Title: Adaptive Hierarchical Cyber Attack Detection and Localization in Active Distribution Systems

Instructor: Dr.Feng George Yu

Assistant Professor: Robert A. Gilliland PhD

ABSTRACT

Development of a cyber security strategy for the active distribution systems is challenging due to the inclusion of distributed renewable energy generations. This paper proposes an adaptive hierarchical cyber attack detection and localization framework for distributed active distribution systems via analyzing electrical waveforms. Cyber attack detection is based on a sequential deep learning model, via which even minor cyber attacks can be identified. The two-stage cyber attack localization algorithm first estimates the cyber attack sub-region, and then localize the specified cyber attack within the estimated subregion. We propose a modified spectral clustering-based network partitioning method for the hierarchical cyber attack ‘coarse ‘localization. Next, to further narrow down the cyber-attack location, a normalized impact score based on waveform statistical metrics is proposed to obtain a ‘fine’ cyber-attack location by characterizing different waveform properties. Finally, compared with classical and state-of-art methods, a comprehensive quantitative evaluation with two case studies shows promising estimation results of the proposed framework.

CHAPTER -1
INTRODUCTION

CYBER attack localization is important to protect smart distribution grids, but also a challenging task because of the inherent distributed energy resources (DER) and topology complexities [1], [2]. Raw electrical waveforms, signals of electrical networks, together with those in cyber networks provide great potentials in cyber attack detection [3]. For example, devices in power networks must leave clues of their operational status and health (including faults or attacks) information in the raw electrical waveform signals: a cyber-device in fault or under attack will cause unusual energy consumption pattern in power networks [4]; a power electronics or electric machine in fault or under attack may cause unusual harmonics or energy profile in electrical networks [5].

By analyzing the electrical waveform signals and their root cause, waveform analytics can present utilities with a complete picture of the health and status of their system, both during outages and normal operating conditions. It could also provide a variety of operational benefits to system operators, asset management personnel, and repair crew. Electronic sensors placed on power grids and distribution systems can either measure the electricity properties, such as phasor measurement unit (PMU) sensors [6], [7] or directly record the raw electrical waveform using waveform measurement unit (WMU) [8]– [12], depending on the needed fidelity of monitoring applications. Thanks to developed network connectivity, the streaming monitoring data flow can be obtained and analyzed online and in real-time [13].

The network of the waveform sensors form an Internet of Things (IoT) system [4], [14], where the waveform sensors are viewed as networked IoT sensing devices. Therefore, we can potentially use the information embedded in electrical signals to enable security monitoring, diagnosis, and prognosis in the power networks. The possibility may be well beyond what we can imagine now. It broadly applies to many cyber-physical systems (CPS) and applications, such as power distribution networks, multi-stage manufacturing systems, electric vehicles, and so on [15]–[17]. Cyber attacks towards connected IOT devices trigger anomalies in system statistics, energy consumption, as well as electrical waveforms [4], [14], [18], [19]. Thus, recorded waveform which carries high fidelity current and voltage information should be adequate for cyber attack characterization. Furthermore, [developed. However, N based algorithms typically require a large amount of training data to capture the sophisticated features, which cannot be fully simulated or acquired from real applications. Thus, combining the rule-based signal processing methods and machine learning methods could lead to a solution tackling the challenging problem using an affordable data size.

There have been numerous works targeting the event and cyber attack localization problem [1], [2], [27]. Dynamic data analytics based localization is always a major branch for the distribution networks [1], DC microgrid [2], islanded microgrid [27]. This paper proposes a new adaptive hierarchical framework for efficient and accurate cyber attack detection and localization by taking

advantage of the electrical waveforms (Fig. 1). The proposed approach has a hierarchical architecture that divides the whole network into sub-groups and then locates the cyber attack within one local cluster. Based on a modified unsupervised clustering and an deep learning based anomaly detection method, cyber attacks in the active distribution systems can be adaptively detected and located. The performance of the proposed approach has been tested by multiple cyber attack scenarios in two representative case studies.

Our contributions are summarized as follows:

_ We propose an adaptive hierarchical cyber attack detection and localization framework for active distribution systems with DERs using the electrical waveform;
_ High fidelity models of DER and cyber attacks are built to analyze the impacts of cyber attacks towards the distribution networks;
_ Extensive experiments are utilized to evaluate the proposed approach performances with quantitative analytics;

The remainder of this paper is organized as follows. In Section II, the cyber attack model of active distribution systems is discussed. In Section III, we describe the proposed approaches with the details of each key component, which are cyber attack detection, network partition and cyber attack localization. Experiments and evaluations can be found in Section IV. In the end, a conclusion is drawn in Section V.

CHAPTER-02

Literature Survey

Cybersecurity risks have increased as a result of the growing integration of distributed energy resources (DERs) in active distribution systems. With the introduction of intelligent devices and dynamic topologies, the formerly passive traditional distribution networks are increasingly more vulnerable [1].

Waveform-Based Cybersecurity: New research shows that unprocessed electrical waveform signals contain a wealth of data that reflects the state of devices' operations. Using sophisticated signal analysis techniques, anomalies in these signals—caused by physical failures or cyberattacks—can be identified [2], [3]. High-frequency monitoring is made possible by sensors such as Waveform Measurement Units (WMUs) and Phasor Measurement Units (PMUs), which make real-time attack detection possible [4].

Data-Driven Detection Methods: For the detection of cyberattacks, rule-based algorithms [5] and signal-property-based approaches [6] have been investigated. These techniques, however, frequently aren't flexible enough to handle unexpected attacks. Complex abnormalities have been successfully captured by machine learning (ML) approaches such as convolutional neural networks (CNNs) and autoencoders [7]. However, the need for big training datasets and the problem of simulating every potential attack scenario continue to be major obstacles.

Cyber Attack Localization Techniques: Resilient systems depend on effective localization techniques. For quicker localization, conventional methods that use K-means clustering and spectral clustering [8] divide the distribution network into smaller sub-networks. However, particularly in systems with high DER penetration, these approaches frequently fall short in dynamically adapting to changes brought about by cyberattacks.

False Data Injection (FDI) Attacks: Distributed control systems are seriously threatened by FDI attacks, in which malevolent actors alter sensor readings. Such attacks can result in equipment damage, false alarms, or even system-wide blackouts, according to research [9]. By regularly changing system parameters, Moving Target Defense (MTD) techniques have been put forth to improve system resilience [10].

The shortcomings of current techniques emphasize the need for hierarchical, adaptive solutions that may use electrical waveform data to precisely detect and locate cyberattacks in real time.

CHAPTER-3

SYSTEM ANALYSIS AND DESIGN

EXISTING SYSTEM

Cyber and physical attacks threaten the security of distribution power grids. The emerging renewable energy sources such as photovoltaics (PVs) introduce new potential vulnerabilities. Based on the electric waveform data measured by waveform sensors in the distribution power networks, in this article, an existing system develops a novel high-dimensional data-driven cyber physical attack detection and identification (HCADI) approach.

First, we analyze the cyber and physical attack impacts (including cyber attacks on the solar inverter causing unusual harmonics) on electric waveforms in the distribution power grids. Then, we construct a highdimensional streaming data feature matrix based on signal analysis of multiple sensors in the network.

Next, we propose a novel mechanism including leverage score-based attack detection and binary matrix factorization-based attack diagnosis. By leveraging the data structure and binary coding, our HCADI approach does not need the training stage for both detection and the root cause diagnosis, which is needed for machine learning/deep learning-based methods. To the best of our knowledge, it is the first attempt to use raw electrical waveform data to detect and identify the power electronics cyber/physical attacks in distribution power grids with PVs.

Disadvantages

The system is not implemented Network Partition based on Modified Spectral Clustering.

The system is not implemented Cyber Attack Localization within Sub-regions.

