in-memory processing that tackles MapReduce framework problem. We compared the SHFIM performance with Spark-based Eclat and dEclat algorithms for the four benchmark datasets. Experimental results proved that SHFIM outperforms Eclat and dEclat Spark-based algorithms in both dense and sparse datasets in terms of execution time. FIM is the most common technique used in discovering frequent patterns from transactional datasets. Frequent patterns have a wide effect in many applications including classifications, market basket analysis, mobile computing, web mining, etc. Apriori is computing intensive algorithm; therefore, lots of resources (Memory and processing) are required. Moreover, Apriori uses horizontal data representation and has some challenges such as multiple dataset scans and candidate generating in each iteration, which makes Apriori suffer from big data. Vertical data representation is smaller than horizontal representation in size and carries information through tidsets about each itemset support. Eclat uses vertical data representation (tidset) and achieved observed performance in sparse datasets, but in dense datasets, it suffers when intermediate results of tidsets become too large for memory. In this paper, we proposed SHFIM (Spark-based Hybrid Frequent Itemset Mining) a novel algorithm that utilizes both the horizontal and vertical layouts to solve the drawbacks in both Apriori and Eclat. SHFIM is a three phases algorithm, which works perfectly in a distributed environment. Phases one and two use the horizontal layout to extract the first and second frequent itemset. Phase three is an iterative process to extract k frequent itemset in k iterations. This phase uses mainly diffset to enhance execution time and memory consumption because it shrinks itemsets into smaller sets that will be mined in less time and consume less space. The support of an itemset is calculated by exploiting the vertical layout in every worker node. As the vertical layout size is smaller than the horizontal layout, therefore it requires less memory and less execution time. We developed SHFIM on Spark framework due to its ability to deal with the iterative problem better than Hadoop MapReduce. Spark is 100 times quicker than Hadoop in data processing and has lots of features such as in-memory processing, RDD data structure, broadcasting variables, partitioning of data, and applied Spark best practices to reduce data shuffling between nodes. These features make the Spark the best choice for us that help SHFIM to deal with big data efficiently and increase its execution time performance. Extensive experiments have been conducted between SHFIM, Eclat,

and dEclat over Spark clusters for dense and sparse datasets. The Experimental results proved that SHFIM can compete well in both dense and sparse datasets and shows noticeably better performance in either lower or higher min_sup in terms of execution time than others in datasets that have lots of variable-length transactions which is the nature of big data. In the future work, we are planning to enhance the SHFIM be more efficient. The results proved that the use of tidset, diffset, and Bloom Filter accelerate the speed of FIM in big datasets. We plan to choose between tidset and diffset on the itemset itself rather than the whole dataset instead of applying the diffset and continue using diffset from the third iteration in the whole dataset.

PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++or Java. It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt −

Python 2.4.3 (#1, Nov 11 2010, 13:34:43)

[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>>

Type the following text at the Python prompt and press the Enter −

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");. However in Python version 2.4.3, this produces the following result −

```
Hello, Python!
```

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file −

Live Demo

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows −

```
$ python test.py
```

This produces the following result −

```
Hello, Python!
```

Let us try another way to execute a Python script. Here is the modified test.py file −

Live Demo

```
#!/usr/bin/python

print "Hello, Python!"
```

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows −

```
$ chmod +x test.py     # This is to make file executable
$./test.py
```

This produces the following result −

```
Hello, Python!
```

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers −

Class names start with an uppercase letter. All other identifiers start with a lowercase letter.

Starting an identifier with a single leading underscore indicates that the identifier is private.

Starting an identifier with two leading underscores indicates a strongly private identifier.

If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and exec not

assert finally or

break for pass

class from print

continue global raise

def if return

del import try

elif in while

else is with

except lambda yield

Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

```
if True:
   print "True"
else:
   print "False"
```

However, the following block generates an error −

```
if True:
print "Answer"
print "True"
else:
print "Answer"
print "False"
```

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks −

Note − Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

```
#!/usr/bin/python
import sys
try:
   # open file stream
   file = open(file_name, "w")
except IOError:
   print "There was an error writing to", file_name
   sys.exit()
print "Enter '", file_finish,
print "' When finished"
```

```
while file_text != file_finish:
   file_text = raw_input("Enter text: ")
   if file_text == file_finish:
      # close the file
      file.close
      break
   file.write(file_text)
   file.write("\n")
file.close()
file_name = raw_input("Enter filename: ")
if len(file_name) == 0:
   print "Next time please enter something"
   sys.exit()
try:
   file = open(file_name, "r")
except IOError:
   print "There was an error reading file"
   sys.exit()
file_text = file.read()
```

```
file.close()

print file_text
```

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example −

```
total = item_one + \
    item_two + \
    item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example −

```
days = ['Monday', 'Tuesday', 'Wednesday',
    'Thursday', 'Friday']
```

Quotation in Python

Python accepts single ('), double (") and triple ('" or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal −

word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""

Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

First comment

print "Hello, Python!" # second comment

This produces the following result −

Hello, Python!

You can type a comment on the same line after a statement or expression −

name = "Madisetti" # This is again comment

You can comment multiple lines as follows −

This is a comment.

This is a comment, too.

This is a comment, too.

I said that already.

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

'''

This is a multiline

comment.

'''

Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Waiting for the User

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action −

raw_input("\n\nPress the enter key to exit.")

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon.

import sys; x = 'foo'; sys.stdout.write(x + '\n')

Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite. For example −

```
if expression :
   suite
elif expression :
   suite
else :
```

suite

Command Line Arguments

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

Options and arguments (and corresponding environment variables):

-c cmd : program passed in as string (terminates option list)

-d : debug output from parser (also PYTHONDEBUG=x)

-E : ignore environment variables (such as PYTHONPATH)

-h : print this help message and exit

You can also program your script in such a way that it should accept various options. Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.

Python Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
```

list2 = [1, 2, 3, 4, 5];

list3 = ["a", "b", "c", "d"]

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example –

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5);

tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing −

tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value −

tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7);

print "tup1[0]: ", tup1[0];

print "tup2[1:5]: ", tup2[1:5];

When the above code is executed, it produces the following result −

tup1[0]: physics

tup2[1:5]: [2, 3, 4, 5]

Updating Tuples

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']

print "dict['Age']: ", dict['Age']
```

When the above code is executed, it produces the following result −

```
dict['Name']:  Zara

dict['Age']:  7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Alice']: ", dict['Alice']
```

When the above code is executed, it produces the following result −

```
dict['Alice']:

Traceback (most recent call last):
```

```
  File "test.py", line 4, in <module>
    print "dict['Alice']: ", dict['Alice'];
KeyError: 'Alice'
```

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example –

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

When the above code is executed, it produces the following result −

```
dict['Age']:  8
dict['School']:  DPS School
```

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'

dict.clear();     # remove all entries in dict

del dict ;        # delete entire dictionary


print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School']
```

This produces the following result. Note that an exception is raised because after del dict dictionary does not exist any more −

```
dict['Age']:

Traceback (most recent call last):

  File "test.py", line 8, in <module>

    print "dict['Age']: ", dict['Age'];

TypeError: 'type' object is unsubscriptable
```

Note − del() method is discussed in subsequent section.

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example −

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}

print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
dict['Name']:  Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example −

```
dict = {['Name']: 'Zara', 'Age': 7}

print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
Traceback (most recent call last):

  File "test.py", line 3, in <module>

    dict = {['Name']: 'Zara', 'Age': 7};

TypeError: unhashable type: 'list'
```

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
tup1 = (12, 34.56);

tup2 = ('abc', 'xyz');


# Following action is not valid for tuples

# tup1[0] = 100;


# So let's create a new tuple as follows

tup3 = tup1 + tup2;
```

```
print tup3;
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement. For example −

```
tup = ('physics', 'chemistry', 1997, 2000);

print tup;

del tup;

print "After deleting tup : ";

print tup;
```

