Analyzing cyber incident data sets is an important method for deepening our understanding of the evolution of the threat situation. This is a relatively new research topic, and many studies remain to be done. In this paper, we report a statistical analysis of a breach incident data set corresponding to 12 years (2005–2017) of cyber hacking activities that include malware attacks. We show that, in contrast to the findings reported in the literature, both hacking breach incident inter-arrival times and breach sizes should be modeled by stochastic processes, rather than by distributions because they exhibit autocorrelations. Then, we propose particular stochastic process models to, respectively, fit the inter-arrival times and the breach sizes. We also show that these models can predict the inter-arrival times and the breach sizes. In order to get deeper insights into the evolution of hacking breach incidents, we conduct both qualitative and quantitative trend analyses on the data set. We draw a set of cybersecurity insights, including that the threat of cyber hacks is indeed getting worse in terms of their frequency, but not in terms of the magnitude of their damage.

**INTRODUCTION**

**PURPOSE:**

DAta breaches are one of the most devastating cyber incidents. The Privacy Rights Clearinghouse [1] reports 7,730 data breaches between 2005 and 2017, accounting for 9,919,228,821 breached records. The Identity Theft Resource Center and Cyber Scout [2] reports 1,093 data breach incidents in 2016, which is 40% higher than the 780 data breach incidents in 2015. The United States Office of Personnel Management (OPM) [3] reports that the personnel information of 4.2 million current and former Federal government employees and the background investigation records of current, former, and prospective federal employees and contractors (including 21.5 million Social Security Numbers) were stolen in 2015

**AIM:**

Analyzing cyber incident datasets is an important method for deepening our understanding of the evolution of the threat situation.

**OBJECTIVE**

The monetary price incurred by data breaches is also substantial. IBM [4] reports that in year 2016, the global average cost for each lost or stolen record containing sensitive or confidential information was $158. NetDiligence [5] reports that in year 2016, the median number of breached records was 1,339, the median per-record cost was $39.82, the average breach cost was $665,000, and the median breach cost was $60,000. While technological solutions can harden cyber systems against attacks, data breaches continue to be a big problem. This motivates us to characterize the evolution of data breach incidents

**BACKGROUND OF THE PROJECT**

Prior works closely related to the present study. Maillart and Sornette [7] analyzed a dataset [8] of 956 personal identity loss incidents that occurred in the United States between year 2000 and 2008. They found that the personal identity losses per incident, denoted by X, can be modeled by a heavy tail distribution Pr(X > n) ∼ n −α where α = 0.7 ± 0.1. This result remains valid when dividing the dataset per type of organizations: business, education, government, and medical institution. Because the probability density function of the identity losses per incident is static, the situation of identity loss is stable from the point of view of the breach size. Edwards et al. [9] analyzed a different breach dataset [1] of 2,253 breach incidents that span over a decade (2005 to 2015). These breach incidents include two categories: negligent breaches (i.e., incidents caused by lost, discarded, stolen devices, or other reasons) and malicious breaching (i.e., incidents caused by hacking, insider and other reasons). They showed that the breach size can be modeled by the log-normal or log-skewnormal distribution and the breach frequency can be modeled by the negative binomial distribution, implying that neither the breach size nor the breach frequency has increased over the years. Wheatley et al. [10] analyzed an organizational breach incidents dataset that is combined from [8] and [1] and spans over a decade (year 2000 to 2015). They used the Extreme Value Theory [11] to study the maximum breach size, and further modeled the large breach sizes by a doubly truncated Pareto distribution. They also used linear regression to study the frequency of the data breaches, and found that the frequency of large breaching incidents is independent of time for the United States organizations, but shows an increasing trend for non-US organizations. There are also studies on the dependence among cyber risks. Bohme and Kataria [12] studied the dependence between cyber ¨ risks of two levels: within a company (internal dependence) and across companies (global dependence). Herath and Herath [13] used the Archimedean copula to model cyber risks caused by virus incidents, and found that there exists some dependence between these risks. Mukhopadhyay et al. [14] used a copula-based Bayesian Belief Network to assess cyber vulnerability. Xu and Hua [15] investigated using copulas to model dependent cyber risks. Xu et al. [16] used copulas to investigate the dependence encountered when modeling the effectiveness of cyber defense early-warning. Peng et al. [17] investigated multivariate cybersecurity risks with dependence. Compared with all these studies mentioned above, the present paper is unique in that it uses a new methodology to analyze a new perspective of breach incidents (i.e., cyber hacking breach incidents). This perspective is important because it reflects the consequence of cyber hacking (including malware). The new methodology found for the first time, that both the incidents inter-arrival times and the breach sizes should be modeled by stochastic processes rather than distributions, and that there exists a positive dependence between them. Other prior works related to the present study. Eling and Loperfido [18] analyzed a dataset [1] from the point of view of actuarial modeling and pricing. Bagchi and Udo [19] used a variant of the Gompertz model to analyze the growth of computer and Internet-related crimes. Condon et. al [20] used the ARIMA model to predict security incidents based on a dataset provided by the Office of Information Technology at the University of Maryland. Zhan et al. [21] analyzed the posture of cyber threats by using a dataset collected at a network telescope. Using datasets collected at a honeypot, Zhan et al. [22], [23] exploited their statistical properties including longrange dependence and extreme values to describe and predict the number of attacks against the honeypot; a predictability evaluation of a related dataset is described in [24]. Peng et al. [25] used a marked point process to predict extreme attack rates. Bakdash et al. [26] extended these studies into related cybersecurity scenarios. Liu et al. [27] investigated how to use externally observable features of a network (e.g., mismanagement symptoms) to forecast the potential of data breach incidents to that network. Sen and Borle [28] studied the factors that could increase or decrease the contextual risk of data breaches, by using tools that include the opportunity theory of crime, the institutional anomie theory, and the institutional theory.

