ARP SPOOFING DETECTOR

Design Document

30th March

Version Major

Serial No.	Name	Enrollment Number
1	Abhishek Kanhar	16114005
2	Anil Kumar	16114012
3	Anoop Singh	16114016
4	Hasanwala Faizal	16114030
5	Rahul Singh	16114052

1) Summary

- ## Spoof Detection Software

  As software will run in background on a PC of a employee. It should not interfere with normal tasks of that machine. Software should register itself as daemon which starts running at startup itself. Process of sniffing packets should not alter the PC's capability of interacting with network.

  ### Main characteristics of the spoof detection software:

  1) Listen to all outgoing packets. (Uses pcapy library)
  2) Listen to all incoming packets. (Uses pcapy library)
  3) Uses two pass filtering method to find spoofed/malformed packets.
  4) Uses response module to notify user and central web application.
  5) Saves credentials in installation directory.

  Software uses module approach to isolate independent working code.

  ### Modules:

  - Packet sniffing module
    - Sniffing all incoming and outgoing traffic from data link layer.
    - Passes ARP and ICMP packets for further analysis.
  - Packet parser module
    - Parses ARP, ICMP packets and return relevant data.
  - Spoof detection engine
    - Runs test on packet for find if packet is spoofed.
  - Response module
    - Warns users about attack.
    - Send alerts and stats to central web application.
  - Packet structure module
    - Contains packet structure of ARP and ICMP.
    - Can create ARP and ICMP packets.
  - Installation module
    - Setups software on user's PC.

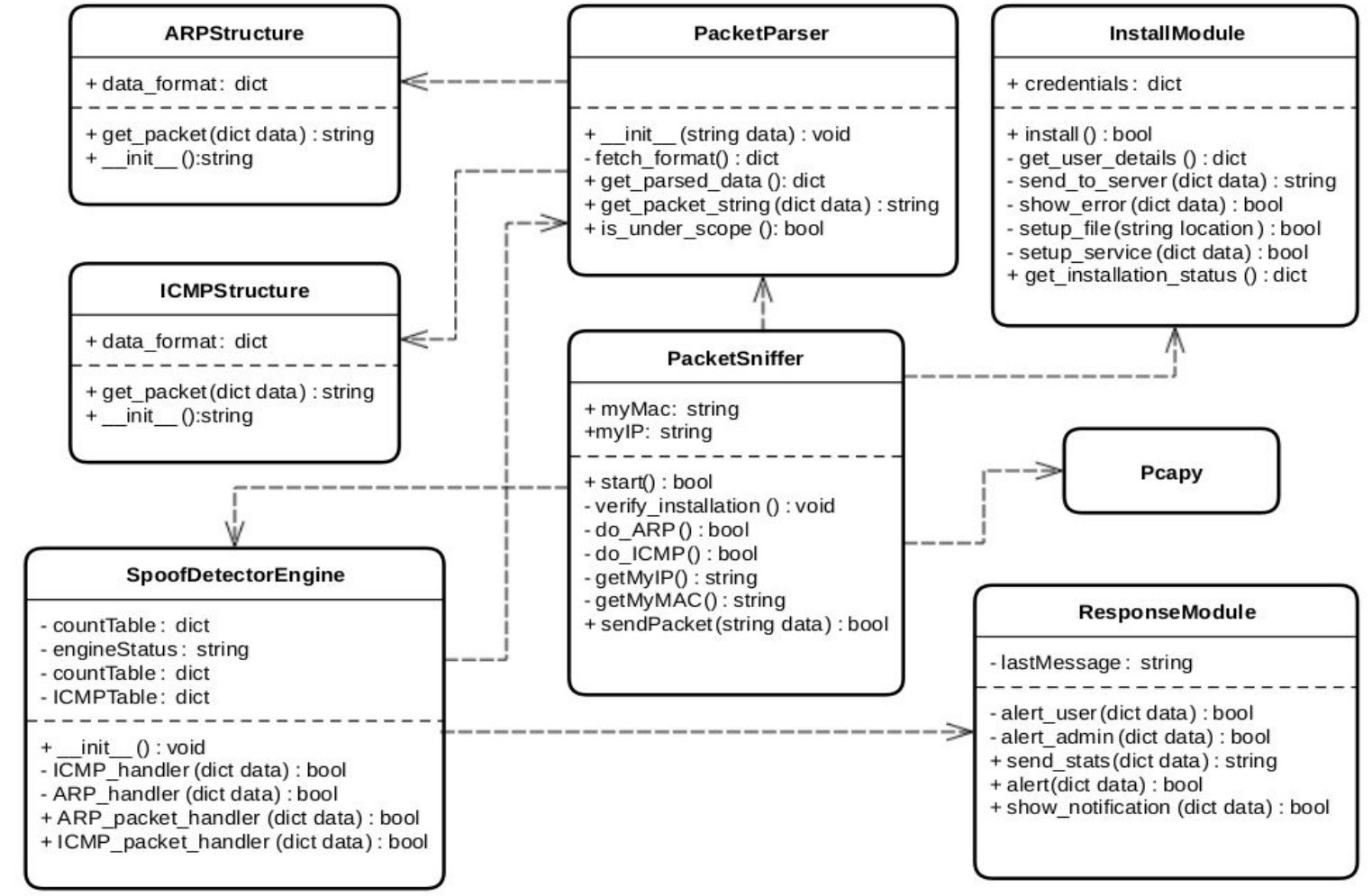
- Creates relevant services(daemon) on PC.