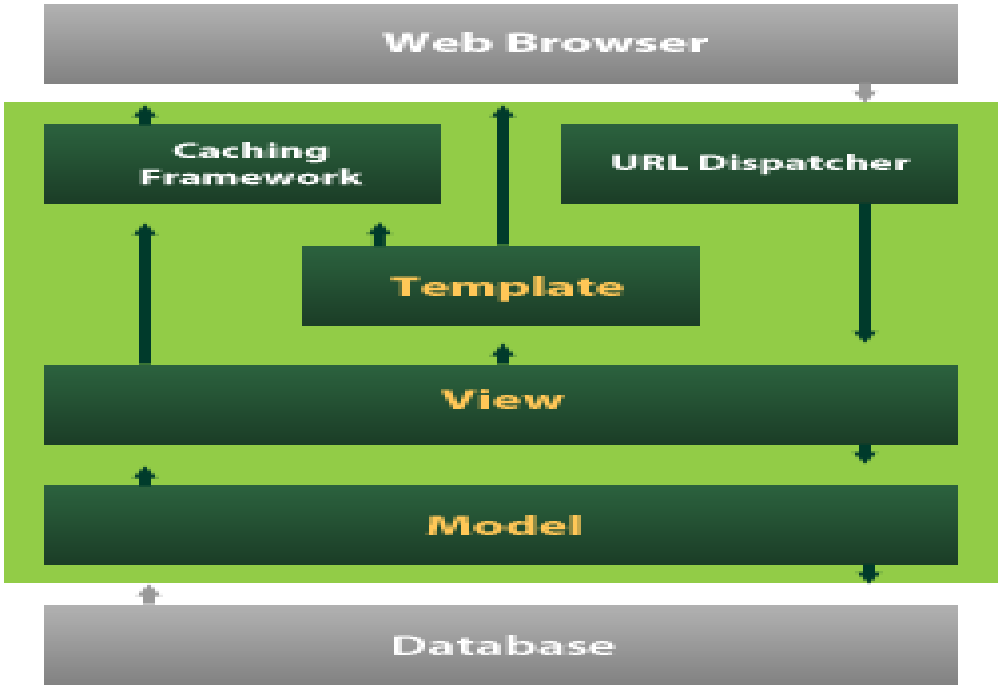
This produces the following result. Note an exception raised, this is because after del tup tuple does not exist any more −

```
('physics', 'chemistry', 1997, 2000)

After deleting tup :

Traceback (most recent call last):

  File "test.py", line 9, in <module>

    print tup;

NameError: name 'tup' is not defined
```

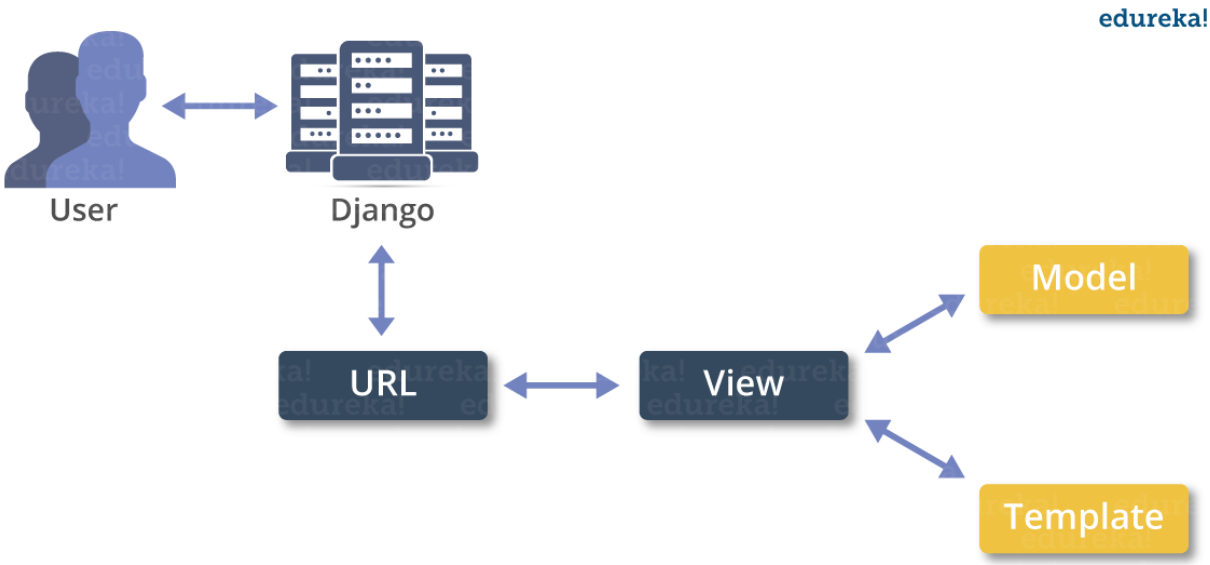
DJANGO

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusabilityand "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models.



Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models



Create a Project

Whether you are on Windows or Linux, just get a terminal or a cmd prompt and navigate to the place you want your project to be created, then use this code −

$ django-admin startproject myproject

This will create a "myproject" folder with the following structure −

myproject/

manage.py

myproject/

__init__.py

settings.py

urls.py

wsgi.py

The Project Structure

The “myproject” folder is just your project container, it actually contains two elements −

manage.py − This file is kind of your project local django-admin for interacting with your project via command line (start the development server, sync db...). To get a full list of command accessible via manage.py you can use the code −

$ python manage.py help

The “myproject” subfolder − This folder is the actual python package of your project. It contains four files −

__init__.py − Just for python, treat this folder as package.

settings.py − As the name indicates, your project settings.

urls.py − All links of your project and the function to call. A kind of ToC of your project.

wsgi.py − If you need to deploy your project over WSGI.

Setting Up Your Project

Your project is set up in the subfolder myproject/settings.py. Following are some important options you might need to set −

DEBUG = True

This option lets you set if your project is in debug mode or not. Debug mode lets you get more information about your project's error. Never set it to ‘True’ for a live project. However, this has to be set to ‘True’ if you want the Django light server to serve static files. Do it only in the development mode.

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.sqlite3',
      'NAME': 'database.sql',
      'USER': '',
      'PASSWORD': '',
      'HOST': '',
      'PORT': '',
   }
}
```

Database is set in the ‘Database’ dictionary. The example above is for SQLite engine. As stated earlier, Django also supports −

MySQL (django.db.backends.mysql)

PostGreSQL (django.db.backends.postgresql_psycopg2)

Oracle (django.db.backends.oracle) and NoSQL DB

MongoDB (django_mongodb_engine)

Before setting any new engine, make sure you have the correct db driver installed.

You can also set others options like: TIME_ZONE, LANGUAGE_CODE, TEMPLATE…

Now that your project is created and configured make sure it's working −

$ python manage.py runserver

You will get something like the following on running the above code −

Validating models...

0 errors found

September 03, 2015 - 11:41:50

Django version 1.6.11, using settings 'myproject.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

A project is a sum of many applications. Every application has an objective and can be reused into another project, like the contact form on a website can be an application, and can be reused for others. See it as a module of your project.

Create an Application

We assume you are in your project folder. In our main “myproject” folder, the same folder then manage.py −

$ python manage.py startapp myapp

You just created myapp application and like project, Django create a “myapp” folder with the application structure −

myapp/

__init__.py

admin.py

models.py

tests.py

views.py

__init__.py − Just to make sure python handles this folder as a package.

admin.py − This file helps you make the app modifiable in the admin interface.

models.py − This is where all the application models are stored.

tests.py − This is where your unit tests are.

views.py − This is where your application views are.

Get the Project to Know About Your Application

At this stage we have our "myapp" application, now we need to register it with our Django project "myproject". To do so, update INSTALLED_APPS tuple in the settings.py file of your project (add your app name) −

```
INSTALLED_APPS = (
   'django.contrib.admin',
   'django.contrib.auth',
   'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'myapp',
)
```

Creating forms in Django, is really similar to creating a model. Here again, we just need to inherit from Django class and the class attributes will be the form fields. Let's add a forms.py file in myapp folder to contain our app forms. We will create a login form.

myapp/forms.py

```
#-*- coding: utf-8 -*-
from django import forms

class LoginForm(forms.Form):
    user = forms.CharField(max_length = 100)
    password = forms.CharField(widget = forms.PasswordInput())
```

As seen above, the field type can take "widget" argument for html rendering; in our case, we want the password to be hidden, not displayed. Many others widget are present in Django: DateInput for dates, CheckboxInput for checkboxes, etc.

Using Form in a View

There are two kinds of HTTP requests, GET and POST. In Django, the request object passed as parameter to your view has an attribute called "method" where the type of the request is set, and all data passed via POST can be accessed via the request.POST dictionary.

Let's create a login view in our myapp/views.py −

```
#-*- coding: utf-8 -*-

from myapp.forms import LoginForm

def login(request):
   username = "not logged in"

   if request.method == "POST":
      #Get the posted form
```

```
    MyLoginForm = LoginForm(request.POST)

    if MyLoginForm.is_valid():
      username = MyLoginForm.cleaned_data['username']
  else:
    MyLoginForm = Loginform()

  return render(request, 'loggedin.html', {"username" : username})
```

The view will display the result of the login form posted through the loggedin.html. To test it, we will first need the login form template. Let's call it login.html.