Proposed System

The system proposes an adaptive hierarchical cyber attack detection and localization framework for active distribution systems with DERs using the electrical waveform;

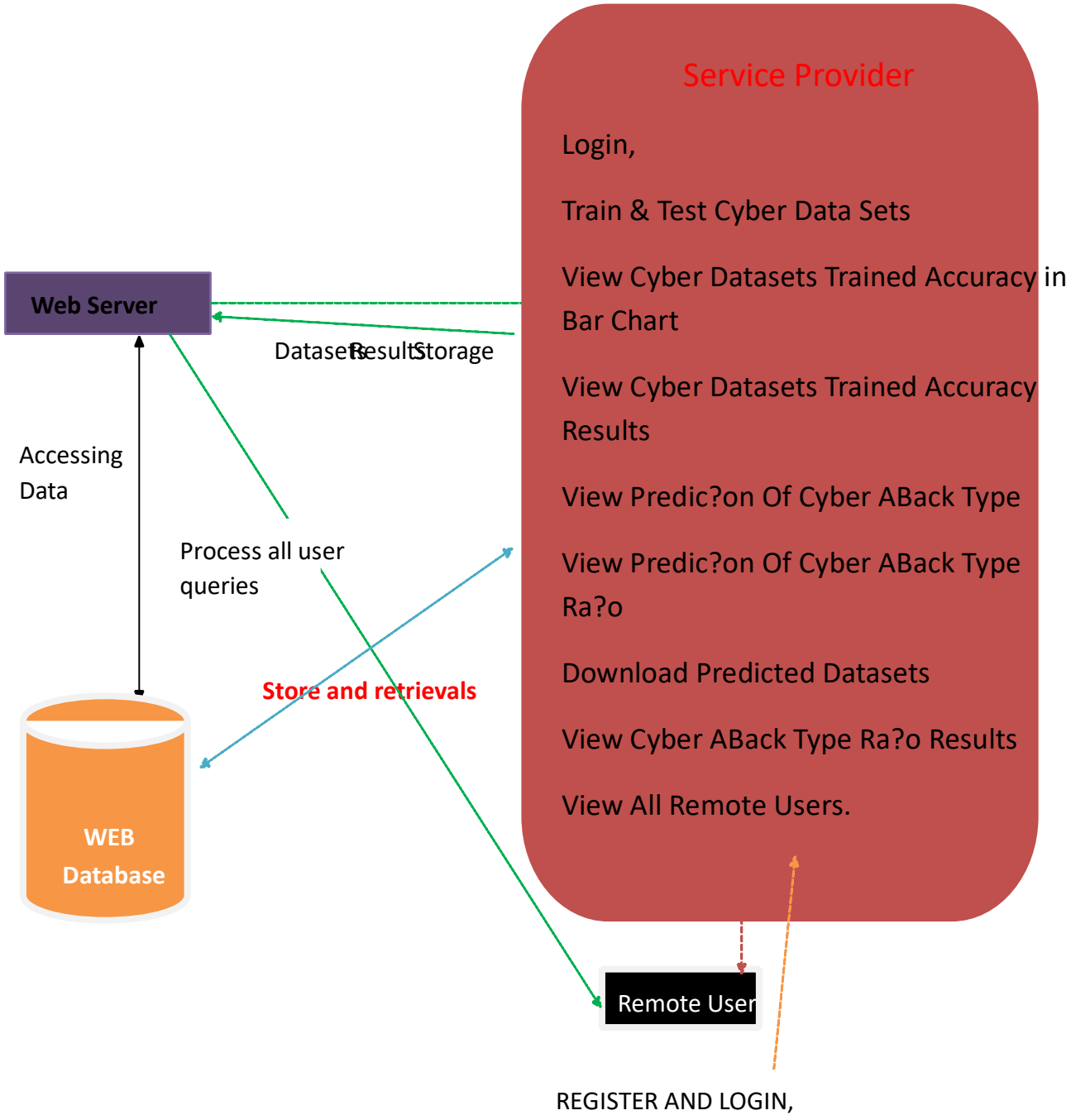
High fidelity models of DER and cyber attacks are built to analyze the impacts of cyber attacks towards the distribution networks;

Extensive experiments are utilized to evaluate the proposed approach performances with quantitative analytics.

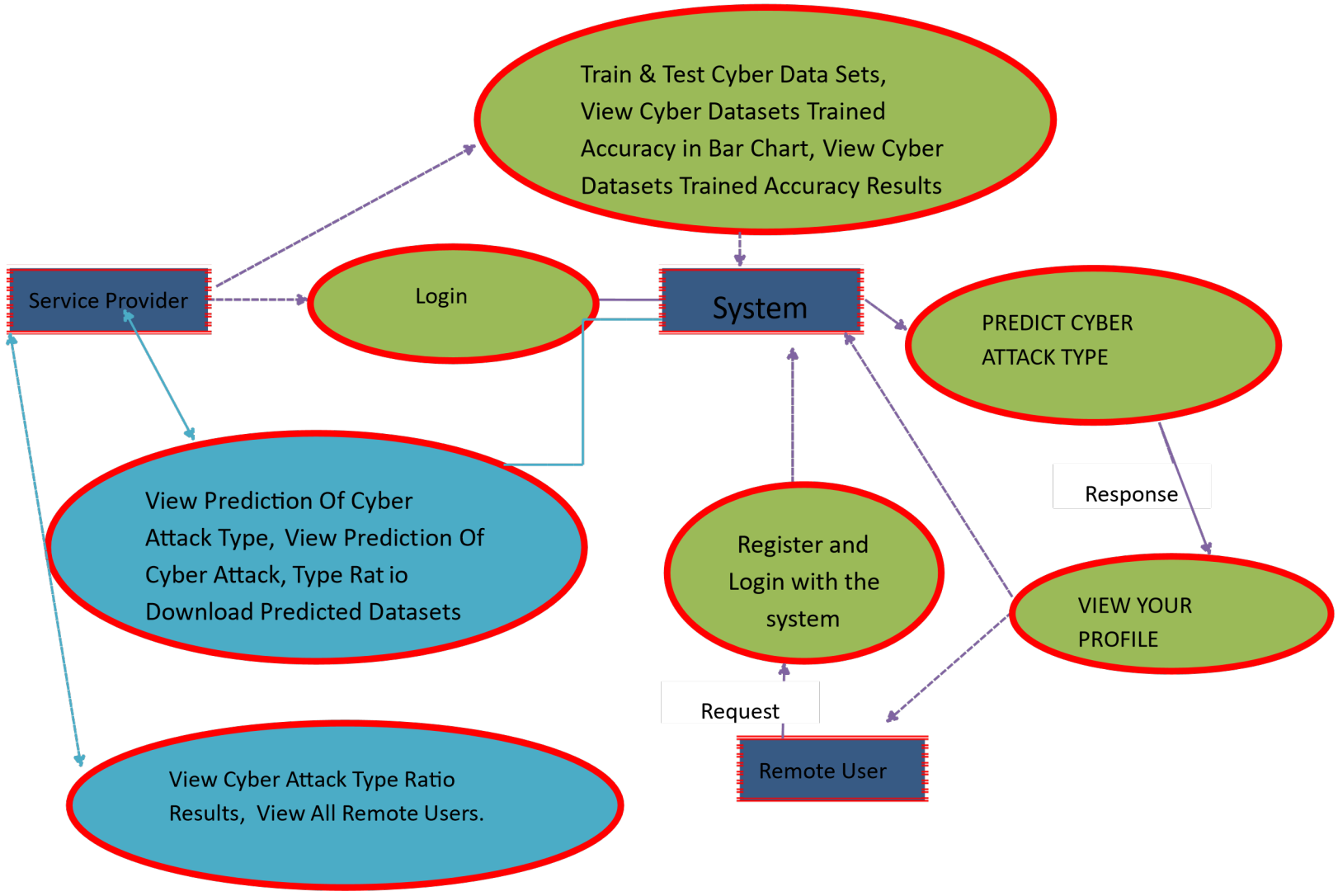
Advantages

In the proposed system, the cyber attack can be detected based on the deviation of the monitoring metrics from steady-state, which, in our study, is an anomaly detection problem.

