Open Wifi SSID Broadcast vulnerability

SSN Project Assessment 2012

Nikos Sidiropoulos
nikos.sidiropoulos@os3.nl

Michał Mioduszewski
michal.mioduszewski@os3.nl

Paweł Oljasz
pawel.oljasz@os3.nl

Edwin Schaap
Edwin.Schaap@os3.nl

December 24, 2012

Abstract

It is far from being common, yet seen that vendors prefer to deliver better user experience rather than better security. Based on this fact, an investigation was performed, which confirmed, (in the SNE lab) that some devices, do broadcast their saved SSIDs in order to guarantee a faster scanning/connection without user interaction or awareness. This however, raises up questions in terms of profiling, and specifically for Open Wifi SSIDs, reveals a security vulnerability . So, what are devices actually giving out and how can someone make use of that are the two main concerns.

1 Introduction

Recent observations has shown[1] that Wifi devices with enabled active scanning function expose trivial information for the users, yet valuable information for the attackers. Gathering the same information in a more efficient way (less time needed, effortless) originate the project idea. Clearly, the idea was to devise an automated method (device) which utilizes the vulnerability of active scanning (probe requests) and automatic connection to open Wifi Access Points from Wifi capable devices. Obviously, by creating a device that can sniff and automatically create spoofed SSIDs, one can grasp the "size of

this vulnerability by scanning many devices in a small period of time. Apart from just a research conclusion, this kind of device can have many practical security related applications. For example, any public entity like an Institution or Company, (without any technical knowledge) can assess the level of security that their individuals (students,workers) have, from the aspect of Wifi active scanning that is, through their Wifi devices, and specifically in high level security entities(banks) , where enforcing a security measure is considered a necessity. Moreover, no physical presence is needed, accomplishing a less profound (stealth mode) security analysis which will provide better results. Needless to say, that it can clearly cut down the costs or working hours close to none.

2 Background

2.1 Client and Access Point Association

A key part of the IEEE802.11 process is discovering a WLAN and subsequently connecting to it. This process starts by the WLAN network that actually sends frames called Beacons in order to advertise its presence by revealing its SSID. After that the WLAN client sends probe request frames (two main category types) in order to identify (scan/search) or connect to its preferable WLAN. Lastly authentication and association are the two final steps for an open Association process which we won't cover, considering this has no direct interest in our research project.

2.2 IEEE 802.11 Beacons

The primary purpose of the beacon is to allow WLAN clients to learn which networks and access points are available in a given area. Access points may broadcast beacons periodically. Although beacons may regularly be broadcasted by an access point, the frames for probing, authentication, and association are used only during the association (or reassociation) process.

2.3 IEEE 802.11 Probes

There are actually two ways for a client to scan for available APs, active and passive:

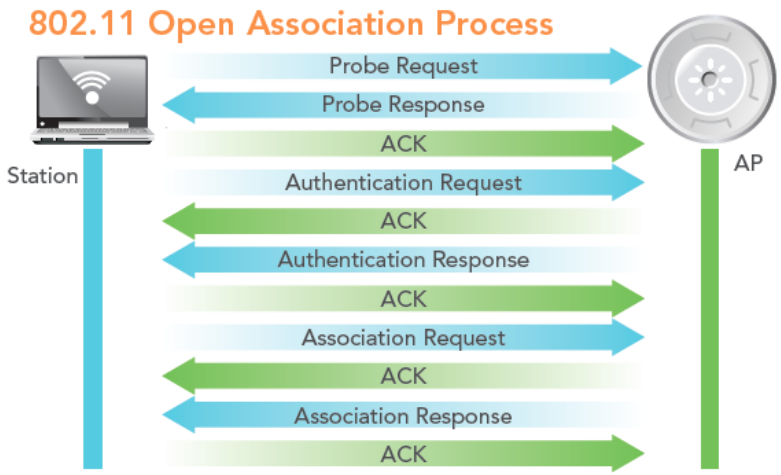

Figure 1: Association Process

2.3.1 Active Scan

During active scan, the client tunes its IEEE 802.11 radio to the channel being scanned and broadcasts a probe request in order to listen to any available probe responses from APs on the specific channel with the a matching SSID[2, p. 980]. The time that the client should wait for a response is vendor specific, but usually is around the value of 10ms. There are two types of probe request frames, a) directed probe and b)broadcast probe.