```
<html>
  <body>

    <form name = "form" action = "{% url "myapp.views.login" %}"
      method = "POST" >{% csrf_token %}

      <div style = "max-width:470px;">
        <center>
```

```
      <input type = "text" style = "margin-left:20%;"
        placeholder = "Identifiant" name = "username" />
    </center>
</div>

<br>

<div style = "max-width:470px;">
    <center>
      <input type = "password" style = "margin-left:20%;"
        placeholder = "password" name = "password" />
    </center>
</div>

<br>

<div style = "max-width:470px;">
    <center>
```

```
            <button style = "border:0px; background-color:#4285F4; margin-top:8%;
              height:35px; width:80%;margin-left:19%;" type = "submit"
              value = "Login" >
              <strong>Login</strong>
            </button>


          </center>
        </div>


      </form>


    </body>
</html>
```

The template will display a login form and post the result to our login view above. You have probably noticed the tag in the template, which is just to prevent Cross-site Request Forgery (CSRF) attack on your site.

```
{% csrf_token %}
```

Once we have the login template, we need the loggedin.html template that will be rendered after form treatment.

```
<html>

   <body>
      You are : <strong>{{username}}</strong>
   </body>

</html>
```

Now, we just need our pair of URLs to get started: myapp/urls.py

```
from django.conf.urls import patterns, url
from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',
   url(r'^connection/',TemplateView.as_view(template_name = 'login.html')),
   url(r'^login/', 'login', name = 'login'))
```

When accessing "/myapp/connection", we will get the following login.html template rendered −

Setting Up Sessions

In Django, enabling session is done in your project settings.py, by adding some lines to the MIDDLEWARE_CLASSES and the INSTALLED_APPS options. This should be done while creating the project, but it's always good to know, so MIDDLEWARE_CLASSES should have −

'django.contrib.sessions.middleware.SessionMiddleware'

And INSTALLED_APPS should have −

'django.contrib.sessions'

By default, Django saves session information in database (django_session table or collection), but you can configure the engine to store information using other ways like: in file or in cache.

When session is enabled, every request (first argument of any view in Django) has a session (dict) attribute.

Let's create a simple sample to see how to create and save sessions. We have built a simple login system before (see Django form processing chapter and Django Cookies Handling chapter). Let us save the username in a cookie so, if not signed out, when accessing our login page you won’t see the login form. Basically, let's make our login system we used in Django Cookies handling more secure, by saving cookies server side.

For this, first lets change our login view to save our username cookie server side −

```
def login(request):
   username = 'not logged in'

   if request.method == 'POST':
      MyLoginForm = LoginForm(request.POST)

      if MyLoginForm.is_valid():
         username = MyLoginForm.cleaned_data['username']
         request.session['username'] = username
      else:
         MyLoginForm = LoginForm()

   return render(request, 'loggedin.html', {"username" : username}
```

Then let us create formView view for the login form, where we won't display the form if cookie is set −

```
def formView(request):
   if request.session.has_key('username'):
      username = request.session['username']
      return render(request, 'loggedin.html', {"username" : username})
   else:
      return render(request, 'login.html', {})
```

Now let us change the url.py file to change the url so it pairs with our new view −

