Innopolis University

Secure Systems and Networks Research project

Submitted By:
Vasiliy Podtikhov,
Bulat Saifullin,
Timur Samigullin

Submitted To:
Rasheed Hussain
Azat Safin
Konstantin Urysov
Kirill Saltanov

December 9, 2016

Contents

Abstract

In modern days social networks become widely used. Practically almost all employers using them. But they can be used to formating public opinion in way not acceptable by company, or by accident share some confidentially information. This often happened because ordinary employee don't unaware of global company goals.

In this work we will try to link a social identity to an IP address by analysis of user traffics. This will help us to establish leakage, find disgruntled employees and change company politics to prevent this situations.

Introduction

Mapping IP address to account on social network is generally believed to be difficult for an individual with no dedicated infrastructure or privileged information. Social networks owners such as Vk.com and Facebook.com have this information, but they always hide it except in the case of a legal decision. But this information may be very handy in big corporations. In average 60% of employee actively use social networks [1]. And sometime employees post trade secret in social network, usually they use fake name. But if the employee go in account while he in the corporation's network mapping IP address to account on social network, can help us to find him.

Related Work

Today we widely used Netflow analysis for security reasons [2][3]. But only recently science works was introduced whom main goal was determine users action in social networks [4] [5]. Unfortunately method who helped us to identify user never was introduced. In this paper we tried to find a solution for this problem.

Research question

Our research question is:

- How to map IP addresses to profile in social network by analysing users activity?

To answer this, the following subquestions should be answered:

- Find connection between user traffic and profile changes.
- What sending data affects changes in profile?
- How to analysis user's net-flow traffics?
- How to analysis a profile in social network?

Methodologies

NetFlow is a feature that was introduced on Cisco routers that provides the ability to collect IP network traffic as it enters or exits an interface. By analyzing the data provided by NetFlow, a network administrator can determine things such as the source and destination of traffic, class of service, and the causes of congestion. A typical flow monitoring setup (using NetFlow) consists of three main components[6]:

- Flow exporter: aggregates packets into flows and exports flow records towards one or more flow collectors.
- Flow collector: responsible for reception, storage and pre-processing of flow data received from a flow exporter.
- Analysis application: analyzes received flow data in the context of intrusion detection or traffic profiling, for example.

We analysed netflow dumps in corporation environment and tried to check if connection was established in period of time and check presence of person in this time period on site. The main purpose is to find correlations between posted time and netflow traffic.

In this work each member of our research group takes responsibility for the most known social networks(vk.com, instagram.com, Facebook.com).

Team members contribution:

- V. Podtikhov - Facebook, Selenium+Phantomjs
- B. Saifullin - VK, pynfdump
- T. Samigullin - Instagram, Netflow sample.

VK

Vk.com is the most popular social network in Russia. Today vk has about 400 millions accounts. 80 millions visitors come to the site every day.

Getting data from vk

Vk.com provide great API for developers. API interface allows information to be received from the database vk.com with the help of http-requests to the special server. We do not need to know in detail how the base is constructed and from which table and field types it consists of. It is enough that API-request knows it. The request syntax and the type of data being returned are strictly determined by the service itself. For example, to receive data about the user with the ID number "396547478", we need to make a request of this type:

```
https://api.vk.com/method/users.get?user_ids=396547478&fields=online,
    last_seen&v=5.60
```

Lets have a look at the individual parts:

- `api.vk.com/methods` API server address
- `users.get` name of API VKontaktes method. Methods represent conditional commands that correspond with an operation from the database to receive, record or delete information. For example, users.get is a method to receive information about a user.
- `user_ids=396547478&fields=online,last_seen&v=5.60` parameter request.

In its response, the server returns JSON-object with the requested data (or a message about a mistake if something went wrong).
The response to our request looks like this:

```
{"response":[{"id":396547478,"first_name":"Ssn","last_name":"Project","online
    ":0,"last_seen":{"time":1480705322,"platform":7}}]}
```

Lets have a close look at fields that is interesting for us:

- `online` - information whether the user is online. Returned values: 1 - online, 0 - offline. If user utilizes a mobile application or site mobile version, it returns `online_mobile` additional field that includes 1.
- `last_seen` - last visit date. Returns `last_seen` object with the following fields:
  - `time` - last visit date (in Unix time).
  - `platform` - type of the platform that used for the last authorization.

In this paper we will use only two methods: users.get which returns detailed information on users and wall.get which returns a list of posts on a user wall or community wall.

Find IP range of vk

Vk.com has two IP range 87.240.128.0/18 and 95.213.0.0/18. To figure it out we first ping vk.com, when we take given IP and check information about it by using utility `whois` it gives us the network mask for this IP. Also we check that vk.com dose not have any another IP by using utility `dig`, we check that it don't return any another IP addresses.

Experimental results

For the experiments we created test profile https://vk.com/id396547478 and fill wall with some text posts in different times of days. For netflow collection we will use nfdump[12], and for real test we will use real netflow traffic from university Innopolis (IU).

Matching IP to profile by analyzing wall posts

Wall post can be different format(music, video, link, repost, photo, text message, etc). The text posts is the popular one and it is very hard for analysis, because it is very similar to personal text message that people send to each other every second in private dialogs. So you can not see difference between them in the netflow traffic. We are lucky if a user posts photo, it has 3 fields that we can analyze: post publish time, photo upload time, photo size. But we will look only at difficult case when all post it is small text message similar to the message that users exchange in dialogs. If we able match IP in this case we will able do it in easier case, because text message posts has only one field for analysis: post publish time. For analyze of user wall post and netflow traffic was written program in python language that take last n post, get timestamp from it and for each timestamp it create list with all unique IP that do request to vk in this second(±1 s.) Then it print IPs that was spotted in each list. In figure 5.1 you can see we correlation between matched IP and number of post.