**SCOPE OF THE PROJECT**

The present study is motivated by several questions that have not been investigated until now, such as: Are data breaches caused by cyber attacks increasing, decreasing, or stabilizing? A principled answer to this question will give us a clear insight into the overall situation of cyber threats. This question was not answered by previous studies. Specifically, the dataset analyzed in [7] only covered the time span from 2000 to 2008 and does not necessarily contain the breach incidents that are caused by cyber attacks; the dataset analyzed in [9] is more recent, but contains two kinds of incidents: negligent breaches (i.e., incidents caused by lost, discarded, stolen devices and other reasons) and malicious breaching. Since negligent breaches represent more human errors than cyber attacks, we do not consider them in the present study. Because the malicious breaches studied in [9] contain four sub-categories: hacking (including malware), insider, payment card fraud, and unknown, this study will focus on the hacking sub-category (called hacking breach dataset thereafter), while noting that the other three sub-categories are interesting on their own and should be analyzed separately

**MODULES:**

1. **UPLOAD DATA**

The data resource to database can be uploaded by both administrator and authorized user. The data can be uploaded with key in order to maintain the secrecy of the data that is not released without knowledge of user. The users are authorized based on their details that are shared to admin and admin can authorize each user. Only Authorized users are allowed to access the system and upload or request for files.

1. **ACCESS DETAILS**

The access of data from the database can be given by administrators. Uploaded data are managed by admin and admin is the only person to provide the rights to process the accessing details and approve or unapproved users based on their details.

1. **USER PERMISSIONS**

The data from any resources are allowed to access the data with only permission from administrator. Prior to access data, users are allowed by admin to share their data and verify the details which are provided by user. If user is access the data with wrong attempts then, users are blocked accordingly. If user is requested to unblock them, based on the requests and previous activities admin is unblock users.

1. **DATA ANALYSIS**

Data analyses are done with the help of graph. The collected data are applied to graph in order to get the best analysis and prediction of dataset and given data policies. The dataset can be analyzed through this pictorial representation in order to better understand of the data details.

**SYSTEM ANALYSIS**

**REQUIREMENT ANALYSIS**

The project involved analyzing the design of few applications so as to make the application more users friendly. To do so, it was really important to keep the navigations from one screen to the other well ordered and at the same time reducing the amount of typing the user needs to do. In order to make the application more accessible, the browser version had to be chosen so that it is compatible with most of the Browsers.

**REQUIREMENT SPECIFICATION**

**Functional Requirements**

* Graphical User interface with the User.

**Software Requirements**

For developing the application the following are the Software Requirements:

1. Python
2. Django

**Operating Systems supported**

1. Windows 10 64 bit OS

**Technologies and Languages used to Develop**

1. Python

**Debugger and Emulator**

* Any Browser (Particularly Chrome)

**Hardware Requirements**

For developing the application the following are the Hardware Requirements:

* Processor: Intel i3
* RAM: 4 GB
* Space on Hard Disk: minimum 1 TB

**SYSTEM SPECIFICATION:**

**HARDWARE REQUIREMENTS:**

* **System :** Intel i3
* **Hard Disk :** 1 TB.
* **Monitor** : 14’ Colour Monitor.
* **Mouse :** Optical Mouse.
* **Ram :** 4GB.

**SOFTWARE REQUIREMENTS:**

* **Operating system :** Windows 10.
* **Coding Language :** Python.
* **Front-End :** Html. CSS
* **Designing :** Html,css,javascript.
* **Data Base :** SQLite.

**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 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.

**EXISTING SYSTEM:**

The present study is motivated by several questions that have not been investigated until now, such as: Are data breaches caused by cyber-attacks increasing, decreasing, or stabilizing? A principled answer to this question will give us a clear insight into the overall situation of cyber threats. This question was not answered by previous studies. Specifically, the dataset analyzed in [7] only covered the time span from 2000 to 2008 and does not necessarily contain the breach incidents that are caused by cyber-attacks; the dataset analyzed in [9] is more recent, but contains two kinds of incidents: negligent breaches (i.e., incidents caused by lost, discarded, stolen devices and other reasons) and malicious breaching. Since negligent breaches represent more human errors than cyber-attacks, we do not consider them in the present study. Because the malicious breaches studied in [9] contain four sub-categories: hacking (including malware), insider, payment card fraud, and unknown, this study will focus on the hacking sub-category (called hacking breach dataset thereafter), while noting that the other three sub-categories are interesting on their own and should be analyzed separately. Recently, researchers started modeling data breach incidents. Maillart and Sornette studied the statistical properties of the personal identity losses in the United States between year 2000 and 2008. They found that the number of breach incidents dramatically increases from 2000 to July 2006 but remains stable thereafter. Edwards et al. analyzed a dataset containing 2,253 breach incidents that span over a decade (2005 to 2015). They found that neither the size nor the frequency of data breaches has increased over the years. Wheatley et al., analyzed a dataset that is combined from corresponds to organizational breach incidents between year 2000 and 2015. They found that the frequency of large breach incidents (i.e., the ones that breach more than 50,000 records) occurring to US firms is independent of time, but the frequency of large breach incidents occurring to non-US firms exhibits an increasing trend.

**PROPOSED SYSTEM:**

In this paper, we make the following three contributions. First, we show that both the hacking breach incident interarrival times (reflecting incident frequency) and breach sizes should be modeled by stochastic processes, rather than by distributions. We find that a particular point process can adequately describe the evolution of the hacking breach incidents inter-arrival times and that a particular ARMA-GARCH model can adequately describe the evolution of the hacking breach sizes, where ARMA is acronym for “AutoRegressive and Moving Average” and GARCH is acronym for “Generalized AutoRegressive Conditional Heteroskedasticity.”We show that these stochastic process models can predict the inter-arrival times and the breach sizes. To the best of our knowledge, this is the first paper showing that stochastic processes, rather than distributions, should be used to model these cyber threat factors. Second, we discover a positive dependence between the incidents inter-arrival times and the breach sizes, and show that this dependence can be adequately described by a particular copula. We also show that when predicting inter-arrival times and breach sizes, it is necessary to consider the dependence; otherwise, the prediction results are not accurate. To the best of our knowledge, this is the first work showing the existence of this dependence and the consequence of ignoring it. Third, we conduct both qualitative and quantitative trend analyses of the cyber hacking breach incidents. We find that the situation is indeed getting worse in terms of the incidents inter-arrival time because hacking breach incidents become more and more frequent, but the situation is stabilizing in terms of the incident breach size, indicating that the damage of individual hacking breach incidents will not get much worse. We hope the present study will inspire more investigations, which can offer deep insights into alternate risk mitigation approaches. Such insights are useful to insurance companies, government agencies, and regulators because they need to deeply understand the nature of data breach risks.