```
from django.conf.urls import patterns, url
from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',
   url(r'^connection/','formView', name = 'loginform'),
   url(r'^login/', 'login', name = 'login'))
```

When accessing /myapp/connection, you will get to see the following page

SYSTEM ANALYSIS

EXISTING SYSTEM

Cyber security is becoming an ever increasing concern for most organizations and much research has been developed in this field over the last few years. Inside these organizations, the Security Operations Center (SOC) is the central nervous system that provides the necessary security against cyber threats. However, to be effective, the SOC requires timely and relevant threat intelligence to accurately and properly monitor, maintain, and secure an IT infrastructure. This leads security analysts to strive for threat awareness by collecting and reading various information feeds. However, if done manually, this results in a tedious and extensive task that may result in little knowledge being obtained given the large amounts of irrelevant information. Research has shown that Open Source Intelligence (OSINT) provides useful information to identify emerging cyber threats. OSINT is the collection, analysis, and use of data from openly available sources for intelligence purposes [21]. Examples of sources for OSINT are public blogs, dark and deep websites, forums, and social media. In such platforms, any person or entity on the Internet can publish, in real time, information in natural language related to cyber security, including incidents, new threats, and vulnerabilities. Among the OSINT sources for cyber threat intelligence, we can highlight the social media Twitter as one of the most representative [22]. Cyber security experts, system administrators, and hackers constantly use Twitter to discuss technical details about cyber attacks and share their experiences [4].

Utilization of OSINT to automatically identify cyber threats via social media, forums and other openly available sources using text analytics was proposed in different researches. However, most proposals focus on identifying important events related to cyber threats or vulnerabilities but do not propose identifying and profiling cyber threats. Amongst research, [13] proposes an early cyber threat warning system that mines online chatter from cyber actors on social media, security blogs, and dark web forums to identify words that signal potential cyber-attacks. The framework is comprised by woman in components: text mining and warning generation. The text mining phase consist son pre-processing the input data to identify potential threat names by discarding ‘known’’ terms and selecting repeating ‘unknown’’ among different sources as they potentially can be the name of a new or discovered cyber threat. The second component, warning generation, irresponsible for issuing alarms for unknown terms that meet some requirements, like appearing

twice in a given period of time. The approach presented in this research uses keyword filtering as the only strategy to identify cyber threat names, which may result in false positives as unknown words may appear in tweets or other content not necessarily related to cyber security. Additionally, this research does not profile the identified cyber threat. First, the proposed approach does not name the identified threat. Naming the threat is an important step to cyber threat intelligence as it may allow analysts to identify and mitigate campaigns based on the historic modus operandi employed by a given threat or group. Second, the proposed approach relies on an external component to classify tweets as related or not to cyber security as opposed to our approach that proposes a component to classify tweets using machine learning trained with the evolving knowledge from MITRE ATT&CK. Third, instead of using a keyword match to pre-filter threats and a fixed list of threat types, we present an approach to profile the identified cyber threat to spot in which phase of phases of the cyber kill chain the given threat operates in. This is important for a cyber threat analyst as he or she may employ the necessary mitigation steps depending on the threat profile.

DISADVANTAGES

An existing system never implemented Multi-Class machine learning (ML) algorithms – the next An existing system didn't implement¬steps in the pipeline. The following method PROCESSIDENTIFIED ANDCLASSIFIEDTHREATS.

PROPOSED SYSTEM:

The overall goal of this works to propose an approach to automatically identify and profile emerging cyber threats based on OSINT (Open Source Intelligence) in order to generate timely alerts to cyber security engineers. To achieve this goal, we propose a solution whose macro steps are listed below. 1) Continuously monitoring and collecting posts from prominent people and companies on Twitter to mine unknown terms related to cyber threats and malicious campaigns; 2) Using Natural Language Processing (NLP) and Machine Learning (ML) to identify those terms most likely to be threat names and discard those least likely; 3) Leveraging MITRE ATT&CK techniques' procedures examples to identify most likely tactic employed by the discovered threat; 4) Generating timely alerts for new or developing threats along with its characterization or goals associated with a risk rate based on how fast the threat is evolving since its identification.