Algorithm:

```
Spoof detector engine

__init__():
   engineStatus = 'running'
   countTable = {}
   ICMPTable = {}
   myMac = getMyMac()
   myIP = getMyIP()

ARP_packet_handler(string data):
   arp_parsed_packet = PacketParser.get_parsed_data(string data)

   if arp_parsed_packet.request==true and arp_parsed_packet.eth.source==myMac:
     begin
          countTable[arp_parsed_packet.arp.target.IP] = true
          return True
     end
   else arp_parsed_packet.reply==true and arp_parsed_packet.eth.target in [myMac, broadcastMAC]:
     begin
          ARP_handler(arp_parsed_packet)
          return True
     end
   return False

ICMP_packet_handler(string data):
   icmp_parsed_packet = PacketParser.get_parsed_data(string data)

   icmp_parsed_packet.reply==true and icmp_parsed_packet.eth.target==myMac:
     begin
          ICMP_handler(arp_parsed_packet)
          return True
     end
   return False

ARP_handler(dict parsed_data):
   do packet consistency check
   if unsafe
     begin
```

```
                ResponseModule.alert(dict parsed_data)
                return False
        end

    if countTable[parsed_data.arp.sourceIP] == true:
        send ICMP echo packet to the sourceIP
        create a new thread to listen for reply
        if no reply in time:
                countTable[parsed_data.arp.sourceIP] = False
                ResponseModule.alert(dict parsed_data)
                return False
        else
                countTable[parsed_data.arp.sourceIP] = True
                declare entry as safe
    else
        ResponseModule.alert(dict parsed_data)
        return False

ICMP_handler(dict parsed_data):

    do packet consistency check

    if unsafe
        begin
                ResponseModule.alert(dict parsed_data)
                return False
        end

    if ICMPTable[parsed_data.icmp.sourceIP] exists
    begin
        ICMPTable[parsed_data.icmp.sourceIP].state = safe

        remove entry after 2 seconds
        return True
    end

    return False
```

Class diagram:



- ## Web-Based Backend

  The web-based backend will be built on Django=2.0.1 using DRF(Django Rest Framework). The backend will receive the data from the ARP Spoofing software and store it in the database. It will also allow the user to login and send the past infection history for himself or for the whole company(Depending on whether the user is an admin or a normal user) to the frontend where the appropriate charts will be plotted.

  Main characteristics of the Web-Based Backend:

  1. API to which the ARP software will be sending the data to store it in the database.
  2. API for registration
  3. API for login
  4. API for dashboard data.
  5. API for infection data for the previous month of the user(Normal user) or the Company(Admin user).