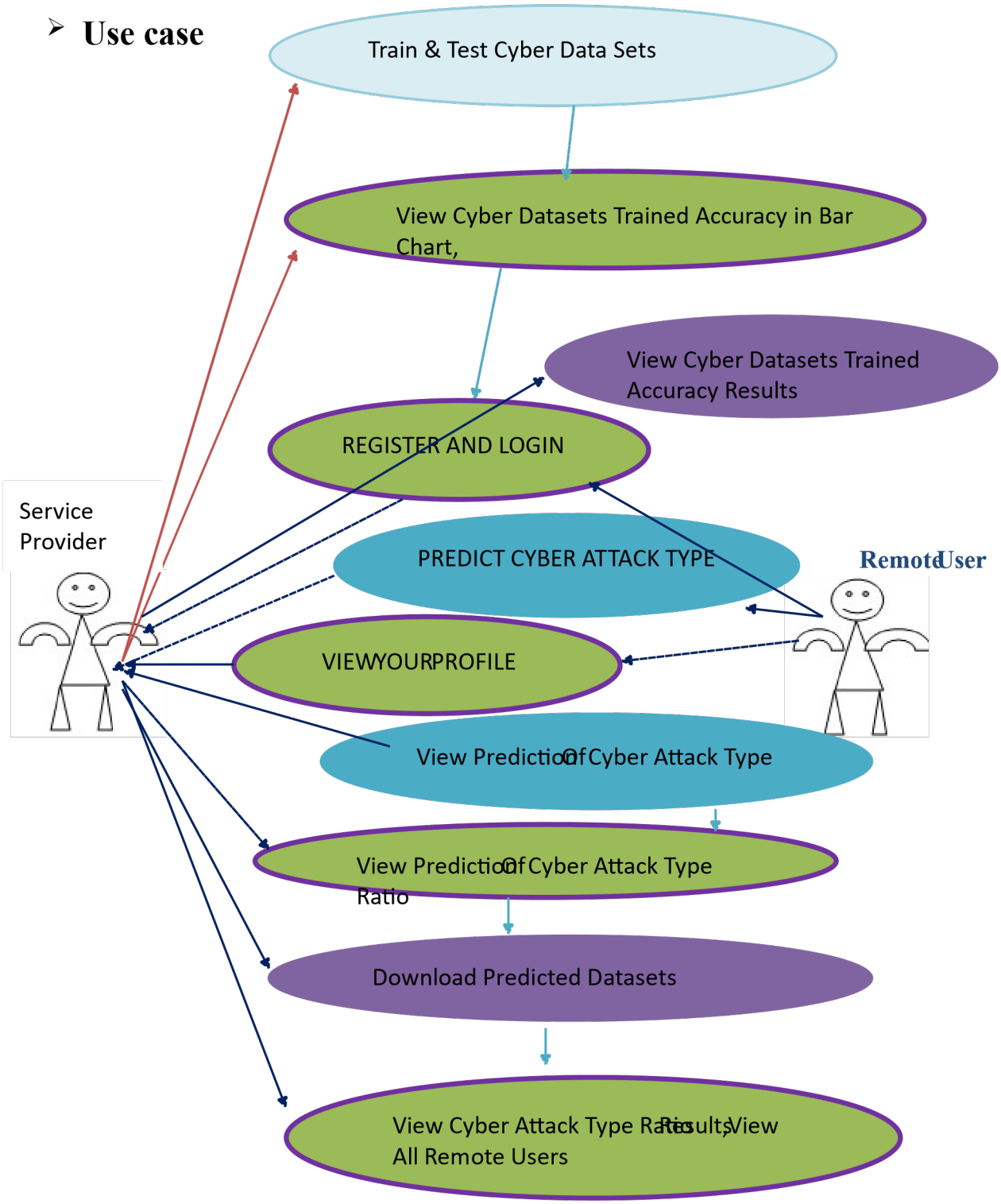
SYSTEM ARCHITECTURE



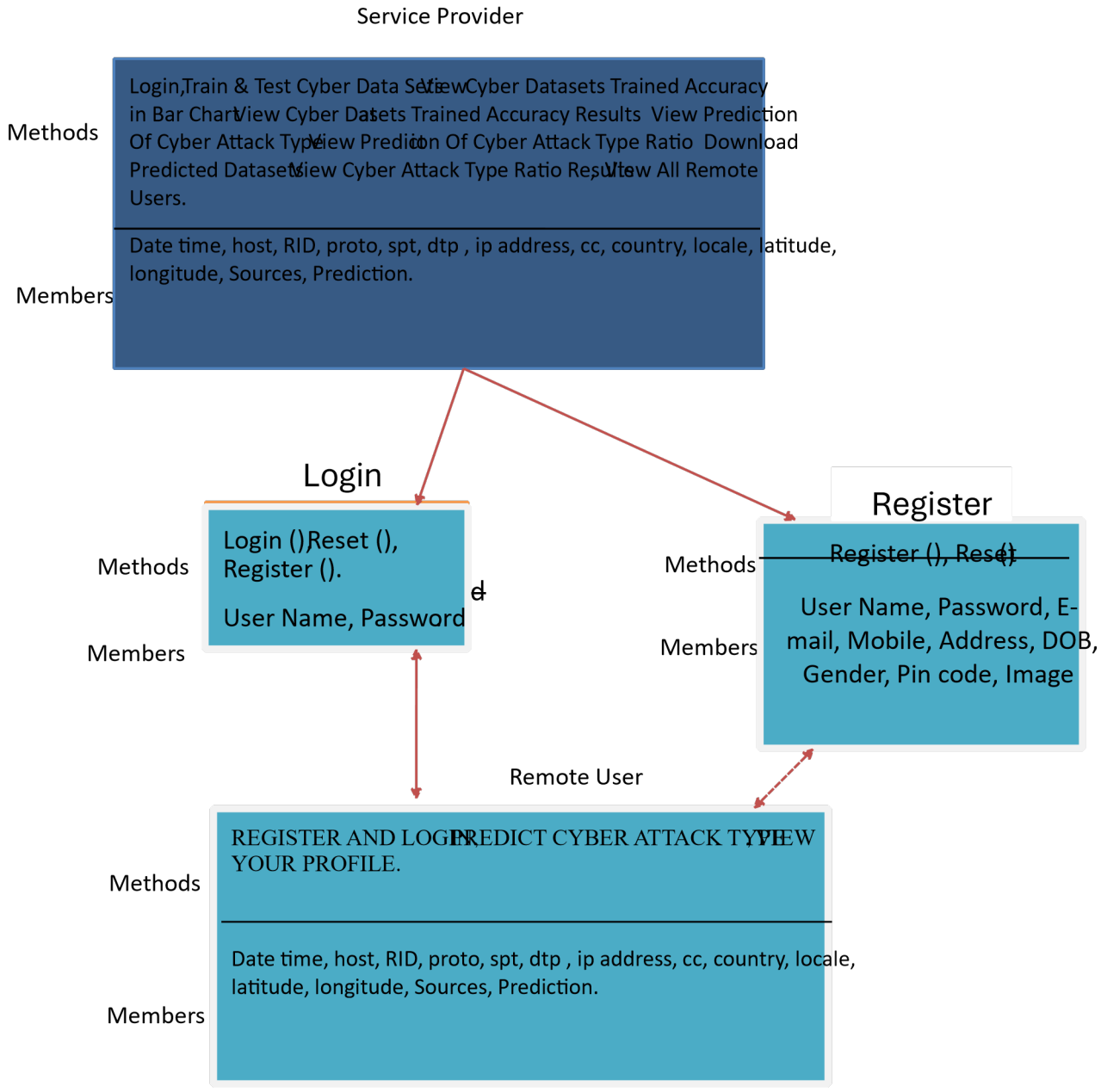
Data flow


Train & Test Cyber Data Sets, View Cyber Datasets Trained Accuracy in Bar Chart, View Cyber Datasets Trained Accuracy Results
Service Provider
Login
System
PREDICT CYBER ATTACK TYPE
View Prediction Of Cyber Attack Type, View Prediction Of Cyber Attack, Type Rat io Download Predicted Datasets
Response
Register and Login with the system
VIEW YOUR PROFILE
Request
Remote User
View Cyber Attack Type Ratio Results, View All Remote Users.