ADVANTAGES

To conduct a cyber-attack, malicious actors typically have to 1) Identify vulnerabilities, 2) acquire the necessary tools and tradecraft to successfully exploit them, 3) choose a target and recruit participants, 4) Create or purchase the infrastructure needed, and 5) Plan and execute the attack. Other actors— system administrators, security analysts, and even victims— may discuss vulnerabilities or coordinate a response to attacks.

A. Time-Window Data Retrieval

   The first step of our pipeline consists in collecting Twitter messages from the Tweets Database which were posted within a given time range. Considering that our objective is to provide a continuous threat identification and alerting system, the time range will be a sliding time window considering the end time of the previous time range as the start time for the next time range. All the resulting Twitter messages will follow to the Unknown Word Selection, described in the next subsection.

B. Word Selection

   The objective of this component is to identify unknown words or terms appearing in collected Twitter messages as they, accordingly to further analysis, may represent the name of the identified emerging threat. The idea of identifying those unknown terms came from the analysis of cyber threats names, which usually receive very strange names - either given by their creators of by the cyber security experts who first spotted them. Wannacry, NotPetya, Cookthief, Emotet, lokibot, and 16shop are some examples of threat names. For the proposed architecture, a term is considered unknown if it passes through the Unknown Word Selection pipeline, which comprises the following procedures: Normalization, URL/E-mail/Author filtering, NLTK Word Tokenize, Correct Word Filtering, Stop-words and punctuation filtering, NER (Named Entity Recognition) filtering and, finally, Dictionary words filtering, as described below.

C. Normalization

   Considering that we are using Twitter messages posted by a variety of people and that Twitter itself imposes a length limit for the post message (nowadays 280 characters), it is very common to have terms for the same meaning written and shortened in different forms. For example, ’C2 server’, ’C&C server’ are written in different forms but, in the context

of cyber security, mean the same thing: ’command and control server’. Command and control servers are computers controlled by an attacker or cybercriminal which is used to send commands to systems compromised by malware and receive stolen data from a target network [39].

D. NLTK Word Tokenize

This step consists in splitting each collected tweet into words. The process of splitting sentences into words or just word tokenize is very commonly used by natural language processing solutions. To employ word tokenization into the proposed solution, we use Natural Language ToolKit (NLTK) Python module 11. The output of this step is, for each tweet, an array of its words or tokens. See in the example below how the content of a tweet is split into tokens: Tweet: "The RobbinHood ransomware is using a vulnerable legacy Gigabyte driver in order to get around antivirus protections".

E. NER Filtering

After applying the above filters in the pipeline, we noticed that, among unknown words, there were many organizations’ names like Microsoft, Google, and so on. Although they were really unknown words for the filters used until this point of the pipeline, we should eliminate them because, knowingly, they did not represent threat names. There is a field called Named Entity Recognition (NER) which is considered a fundamental task in a natural language processing (NLP) system. NER is a subproblem of information extraction and involves processing structured and unstructured data to identify expressions that refer to people, places, organizations, and companies [40]. Thus, applying NER to our pipeline would help reduce the number of companies being considered ’unknown words’.

F. TF-IDF

In this subsection, we describe the use of TF-IDF (Term Frequency-Inverse Document Frequency) to transform the text documents coming from both MITRE ATT&CK corpus and Twitter messages into a vectorized representation needed by both One-Class and Multi-Class machine learning (ML) algorithms - the next steps in the pipeline. Machine learning algorithms, more specifically the ones used in this work, operate on a numeric feature space, expecting input as a two-dimensional array where rows are instances and columns feature. To perform ML on the text we need to transform our documents into vector representations such that we can apply numeric machine learning in a process called

feature extraction or vectorization [43]. To perform the feature extraction, we employed a method called TF-IDF (Term Frequency-Inverse Document Frequency) [44]. TF-IDF is a numerical representation of the importance (weight) of a term t in a specific document d within a corpus of documents.

G. Classification process

- Logistic Regression Algorithm

  It is a SML model that is very commonly or widely used for the classification. Performance of LR model for linearly separable classes is very well and even easy to implement. Specially, in industry it is most commonly used. In general LR is used for binary classification as it is a linear model but using technique OvR it may be used for classification of multi class [9]. LR is applied on dataset by considering three different train test ratio (80:20, 60:40, and 70:30) to predict whether the bank currency is forge or genuine. For train test ratio 80:20 ROC curve and learning curves are drawn. Accuracy of LR is observed around 98% .