**TECHNOLOGY USED**

**PYTHON**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An [interpreted language](https://en.wikipedia.org/wiki/Interpreted_language), Python has a design philosophy that emphasizes code [readability](https://en.wikipedia.org/wiki/Readability) (notably using [whitespace](https://en.wikipedia.org/wiki/Whitespace_character) indentation to delimit [code blocks](https://en.wikipedia.org/wiki/Code_block) rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer [lines of code](https://en.wikipedia.org/wiki/Source_lines_of_code) than might be used in languages such as [C++](https://en.wikipedia.org/wiki/C%2B%2B)or [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many [operating systems](https://en.wikipedia.org/wiki/Operating_system). [CPython](https://en.wikipedia.org/wiki/CPython" \o "CPython), the [reference implementation](https://en.wikipedia.org/wiki/Reference_implementation) of Python, is [open source](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Python_Software_Foundation). Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural](https://en.wikipedia.org/wiki/Procedural_programming), and has a large and comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

**Interactive Mode Programming**

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

$ python

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.

Live Demo

#!/usr/bin/python

# 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 −

#!/usr/bin/python

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 −

$ python -h

usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...

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 −

Live Demo

#!/usr/bin/python

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 −

Live Demo

#!/usr/bin/python

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 −

Live Demo

#!/usr/bin/python

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 −

Live Demo

#!/usr/bin/python

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 −

Live Demo

#!/usr/bin/python

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 −

Live Demo

#!/usr/bin/python

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 −

Live Demo

#!/usr/bin/python

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 −

Live Demo

#!/usr/bin/python

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 −

Live Demo

#!/usr/bin/python

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 [reusability](https://en.wikipedia.org/wiki/Reusability" \o "Reusability)and "pluggability" of components, rapid development, and the principle of [don't repeat yourself](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself). Python is used throughout, even for settings files and data models.



Django also provides an optional administrative [create, read, update and delete](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) interface that is generated dynamically through [introspection](https://en.wikipedia.org/wiki/Introspection_(computer_science)) 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',

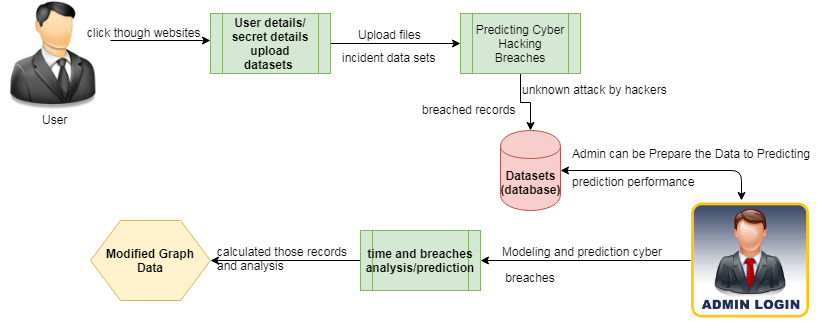
url(r'^connection/','formView', name = 'loginform'),

url(r'^login/', 'login', name = 'login'))

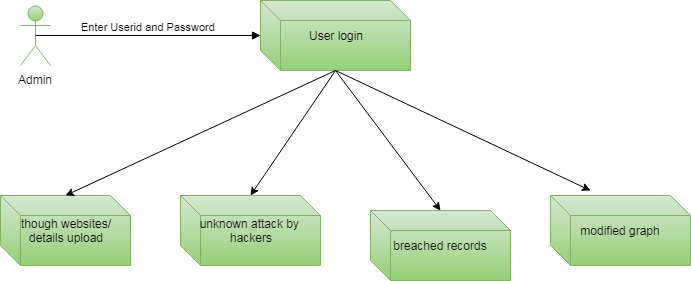
When accessing /myapp/connection, you will get to see the following page