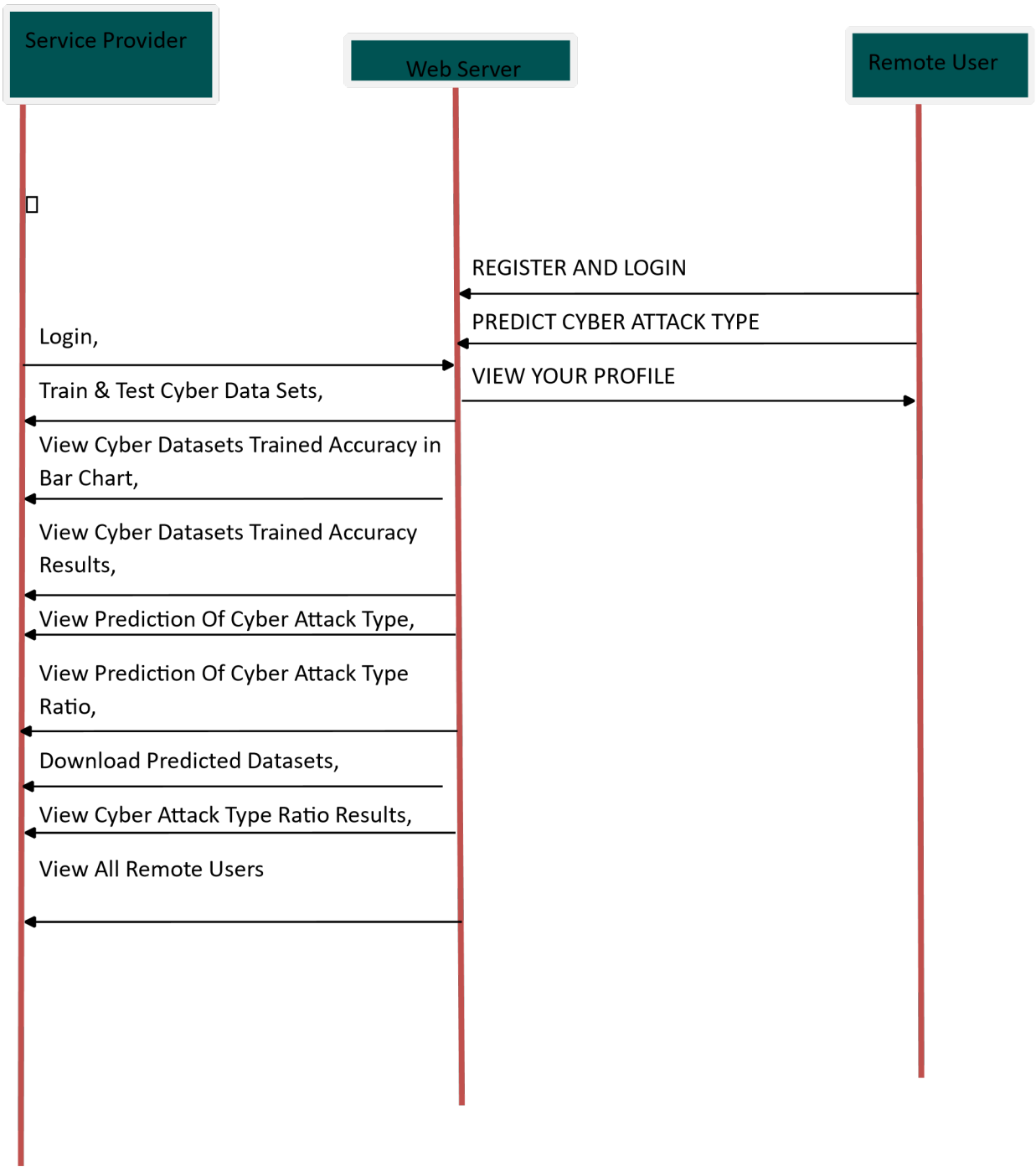
➢ **Use case**


Train & Test Cyber Data Sets
View Cyber Datasets Trained Accuracy in Bar Chart,
View Cyber Datasets Trained Accuracy Results
REGISTER AND LOGIN
Service Provider
PREDICT CYBER ATTACK TYPE
Remote User
VIEWYOURPROFILE
View Prediction Of Cyber Attack Type
View Prediction Of Cyber Attack Type Ratio
Download Predicted Datasets
View Cyber Attack Type Ratio Results, View All Remote Users