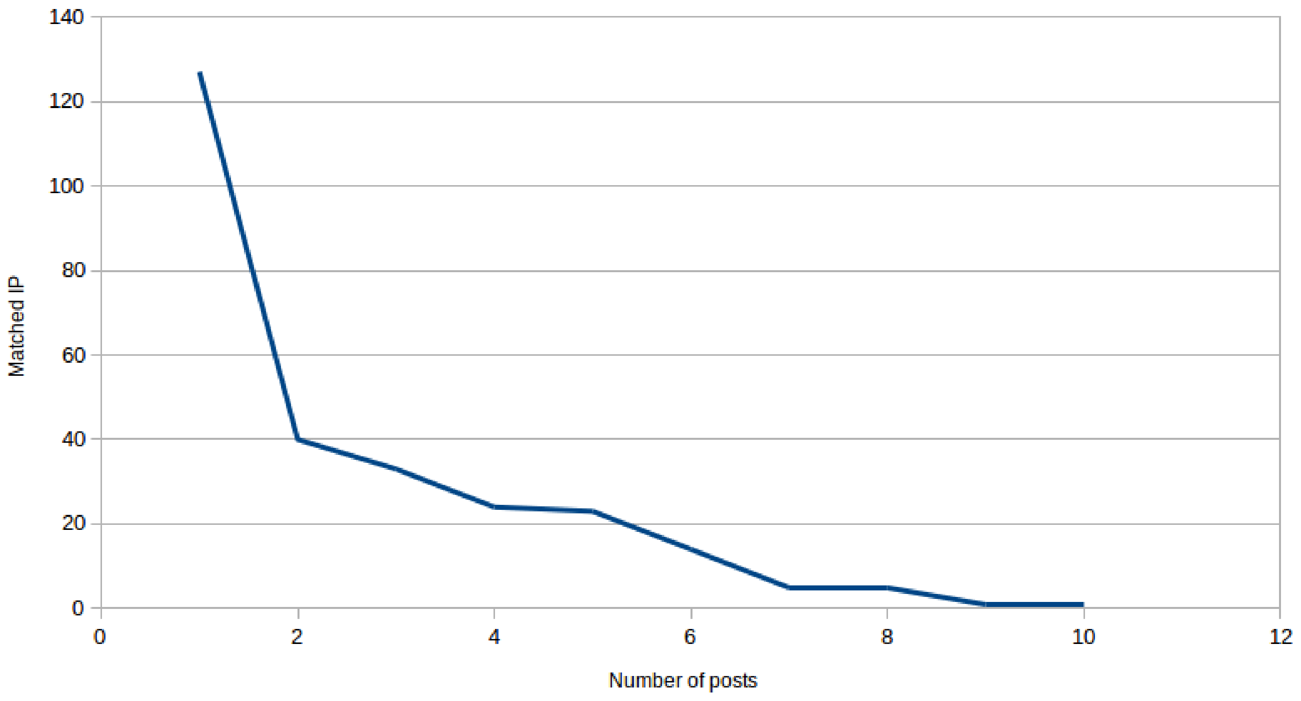


Figure 5.1: correlation between matched IP and number of post

Problems

The main problem we did not know that the user post posts from the same IP, only that we can know it was or PC or mobile app.

The second main problem that we have too many active user in the same time. As you can see from figure 5.2 in peak time we have about 350 unique IP do request to vk server.

The third problem, it can be less then 9 post with the same IP addresses (from diagram 5.1 we saw that we need more then 9 post to match IP). User can do post from different locations, not only in work place and DHCP can change IP too often.

Solutions for this problems will be consider in next section.

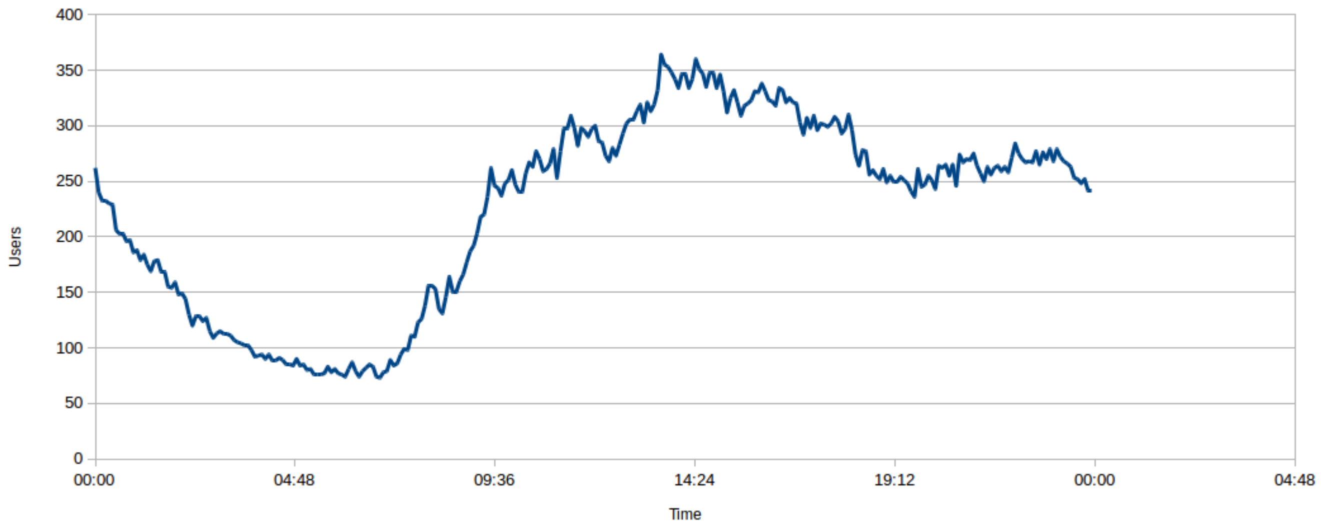


Figure 5.2: Number of vk active users in IU

Matching IP to profile while user is online

We can handle all problems that listed above by collecting enough data while user is online. It is highly unlikely that user will post 9 post in his wall per one connections, but he can comment under some post or do post in another wall. But it still hard to collect and analyze data. We can use social engineering and try to make chat with him, in this case each message can be considered as wall post message(each message has send timestamp like post publish(send) timestamp). But still it can fail if user will not answer.