Directed probe: The client sends a named-specific SSID probe request[2, p. 979] where only the AP(s) that have the requested SSID should reply to this request with a probe-response. (Figure 2)

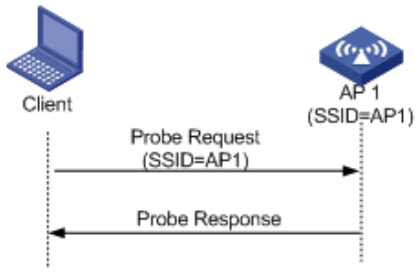

Figure 2: Direct Probe

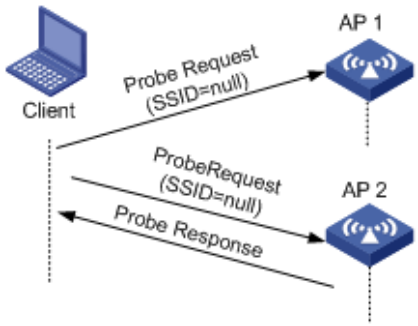

Figure 3: Broadcast Probe

Broadcast probe: The client sends a broadcast probe request with a **null** SSID where all APs receiving this, would respond with a probe-response for each of the SSID they support. (Figure 3)

2.3.2 Passive Scan

In passive scanning, the client also tunes into the radio frequency it wants to scan through, yet instead of sending a probe request it waits[2, p. 978] for a broadcast beacon from the available APs.

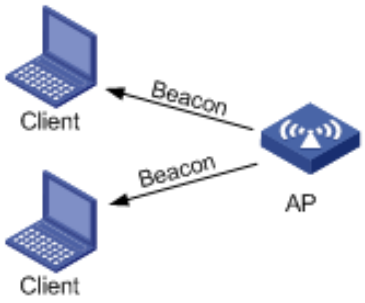

Figure 4: Passive Scan

3 Method

3.1 Detecting devices which send Probe Request

3.1.1 The Setup

This part of the project deals with gathering data of all types of devices which are sending out Probe Requests. The setup that was used for this was mostly based on a software configuration so it could be setup on a router with a custom operating system (like OpenWRT) or on a laptop. A configuration is setup in such a way that three types of data could be gathered.

The first part is a network sniffer. A process running *tcpdump* listens on the wireless monitor device to capture all traffic packages sent out by other devices. This is then put through a filter so only the Probe Requests are save to a log file on the filesystem. This file is then formatted to contain only the MAC address of the device and the SSID send out by the Probe Request.

The second part is the device identification. For this, an access-point is setup so devices can connect to it. A DHCP server is used to give every device an unique ip address by using an infinite lease. The leases are recorded in a file which contain both the MAC address and the IP address. Then all the traffic is routed to a local webserver (in this case Lighttpd) which has logging enabled. This log file contains the IP address and the UserAgent string of the HTTP client of the device. So in essence, if a device is connected to the access-point and request a webpage, it hits the local webserver and records the request.

3.1.2 Gathering Data

To gather data, this setup was exposed on different locations. The locations were chosen the be public but also interact with people. On every location people were asked to connect to the access-point to determine the device type. The locations involved where the UvA, VU, Amsterdam Central and Schiphol Airport. Before devices were connecting to the access-point, the sniffer was setup to capture packages and kept running a short time after the last client had connected. The reason for this is to make sure the sniffer is able to capture Probe Request, since those are sent out often every 60 seconds.

3.1.3 Processing Data

After gathering the data, there are three files for every logging sessions, namely the SSIDs sent out, the DHCP leases and the HTTP UserAgents. The files from all sessions were then combined into three large files and formatted in such a way that they could be imported into a database. For this project, a MySQL database is used. Every file was imported into the database and represents a table. With the data inside a database it is possible to query the data easily.

3.2 Detect devices which connect to rogue AP

In order to realize which devices do connect to rogue access points, a plethora of devices with various operating systems were put into the assessment procedure. An access point is configured with an SSID which is already known by the device used. The AP is set to be an unprotected network. When the device was automatically connected, the type of operating system along with the version was recorded.

3.3 Automated Spoofing

The concept of the Dynamic Access Point is to automate the process of gathering data for SSID and mac address, which are responsible for probe requesting for those specific SSID. Next step is to automate the spoofing process of those SSIDs to find out which of them might by open. If a device makes the connection, it can be concluded that specific SSID is open and that the setup indeed works.

To automate the spoofing principle a piece of hardware is prepared with custom software to achieve this. This piece of hardware is called the Dynamic Access Point (DAP). It is responsible to adjust the spoofing process according to its input. The input can be user defined and/or captured information from the wireless interface. The DAP created in this research can operate in two attack modes: general-mode and direct-mode. In the general-mode all Probe Requests are gathered and are used to set the spoofed SSIDs from. In the direct-mode, only Probe Requests are used from a specific mac-address.