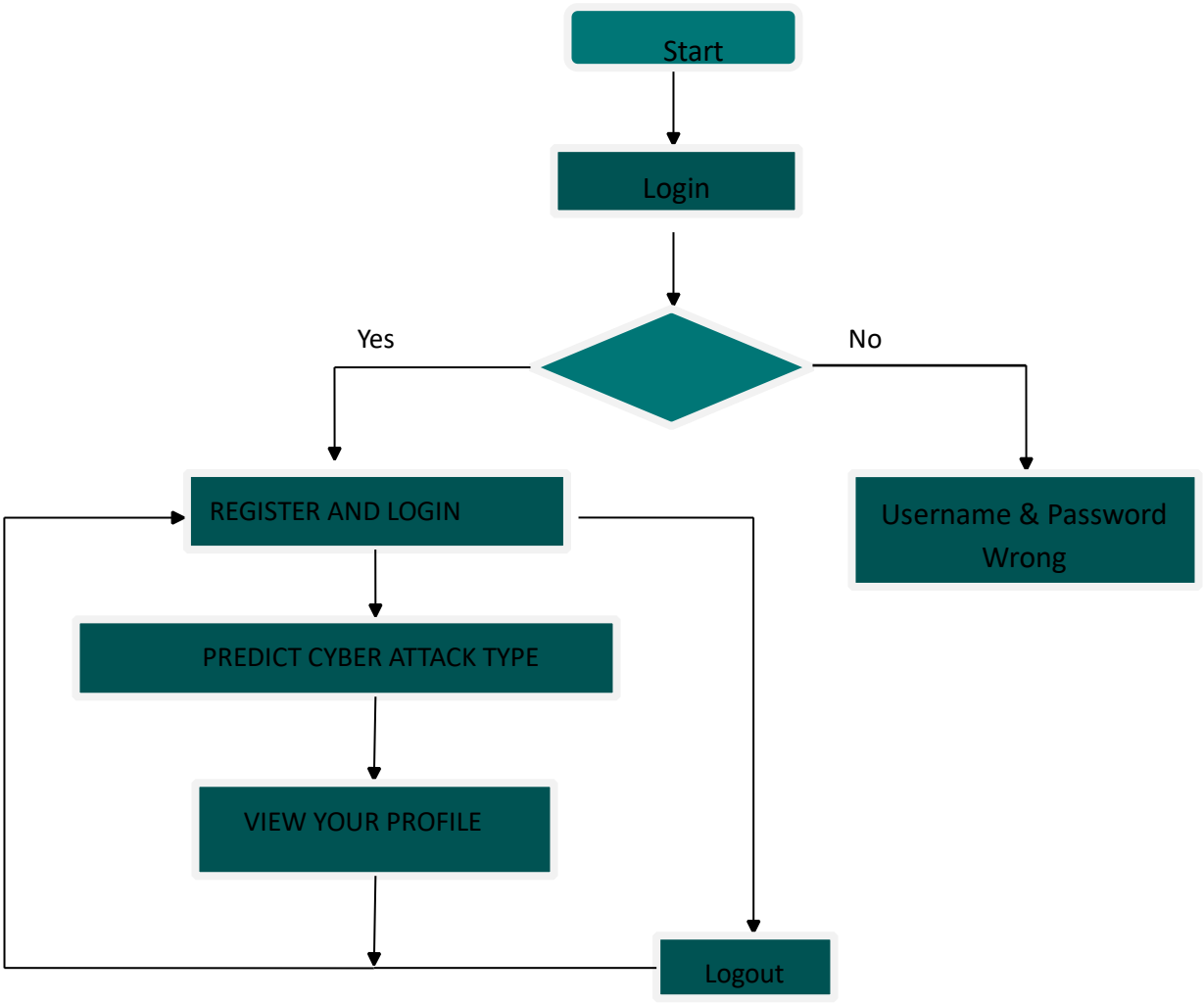
Class Diagram



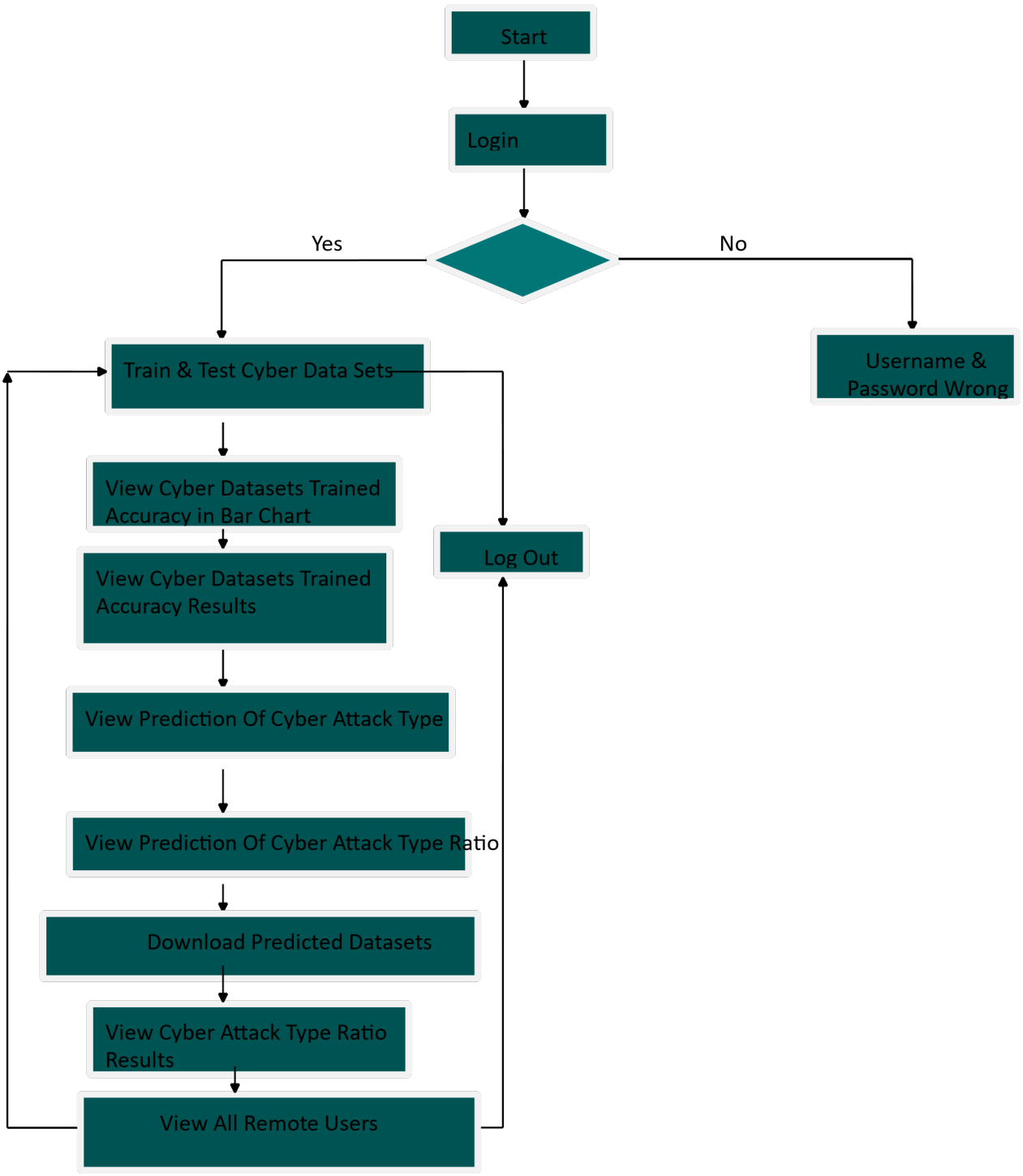
Sequence Diagram



Flow chart: Remote User



➢ Flow Chart : Service Provider



SYSTEM REQUIREMENTS

H/W System Configuration:-

Processor	- Pentium –IV
RAM	- 4 GB (min)
Hard Disk	- 20 GB
Key Board	- Standard Windows Keyboard
Mouse	- Two or Three Button Mouse
Monitor	- SVGA

SOFTWARE REQUIREMENTS:

Operating system	:	Windows 7 Ultimate.
Coding Language	:	Python.
Front-End	:	Python.
Back-End	:	Django-ORM
Designing	:	Html, css, javascript.
Data Base	:	MySQL (WAMP Server).

CHAPTER-4

IMPLEMENTATION

Modules:

Service Provider

In this module, the Service Provider has to login by using valid user name and password. After login successful he can do some operations such as
Login, Train & Test Cyber Data Sets, View Cyber Datasets Trained

Accuracy in Bar Chart, View Cyber Datasets Trained Accuracy Results,

View Prediction Of Cyber Attack Type, View Prediction Of Cyber Attack Type Ratio, Download Predicted Datasets, View Cyber Attack Type Ratio Results, View All Remote Users.

View and Authorize Users

In this module, the admin can view the list of users who all registered. In this, the admin can view the user's details such as, user name, email, address and admin authorizes the users.

Remote User

In this module, there are n numbers of users are present. User should register before doing any operations. Once user registers, their details will be stored to the database. After registration successful, he has to login by using authorized user name and password. Once Login is successful user will do some operations like REGISTER AND LOGIN, PREDICT CYBER ATTACK TYPE, VIEW YOUR PROFILE.

CHAPTER-5

SOFTWARE ENVIRONMENT

1.1 PYTHON

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive: You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

1.2 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

1.3 Python Features

Python's features include:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.

- Easy-to-maintain: Python's source code is fairly easy-to-maintain.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- Scalable: Python provides a better structure and support for large programs than shell scripting.

Python has a big list of good features:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

3.1 LIST

The list is a most versatile data type available in Python which can be written as a list of commaseparated 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"] Basic List Operations

Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string. Python Expression

Results Description len([1, 2, 3]) 3 Length

[1, 2, 3] + [4, 5, 6] [1, 2, 3, 4, 5, 6] Concatenation

['Hi!'] * 4 ['Hi!', 'Hi!', 'Hi!', 'Hi!'] Repetition

3 in [1, 2, 3] True Membership for x in

[1, 2, 3]: print x, 1 2 3 Iteration

Built-in List Functions & Methods:

Python includes the following list functions −

SN Function with Description

1 cmp(list1, list2)

Compares elements of both lists.

2 len(list)

Gives the total length of the list.

3 max(list)

Returns item from the list with max value.

4 min(list)

Returns item from the list with min value.

5 list(seq)

Converts a tuple into list.

Python includes following list methods

SN Methods with Description

1 list.append(obj)

Appends object obj to list

2 list.count(obj)

Returns count of how many times obj occurs in list

3 list. extend(seq)

Appends the contents of seq to list

4 list.index(obj)

Returns the lowest index in list that obj appears

5 list.insert(index, obj)

Inserts object obj into list at offset index

6 list.pop(obj=list[-1])

Removes and returns last object or obj from list

7 list.remove(obj)

Removes object obj from list

8 list.reverse()

Reverses objects of list in place

9 list.sort([func])

Sorts objects of list, use compare function if given

3.2 TUPLES

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 we 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 code is executed, it produces the following result −

tup1[0]: physics tup2[1:5]: [2, 3, 4, 5]

Updating Tuples:

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

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');
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

Basic Tuples Operations:

Python Expression	Results	Description
len((1, 2, 3))	3	Length

(1, 2, 3) + (4, 5, 6) (1, 2, 3, 4, 5, 6) Concatenation

('Hi!',) * 4 ('Hi!', 'Hi!', 'Hi!', 'Hi!') Repetition

3 in (1, 2, 3) True Membership for x in

(1, 2, 3): print x, 1 2 3 Iteration

Built-in Tuple Functions

SN	Function with Description
1	cmp(tuple1, tuple2):Compares elements of both tuples.
2	len(tuple):Gives the total length of the tuple.
3	max(tuple):Returns item from the tuple with max value.
4	min(tuple):Returns item from the tuple with min value.
5	tuple(seq):Converts a list into tuple.

3.2 DICTIONARY

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or 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']
```

Result – dict['Name']: Zara

dict['Age']: 7

Updating Dictionary

We 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'] Result −

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

Delete Dictionary Elements

We 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']

Built-in Dictionary Functions & Methods – Python

includes the following dictionary functions −

SN	Function with Description
1	cmp(dict1, dict2)

Compares elements of both dict.

2 len(dict)

Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

3 str(dict)

Produces a printable string representation of a dictionary

4 type(variable)

Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python includes following dictionary methods −

SN Methods with Description

1 dict.clear():Removes all elements of dictionary dict

2 dict. Copy():Returns a shallow copy of dictionary dict

3 dict.fromkeys():Create a new dictionary with keys from seq and values set to value.

4 dict.get(key, default=None):For key key, returns value or default if key not in dictionary

5 dict.has_key(key):Returns true if key in dictionary dict, false otherwise

6 dict.items():Returns a list of dict's (key, value) tuple pairs

7 dict.keys():Returns list of dictionary dict's keys

8 dict.setdefault(key, default=None):Similar to get(), but will set dict[key]=default if key is not already in dict

9 dict.update(dict2):Adds dictionary dict2's key-values pairs to dict 10

dict.values():Returns list of dictionary dict's values

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

Defining a Function

Simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).

- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

def functionname(parameters

): "function_docstring"

function_suite return

[expression]

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function −

Function definition is here

def printme(str):

"This prints a passed string into this

function" print str return;

Now you can call printme function

printme("I'm first call to user defined function!")

printme("Again second call to the same

function")

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

I'm first call to user defined function!

Again second call to the same function

Function Arguments

You can call a function by using the following types of formal arguments:

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

Global variables Local variables

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example − total = 0; # This is global variable. # Function definition is here def sum(arg1, arg2):

Add both the parameters and return them."

total = arg1 + arg2; # Here total is local variable.

print "Inside the function local total : ",

total return total; sum(10, 20);

print "Outside the function global total : ", total

Result −

Inside the function local total : 30

Outside the function global total : 0

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily

named attributes that you can bind and reference.Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example:

The Python code for a module named aname normally resides in a file named aname.py.

Here's an example of a simple module, support.py def print_func(par): print "Hello : ", par

return

The import Statement

The import has the following syntax:

import module1[, module2[,... moduleN]

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module support.py, you need to put the following
command at the top of the script −

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

Packages in Python

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-sub packages.

Consider a file Pots.py available in Phone directory. This file has following line of source code −

```
def Pots():
   print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as above −

- Phone/Isdn.py file having function Isdn()
- Phone/G3.py file having function G3()

Now, create one more file __init__.py in Phone directory −

- Phone/__init__.py

To make all of your functions available when you've imported Phone,to put explicit
import statements in __init__.py as follows − from Pots import Pots

from Isdn import Isdn

from G3 import G3

After you add these lines to __init__.py, you have all of these classes available when you import the Phone package.

Now import your Phone Package.

import Phone

Phone.Pots()

Phone.Isdn()

Phone.G3()

RESULT:

I'm Pots Phone

I'm 3G Phone

I'm ISDN Phone

In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

This chapter covers all the basic I/O functions available in Python.

Printing to the Screen

The simplest way to produce output is using the print statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows − print "Python is really a great language,", "isn't it?" Result:

Python is really a great language, isn't it?

Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

- raw_input
- input

The raw_input Function

The raw_input([prompt]) function reads one line from standard input and returns it as a string (removing the trailing newline). str = raw_input("Enter your input: "); print "Received input is : ", str

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this −

Enter your input: Hello Python

Received input is : Hello Python

The input Function

The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

str = input("Enter your input: ");

print "Received input is : ", str

This would produce the following result against the entered input −

Enter your input: [x*5 for x in range(2,10,2)]

Recieved input is : [10, 20, 30, 40]

Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.

The open Function

Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object, which would be utilized to call other support methods associated with it. Syntax

file object = open(file_name [, access_mode][, buffering])

Here are parameter details:

- file_name: The file_name argument is a string value that contains the name of the file that you want to access.
- access_mode: The access_mode determines the mode in which the file has to be opened,

i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).

- buffering: If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file −

Modes Description

r Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

rb Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

r+ Opens a file for both reading and writing. The file pointer placed at the beginning of the file.

rb+ Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

w Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

wb Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

w+ Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

wb+ Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

a Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

a+ Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

ab+ Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The file Object Attributes

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute Description

file.closed Returns true if file is closed, false otherwise. file.mode

Returns access mode with which file was opened. file.name

Returns name of the file. file.softspace Returns false if space explicitly

required with print, true otherwise.

Example

Open a file fo = open("foo.txt",

"wb") print "Name of the file: ",

fo.name print "Closed or not : ",

fo.closed print "Opening mode : ",

fo.mode print "Softspace flag : ",

fo.softspace

This produces the following result −

Name of the file: foo.txt

Closed or not : False

Opening mode : wb

Softspace flag : 0

The close() Method

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

Example

Open a file fo = open("foo.txt",

"wb") print "Name of the file: ",

fo.name

Close opend file

fo.close()

Result −

Name of the file: foo.txt

Reading and Writing Files

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

The write() Method

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.The write() method does not add a newline character ('\n') to the end of the string Syntax fileObject.write(string);

Here, passed parameter is the content to be written into the opened file. Example

Open a file

fo = open("foo.txt", "wb")

fo.write("Python is a great language.\nYeah its great!!\n");

Close opend file fo.close()

The above method would create foo.txt file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

Python is a great language.

Yeah its great!!

The read() Method

The read() method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data. Syntax

fileObject.read([count]);

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

Example

Let's take a file foo.txt, which we created above.

Open a file fo =

open("foo.txt", "r+") str =

fo.read(10); print "Read

String is : ", str # Close

opend file fo.close()

This produces the following result −

Read String is : Python is

File Positions

The tell() method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The seek(offset[, from]) method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes are to be moved.

If from is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example

Let us take a file foo.txt, which we created above.

Open a file fo =

open("foo.txt", "r+") str =

fo.read(10); print "Read

String is : ", str

```
# Check current position
position = fo.tell();
print "Current file position : ", position

# Reposition pointer at the beginning once
again position = fo.seek(0, 0); str = fo.read(10);
print "Again read String is : ", str
# Close opend file
fo.close()
```

This produces the following result −

```
Read String is :  Python is
Current file position :  10
Again read String is :  Python is
```

Renaming and Deleting Files

Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

The rename() Method

The rename() method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file test1.txt:

```
import os
```

Rename a file from test1.txt to test2.txt

os.rename("test1.txt", "test2.txt")

The remove() Method

You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument. Syntax

os.remove(file_name)

Example

Following is the example to delete an existing file test2.txt −

#!/usr/bin/python

import os

Delete file test2.txt

os.remove("text2.txt")

Directories in Python All files are

contained within various

directories, and Python has no

problem handling these too. The

os module has several methods

that help you create, remove, and

change directories.

The mkdir() Method

You can use the mkdir() method of the os module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created. Syntax os.mkdir("newdir")

Example

Following is the example to create a directory test in the current directory −

```
#!/usr/bin/python
import os

# Create a directory "test"
os.mkdir("test")
```

The chdir() Method

You can use the chdir() method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory. Syntax os.chdir("newdir")

Example

Following is the example to go into "/home/newdir" directory −

```
#!/usr/bin/python
import os

# Changing a directory to
"/home/newdir"
os.chdir("/home/newdir") The getcwd()
```

Method

The getcwd() method displays the current working

directory. Syntax os.getcwd()

Example

Following is the example to give current directory −

```
import os

# This would give location of the current directory
os.getcwd()
```

The rmdir() Method

The rmdir() method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax:

os.rmdir('dirname')

Example

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

```
import os

# This would  remove "/tmp/test"  directory.

os.rmdir( "/tmp/test"  )
```

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows −

- File Object Methods: The file object provides functions to manipulate files.
- OS Object Methods: This provides methods to process files as well as directories.

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −

- Exception Handling: This would be covered in this tutorial. Here is a list standard Exceptions available in Python: Standard Exceptions.
- Assertions: This would be covered in Assertions in Python

List of Standard Exceptions −

EXCEPTION NAME DESCRIPTION

Exception Base class for all exceptions

StopIteration Raised when the next() method of an iterator does not point to any object.

SystemExit Raised by the sys.exit() function.

StandardError Base class for all built-in exceptions except StopIteration and SystemExit.

ArithmeticError Base class for all errors that occur for numeric calculation.

OverflowError Raised when a calculation exceeds maximum limit for a numeric type.

FloatingPointError Raised when a floating point calculation fails.

ZeroDivisionError Raised when division or modulo by zero takes place for all numeric types.

AssertionError Raised in case of failure of the Assert statement.

AttributeError Raised in case of failure of attribute reference or assignment.

EOFError Raised when there is no input from either the raw_input() or input() function and the end of file is reached.

ImportError Raised when an import statement fails.

KeyboardInterrupt Raised when the user interrupts program execution, usually by pressing Ctrl+c.

LookupError Base class for all lookup errors.

IndexError

KeyError Raised when an index is not found in a sequence.

Raised when the specified key is not found in the dictionary.

NameError Raised when an identifier is not found in the local or global namespace.

UnboundLocalError

EnvironmentError Raised when trying to access a local variable in a function or method but no value has been assigned to it.

Base class for all exceptions that occur outside the Python environment.

IOError

IOError Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.

Raised for operating system-related errors.

SyntaxError

IndentationError Raised when there is an error in Python syntax.

Raised when indentation is not specified properly.

SystemError Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.

SystemExit Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.

TypeError Raised when an operation or function is attempted that is invalid for the specified data type.

ValueError Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.

RuntimeError Raised when a generated error does not fall into any category.

NotImplementedError Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as −

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:

- Importing the API module.
- Acquiring a connection with the database.
- Issuing SQL statements and stored procedures.
- Closing the connection

CHAPTER-6

SYSTEM STUDY AND TESTING

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 TESTING

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.

6.1 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.

6.2 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.

6.3 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.

SYSTEM TESTING METHODOLOGIES

The following are the Testing Methodologies:

1.Unit Testing.

2.Integration Testing.

3.User Acceptance Testing.

4.Output Testing.

5.Validation Testing.

Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing.

During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. All error handling paths are also tested.

Integration Testing

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

1)Top Down Integration

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner.

In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

2. Bottom-up Integration

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom up integration strategy may be implemented with the following steps:

- The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.
- A driver (i.e.) the control program for testing is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure

The bottom up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

OTHER TESTING METHODOLOGIES

User Acceptance Testing

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

Output Testing

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format. Validation Checking

Validation checks are performed on the following fields.

Text Field:

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

Numeric Field:

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. Testing involves executing the real data information is used in the

program the existence of any program defect is inferred from the output. The testing should be planned so that all the requirements are individually tested.

A successful test is one that gives out the defects for the inappropriate data and produces and output revealing the errors in the system.

Preparation of Test Data

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

Using Live Test Data:

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a set of data from their normal activities. Then, the systems person uses this data as a way to partially test the system. In other instances, programmers or analysts extract a set of live data from the files and have them entered themselves.

It is difficult to obtain live data in sufficient amounts to conduct extensive testing. And, although it is realistic data that will show how the system will perform for the typical processing requirement, assuming that the live data entered are in fact typical, such data generally will not test all combinations or formats that can enter the system. This bias toward typical values then does not provide a true systems test and in fact ignores the cases most likely to cause system failure.

Using Artificial Test Data:

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program.

The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the systems specifications.

The package “Virtual Private Network” has satisfied all the requirements specified as per software requirement specification and was accepted.

USER TRAINING

Whenever a new system is developed, user training is required to educate them about the working of the system so that it can be put to efficient use by those for whom the system has been primarily designed. For this purpose the normal working of the project was demonstrated to the prospective

users. Its working is easily understandable and since the expected users are people who have good knowledge of computers, the use of this system is very easy.