After analyze of vk-API by me was found one bug in vk.com (or feature?). In vk page if user is offline in status bar we can see his last seen time, but this field is available from API even when user is online and it update after each activity of user(send message to smb, reload page, do comment or post, open smb page, etc). So we can see that user do something in the site, of curse we don't know what exactly he do, we can just guessing. But now we know what vk server receive some data from user in this exact time.

For exploit this bug we will write two python script, one will check user last seen field and write it in the file each time it change, second program will analyze the file. Because all timestamp will be get in short period it is possible that a lot of another users has connections in this time (they can just watch film in vk) so we will need more timestamp, but now it is not a problem. The correlation between matched IP and number of timestamp you can see in figure 5.3. In this figure was consider the case when user is texting with someone. We can see time of each message he send, usually people send 2-3 message per minute and we will get around 50 timestamps in 20 minutes. In this case we need only 42 time stamps to match the IP.

Program usage:

```
$ python user_follow_file.py [vk id]
$ python user_last_seen.py [vk id]
```

Output:

```
1 188.130.155.46 50/50 100%
2 10.241.1.2 48/50 96%
```

```
3 10.242.1.90 47/50 94%
4 10.90.131.101 46/50 92%
```

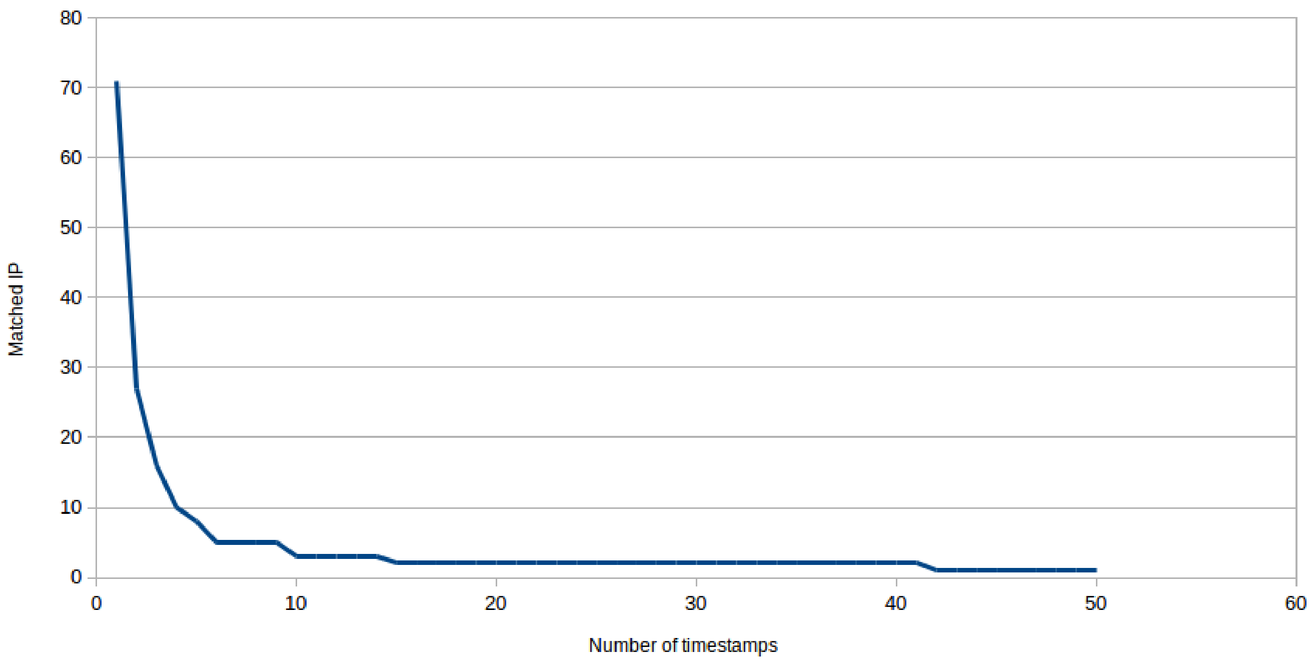


Figure 5.3: The correlation between matched IP and number of timestamp

Instagram

Instagram is an online mobile photo-sharing, video-sharing, and social networking service that enables its users to take pictures and videos, and share them either publicly or privately on the app, as well as through a variety of other social networking platforms, such as Facebook, Twitter, Tumblr, and Flickr. [7]

Getting data from Instagram

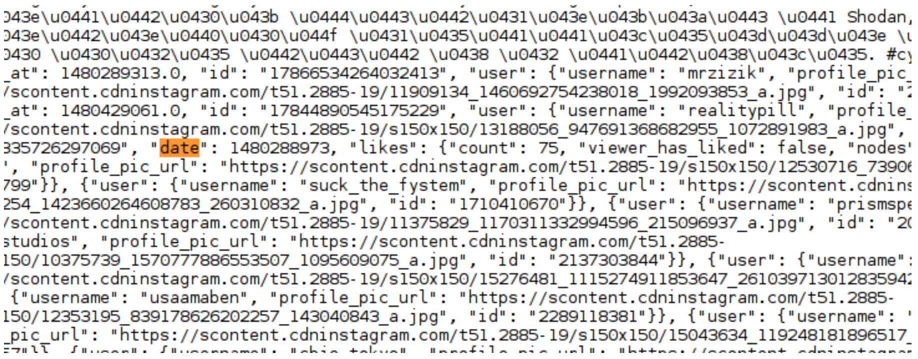
To find correlation between posted time and netflow traffic, we need to get exact time, when user post photo.