  The code will have the following significant features:

  - User Model
    - Will contain the data of the users (Employees or Admin).
    - Attributes of the Model
      - Basic Info (Name, Email, Password etc)
      - user_type (Admin or Normal User)
      - machine_status (Current status of the user's system(Infected/ Not Infected)
      - A method get_infections (Will return the infection data(contained in Infection Model objects) for the previous month)
  - Infection Model
    - Will contain all the infections
    - Attributes of the Model
      - Data/Time of infection
      - infected_user (Foreign Key to User Model)
      - case_status (Fixed or Not Fixed)
  - Message Model
    - Is meant for admin only

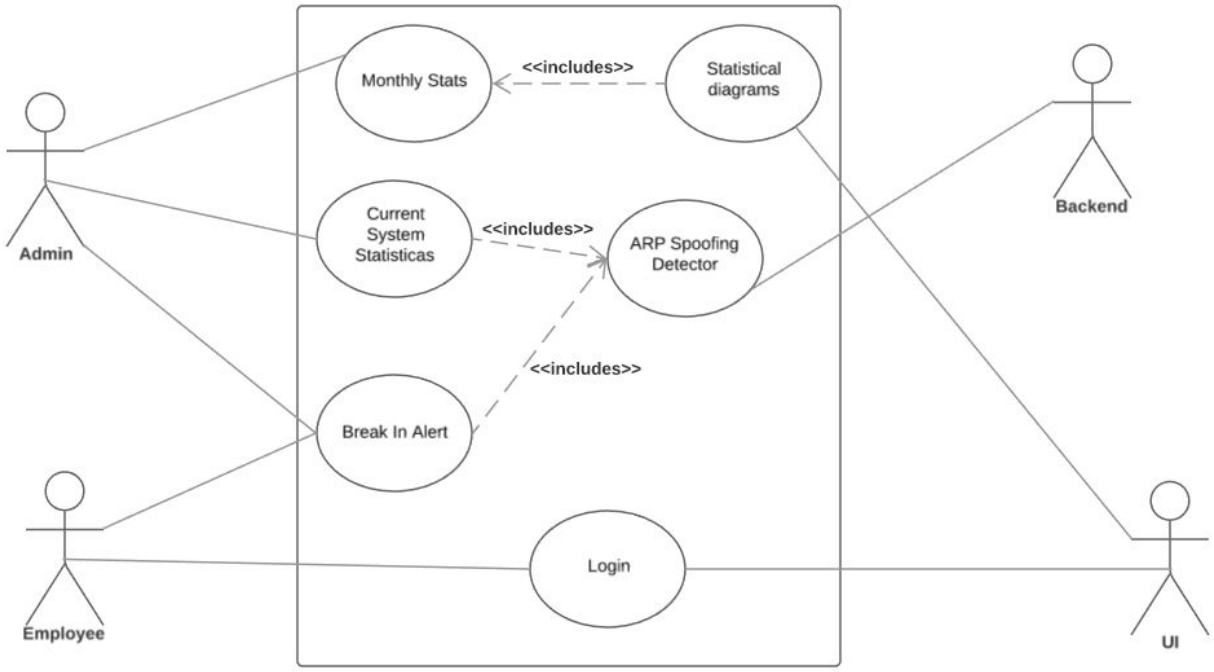
    - Will contain messages on breach detection or on fix of breach.
    - Attributes of the Model
        - message_type (infection or fix_successful)
        - User_id (ForeignKey to User Model)
        - Date/Time (Datetime of the infection/fix)
- APIs
    - SubmitInfectionData [POST]
        - Will be called by the ARP Spoofing Software.
        - Get the POST data and save it to database in Infections table.
        - Generate a message to admin to shut down/ disconnect the PC.
        - Request status codes
            - 200: Submitted successfully
            - 400: Bad Request, data not in format
    - UserRegister [POST]
        - Create a new user for the submitted POST data
        - Return the user data of the new user created in JSON.
        - Request status codes
            - 201: Created
            - 400: Bad Request, incorrect format or user already exists.
    - User_Login [POST]
        - Check the credentials and login a user
        - Return the user data in JSON.
        - Request status codes
            - 200: Ok, logged in
            - 404: User Not found, Incorrect credentials
    - GetHistory [GET]
        - Check if the request is made by an admin or an employee.
        - If admin, return all the infection data grouped by employee_id for the last month in JSON.
        - If employee, return infection data for his system for the last month in JSON.
        - Request status codes
            - 200: Ok, return data
            - 401: Unauthorized, user not logged in.
    - GetUser [GET and POST]
        - GET
            - Get user data of the logged in user.