**SYSTEM DESIGN**

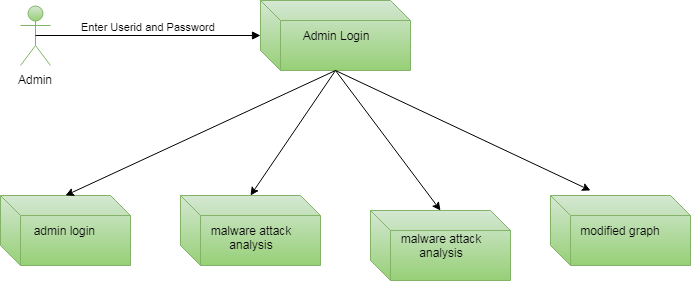
1. ARCHITECTURE DIAGRAM



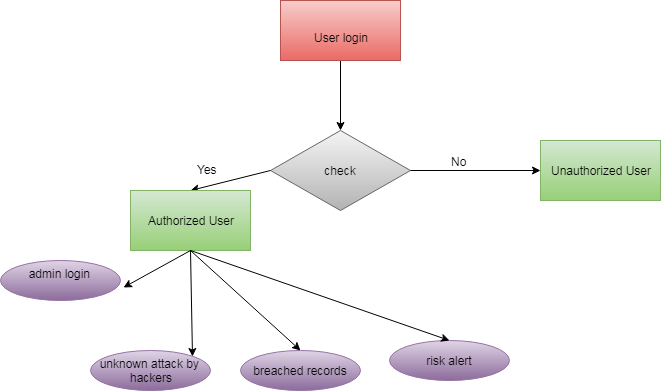
1. COMPONENT DIAGRAM
   1. User



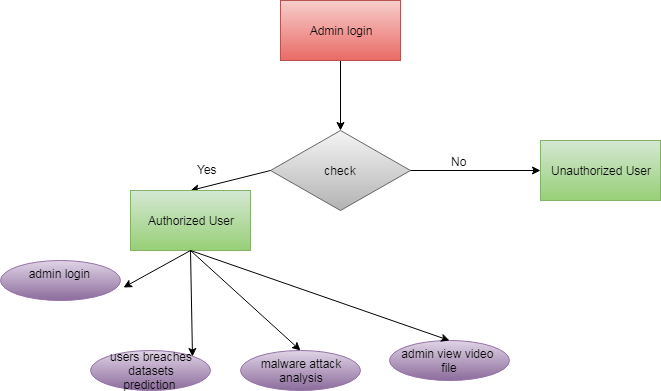
* 1. Admin



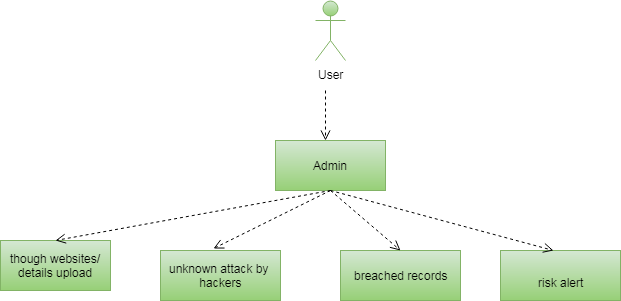
1. ER DIAGRAM
   1. User



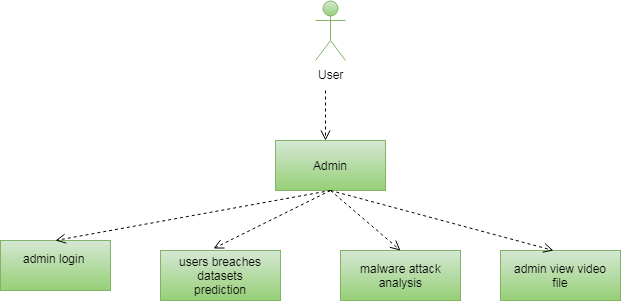
* 1. Admin



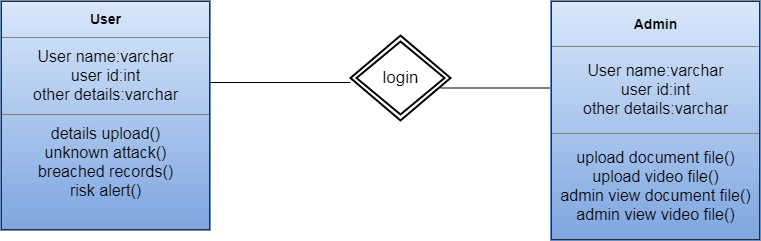
1. USE CASE DIAGRAM
   1. User



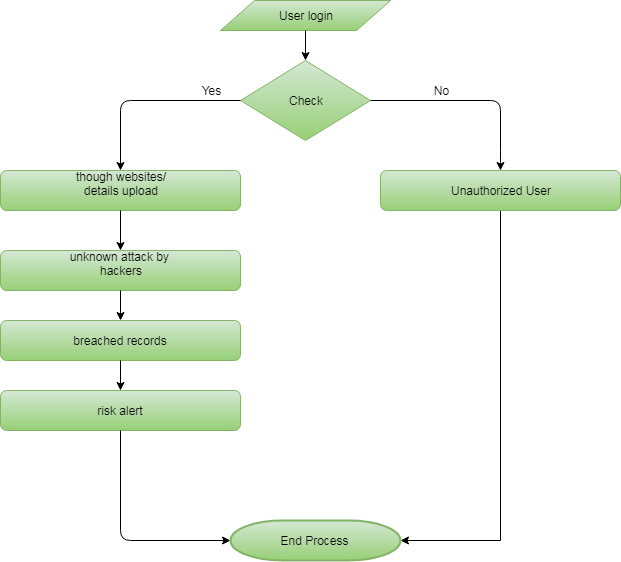
* 1. Admin



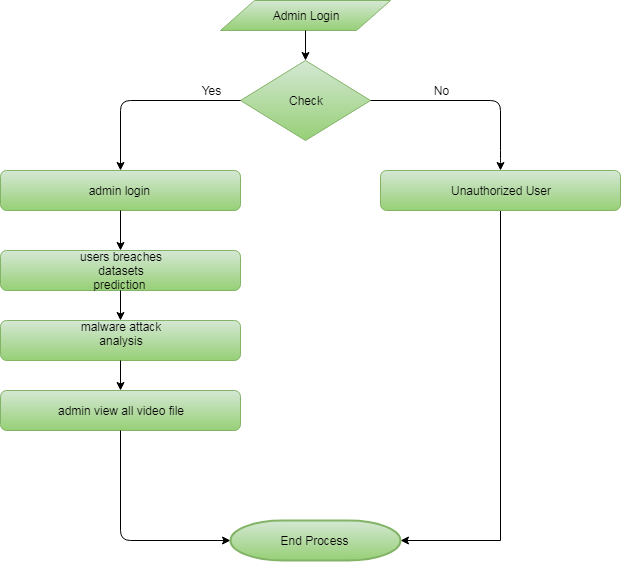
1. CLASS DIAGRAM



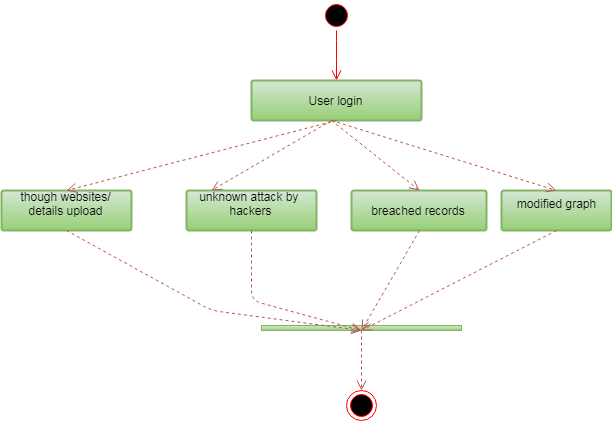
1. DATA FLOW DIAGRAM
   1. User



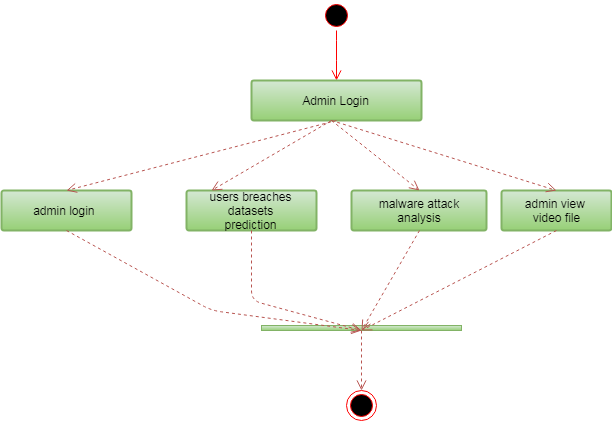
* 1. Admin



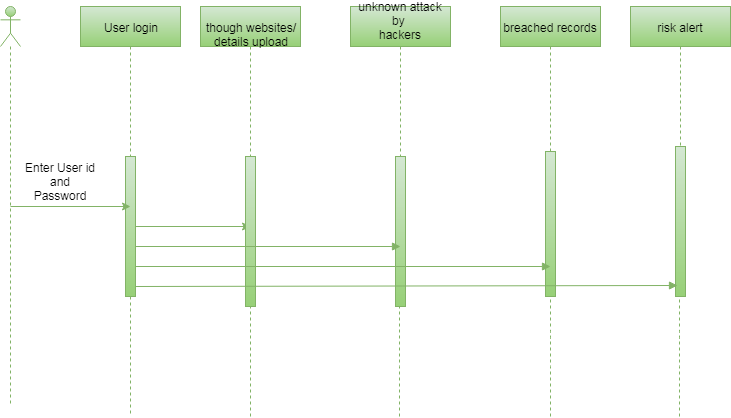
1. ACTIVITY DIAGRAM
   1. User



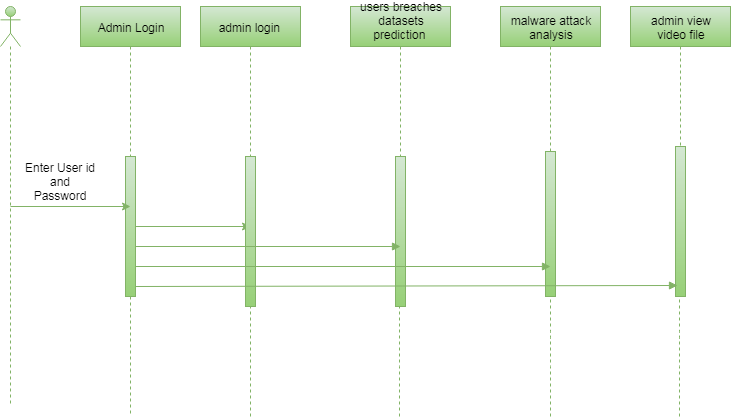
* 1. Admin



1. SEQUENCE DIAGRAM
   1. User



* 1. Admin



**IMPLEMENTATION:**

**SOURCE CODE:**

import re

from django.contrib import messages

from django.contrib.auth import authenticate

from django.db.models import Q, Count

from django.shortcuts import render, redirect

# Create your views here.

from Cyber\_Users.forms import UserRegister\_Form

from Cyber\_Users.models import UserRegister\_Model, UserAdd\_Model

def user\_login(request):

if request.method == "POST":

name = request.POST.get('name')

password = request.POST.get('password')

try:

check = UserRegister\_Model.objects.get(name=name,password=password)

request.session['userid'] = check.id

return redirect('user\_adddata')

except:

pass

user = authenticate(name=name,password=password)

if user is not None:

if user.is\_active:

return redirect('user\_adddata')

else:

messages.error(request, 'username or password are not match')

return redirect('user\_login')