Fisrst, all information about post was getting through API functional of Instagram web-site. For this reason third-part application should be registered at instagram development page, and access-token should be got. Request method for getting info about specific post shown bellow:

```
https://api.instagram.com/v1/media/shortcode/BNPjFcrh22_?access_token
    =3955223166.3a064fe.2562f48363ac48f8b002f713fddeae2e
```

With testing accounts everything was fine, but when I tried to access to another real account I have a API error. Instagram API has one speciality: there is a sandbox environment for testing reasons, and for getting data from every page, first, he should be invited to sandbox and he should accept requests from application, even if page open for everybody. I think, application should not ask requests for viewing user's page.

The way out is to parse raw html page, and get data from it. In source of html page I saw, that there is raw json data.



)43e\u0441\u0442\u0430\u043b \u0444\u0443\u0442\u0431\u043e\u043b\u043a\u0443 \u0441 Shodan,
)43e\u0442\u043e\u0440\u0430\u044f \u0431\u0435\u0441\u0441\u043c\u0435\u043d\u043d\u043e \u
)430 \u0430\u0432\u0435 \u0442\u0443\u0442 \u0438 \u0432 \u0441\u0442\u0438\u043c\u0435. #cy
_at": 1480289313.0, "id": "17866534264032413", "user": {"username": "mrzizik", "profile_pic_
/scontent.cdninstagram.com/t51.2885-19/11909134_1460692754238018_1992093853_a.jpg", "id": "
at": 1480429061.0, "id": "17844890545175229", "user": {"username": "realitypill", "profile
/scontent.cdninstagram.com/t51.2885-19/s150x150/13188056_947691368682955_1072891983_a.jpg",
335726297069", "date": 1480288973, "likes": {"count": 75, "viewer_has_liked": false, "nodes'
', "profile_pic_url": "https://scontent.cdninstagram.com/t51.2885-19/s150x150/12530716_7390
799"}}, {"user": {"username": "suck_the_fystem", "profile_pic_url": "https://scontent.cdnins
254_1423660264608783_260310832_a.jpg", "id": "1710410670"}}, {"user": {"username": "prismspe
/scontent.cdninstagram.com/t51.2885-19/11375829_1170311332994596_215096937_a.jpg", "id": "2C
studios", "profile_pic_url": "https://scontent.cdninstagram.com/t51.2885-
150/10375739_1570777886553507_1095609075_a.jpg", "id": "2137303844"}}, {"user": {"username":
/scontent.cdninstagram.com/t51.2885-19/s150x150/15276481_1115274911853647_261039713012835942
 {"username": "usaamaben", "profile_pic_url": "https://scontent.cdninstagram.com/t51.2885-
150/12353195_839178626202257_143040843_a.jpg", "id": "2289118381"}}, {"user": {"username": '
_pic_url": "https://scontent.cdninstagram.com/t51.2885-19/s150x150/15043634_119248181896517_

It means, that I can simply get all data from specific web page even without any authorization. For parsing I used xpath and re library from python.

```
script = html.xpath('//script[contains(., "window._sharedData")]/text()')[0]
data = re.search(r"window._sharedData = (.*?);\$", script).group(1)
data = json.loads(data)
```

All data was stored in **data** array.

Finding range of Instagram IP

We ask to give us all university netflow traffic for analyzing, it is stored on our computers. The difference of Instagram in comparison with other social network, is that Instagram has very narrow area of usage. In this network users can only add and comment their photos. Every user's action connected with photos. It means, that instagram need less computer capacity, than other networks. And also Instagram now belong to facebook. The problem was to find exact range of Instagram IP addresses.

Instagram hasn't got it's own autonomous system, but most number of requests send to 31.13.93.72 or 31.13.92.32 or 31.13.93.54. For the first sight we can assume that we should only restrict 31.13.92.0/24 or 31.13.93.0/24. But that is not a solution, because not every address in this network belongs to Instagram.

So, I deided to find all Instagram IP addresses by myself. I extract all unique destination IP addresses from netflow traffic and get 106 MB file with 6291215 lines. I write a script to revesre-resolve all IP addresses and find Instagram string in it. It was a bad idea. Script worked for three days, but process was not finished. And during this I find another solution for this. I decided to use all 31.13.0.0/16 network, and resolve Instagram IP addresses after extracting data from netflow.

Extracting data from netflow

In my script I used only raw nfdump. You can see the whole string filter bellow:

```
\$ nfdump -R /var/flows/MYROUTER "dst net 31.13.0.0/16 and port 443" -o csv -
    t 2016/11/27.22:47:26-2016/11/27.22:47:56 -s record/bytes | head -n -3 |
    sed '1d'
```

The result of such execution:

```
('2016-11-29 15:57:36', '10.240.20.237', '31.13.72.53', '40166', '255667')
('2016-11-29 15:57:22', '10.91.35.114', '31.13.72.8', '57339', '15368')
('2016-11-29 15:57:24', '10.240.20.133', '31.13.92.11', '45988', '8917')
('2016-11-29 15:57:47', '10.240.16.55', '31.13.92.51', '54943', '8843')
('2016-11-29 15:57:35', '10.240.18.181', '31.13.72.53', '62515', '6779')
('2016-11-29 15:57:33', '10.240.16.208', '31.13.72.53', '37487', '5127')
('2016-11-29 15:57:28', '10.242.1.233', '31.13.72.12', '38472', '5122')
```

After filtering only Instagram IP addresses it became:

```
('2016-11-29 15:57:36', '10.240.20.237', '31.13.72.53', '40166', '255667')
('2016-11-29 15:57:47', '10.240.16.55', '31.13.92.51', '54943', '8843')
('2016-11-29 15:57:35', '10.240.18.181', '31.13.72.53', '62515', '6779')
('2016-11-29 15:57:33', '10.240.16.208', '31.13.72.53', '37487', '5127')
```

The main thing, that I should solve is to find necessary time range. At the moment when user post photo to his Instagram account, long tcp connection should occur, so this connection can start early or end later, that exact post time. With empirical analysis I detect that I should take 20 seconds offset before exact time post and 10 sec offset after timepost. This range give valid results.

