iOS上基于NetworkExtension的实现流量分类的系统

> 备注信息，后续删除
>
> 命名：iTrafficeClassifier
>
> iOS app服务器：app.ainassine.cn
>
> app下载地址：https://app.ainassine.cn/download.html
>
> app.ainassine.cn => 119.23.215.159
>
> PerAppProxy端口：10808
>
> bundleID查询地址：http://www.ainassine.cn/AppStore_Crawler_Demo/index.html
>
> bundleID查询服务器端口：5000
>
> 目前在iOS <= 12.1版本上 `PerAppProxy` 存在漏洞，在 `iOS 12.1.1 beta 2` 版本上有修复，如果用户需要使用基于 `UDP` 协议实现的语音、视频通话的话，需要升级至 `iOS 12.1.1 beta 2`，否则会引起iPhone重启。

0x00 概述

项目需求背景描述，工具功能概述，工具运行环境概述，限制概述。

本系统为网络流量分类研究提供了一种可以可靠标记流量类别的工具。Android平台上已经有相应的工具，但是iOS平台上能够为流量按照不同APP进行分类的应用还很鲜见。除了对流量进行标记之外，本应用还提供内容过滤的功能，也会对用户的位置信息进行记录，方便后续分析流量与位置关系。

0x01 系统架构

客户端设计、功能，服务器设计、功能，客户端服务器通信设计。

客户端主要功能模块由PacketTunnel、PerAppProxy支撑。

> TODO：
>
> ~~1. 百度地图API接入完成位置信息获取~~
> 2. ContentFilter接入完成内容过滤

服务器主要包括**代理服务器**、**BundleID查询服务器**、**应用服务器**、**流量抓取服务器**等。

代理服务器完成SOCKS5协议通信。

BundleID查询服务器使用频率应该不高，属于附属产品，可以提供给其他用户查询某些中国区APP的bundle ID，由Python 3.6.5 提供服务。对于本系统，主要承担在iOS端工具编译时提供目标的bundle ID的任务。

应用服务器是为了支撑iOS端工具使用而建立的一系列脚本的集合。

> 目前阶段：
>
> 1. 确定APP终端所在公网IP（IP+端口，如果终端在局域网环境中的话）。
> 2. 接收定位数据，并按照imei号码存储。

流量抓取服务器主要完成对SOCKS5服务器转发的流量的抓取工作。

> 在这个过程中，明确的应该是目的应用服务器（终端上APP真正请求的服务器）的地址。由此就可以完成**服务器地址**和**应用**的对应关系的确定。

0x02 客户端

PakcetTunnel、PerAppProxy。

记录每一个packet或者datagram的详细信息，包含app信息。

位置记录

利用百度定位SDK实现。百度对iOS原生的定位服务进行了封装，在其基础上提供了语义化的定位结果，但是百度没有对iOS中allowdeferredlocationupdates的函数进行封装，导致如果本APP被杀死，将无法继续进行定位服务。**但是只要程序在后台，就能保证长期活动。**

定位返回数据
1. location:

> location是iOS原生的定位结果，属于CLLocation类型。以下内容均可以在CLLocation中找到，有更详细了解的需求请移步。

主要属性:
`coordinate: CLLocationCoordinate2D` : 定位获得的二维地理坐标，经纬度。
`altitude: CLLocationDistance` : 定位获得的海拔高度。
`horizontalAccuracy: CLLocationAccuracy` : 水平定位精度，单位为 `米` 。
`verticalAccuracy: CLLocationAccuracy` : 垂直定位精度，单位为 `米` 。
`floor: CLFloor?` : 定位获得的楼层数，不是所有定位都有结果，也不是所有的iPhone都支持。
`speed: CLLocationSpeed` : 定位获得的设备移动速度，单位为 `米每秒` 。
`course: CLLocationDirection` : 定位获得的设备的方向，以与正北方向的相对夹角衡量。
`timestamp: Date` : 定位结果产生的时间点。

2. rgcData:

> rgcData属于Baidu定位SDK定义的BMKLocationReGeocode类，是对iOS定位结果的一种补充，对定位结果进行了语义解释。

`country: NSString` : 定位所在的国家。
`countryCode: NSString` : 国家编码。
`province: NSString` : 省份名称。
`city: NSString` : 城市名称。
`district: NSString` : 区名称。
`street: NSString` : 街道名称。
`streetNumber: NSString` : 街区号码。
`cityCode: NSString` : 城市编码。
`adCode: NSString` : 行政区划编码。
`locationDescribe: NSString` : 定位地点在什么地方周围的语义化描述信息。
`poiList: NSArray<BMKLocationPoi*>` : 语义化结果，表示该定位点周围的poi列表。

定位数据上传
定位数据是由BaiduLocation SDK产生的，每隔大约15s会有一次定位结果。本APP不在手机上保留定位结果，在获得定位结果后就通过HTTP POST以JSON的形式上传到服务器，JSON格式如下：

```
{
  "idfa": xxxxxxxxx,
  "location": {
    "coordinate": (double,double),
    "horizontalAccuracy": double,
    "altitude": double,
    "verticalAccuracy": double,
    "floor": int?,
    "speed": double,
    "course": double,
    "timestamp": Date
  },
  "rgcData": {
    "country": String,
    "countryCode": String,
    "province": String,
    "city": String,
    "cityCode": String,
    "district": String,
    "street": String,
    "streetNumber": String,
    "adCode": String,
    "locationDescribe": String,
    "poiList": list,[]cd
  }
}
```

~~其中 `imei` 用于确定iPhone终端。也可以获得设备的 `UUID` 作为唯一标志。~~
根据iOS - 获取设备标识符UUID/UDID/IMEI等，iOS上获取 `IMEI` 、 `IMSI` 、 `UDID` 已经不再可能。
而 `UUID` 作为唯一标志符比较繁琐，参考获取iOS设备唯一标示UUID——Swift版这里使用 `IDFA` 作为唯一标志符，**要求用户允许追踪**。 `IDFA` 会在用户卸载iPhone上所有相关开发商的APP后重置。 `353B317C-0B24-4C0F-B840-92A77F6350BC` 是一个样例。

内容过滤

数据库

0x03 服务端

代理服务器

> 服务器地址：119.23.215.159:10808

SOCKS5服务器源码来自Github。为了适应aliyun服务器*专用网络*没有公网网卡的缺陷，改动了其中开启

UDP服务的代码，使其监听端口为 `0.0.0.0` 而非某个公网IP。根据StackOverflow，为了保证UDP通话通畅进行，修改了SOCKS5实现中关于UDP连接超时的设置，由原来的 `60` 修改为 `6000` ，单位为 `秒` 。

SOCKS5参考

RFC 1928，定义了 `SOCKS5` 协议的基本通信流程。

RFC 1929， `SOCKS5` 用户名密码认证过程。

RFC 1961， `SOCKS5` GSS-API方式认证过程。

流量抓取服务器

> 利用TCPDump配合脚本实现
> -d、-dd、-ddd意义不明。

TCPDump的使用:

参照说明。

重要选项:

`-w file` ： 保存抓取结果到文件*file*中。

`-B buffer_size` 或 `--buffer-size=buffer_size` ： 设置操作系统抓取的缓冲区大小，单位为KiB（1024字节）。

`-c count` ： 抓取*count*个packet后退出。

`-C file_size` ： 在向文件写入一个新的packet前，检查文件大小是否超过*file_size*设置，如果超过，则关闭当前文件并打开一个新的文件进行写入。后续文件命名为 `-w` 设定的文件名加序号。*file_size*的单位为**百万字节**（1,000,000字节，而非1,048,576字节）。

`-D` 或 `--list-interfaces` ： 列出系统中可用的网络接口，以及在哪些接口上可以进行流量抓取。对于每个接口，会打印序号、接口名，可能还会打印有关接口的描述信息。接口名或者接口序号可以被提供给 `-i` 参数用来指定在哪个接口上进行流量抓取。如果*tcpdump*在编译过程中使用的*libpcap*过老，缺少pcap_findalldevs(3PCAP)函数， `-D` 参数将不被支持。