MAINTAINENCE

This covers a wide range of activities including correcting code and design errors. To reduce the need for maintenance in the long run, we have more accurately defined the user's requirements during the process of system development. Depending on the requirements, this system has been developed to satisfy the needs to the largest possible extent. With development in technology, it may be possible to add many more features based on the requirements in future. The coding and designing is simple and easy to understand which will make maintenance easier.

TESTING STRATEGY :

A strategy for system testing integrates system test cases and design techniques into a well planned series of steps that results in the successful construction of software. The testing strategy must cooperate test planning, test case design, test execution, and the resultant data collection and evaluation .A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

SYSTEM TESTING:

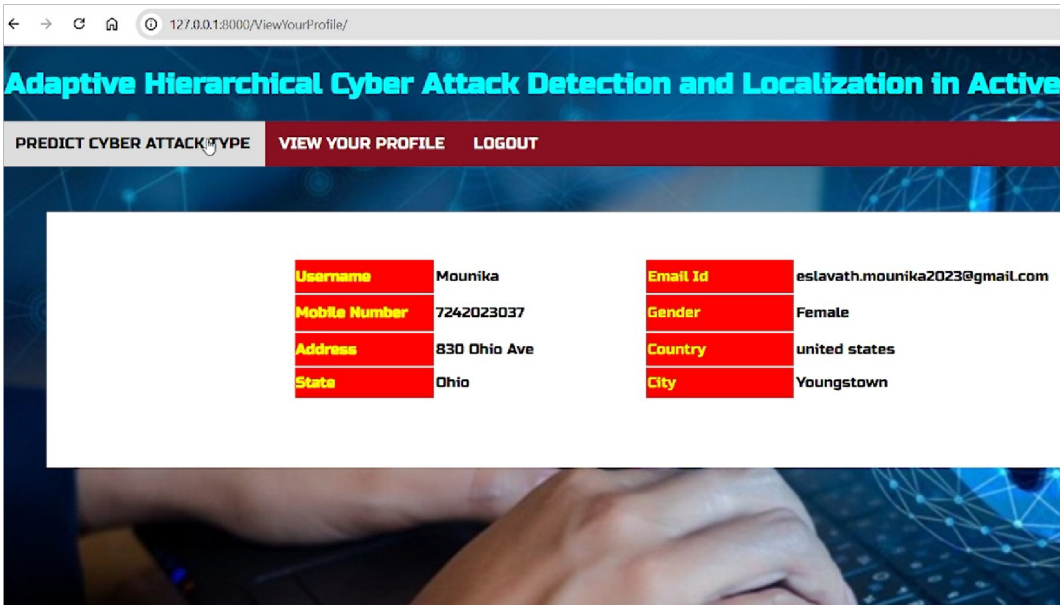
Software once validated must be combined with other system elements (e.g. Hardware, people, database). System testing verifies that all the elements are proper and that overall system function performance is achieved. It also tests to find discrepancies between the system and its original objective, current specifications and system documentation.

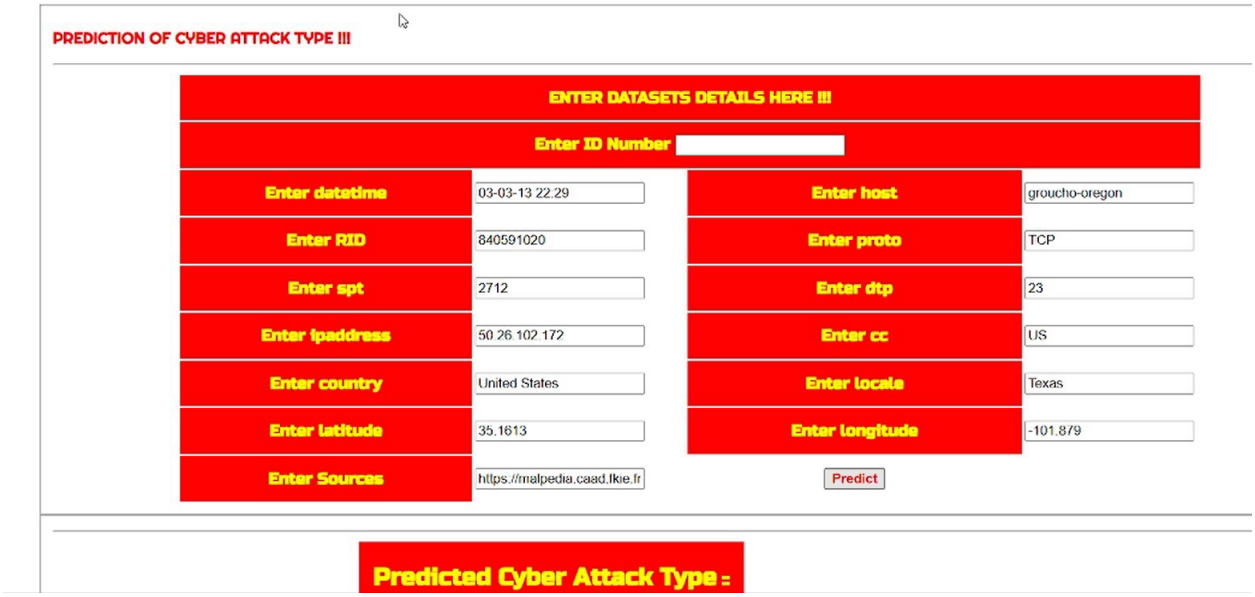
UNIT TESTING:

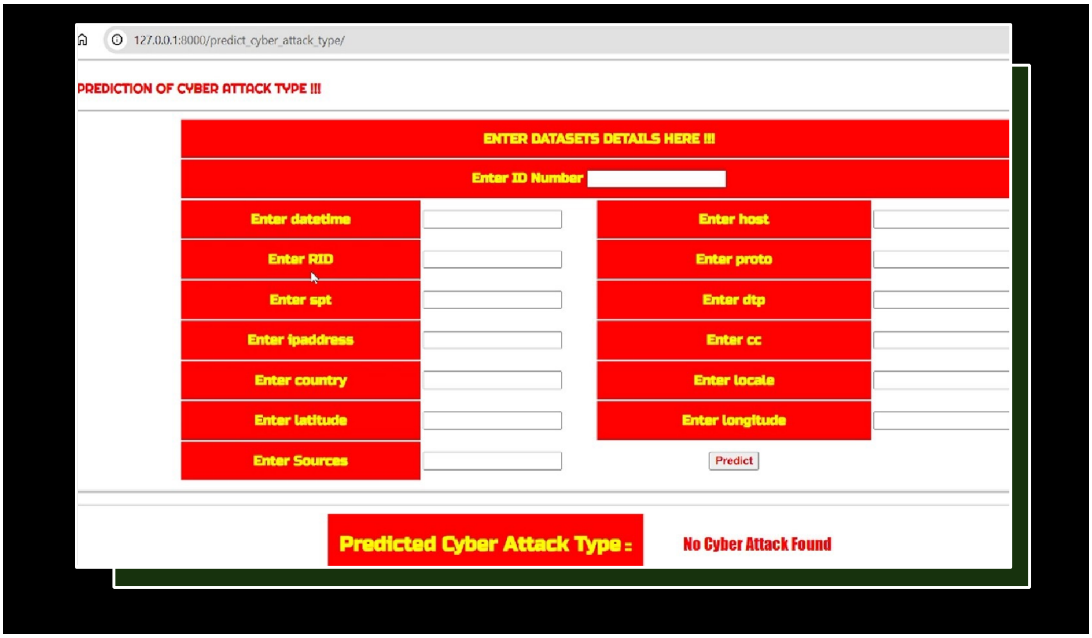
In unit testing different are modules are tested against the specifications produced during the design for the modules. Unit testing is essential for verification of the code produced during the coding phase, and hence the goals to test the internal logic of the modules. Using the detailed design description as a guide, important Conrail paths are tested to uncover errors within the boundary of the modules. This testing is carried out during the programming stage itself. In this type of testing step, each module was found to be working satisfactorily as regards to the expected output from the module.