Experimental results

In my part of project I trying to map internal IP address on company network to the post in Instagram. To accomplish this I should take time range in 30 seconds, with 20 seconds offset between exact post time and 10 seconds offset after post time. You can see the whole log of program bellow:

```
\$ bin/python netflow.py
URL to analyze: https://www.instagram.com/p/BNZRbaVgTbv
The post was created at 2016/11/29.12:57:40 GMT
Getting the timerange from netflow dumps: before offset = 20 after offset =
    10 GMT offset of netflow server = 3
nfdump -R /var/flows/MYROUTER "dst net 31.13.0.0/16 and port 443" -o csv -t
    2016/11/29.15:57:20-2016/11/29.15:57:50 -s record/bytes | head -n -3 | sed
     '1d'

 At this period of time the following IP addresses was going to instagram
     website:

('2016-11-29 15:57:36', '10.240.20.237', '31.13.72.53', '40166', '255667')
('2016-11-29 15:57:47', '10.240.16.55', '31.13.92.51', '54943', '8843')
('2016-11-29 15:57:35', '10.240.18.181', '31.13.72.53', '62515', '6779')
('2016-11-29 15:57:33', '10.240.16.208', '31.13.72.53', '37487', '5127')
('2016-11-29 15:57:34', '10.240.16.157', '31.13.93.52', '53044', '4568')
('2016-11-29 15:57:27', '10.91.42.54', '31.13.92.51', '59556', '4269')
('2016-11-29 15:57:30', '10.240.23.33', '31.13.92.51', '60524', '4019')

 But only following IP addresses get enough bytes from the website:

('2016-11-29 15:57:36', '10.240.20.237', '31.13.72.53', '40166', '255667')
```

Facebook

Facebook today it's the largest and most famous social network with more than one billion active users per month [8]. Before April 30 2014 it was quite easy to get information about public available user posts. However when Facebook upgraded Graph API to version 2 all applications now must get **User Access Token** token with **user_posts** permission. If application don't get this permission empty data array will be returned.

It's obvious what we try to reduce user interaction to minimum in our work. To do this we decide analyze HTML page.

Facebook HTML page

All posts in facebook timeline returned in `<dig>` tag with "userContentWrapper _5pcr" class. We are interested in nested tag `<abbr>` with class _5ptz, this tag contain attribute **data-utime** which in turn contain timestamps of posts in Unix time format. To get this tags in Python code I use combination of Selenium [9] and Phantom.js [10].

Program flow

As input my program take link to Facebook account. To get all timestamps from user page we must be log in into Facebook. After we successfully log in, we try to download all dynamically loadable posts of that user. To do so we scroll down page until java-scripts download all available information. After that we get list with data-utime attributes. Now we have information about user time presence on page.

Netflow: filtering and compare

After we get all data-utime attribute, we should start thinking about reducing Netflow records. First step it's leave only those records which time coincidence with time presence. We can do this with pynfdump package for Python 2. We get all files with Netflow records with a five-minute offset relative to the time standing on the site, this needed because all Netflow records saved in files with the corresponding date of the name. Netflow collector save this files every five minutes, so all names multiple of five minutes.

After getting all required files we additionally reduce amount of Netflow records. To do so we must know IP range of Facebook. As documented on Facebook's Developer site [11], autonomous system AS32934 belongs to Facebook. To find IP range list we can use whois program:

```
whois -h whois.radb.net -- '-i origin AS32934' \textbar grep \^{}route
```

Now we get only those records for which time coincides with data-utime attribute plus small offset necessary to compensate for the time delay in the network.

On each value of data-utime attribute we get set of possible IP addresses. After finding all corresponding to data-utime IP sets we build massive with next structure: `<IP address> - <Number of meetings in sets>`, and sort this massive by number of meetings. On top we get IP addresses which correspond with the account more likely.

Experimental results

As may be seen, Fasebook appear very restricted and closed social network, at least for third-party application. Facebook provide little information about his user for unauthorized applications. All we get from Facebook site it's only time posting of publicly available posts.

Conclusions

In our work we find solution to map IP address of internal network user for following social networks: Facebook, Vk and Instagram. The solution is suitable for enterprise companies that has its own network infrastructure and who care about information security and data losses. We can identify the IP address of user with high probability.

Bibliography

[1] Dont Let Your Sensitive Information Leak onto Social Networks: Information Leakage with Zecurion Zgate

[2] W. Zhengi, W. Xinyu NetFlow Based Intrusion Detection System

[3] G. Munz, G. Carle Real-time Analysis of Flow Data for Network Attack Detection

[4] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing user navigation and interactions in on-line social networks. Inf. Sci., July 2012.

[5] N. V. Verde, G. Ateniese, E.Gabrielli, L. V. Mancini. No natd user left behind: Fingerprinting users behind NAT from netflow records alone, 2014

[6] Hofstede, Rick; Celeda, Pavel; Trammell, Brian; Drago, Idilio; Sadre, Ramin; Sperotto, Anna; Pras, Aiko. Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX". IEEE Communications Surveys Tutorials. IEEE Communications Society.