- Decision Tree Algorithm:

  It is a classification model having a structure like a tree. DT is incrementally developed by breaking down the data set into smaller subsets. DT results are having two types of nodes Decision nodes and leaf nodes. For an example consider a decision node i.e., Outlook and it have branches as Rainy, Overcast and Sunny representing values of the tested feature. Hours Played i.e., a leaf node it gives the decision on numerical targeted value. DT can handle both numerical as well as categorical data [8]. DT is applied on dataset by considering three different train test ratio (80:20, 60:40, and 70:30) to predict whether the bank currency is forge or genuine. For train test ratio 80:20 ROC curve and learning curves are drawn. Accuracy of DT has been observed around 99%.

- Random Forest Algorithm

  Random Forest is that the prevalent supervised technique. it's useful for mainly doing classification challenges and also regression challenges. RF is one amongst the classifiers which holds multiple decision trees in each subset of an assumed data set and computes the everyday value that enhances prediction accurateness for the dataset. The random forest doesn't depend upon decision trees. Instead, it gets a

prediction from every tree so forecasts the last result which is made upon polls of prevalence estimations. The more trees within the forest, the upper the accuracy and avoid over fitting problems. it's supported the ensemble technique concept, which mixes multiple classifiers to unravel a thorny problem and improves model performance.

H. Performance Evaluation

Once the model has been built the accuracy of the model has to be evaluated by the performance metrics in deep learning and machine learning methods. We have used F1-Score, precision, recall, confusion matrix and accuracy score.

Confusion Matrix: Confusion matrix is a very intuitive cross tab of actual class values and predicted class values. It contains the count of observations that fall in each category. Build a model → make class predictions on test data using the model → create a confusion matrix for each model.

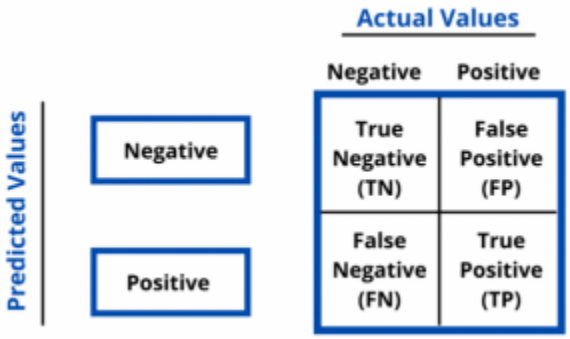


Figure: Confusion matrix

Accuracy: It is one of the important parameters to determine the accuracy of the classification problems. It defines how often the model predicts the correct output. It can be calculated as the ratio of the number of correct predictions made by the classifier to all number of predictions made by the classifiers. The formula is given below:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Precision: It can be defined as the number of correct outputs provided by the model or out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula:

$$\text{Precision} = \frac{TP}{TP+FP}$$

Recall: It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

$$\text{Recall} = \frac{TP}{TP+FN}$$

F1-measure: If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

$$\text{F1-Score} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

SYSTEM DESIGN

UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

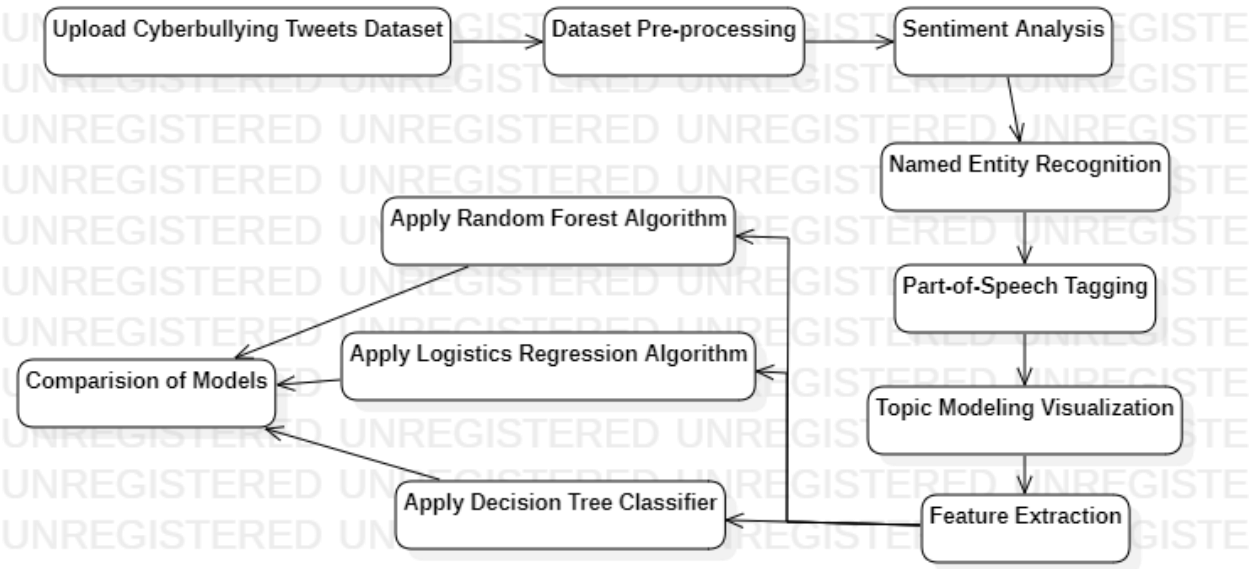
The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.

4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

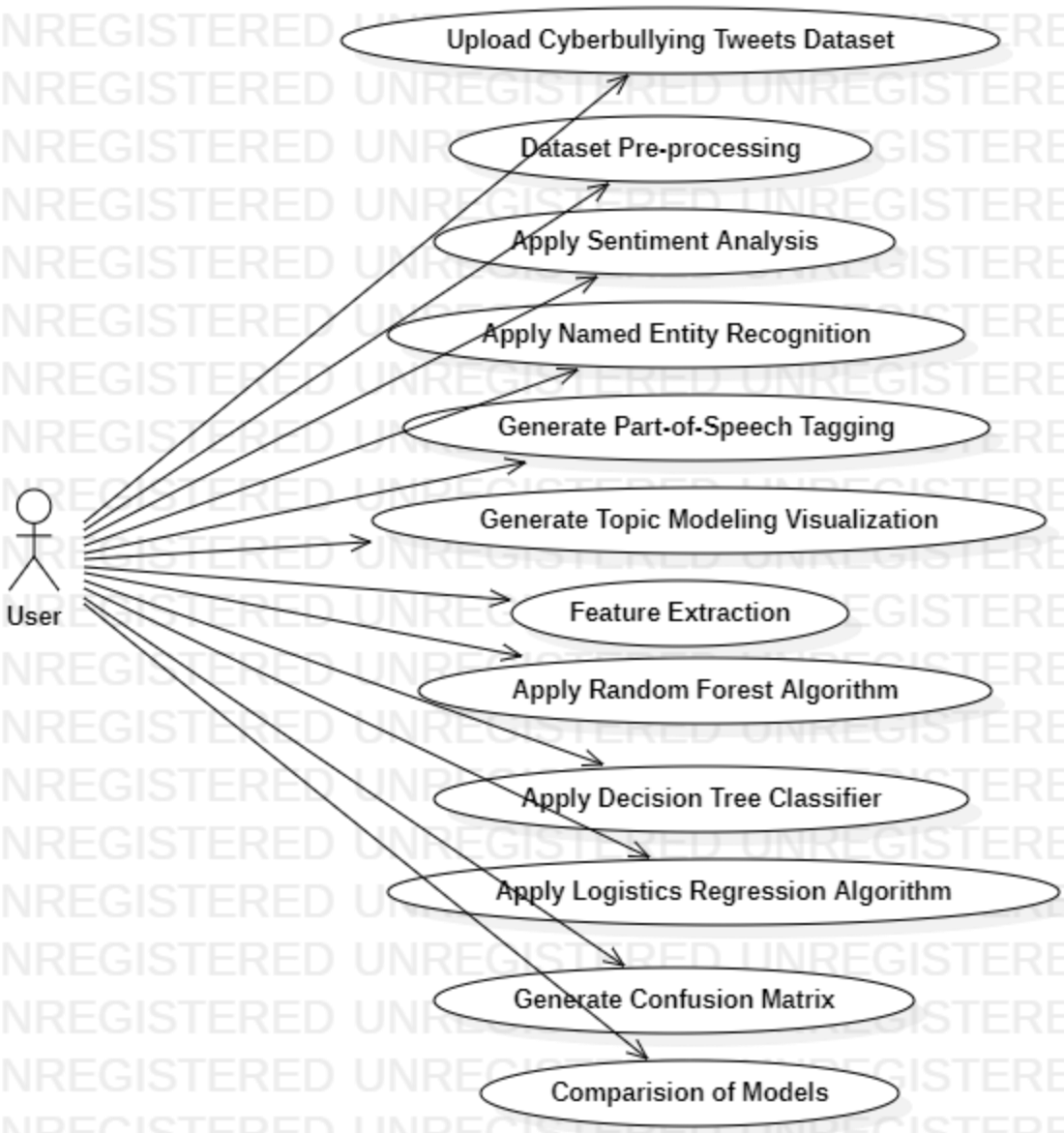
DATA FLOW DIAGRAM:

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.



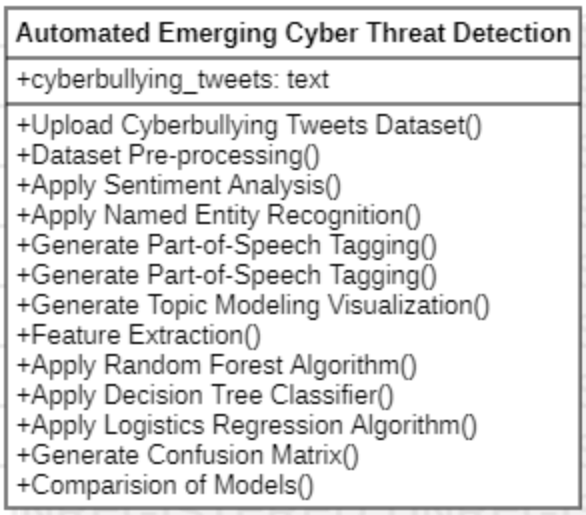
USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



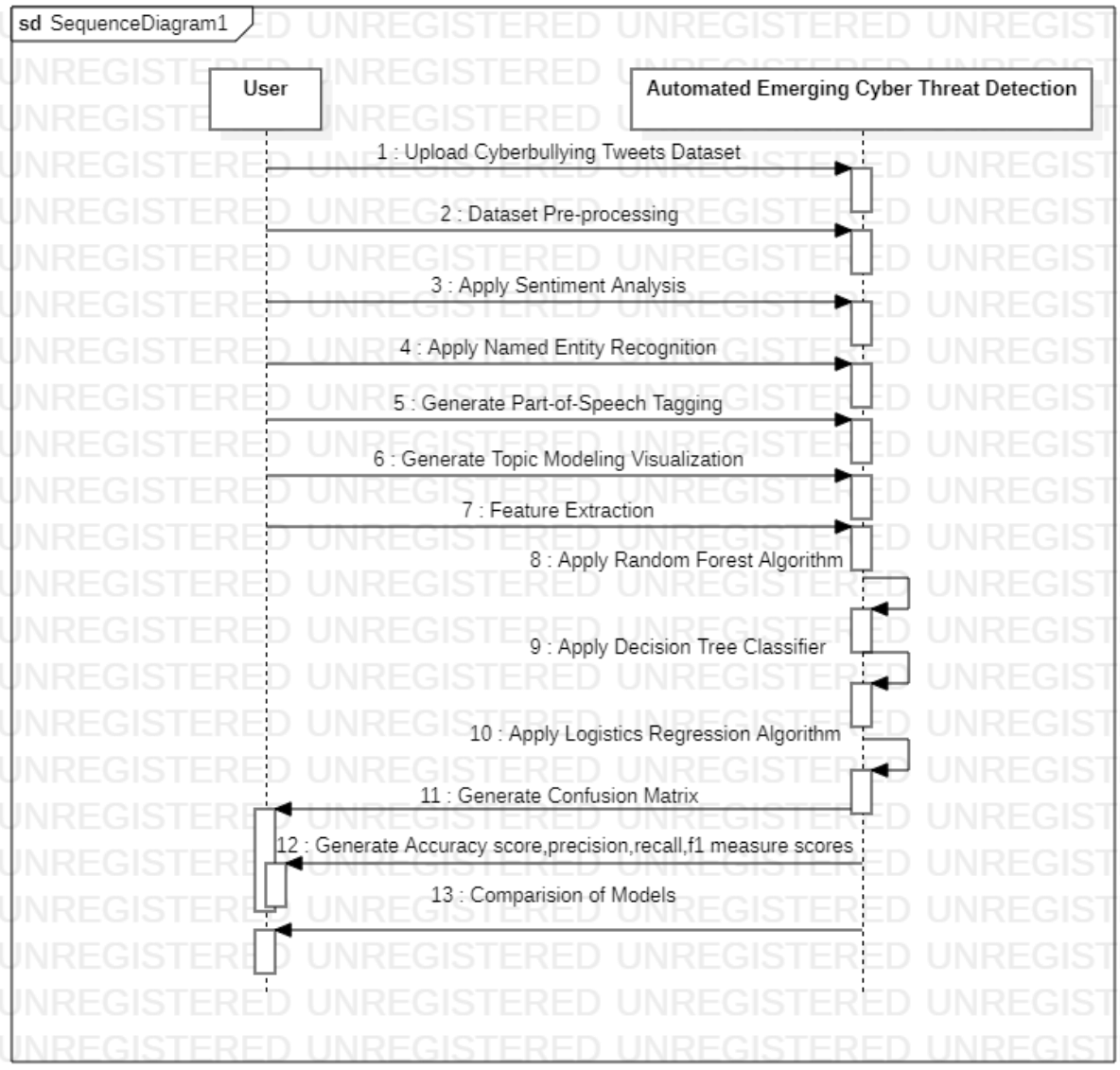
CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.



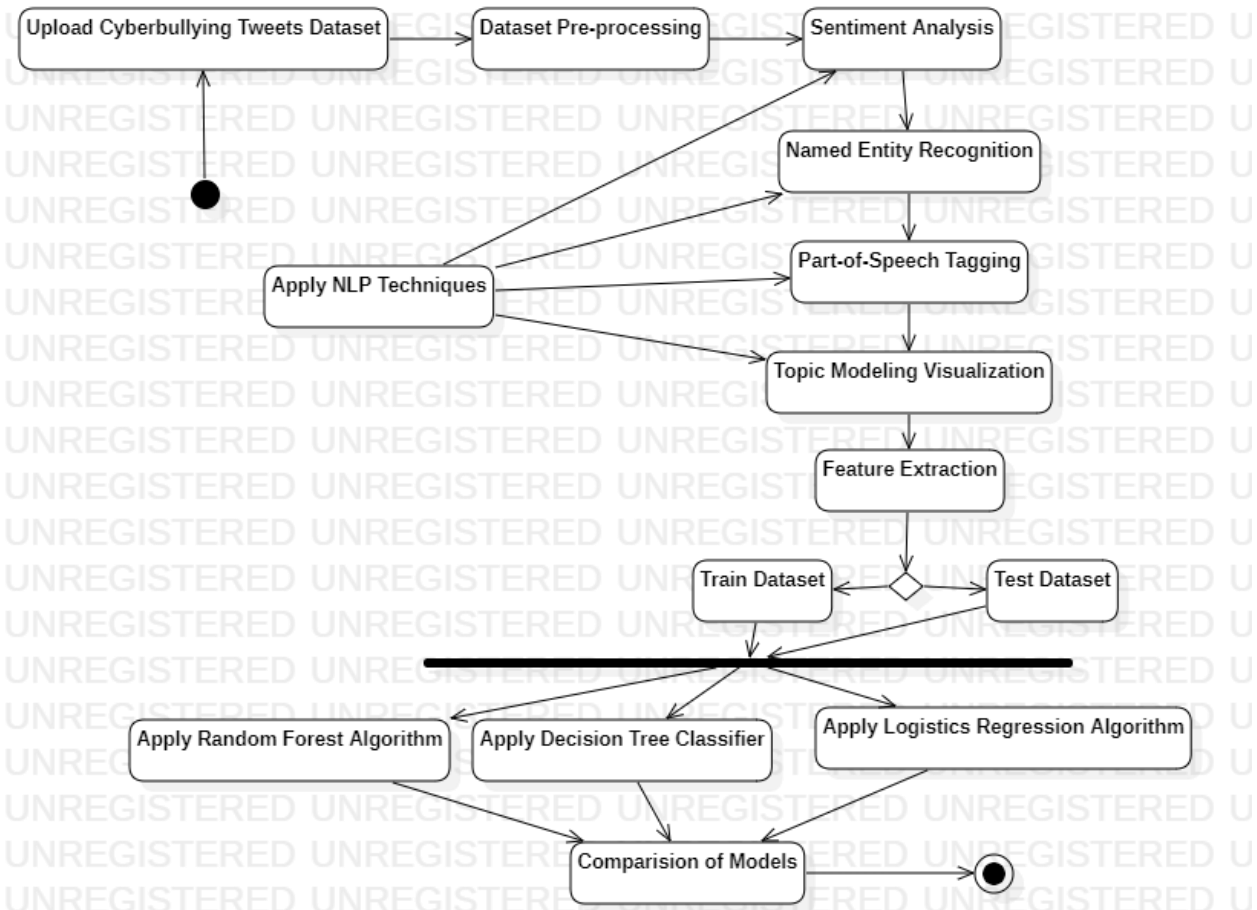
SEQUENCE DIAGRAM:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



INPUT AND OUTPUT DESIGN

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The

input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

1.Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3.When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

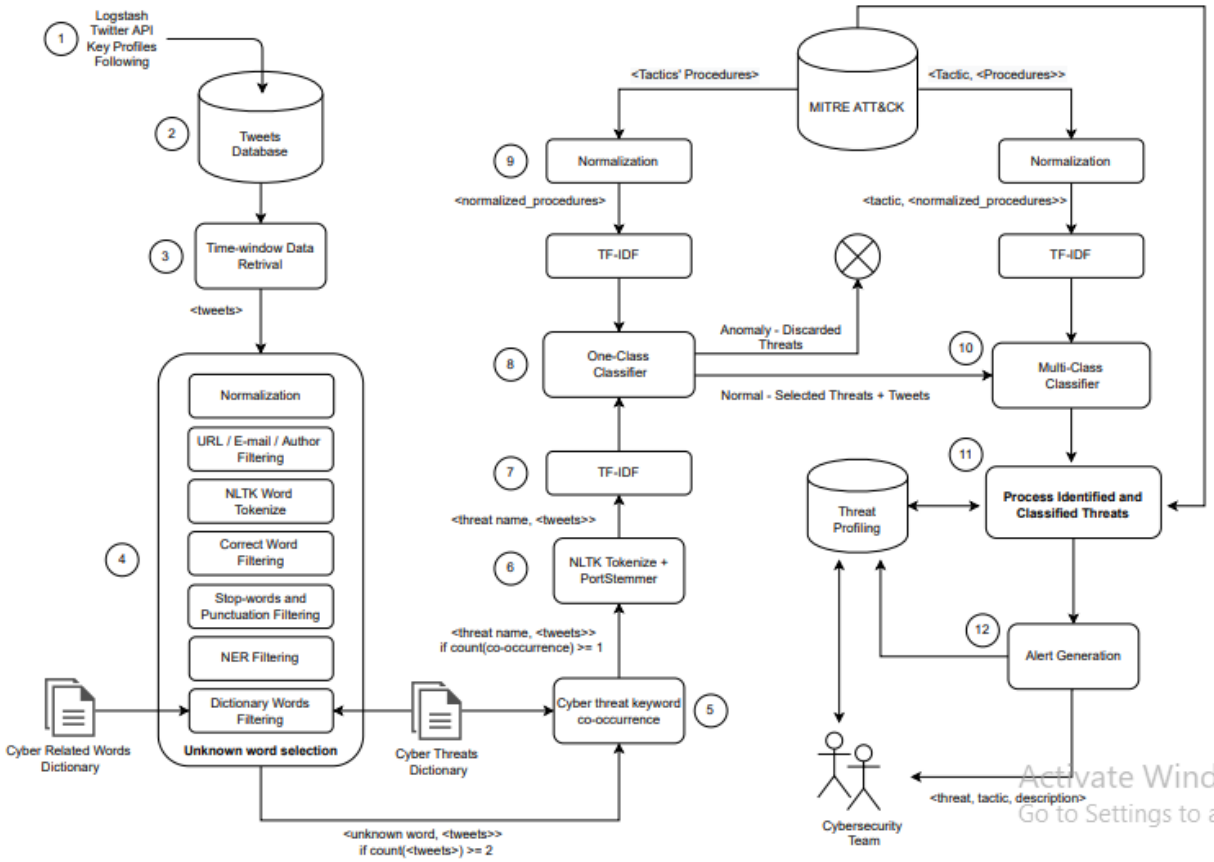
2.Select methods for presenting information.

3.Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

SYSTEM ARCHITECTURE



SYSTEM SPECIFICATION

HARDWARE REQUIREMENTS:

- System : Intel i3 to untill
- Hard Disk : 10 GB minimum.
- Monitor : 14/10/12/15’ Colour Monitor.
- Mouse : Optical Mouse.
- Ram : 4GB MINIMUM.

SOFTWARE REQUIREMENTS:

- Operating system : Windows 7/8/10/11.
- Coding Language : Python 3.7.
- Type of Application : GUI Application
- Front-End Technologies : Tkinter API
- Backend Technologies :matplotlib,pandas,nltk sckit-learn..etc
- IDE Tool : PyCharm community edition 2021

SYSTEM STUDY

FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are,

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must

not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

SYSTEM TEST

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

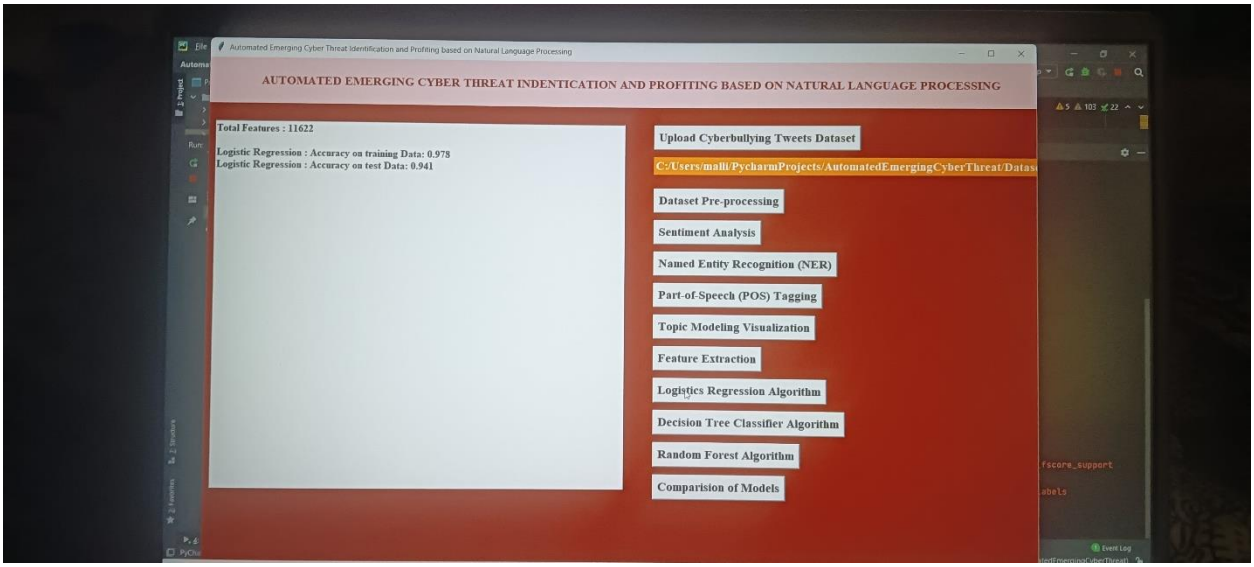
Test Results: All the test cases mentioned above passed successfully. No defects encountered.

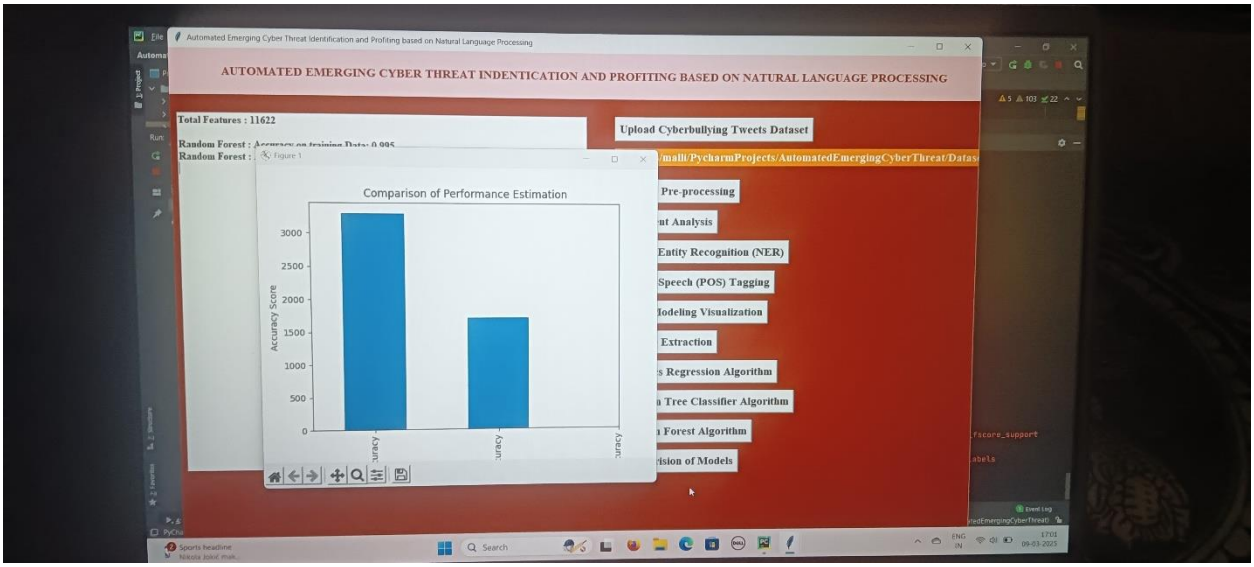
Acceptance Testing

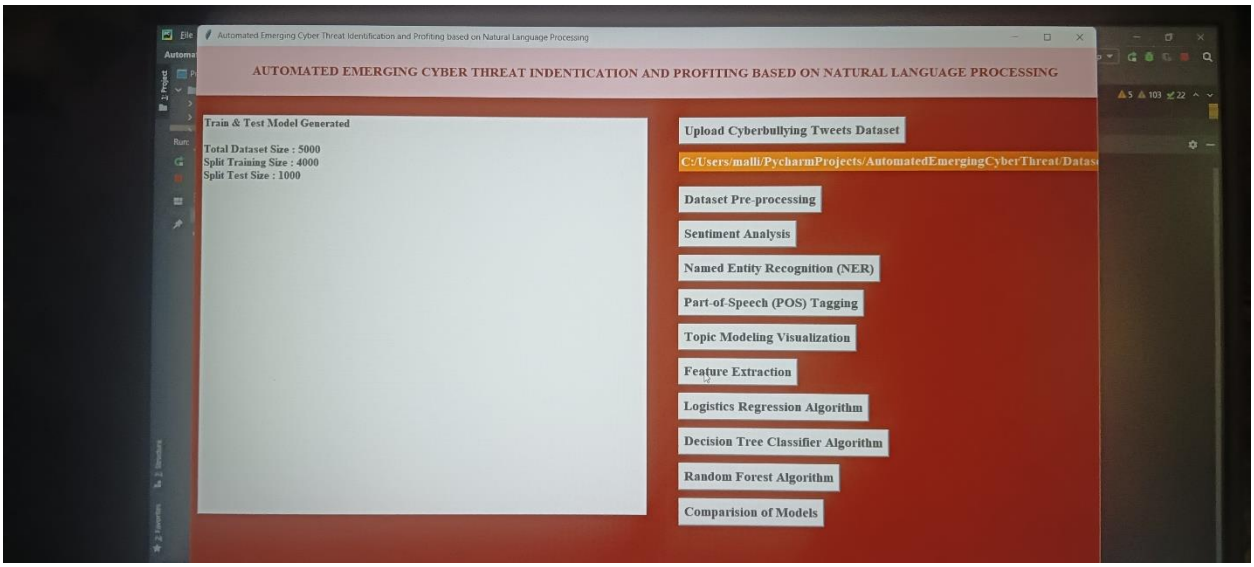
User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