return render(request, 'users/user\_login.html')

def user\_register(request):

if request.method == "POST":

forms = UserRegister\_Form(request.POST)

if forms.is\_valid():

forms.save()

messages.success(request, 'You have been successfully registered')

return redirect('user\_login')

else:

forms = UserRegister\_Form()

return render(request,'users/user\_register.html',{'form':forms})

def user\_adddata(request):

userid = request.session["userid"]

obj = UserRegister\_Model.objects.get(id=userid)

attack1 = []

attack2, attack3, attack4, attack5, attack6, attack7, attack8, attack9 = [], [], [], [], [], [], [], []

splt = ''

Entity = ''

Year = 0

Records = ''

Organizationtype = ''

Method = ''

txt =''

Adddata = ''

ans = ''

Time = ''

if request.method == "POST":

Entity = request.POST.get("entity")

Year = request.POST.get("year")

Records = request.POST.get("records")

Organizationtype = request.POST.get("organizationtype")

Method = request.POST.get("method")

txt = request.POST.get("name")

Time = request.POST.get("time")

splt = (re.findall(r"[\w']+", str(txt)))

for f in splt:

if f in ('IPid', 'FDDI', 'x25', 'rangingdistance'):

attack1.append(f)

elif f in ('tcpchecksum', 'mtcp', 'controlflags', 'tcpoffset', 'tcpport'):

attack2.append(f)

elif f in ('ICMPID', 'udptraffic', 'udpunicorn', 'datagramid', 'NTP', 'RIP', 'TFTP'):

attack3.append(f)

elif f in ('GETID', 'POSTID', 'openBSD', 'appid', 'sessionid', 'transid', 'physicalid'):

attack4.append(f)

elif f in ('SYN', 'ACK', 'synpacket', 'sycookies'):

attack5.append(f)

elif f in ('serverattack', 'serverid', 'blockbankwidth'):

attack6.append(f)

elif f in ('monlist', 'getmonlist', 'NTPserver'):

attack7.append(f)

elif f in ('portid', 'FTPID', 'tryion', 'fragflag'):