[7] http://www.businessinsider.com/instagram-2010-11

[8] https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/ - Statistic about amount of users in social networks

[9] http://www.seleniumhq.org/ - main page of Selenium browser automation project

[10] http://phantomjs.org/ - main page of headless WebKit

[11] https://developers.facebook.com/docs/sharing/webmasters/crawler - information about Facebook for developers

[12] https://pythonhosted.org/pynfdump/ - official site of package pynfdump

Our programms

Facebook code

```
    import time
    import pynfdump
    from selenium import webdriver
    import datetime
    from datetime import timedelta
    import operator

    """ list of facebook IP
     can be received "whois -h whois.radb.net -- '-i origin AS32934' | awk '/^route:/ {print $2;}' | sort | uniq"""
    facebookIP = ("103.4.96.0/22", "129.134.0.0/16", "157.240.0.0/16", "157.240.0.0/24",
    "157.240.10.0/24", "157.240.1.0/24", "157.240.2.0/24", "157.240.3.0/24",
    "157.240.4.0/24", "157.240.5.0/24", "157.240.6.0/24", "157.240.7.0/24",
    "157.240.8.0/24", "157.240.9.0/24", "173.252.64.0/18", "173.252.64.0/19",
    "173.252.70.0/24", "173.252.96.0/19", "179.60.192.0/22", "179.60.192.0/24",
    "179.60.193.0/24", "179.60.194.0/24", "179.60.195.0/24", "185.60.216.0/22",
    "185.60.216.0/24", "185.60.217.0/24", "185.60.218.0/24", "185.60.219.0/24",
    "204.15.20.0/22", "31.13.24.0/21", "31.13.64.0/18", "31.13.64.0/19",
    "31.13.64.0/24", "31.13.65.0/24", "31.13.66.0/24", "31.13.67.0/24", "31.13.68.0/24",
    "31.13.69.0/24", "31.13.70.0/24", "31.13.71.0/24", "31.13.72.0/24", "31.13.73.0/24",
    "31.13.74.0/24", "31.13.75.0/24", "31.13.76.0/24", "31.13.77.0/24", "31.13.78.0/24",
    "31.13.79.0/24", "31.13.80.0/24", "31.13.81.0/24", "31.13.82.0/24", "31.13.83.0/24",
    "31.13.84.0/24", "31.13.85.0/24", "31.13.86.0/24", "31.13.87.0/24", "31.13.88.0/24",
    "31.13.89.0/24", "31.13.90.0/24", "31.13.91.0/24", "31.13.92.0/24", "31.13.93.0/24",
    "31.13.94.0/24", "31.13.95.0/24", "31.13.96.0/19", "45.64.40.0/22", "66.220.144.0/20",
    "66.220.144.0/21", "66.220.149.11/16", "66.220.152.0/21", "66.220.158.11/16",
    "66.220.159.0/24", "69.171.224.0/19", "69.171.224.0/20", "69.171.224.37/16",
    "69.171.229.11/16", "69.171.239.0/24", "69.171.240.0/20", "69.171.242.11/16",
    "69.171.253.0/24", "69.171.255.0/24", "69.63.176.0/20", "69.63.176.0/21",
    "69.63.176.0/24", "69.63.178.0/24", "69.63.184.0/21", "69.63.186.0/24", "74.119.76.0/22")
    url = "https://www.facebook.com/lonesexternals"  # page of person
    username = "email"  # class of login field on facebook
    password = "pass"  # class of password field on facebook
    login = "u_0_0"  # class of login button
    timestampClass = "_5ptz"  # class of time field
    timeName = "data-utime"  # attribute with utime
    netflowStore = "/mnt/flows"
    routerList = ['ROUTER']
    dateConverted = list()  # utime into string
    netflowDate = 1480317600  # earliest netflow time
    ipList = list()
    userList = dict()
    delay = 1  # max delay between connection ends and data posting
    precise = 0.8  # reasonable deviation


    def gettimelist(address):
        """ Parse given page in virtual browser and get list of date"""
        browser = webdriver.\
            PhantomJS(executable_path='./phantomjs-2.1.1-linux-x86_64/bin/
                phantomjs')
        browser.get(address)
        elem = browser.find_element_by_name(username)
        elem.send_keys("mr.external@mail.ru")
        elem = browser.find_element_by_name(password)
        elem.send_keys("248163264")
        elem = browser.find_element_by_id(login)
        elem.submit()
        time.sleep(3)
        last_height = browser.execute_script("return document.body.
            scrollHeight")
        # scroll page to bottom, to download all dynamical updates
        while True:
            browser.execute_script("window.scrollTo(0, document.body.
                scrollHeight);")
            time.sleep(1)
            new_height = browser.execute_script("return document.body.
                scrollHeight")
            if new_height == last_height:
                break
            else:
                last_height = new_height
        elem = browser.find_elements_by_class_name(timestampClass)
        datelist = list()
        for timestamp in elem:
            datelist.append(timestamp.get_attribute(timeName))
        browser.quit()
        return datelist


    dateList = gettimelist(url)
    # building search string from known facebook IP for netflow parser
    searchString = "dst net " + facebookIP[0]  # + " or src net "+ facebookIP
        [0]
    for i in facebookIP:
        searchString += " or dst net " + i  # + " or src net "+ i
    # convert time to string format if time greater than first time in
        netflow
    for i in dateList:
        if int(i) > netflowDate:
            dateConverted.append(datetime.datetime.fromtimestamp(int(i)))
        else:
            break
    netflowSet = pynfdump.Dumper(netflowStore, sources=routerList)
    # get list of list  with suitable ip for each known date
    for i in range(len(dateConverted)):
        ipList.append(list())
        netflowSet.set_where(
            start=(dateConverted[i] - timedelta(minutes=dateConverted[i].
                minute % 5)).strftime('%Y-%m-%d %H:%M'),
            end=(dateConverted[i] + timedelta(minutes=5 - dateConverted[i].
                minute % 5)).strftime('%Y-%m-%d %H:%M'))
        ipSet = netflowSet.search(searchString)
        for r in ipSet:
            # get netflow in time [utime;utime+delay]
            if (int(r['first'].strftime('%s'))) <= int(dateConverted[i].
                strftime('%s')) <= (
                        int(r['last'].strftime('%s')) + delay):
                ipList[i].append(r['srcip'])
    for i in range(len(ipList) - 1):
        # create dictionary with all meet IP address and his frequency
        temp = set(ipList[i])
        for k in temp:
            if k in userList:
                userList[k] += 1
            else:
                userList[k] = 1
    for i in userList:
        sorted_x = sorted(userList.items(), key=operator.itemgetter(1),
            reverse=True)
    for i in sorted_x:
        if i[1] > sorted_x[0][1] * precise:
            print i
        else:
            break
```