RESULT


Automated Emerging Cyber Threat Identification and Profiting based on Natural Language Processing
AUTOMATED EMERGING CYBER THREAT INDENTICATION AND PROFITING BASED ON NATURAL LANGUAGE PROCESSING
Total Features : 11622
Logistic Regression : Accuracy on training Data: 0.978
Logistic Regression : Accuracy on test Data: 0.941
Upload Cyberbullying Tweets Dataset
C:/Users/malli/PycharmProjects/AutomatedEmergingCyberThreat/Datas
Dataset Pre-processing
Sentiment Analysis
Named Entity Recognition (NER)
Part-of-Speech (POS) Tagging
Topic Modeling Visualization
Feature Extraction
Logistics Regression Algorithm
Decision Tree Classifier Algorithm
Random Forest Algorithm
Comparision of Models


Automated Emerging Cyber Threat Identification and Profiting based on Natural Language Processing
AUTOMATED EMERGING CYBER THREAT INDENTICATION AND PROFITING BASED ON NATURAL LANGUAGE PROCESSING
Total Features : 11622
Upload Cyberbullying Tweets Dataset
Figure 1
Comparison of Performance Estimation
Accuracy Score
3000
2500
2000
1500
1000
500
0


Automated Emerging Cyber Threat Identification and Profiting based on Natural Language Processing
AUTOMATED EMERGING CYBER THREAT INDENTICATION AND PROFITING BASED ON NATURAL LANGUAGE PROCESSING
Train & Test Model Generated
Total Dataset Size : 5000
Split Training Size : 4000
Split Test Size : 1000
Upload Cyberbullying Tweets Dataset
C:/Users/malli/PycharmProjects/AutomatedEmergingCyberThreat/Datas
Dataset Pre-processing
Sentiment Analysis
Named Entity Recognition (NER)
Part-of-Speech (POS) Tagging
Topic Modeling Visualization
Feature Extraction
Logistics Regression Algorithm
Decision Tree Classifier Algorithm
Random Forest Algorithm
Comparision of Models

Further Enhancement

In future work, we consider it important to advance in tweets selection stages (Unknown Words and One-class), to improve the false positives rate and in the profiling stage, to reach higher accuracy in determining the technique associated with the identified threat. We are working on this way by experimenting with a different NLP approach using the part of speech (POS) algorithm implementation from Spacy29 Python library. The object is to identify the root verb, the subject, and the object of the phrases to select tweets where the action described (the root verb) is referencing the unknown word (the subject).

CONCLUSION

vulnerabilities and threats appearing at any time, keeping up to date on them is a challenging but important task for analysts. Even following the best practices and applying the best controls, a new threat may bring an unusual way to subvert the defenses requiring a quick response. This way, timely information about emerging cyber threats becomes paramount to a complete cybersecurity system. This research proposes an automated cyber threat identification and profiling based on the natural language processing of Twitter messages. The objective is exactly to cooperate with the hard work of following the rich source of information that is Twitter to extract valuable information about emerging threats in a timely manner. This work differentiates itself from others by going a step beyond identifying the threat. It seeks to identify the goals of the threat by mapping the text from tweets to the procedures conducted by real threats described in MITRE ATT&CK knowledge base. Taking advantage of this evolving and collaborative knowledge base to train machine learning algorithms is a way to leverage the efforts of cyber security community to automatically profile identified cyber threats in terms of their intents. To put in test our approach, in addition to the research experiment, we implemented the proposed pipeline and run it for 70 days generating online alerts for the Threat Intelligence Team of a big financial institution in Brazil. During this period, at least three threats made the team take preventive actions, such as the PetitPotam case, described in section V. Our system alerted the team making them aware of PetitPotam 17 days before the official patch was published by Microsoft. Within this period, the defense team was able to implement mitigations avoiding potential exploits and, consequently, incidents. Our experiments showed that the profiling stage reached an F1 score of 77% in correctly profiling discovered threats among 14 different tactics and the percentage of false alerts of 15%.

REFERENCES

[1] Ba Dung Le, Guanhua Wang, Mehwish Nasim, and Ali Babar. Gathering cyber threat intelligence from twitter using novelty classification. arXiv preprint arXiv:1907.01755, 2019.

[2] Gartner Research. Definition: Threat intelligence, 2013.

[3] Robert David Steele. Open source intelligence: What is it? why is it important to the military? American Intelligence Journal, pages 35–41, 1996.

[4] Carl Sabottke, Octavian Suciu, and Tudor Dumitras, . Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In 24th {USENIX} Security Symposium ({USENIX} Security 15), pages 1041–1056, 2015.

[5] Anna Sapienza, Alessandro Bessi, Saranya Damodaran, Paulo Shakarian, Kristina Lerman, and Emilio Ferrara. Early warnings of cyber threats in online discussions. In 2017 IEEE International Conference on Data Mining Workshops (ICDMW), pages 667–674. IEEE, 2017.