In Due Course, latest technology advancements will be taken into consideration. As part of technical build-up many components of the networking system will be generic in nature so that future projects can either use or interact with this. The future holds a lot to offer to the development and refinement of this project

CHAPTER -7 SCREENSHOT


127.0.0.1:8000/ViewYourProfile/
Adaptive Hierarchical Cyber Attack Detection and Localization in Active
PREDICT CYBER ATTACK TYPE
VIEW YOUR PROFILE
LOGOUT
Username
Mounika
Email Id
eslavath.mounika2023@gmail.com
Mobile Number
7242023037
Gender
Female
Address
830 Ohio Ave
Country
united states
State
Ohio
City
Youngstown


PREDICTION OF CYBER ATTACK TYPE !!!
ENTER DATASETS DETAILS HERE !!!
Enter ID Number
Enter datetime
03-03-13 22:29
Enter host
groucho-oregon
Enter RID
840591020
Enter proto
TCP
Enter spt
2712
Enter dtp
23
Enter ipaddress
50.26.102.172
Enter cc
US
Enter country
United States
Enter locale
Texas
Enter latitude
35.1613
Enter longitude
-101.879
Enter Sources
https://malpedia.caad.fkie.fr
Predict
Predicted Cyber Attack Type :


127.0.0.1:8000/predict_cyber_attack_type/
PREDICTION OF CYBER ATTACK TYPE !!!
ENTER DATASETS DETAILS HERE !!!
Enter ID Number
Enter datetime
Enter RID
Enter spt
Enter ipaddress
Enter country
Enter latitude
Enter Sources
Enter host
Enter proto
Enter dtp
Enter cc
Enter locale
Enter longitude
Predict
Predicted Cyber Attack Type :
No Cyber Attack Found

CHAPTER-8

CONCLUSION

In this paper, we proposed an adaptive hierarchical cyber attack localization approach for active distribution systems. Electric waveform signals obtained by WMU sensors are used to capture the abnormal features, which would be otherwise ignored. To improve the eScience, we propose a modified spectral clustering method to first partition the whole large network into smaller ‘coarse’ sub-regions. Next, the accurate ‘fine’ cyber attack location can be determined by calculating and analyzing Impact Score of each sensor in the potential sub-region. Furthermore, we compare our method with other methods in each step in cyber attack detection, sub-graph clustering, and localization, respectively. The results from two representative distribution grids show that our method shows promising performances.

Contribution of team members

1. Mounika Eslavath(Y00868440)

Researching cyberattacks on power systems, locating critical flaws, and assisting in the creation of a hierarchical detection model were my contributions. Using Python and MATLAB, I worked on data pretreatment, model implementation, and assault scenario simulation to verify system **performance.**

2. Kasukurthi Tejaswi(Y00868439)

conducted study on the particular issues raised earlier and found viable solutions. An important aspect of this work was investigating the possible implementation of these techniques, which we went into great length about in the experiments section.

Through my research, I was able to comprehend the project's experimental approaches as well as the general idea of cyberattacks and the several ways they are addressed. I also added to the conclusion section by stressing the significance of these cyberattacks and providing a summary of the main findings.

REFERENCES

[1] I. Džafi´c, R. A. Jabr, S. Henselmeyer, and T. Đonlagi´c, "Fault location in distribution networks through graph marking," IEEE Transactions on Smart Grid, vol. 9, no. 2, pp. 1345–1353, 2016.

[2] R. Bhargav, B. R. Bhalja, and C. P. Gupta, "Novel fault detection and localization algorithm for low voltage dc microgrid," IEEE Transactions on Industrial Informatics, 2019.

[3] G. Wu, G. Wang, J. Sun, and J. Chen, "Optimal partial feedback attacks in cyber-physical power systems," IEEE Transactions on Automatic Control, vol. 65, no. 9, pp. 3919–3926, 2020.

[4] F. Li, Y. Shi, A. Shinde, J. Ye, and W.-Z. Song, "Enhanced cyberphysical security in internet of things through energy auditing," IEEE Internet of Things Journal, vol. 6, no. 3, pp. 5224–5231, 2019. [5] A. J. Wilson, D. R. Reising, R. W. Hay, R. C. Johnson, A. A. Karrar, and T. D. Loveless, "Automated identification of electrical disturbance waveforms within an operational smart power grid," IEEE Transactions on Smart Grid, vol. 11, no. 5, pp. 4380–4389, 2020.

[6] P. Dutta, A. Esmaeilian, and M. Kezunovic, "Transmission-line fault analysis using synchronized sampling," IEEE transactions on power delivery, vol. 29, no. 2, pp. 942–950, 2014.

[7] I. Sadeghkhani, M. E. H. Golshan, A. Mehrizi-Sani, J. M. Guerrero, and A. Ketabi, "Transient monitoring function–based fault detection for inverter interfaced microgrids," IEEE Transactions on Smart Grid, vol. 9, no. 3, pp. 2097–2107, 2016.

[8] A. F. Bastos, S. Santoso, W. Freitas, and W. Xu, "Synchrowaveform measurement units and applications," in 2019 IEEE Power & Energy Society General Meeting (PESGM). IEEE, 2019, pp. 1–5.

[9] Schweitzer Engineering Laboratories, Pullman, WA, USA., "SEL-T400L

Time Domain Line Protection," https://selinc.com/ products/T400L/, Last Access: July 31, 2020.

[10] Candura instruments, Oakville, ON, Canada., "iPSR intelligent Power System Recorder," https://www.candura.com/ products/ipsr.html, Last Access: July 31, 2020.

[11] D. Borkowski, A. Wetula, and A. Bie´n, "Contactless measurement of substation busbars voltages and waveforms reconstruction using electric field sensors and artificial neural network," IEEE Transactions on Smart Grid, vol. 6, no. 3, pp. 1560–1569, 2014.

[12] B. Gao, R. Torquato, W. Xu, and W. Freitas, "Waveform-based method for fast and accurate identification of subsynchronous resonance events," IEEE Transactions on Power Systems, vol. 34, no. 5, pp. 3626–3636, 2019.

[13] F. Li, R. Xie, Z. Wang, L. Guo, J. Ye, P. Ma, and W. Song, "Online distributed iot security monitoring with multidimensional streaming big data," IEEE Internet of Things Journal, vol. 7, no. 5, pp. 4387–4394, 2020.

[14] F. Li, A. Shinde, Y. Shi, J. Ye, X.-Y. Li, and W.-Z. Song, "System statistics learning-based iot security: Feasibility and suitability," IEEE Internet of Things Journal, vol. 6, no. 4, pp. 6396–6403, 2019. [15] F. Li, Q. Li, J. Zhang, J. Kou, J. Ye, W. Song, and H. A. Mantooth, "Detection and diagnosis of data integrity attacks in solar farms based on multilayer long short-term memory network," IEEE Transactions on Power Electronics, vol. 36, no. 3, pp. 2495–2498, 2021.

[16] A. Wang and J. Shi, "Holistic modeling and analysis of multistage manufacturing processes with sparse eSective inputs and mixed profile outputs," IISE Transactions, vol. 53, no. 5, pp. 582–596, 2021.

[17] J. Ye, L. Guo, B. Yang, F. Li, L. Du, L. Guan, and W. Song, "Cyber–physical security of powertrain systems in modern electric vehicles: Vulnerabilities,

challenges, and future visions," IEEE Journal of Emerging and Selected Topics in Power Electronics, vol. 9, no. 4, pp.
4639–4657, 2021.

[18] F. Li, R. Xie, B. Yang, L. Guo, P. Ma, J. Shi, J. Ye, and W. Song, "Detection and identification of cyber and physical attacks on distribution power grids with pvs: An online high-dimensional data-driven approach," IEEE Journal of Emerging and Selected Topics in Power Electronics, Early Access.

[19] J. Zhang, S. Sahoo, J. C.-H. Peng, and F. G. Blaabjerg, "Mitigating concurrent false data injection attacks in cooperative dc microgrids," IEEE Transactions on Power Electronics, 2021, early access.

[20] M. P. Tcheou, L. Lovisolo, M. V. Ribeiro, E. A. Da Silva, M. A. Rodrigues, J. M. Romano, and P. S. Diniz, "The compression of electric signal waveforms for smart grids: State of the art and future trends," IEEE Transactions on Smart Grid, vol. 5, no. 1, pp. 291–302, 2013.

[21] Y.-C. Chang and T.-C. Huang, "An interactive smart grid communication approach for big data traSic," Computers & Electrical Engineering, vol. 67, pp. 170–181, 2018.

[22] H. Maaß, H. K. Cakmak, F. Bach, R. Mikut, A. Harrabi, W. Süß, W. Jakob, K.U. Stucky, U. G. Kühnapfel, and V. Hagenmeyer, "Data processing of high-rate low-voltage distribution grid recordings for smart grid monitoring and analysis," EURASIP Journal on Advances in Signal Processing, vol. 2015, no. 1, pp. 1–21, 2015.

[23] X. Liang, S. A. Wallace, and D. Nguyen, "Rule-based data-driven analytics for wide-area fault detection using synchrophasor data," IEEE
Transactions on Industry Applications, vol. 53, no. 3, pp. 1789–1798, 2016.

[24] B. Wang, H. Wang, L. Zhang, D. Zhu, D. Lin, and S. Wan, "A datadriven method to detect and localize the single-phase grounding fault in distribution

network based on synchronized phasor measurement," EURASIP Journal on Wireless Communications and Networking, vol.
2019, no. 1, p. 195, 2019.

[25] I. Niazazari and H. Livani, "A pmu-data-driven disruptive event classification in distribution systems," Electric Power Systems Research, vol.
157, pp. 251–260, 2018.