`-F file` ： 使用*file*作为输入的正则式文件。

`-i interface` 或 `--interface=interface` ： 监听*interface*。如果没有设置，*tcpdump*将寻找系统中序号最小的接口（本地回环除外）进行监听，结果很有可能是 `eth0` 。

`-n` ： 不要将地址转为名称，比如将IP、端口号转为域名。

`-Q direction` 或 `--direction=direction` ： 选择抓取报文的方向为*direction*，有*in*、*out*、*inout*可选。不是所有的平台都支持。

执行设计:

首先我们的流量是来自客户端，经由SOCKS5协议传输到服务器上，再由服务器与目的服务器进行通信的。这其中设计很多packet换头的操作。在我们的SOCKS5服务器看来，无论客户端身处什么样的网络环境（客户端有可能身处某内网，本身IP是局域网IP），他都只能看到该客户端在公网上的IP和端口，而客户端需要进行交互的服务器无疑是暴露在公网上的，也就是说，目的服务器的IP、端口与其提供的服务之间存在着长期、稳定的对应关系，这样我们就能完成**流量的APP分类**。那么如何在抓取后区别来自不同的客户端的流量呢？在比较粗略的粒度上，如果终端是通过运营商网络直接接入互联网，那么他的IP地址就是其自身的真实IP地址。但是如果终端是经由路由器之类的设备接入网络，那么只有通过其IP地址和端口号去鉴识其身份。现在系统内只是简单获取终端的公网IP，后续需要尝试获取公网IP、端口号，并且证实这一对应关系的正确

性。终端需要周期上报自身的特征以及IP。

> **一个示例的tcpdump执行命令**
> tcpdump -w t.pcap -C 1 [expression]
> *tcpdump*将会抓取满足expression的流量，并将结果保存在*t.pcap*中，如果*t.pcap*大小超过了1百万字节（约为1MB），那么就会保存在新的文件t.pcapnum中，其中num从1自增。
>
> **关于expression**
> 沟通需求后再确定，目前来看可以只抓ipv4的部分。

应用服务器

1 定位数据接收服务器

> 地址：
> http://119.23.215.159/test/checkin/locRec.php
>
> 服务器操作系统：
> LSB Version: :core-4.1-amd64:core-4.1-noarch
> Distributor ID: CentOS
> Description: CentOS Linux release 7.2.1511 (Core)
> Release: 7.2.1511
> Codename: Core
>
> apache版本： Server version: Apache/2.4.6 (CentOS)
> Server built: Jun 27 2018 13:48:59
>
> PHP版本： PHP 5.4.16 (cli) (built: Apr 12 2018 19:02:01)
> Copyright (c) 1997-2013 The PHP Group
> Zend Engine v2.4.0, Copyright (c) 1998-2013 Zend Technologies

定位数据以 `JSON` 格式通过 `HTTP POST` 方式上传到服务器。服务器采用PHP实现，负责接收数据，并且以 `locRec{UUID}.log` 的形式对不同iPhone终端的定位数据进行保存。样例请看这里。

2 公网IP确定服务器

> 地址：
> http://119.23.215.159/test/checkin/checkin.php

类似定位数据接收服务器，应用在启动后会以 `HTTP POST` 的形式，带自己的 `UUID` 请求服务地址，从而获得自己的 `公网IP` 地址。。

0x04 通信设计

流量分类信息通信。[待实现]

位置信息通信

参照定位数据上传

0x05 应用分析

客户端开销 [TODO]

服务器容量 [TODO]

0x06 成果对比

0x07 参考

[1] Reboot caused by my AppProxy handling UDPFlows
[2] Does NetworkExtension know which app the data flow comes from?
[3] SOCKS5 Server
[4] Is there a timeout for UDP in SOCKS5?
[5] RFC 1928
[6] RFC 1929
[7] RFC 1961