[6] Eric Nunes, Ahmad Diab, Andrew Gunn, Ericsson Marin, Vineet Mishra, Vivin Paliath, John Robertson, Jana Shakarian, Amanda Thart, and Paulo Shakarian. Darknet and deepnet mining for proactive cybersecurity threat intelligence. In 2016 IEEE Conference on Intelligence and Security Informatics (ISI), pages 7–12. IEEE, 2016.

[7] Sudip Mittal, Prajit Kumar Das, Varish Mulwad, Anupam Joshi, and Tim Finin. Cybertwitter: Using twitter to generate alerts for cybersecurity threats and vulnerabilities. In 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pages 860–867. IEEE, 2016.

[8] Abbas Attarwala, Stanko Dimitrov, and Amer Obeidi. How efficient is twitter: Predicting 2012 us presidential elections using support vector machine via twitter and comparing against iowa electronic markets. In 2017 Intelligent Systems Conference (IntelliSys), pages 646–652. IEEE, 2017.

[9] Nuno Dionísio, Fernando Alves, Pedro M Ferreira, and Alysson Bessani. Towards end-to-end cyberthreat detection from twitter using multi-task learning. In 2020 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2020.

[10] Onook Oh, Manish Agrawal, and H Raghav Rao. Information control and terrorism: Tracking the mumbai terrorist attack through twitter. Information Systems Frontiers, 13(1):33–43, 2011.

[11] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In Proceedings of the 19th international conference on World wide web, pages 851–860, 2010.

[12] Bertrand De Longueville, Robin S Smith, and Gianluca Luraschi. " omg, from here, i can see the flames!" a use case of mining location based social networks to acquire spatio-temporal data on forest fires. In Proceedings of the 2009 international workshop on location based social networks, pages 73–80, 2009.

[13] Anna Sapienza, Sindhu Kiranmai Ernala, Alessandro Bessi, Kristina Lerman, and Emilio Ferrara. Discover: Mining online chatter for emerging cyber threats. In Companion Proceedings of the The Web Conference 2018, pages 983–990, 2018.

[14] Rupinder Paul Khandpur, Taoran Ji, Steve Jan, Gang Wang, Chang-Tien Lu, and Naren Ramakrishnan. Crowdsourcing cybersecurity: Cyber attack detection using social media. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pages 1049– 1057, 2017.

[15] Quentin Le Sceller, ElMouatez Billah Karbab, Mourad Debbabi, and Farkhund Iqbal. Sonar: Automatic detection of cyber security events over the twitter stream. In Proceedings of the 12th International Conference on Availability, Reliability and Security, pages 1–11, 2017.

[16] Kuo-Chan Lee, Chih-Hung Hsieh, Li-Jia Wei, Ching-Hao Mao, JyunHan Dai, and Yu-Ting Kuang. Sec-buzzer: cyber security emerging topic mining with open threat intelligence retrieval and timeline event annotation. Soft Computing, 21(11):2883–2896, 2017.

[17] Alan Ritter, Evan Wright, William Casey, and Tom Mitchell. Weakly supervised extraction of computer security events from twitter. In Proceedings of the 24th International Conference on World Wide Web, pages 896–905, 2015.

[18] Andrei Queiroz, Brian Keegan, and Fredrick Mtenzi. Predicting software vulnerability using security discussion in social media. In European Conference on Cyber Warfare and Security, pages 628–634. Academic Conferences International Limited, 2017.

[19] Avishek Bose, Vahid Behzadan, Carlos Aguirre, and William H Hsu. A novel approach for detection and ranking of trendy and emerging cyber threat events in twitter streams. In 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pages 871–878. IEEE, 2019.

[20] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. In Technical report. The MITRE Corporation, 2018.

[21] Bert-Jaap Koops, Jaap-Henk Hoepman, and Ronald Leenes. Open-source intelligence and privacy by design. Computer Law & Security Review, 29(6):676–688, 2013.

[22] Rodrigo Campiolo, Luiz Arthur F Santos, Daniel Macêdo Batista, and Marco Aurélio Gerosa. Evaluating the utilization of twitter messages as a source of security alerts. In Proceedings of the 28th Annual ACM Symposium on Applied Computing, pages 942–943, 2013.

[23] Nuno Dionísio, Fernando Alves, Pedro M Ferreira, and Alysson Bessani. Cyberthreat detection from twitter using deep neural networks. In 2019 International Joint Conference on Neural Networks, pages 1–8. IEEE, 2019.

[24] Amirreza Niakanlahiji, Jinpeng Wei, and Bei-Tseng Chu. A natural language processing based trend analysis of advanced persistent threat techniques. In 2018 IEEE International Conference on Big Data (Big Data), pages 2995–3000. IEEE, 2018.

[25] Gbadebo Ayoade, Swarup Chandra, Latifur Khan, Kevin Hamlen, and Bhavani Thuraisingham. Automated threat report classification over multisource data. In 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), pages 236–245. IEEE, 2018.

[26] Vahid Behzadan, Carlos Aguirre, Avishek Bose, and William Hsu. Corpus and deep learning classifier for collection of cyber threat indicators in twitter stream. In 2018 IEEE International Conference on Big Data (Big Data), pages 5002–5007, 2018.

[27] Ashok Deb, Kristina Lerman, and Emilio Ferrara. Predicting cyber-events by leveraging hacker sentiment. Information, 9(11):280, 2018.

[28] Ryan Williams, Sagar Samtani, Mark Patton, and Hsinchun Chen. Incremental hacker forum exploit collection and classification for proactive cyber threat intelligence: An exploratory study. In 2018 IEEE International Conference on Intelligence and Security Informatics (ISI), pages 94–99. IEEE, 2018.

[29] A Rakhlin. Convolutional neural networks for sentence classification. GitHub, 2016.