attack8.append(f)

elif f in ('malwareid', 'gethttpid', 'httpid'):

attack9.append(f)

if len(attack1) > len(attack2) and len(attack1) > len(attack3) and len(attack1) > len(attack4) and len(

attack1) > len(attack5) and len(attack1) > len(attack6) and len(attack1) > len(attack7) and len(

attack1) > len(attack8) and len(attack1) > len(attack9):

ans = "Man-in-the-middle Attack"

elif len(attack2) > len(attack1) and len(attack2) > len(attack3) and len(attack2) > len(attack4) and len(

attack2) > len(attack5) and len(attack2) > len(attack6) and len(attack2) > len(attack7) and len(

attack2) > len(attack8) and len(attack2) > len(attack9):

ans = "Phishing and spear phishing attacks"

elif len(attack3) > len(attack2) and len(attack3) > len(attack1) and len(attack3) > len(attack4) and len(

attack1) > len(attack5) and len(attack1) > len(attack6) and len(attack1) > len(attack7) and len(

attack1) > len(attack8) and len(attack1) > len(attack9):

ans = "Drive-by attack"

elif len(attack4) > len(attack2) and len(attack4) > len(attack3) and len(attack4) > len(attack1) and len(

attack4) > len(attack5) and len(attack4) > len(attack6) and len(attack4) > len(attack7) and len(

attack4) > len(attack8) and len(attack4) > len(attack9):

ans = "Password attack"

elif len(attack5) > len(attack2) and len(attack5) > len(attack3) and len(attack5) > len(attack4) and len(

attack5) > len(attack1) and len(attack5) > len(attack6) and len(attack5) > len(attack7) and len(

attack5) > len(attack8) and len(attack5) > len(attack9):

ans = "SQL injection attack"

elif len(attack6) > len(attack2) and len(attack6) > len(attack3) and len(attack6) > len(attack4) and len(

attack6) > len(attack5) and len(attack6) > len(attack1) and len(attack6) > len(attack7) and len(

attack6) > len(attack8) and len(attack6) > len(attack9):

ans = "Cross-site scripting (XSS) attack"

elif len(attack7) > len(attack2) and len(attack7) > len(attack3) and len(attack7) > len(attack4) and len(

attack7) > len(attack5) and len(attack7) > len(attack6) and len(attack7) > len(attack1) and len(

attack7) > len(attack8) and len(attack7) > len(attack9):

ans = "Eavesdropping attack"

elif len(attack8) > len(attack2) and len(attack8) > len(attack3) and len(attack8) > len(attack4) and len(

attack8) > len(attack5) and len(attack8) > len(attack6) and len(attack8) > len(attack7) and len(

attack8) > len(attack1) and len(attack8) > len(attack9):

ans = "Birthday attack"

elif len(attack9) > len(attack2) and len(attack9) > len(attack3) and len(attack9) > len(attack4) and len(

attack9) > len(attack5) and len(attack9) > len(attack6) and len(attack9) > len(attack7) and len(

attack9) > len(attack8) and len(attack9) > len(attack1):

ans = "Teardrop attack"

else:

ans = "Unmalware"

UserAdd\_Model.objects.create(uregid=obj,entity=Entity,year=Year,records=Records,organizationtype=Organizationtype,method=Method,adddata=txt,attackresult=ans,time=Time)

return render(request,'users/user\_adddata.html')

def user\_page(request):

obj = UserAdd\_Model.objects.all()

return render(request,'users/user\_page.html',{'object':obj})

def malware(request):

obj = UserAdd\_Model.objects.filter(Q(attackresult='Man-in-the-middle (MitM) attack') | Q(attackresult='Phishing and spear phishing attacks') | Q(

attackresult='Drive-by attack') | Q(attackresult='Password attack') | Q(

attackresult='SQL injection attack') | Q(attackresult='Cross-site scripting (XSS) attack') | Q(attackresult='Eavesdropping attack') | Q(

attackresult='Birthday attack') | Q(attackresult='Teardrop attack'))

return render(request,'users/malware.html',{'object':obj})

def unmalware(request):

obj = UserAdd\_Model.objects.filter(attackresult='Unmalware')

return render(request,'users/unmalware.html',{'object':obj})

def breaches\_analysis(request):

chart = UserAdd\_Model.objects.values('attackresult','method').annotate(dcount=Count('attackresult'))

return render(request,'users/breaches\_analysis.html',{'objects':chart})

def chart\_page(request,chart\_type):

chart = UserAdd\_Model.objects.values('year').annotate(dcount=Count('organizationtype'))

return render(request,'users/chart\_page.html',{'chart\_type':chart\_type,'objects':chart})

from tkinter import CASCADE

from django.db import models

# Create your models here.

class UserRegister\_Model(models.Model):

name = models.CharField(max\_length=50)

email = models.EmailField(max\_length=30)

password = models.CharField(max\_length=10)

phoneno = models.CharField(max\_length=15)

address = models.CharField(max\_length=500)

class UserAdd\_Model(models.Model):

uregid = models.ForeignKey(UserRegister\_Model)

entity = models.CharField(max\_length=100)

year = models.IntegerField()

records = models.CharField(max\_length=1000)

organizationtype=models.CharField(max\_length=1000)

method = models.CharField(max\_length=100)

adddata = models.CharField(max\_length=500)

attackresult = models.CharField(max\_length=500)

time = models.CharField(max\_length=100)

**ALGORITHM:**

**SUPPORT VECTOR MACHINE**

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification and regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well (look at the below snapshot). Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). More formally, a support vector machine constructs a hyper plane or set of hyper planes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyper plane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. Whereas the original problem may be stated in a finite dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space.