- Get messages of the user (Employee) or all the messages (Admin).
- Request status codes
  - 200: Ok, Return data
  - 401: Unauthorized, user not logged in

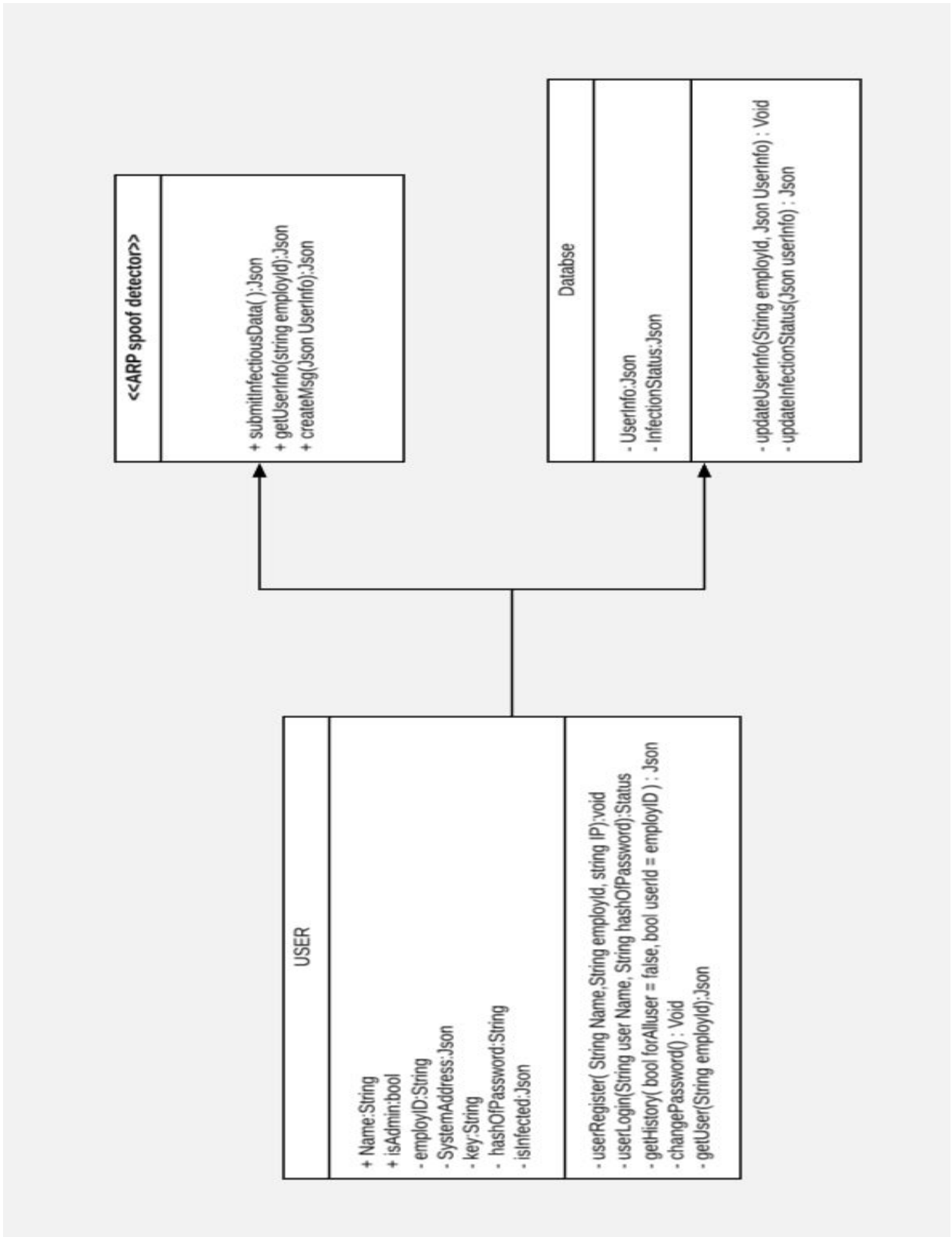
■ POST

- Allowed to admin only
- Get user data of the user with id provided in post
- Request status codes
  - 200: Ok, Return data
  - 401: Unauthorized, user is not admin
  - 404: User with requested id not found.

Use-Case Diagram-



Class Diagram -



- ## Web-Based Frontend

  The web-based frontend will be built using HTML5, Bootstrap 4 and Javascript(A bit of Jquery). Basically, the user will be visiting the frontend pages and these pages will call the Django Backend API's to get data and present it to the user in proper format.

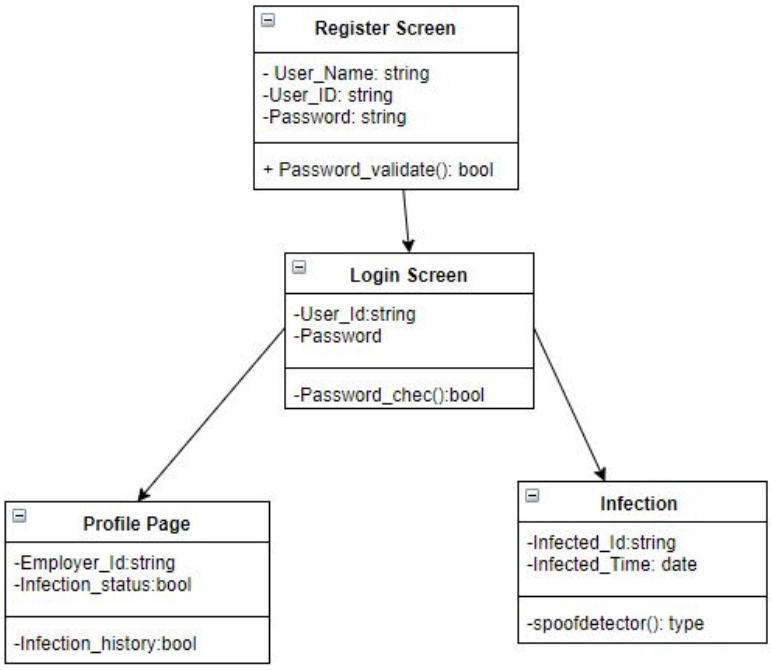
  Main characteristics of the web-based frontend:

  1. The Register screen which will ask for the Employer ID and Password.
  2. The Login screen with proper validations before the user can submit.
  3. The dashboard which will show the user's basic info and messages if any.
  4. The Infection History screen which will have the Infection data received from the GetHistory function of backend and plot it into meaningful graphs (Using JsCharts library)
  5. The profile page which will show the profile of a Employee.

  The web pages are described below in detail:

  - Register Screen
    - A basic register panel with password validations
  - Login Screen
    - A basic login panel with the correct handling of error messages provided by backend in case incorrect credentials are provided.
  - Dashboard
    - The page will have 2 tabs
      - Basic info tab.
      - Message panel tab (Which will show the messages)
  - Infection History Screen
    - It will have two components
      - A table with the infection history
      - A section for charts plotted by jscharts library
  - Profile Page
    - Show the basic data of the Employee.
    - Show the table of infection history

Class Diagram-



2) Questions

Why use ‘pcapy’?

Pcapy provied simple object oriented API. It also works with threads, which allows faster packet handling. It also works on both windows and unix operating systems.

3)Comments

None