Instagram

```
#!/usr/bin/env python
import requests
import pynfdump
import json
from xml.dom import minidom
from pprint import pprint
from datetime import datetime, timedelta
import time
import os
import csv
import socket
from terminaltables import AsciiTable
import json
import re
from lxml import html

netflow_GMT = 3
```

```
first_offset = 10
last_offset = 20

client_id = "3a064fe6d34a42d2a44c013b8ddd5ef7"
client_secret = "712a7789c81942d086326bcd9dd8d272"
access_token = "3955223166.3a064fe.2562f48363ac48f8b002f713fddeae2e"
shortcode = "BNZWZtiAsT3"
url = "https://www.instagram.com/p/BNYpYbSAcCz"
html = html.fromstring(requests.get(url).content)
script = html.xpath('//script[contains(., "window._sharedData")]/text()')[0]
data = re.search(r"window._sharedData = (.*?);$", script).group(1)
data = json.loads(data)
pprint(data['entry_data']['PostPage'][0]['media']['date'])
created_time = data['entry_data']['PostPage'][0]['media']['date']
print "URL to analyze: " + url
print "The post was created at " + str(datetime.fromtimestamp(int(
    created_time) + time.timezone).strftime('%Y/%m/%d.%H:%M:%S')) + " GMT"
first_time = created_time - first_offset
last_time = created_time + last_offset
print "Getting the timerange from netflow dumps: before offset = " + str(
    first_offset) + " after offset = " + str(last_offset) + " GMT offset of
    netflow server = " + str(netflow_GMT)
nfdump_string = "nfdump -R /var/flows/MYROUTER \"dst net 31.13.0.0/16 and
    port 443\" -o csv -t " + str(datetime.fromtimestamp(first_time + time.
    timezone + netflow_GMT * 3600).strftime('%Y/%m/%d.%H:%M:%S')) + "-" + str(
    datetime.fromtimestamp(last_time + time.timezone + netflow_GMT * 3600).
    strftime('%Y/%m/%d.%H:%M:%S')) + " -s record/bytes | head -n -3 | sed '1d
    '"
print nfdump_string
nfdump = os.popen(nfdump_string).read()
nfdump_output = csv.reader(nfdump.split('\n'), delimiter=',')
print "\n At this period of time the following IP addresses was going to
    instagram website: \n"
for row in nfdump_output:
        try:
                hostname = socket.gethostbyaddr(row[4])[0]
                if "instagram" in hostname:
                        print(row[0], row[3],row[4], row[5], row[12])
        except:
                shortcode = ""
print "\n But only following ip addresses get enough bytes from the website:
    \n"
nfdump_output = csv.reader(nfdump.split('\n'), delimiter=',')
for row in nfdump_output:
        try:
                hostname = socket.gethostbyaddr(row[4])[0]
                if "instagram" in hostname and int(row[12])>20000:
                        print(row[0], row[3],row[4], row[5], row[12])
        except:
                continue
```

Vk

Wall.py

```
#!/usr/bin/env python


import sys
import vk
import datetime
import time
import signal
import pynfdump
import operator

POST = 11
SHOW = 1  # SHOW < POST
PACKETS = 100  # Filter parameter 20
FILTER = "dst net 87.240.128.0/18 or dst net 95.213.0.0/18"  # vk ip range

def main():
    if(len(sys.argv) < 2):
        uid = input("id: ")
    else:
        uid = sys.argv[1]

    nfStore = "/mnt/flows"
    routerList = ['ROUTER']

    session = vk.Session()
    api = vk.API(session)
    user = api.users.get(user_ids=uid, fields="online,last_seen")
    # print user[0]['last_seen']['time']
    # print user[0]['online']

    last_seen = int(user[0]['last_seen']['time'])

    print (datetime.datetime.fromtimestamp(last_seen).strftime('%Y-%m-%d %H:%
       M:%S'))

    wall = api.wall.get(owner_id=uid, count=POST, filter="owner")

    last_post = []

    for i in range(0, POST):
        last_post.append(wall[1+i]['date'])
        print (str(i+1)+'. ' + datetime.datetime.fromtimestamp(last_post[i]).
            strftime('%Y-%m-%d %H:%M:%S'))




    # wall
    list_wall=[]
    for i in range(0, POST):
        d.set_where(
            start=datetime.datetime.fromtimestamp(last_post[i]).strftime('%Y-%m-%d %H:') + str(calc_start(last_post[i])),
            end=datetime.datetime.fromtimestamp(last_post[i]+300).strftime('%Y-%m-%d %H:') + str(calc_start(last_post[i]+300)))
        records = d.search(FILTER)
        list_w = []
        for r in records:
            if (int(r['first'].strftime('%s')) - 0) <= last_post[i] and last_post[i] <= (int(r['last'].strftime('%s')) + 1):
                if r['srcip'] not in list_w and r['packets'] < PACKETS:
                    list_w.append(r['srcip'])
        print len(list_w), "matches by post :", i+1
        #print list_w
        list_wall.append(list_w)


    user_ip = dict()
    for i in range(0, POST):
        for ip in list_wall[i]:
            if ip not in user_ip:
                user_ip[ip] = 0
            else:
                user_ip[ip] += 1
    print len(user_ip), "total number ip in wall"



    sorted_user_ip = sorted(user_ip.items(), key=operator.itemgetter(1), reverse=True)

    c = 0
    for ip in sorted_user_ip:
        if ip[1] >= (POST - SHOW):
            c += 1
            print c, "%-13s" % ip[0], "\t%i/%i" % (ip[1] + 1, POST), "%i%s" % ((ip[1] + 2.0)/(POST+1) * 100, '%')



def calc_start(time):
    minute_start1 = (int(datetime.datetime.fromtimestamp(time).strftime('%M')) / 5) * 5
    return minute_start1


def signal_handler(signal, frame):
    print('You pressed Ctrl+C!')
    sys.exit(0)


if __name__ == "__main__":
    signal.signal(signal.SIGINT, signal_handler)
    sys.exit(main())
```