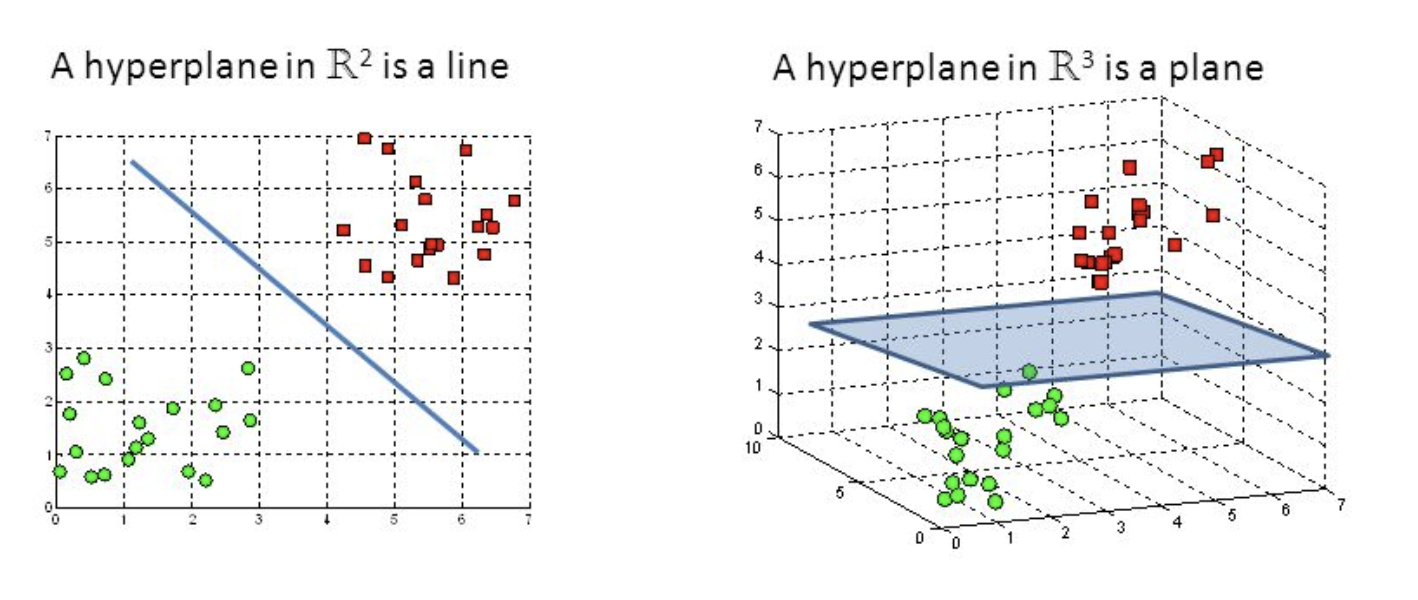
**SUPPORT VECTOR MACHINE?**

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

Possible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes and Support Vectors



Hyperplanes in 2D and 3D feature space

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



Support Vectors

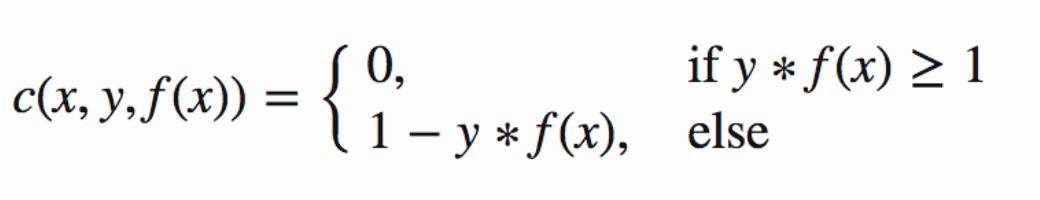
Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

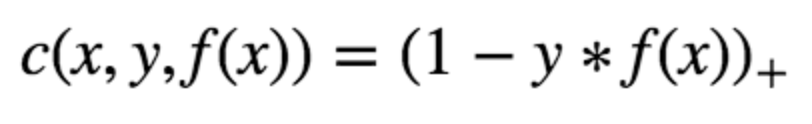
Large Margin Intuition

In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value(0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values([-1,1]) which acts as margin.

Cost Function and Gradient Updates

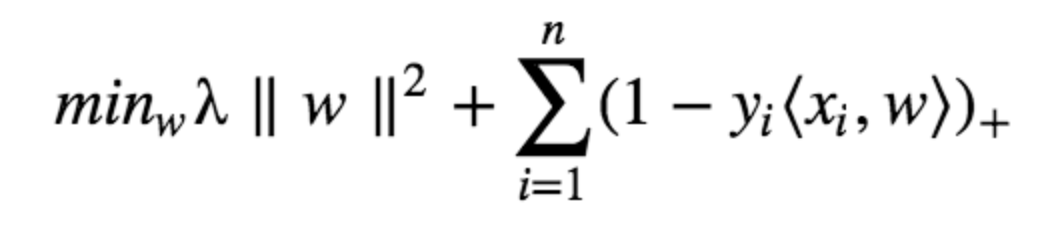
In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.





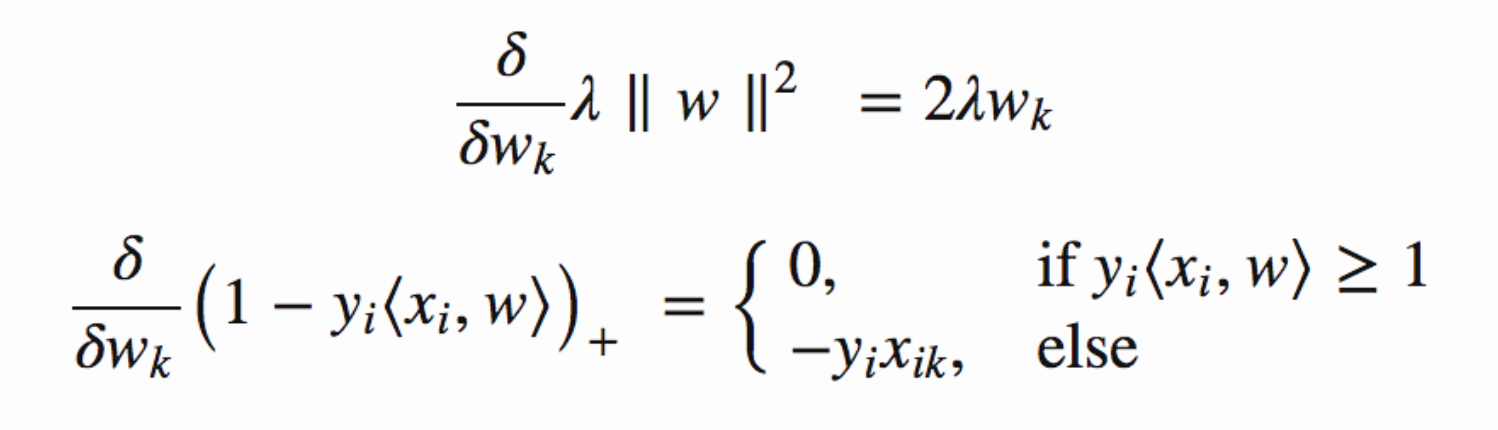
Hinge loss function (function on left can be represented as a function on the right)

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.



Loss function for SVM

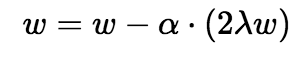
Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.



Gradients

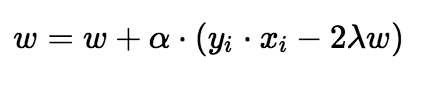
When there is no misclassification, i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.





Gradient Update — No misclassification

When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.



Gradient Update — Misclassification

**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.

**Sample Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.no** | **Test Case** | **Excepted Result** | **Result** | **Remarks(IF Fails)** |
| 1. | User Register | If User registration successfully. | Pass | If already user email exist then it fails. |
| 2. | User Login | If User name and password is correct then it will getting valid page. | Pass | Un Register Users will not logged in. |
| 3. | Start Data Pre process | Image has toupload | Pass | Corona Discharge Image must be select |
| 4 | Models Executed | For out four algorithms has executed and calculated 3 features | Pass | Models executed and predicted the results |
| 5 | Admin login | Admin can login with his login credential. If success he get his home page | Pass | Invalid login details will not allowed here |
| 6. | Admin can activate the register users | Admin can activate the register user id | Pass | If user id not found then it won’t login. |

**CONCLUSION**

We analyzed a hacking breach dataset from the points of view of the incidents inter-arrival time and the breach size, and showed that they both should be modeled by stochastic processes rather than distributions. The statistical models developed in this paper show satisfactory fitting and prediction accuracies. In particular, we propose using a copula-based approach to predict the joint probability that an incident with a certain magnitude of breach size will occur during a future period of time. Statistical tests show that the methodologies proposed in this paper are better than those which are presented in the literature, because the latter ignored both the temporal correlations and the dependence between the incidents interarrival times and the breach sizes. We conducted qualitative and quantitative analyses to draw further insights. We drew a set of cybersecurity insights, including that the threat of cyber hacking breach incidents is indeed getting worse in terms of their frequency, but not the magnitude of their damage. The methodology presented in this paper can be adopted or adapted to analyze datasets of a similar nature. There are many open problems that are left for future research. For example, it is both interesting and challenging to investigate how to predict the extremely large values and how to deal with missing data (i.e., breach incidents that are not reported). It is also worthwhile to estimate the exact occurring times of breach incidents. Finally, more research needs to be conducted towards understanding the predictability of breach incidents (i.e., the upper bound of prediction accuracy [24]). Acknowledgement. We thank the reviewers for their constructive comments that helped improve the paper. In Section V, we incorporated some insightful comments of one reviewer on how to connect the prediction models to real-world cyber defense quantitative risk management.

**REFERENCES**

[1] P. R. Clearinghouse, “Privacy rights clearinghouse’s chronology of data breaches.” https://www.privacyrights.org/data-breaches, Last accessed on November 9, 2017.

[2] I. T. R. Center, “http://www.idtheftcenter.org/2016databreaches.html.”

[3] C. R. Center, “Cybersecurity incidents.” https://www.opm.gov/ cybersecurity/cybersecurity-incidents/, Last accessed on November 9, 2017.

[4] I. Security, “https://www.ibm.com/security/data-breach/index.html.”

[5] N. . C. C. Study, “https://netdiligence.com/wp-content/uploads/2016/10/ P02 NetDiligence-2016-Cyber-Claims-Study-ONLINE.pdf.”

[6] M. Eling and W. Schnell, “What do we know about cyber risk and cyber risk insurance?,” The Journal of Risk Finance, vol. 17, no. 5, pp. 474– 491, 2016.

[7] T. Maillart and D. Sornette, “Heavy-tailed distribution of cyber-risks,” The European Physical Journal B-Condensed Matter and Complex Systems, vol. 75, no. 3, pp. 357–364, 2010.

[8] R. B. Security, “Datalossdb.” https://blog.datalossdb.org/, Last accessed on November 9, 2017.

[9] B. Edwards, S. Hofmeyr, and S. Forrest, “Hype and heavy tails: A closer look at data breaches,” Journal of Cybersecurity, vol. 2, no. 1, pp. 3–14, 2016.

[10] S. Wheatley, T. Maillart, and D. Sornette, “The extreme risk of personal data breaches and the erosion of privacy,” The European Physical Journal B, vol. 89, no. 1, p. 7, 2016.

[11] P. Embrechts, C. Kluppelberg, and T. Mikosch, ¨ Modelling extremal events: for insurance and finance, vol. 33. Springer Science & Business Media, 2013.

[12] R. Bohme and G. Kataria, “Models and measures for correlation ¨ in cyber-insurance.,” in Workshop on the Economics of Information Security (WEIS), 2006.

[13] H. Herath and T. Herath, “Copula-based actuarial model for pricing cyber-insurance policies,” Insurance Markets and Companies: Analyses and Actuarial Computations, vol. 2, no. 1, pp. 7–20, 2011.

[14] A. Mukhopadhyay, S. Chatterjee, D. Saha, A. Mahanti, and S. K. Sadhukhan, “Cyber-risk decision models: To insure it or not?,” Decision Support Systems, vol. 56, pp. 11–26, 2013.

[15] M. Xu and L. Hua, “https://www.soa.org/research-reports/2017/ cybersecurity-insurance/,” 2017