user_follow_file.py

```
#!/usr/bin/env python

from __future__ import print_function

import sys
import vk
import datetime
import time
import signal
import pynfdump
import time
import glob

FLAG = True

def main():
    if(len(sys.argv) < 2):
        uid = input("id: ")
    else:
        uid = sys.argv[1]

    session = vk.Session()
    api = vk.API(session)
    user = api.users.get(user_ids=uid, fields="online,last_seen")
    filename = user[0]['first_name']+"_"+user[0]['last_name']+".txt"
    # print (filename)

    if int(user[0]['online']) == 1:
        print (user[0]['first_name']+" is "+'online')
    else:
        print (user[0]['first_name']+" is "'offline')

    last_seen = int(user[0]['last_seen']['time'])

    print (datetime.datetime.fromtimestamp(last_seen).strftime('%Y-%m-%d %H:%
        M:%S'))

    # f = open("last_seen_time.txt", "a")

    # print(last_seen, file=f)
    old = last_seen
    platform = user[0]['last_seen']['platform']
    print ("Platform:", platform)
    global FLAG
    while FLAG:  # and user[0]['last_seen']['platform'] == 7 : # full web
        version
        try:
            user = api.users.get(user_ids=uid, fields="online,last_seen")
        except:
            print ("Some thing wrong with api:(")

        if platform != user[0]['last_seen']['platform']:
            print(user[0]['first_name'], "change the device")
            break

        new = user[0]['last_seen']['time']
        if old < new:
            f = open(filename, "a")

            print(new, file=f)

            f.close()
            print (datetime.datetime.fromtimestamp(new).strftime('%Y-%m-%d %H
                :%M:%S'), "(%i)" % new)
            old = new

        time.sleep(0.9)
        if int(user[0]['online']) == 0:
            print ("User offline")
            time.sleep(5.9)
            #FLAG = False


def signal_handler(signal, frame):
    global FLAG
    FLAG = False
    print('You pressed Ctrl+C!')
    # sys.exit(0)


if __name__ == "__main__":
    signal.signal(signal.SIGINT, signal_handler)
    sys.exit(main())
```

last_seen_analaser.py

```
#!/usr/bin/env python

import sys
import vk
import datetime
import time
import signal
import pynfdump
import glob
import operator


SHOW = 5
PACKETS = 50  # Filter parameter

def main():
    if len(sys.argv) < 2:
        uid = input("id: ")
    else:
        uid = sys.argv[1]

    nfStore = "/mnt/flows"
    routerList = ['ROUTER']

    session = vk.Session()
    api = vk.API(session)

    filename='last_seen_time.txt'
    user = api.users.get(user_ids=uid, fields="online,last_seen")
    filename = user[0]['first_name']+"_"+user[0]['last_name']+".txt"

    lines = [line.rstrip('\n') for line in open(filename)]
    print lines

    d = pynfdump.Dumper(nfStore, sources=routerList)

    print len(lines), "lines in file"

    list_wall = []
    for t in lines:
        list_w = []
        d.set_where(
            start=datetime.datetime.fromtimestamp(int(t)).strftime('%Y-%m-%d %H:') + str(calc_start(int(t))),
            end=datetime.datetime.fromtimestamp(int(t)+300).strftime('%Y-%m-%d %H:') + str(calc_start(int(t)+300)))
        records = d.search("dst net 87.240.128.0/18 or dst net 95.213.0.0/18")
        for r in records:
            if (int(r['first'].strftime('%s')) - 0) <= int(t) <= (int(r['last'].strftime('%s')) + 1):
                if r['srcip'] not in list_w and r['packets'] < PACKETS:
                    list_w.append(r['srcip'])
        print len(list_w), "matches by time:", datetime.datetime.fromtimestamp(int(t)).strftime('%Y-%m-%d %H:%M:%S')
        # print list_w
        list_wall.append(list_w)

    user_ip = dict()
    for i in range(0, len(lines)):
        for ip in list_wall[i]:
            if ip not in user_ip:
                user_ip[ip] = 0
            else:
                user_ip[ip] += 1
    print len(user_ip), "total number ip in wall"

    sorted_user_ip = sorted(user_ip.items(), key=operator.itemgetter(1), reverse=True)

    c = 0
    for ip in sorted_user_ip:
        c += 1
        if ip[1] >= (len(lines) - SHOW):
            print c, "%-13s" % ip[0], "\t%i/%i" % (ip[1] + 1, len(lines)), "%i%s" % ((ip[1] + 1.0)/(len(lines)) * 100, '%')
        else:
            break
```

```
def calc_start(time):
    minute_start1 = (int(datetime.datetime.fromtimestamp(time).strftime('%M')
       ) / 5) * 5
    return minute_start1


def signal_handler(signal, frame):
    print('You pressed Ctrl+C!')
    sys.exit(0)

if __name__ == "__main__":
    signal.signal(signal.SIGINT, signal_handler)
    sys.exit(main())
```