In this research, a TP-LINK router is used with OpenWRT as its operating system. There are two devices used for each part of the process. The first part is information gathering where the device listens for wireless traffic

and filters it accordingly. A piece of software is written to accomplish this step. The wireless interface is set in monitor mode so all packets are captured. Then all packets are filtered except for the Probe Requests. From these Probe Requests a list is generated with unique SSIDs. The first seven are used for spoofing. If the system is in direct-mode, the SSIDs are only used from a mac-address defined by the user.

The second part of the setup is also a TP-LINK router with OpenWRT. This router is setup to act as an access-point by using hostapd as the service. Hostapd is a userspace process that handles the clients connected to the access-point. The router can hand out clients an IP-address by using DHCP and provides internet access. When the first router has enough SSIDs collected it generates a configuration-file in a format which the hostapd service understands. It sends the file via ssh, by using the scp command, to the second router and reloads the service. After reloading, the second router acts as access-point for the provided SSIDs.

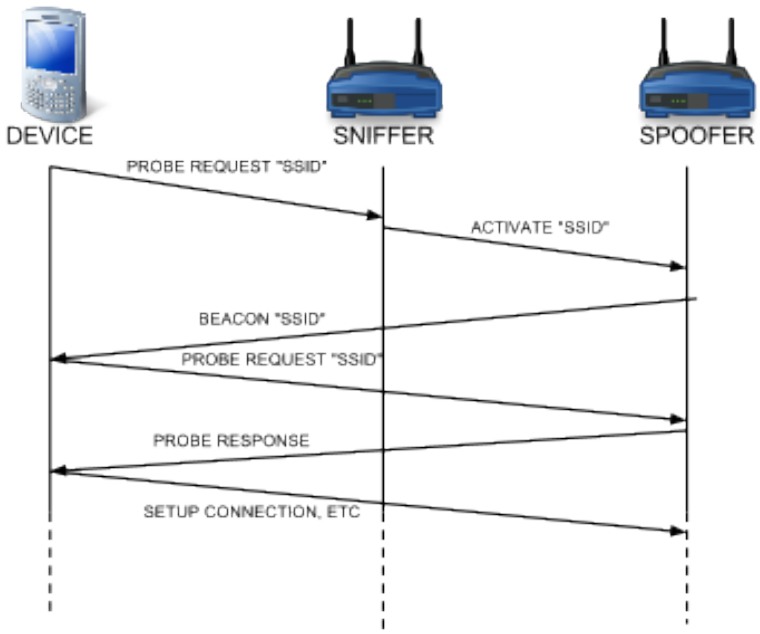

Figure 5: Basic Operation of DAP

4 Findings

4.1 Detecting devices which send Probe Request

The results gathered from the experiment are divided in three parts. First there is a table which contains a MAC address and a broadcasted SSID. Then there is a table with DHCP leases, which consist of MAC addresses with a corresponding IP address. And the third table is filled with an IP address and a UserAgent string from the internet-browser. To analyze this data there were multiple queries executed. First we got the most popular SSIDs by using the following query as seen in Table 1.

```
SELECT ssid, count(*) AS count
FROM (
        SELECT ssid.mac, ssid.ssid
        FROM ssid
        WHERE ssid.ssid != ""
        GROUP BY ssid.mac
) AS P
GROUP BY ssid ORDER BY count DESC
```

SSID	Count
KPN	277
tmobile	229
Linksys	23
ljgvwF02	22
swisscom	19
free-hotspot.com	19
homerun1x	14
de	11
Amsterdam	8
Hotel	7

Table 1: The ten most used SSIDs

Next is a query to get the unique UserAgent strings from clients which are also sending out a Probe Request. The results are formatted to only show the operating system and can be observed in Table 2.

```
SELECT useragent.useragent
```

```
FROM ssid, dhcp, useragent
WHERE
        ssid.mac = dhcp.mac
       AND dhcp.ip = useragent.ip
       AND ssid.ssid != ""
GROUP BY useragent.useragent
```

Operating Systems	
Android 2.1	BlackBerry 7.1.0
Android 2.3	BlackBerry 7.1.0.336
Android 2.3.3	iOS 4.0
Android 2.3.6	iOS 5.0.1
Android 4.0.3	iOS 5.1
Android 4.0.4	iOS 5.1.1
Android 4.1.1	iOS 6.0
Android 4.1.2	iOS 6.0.1
Android 4.2	Mac OS X 10.6.8
BlackBerry 5.0.0.592	Windows 7
BlackBerry 6.0.0.668	

Table 2: Operating systems sending Probe Requests

To give perspective to this data, marketshare data is gathered and compared with the operating systems used. This results in values representing the marketshare of devices which are known to be sending Probe Requests and a market share of devices which are not seen in this research. These values are presented in Figure 6. The operating systems which are sending Probe Requests consist 68.8% of the market.

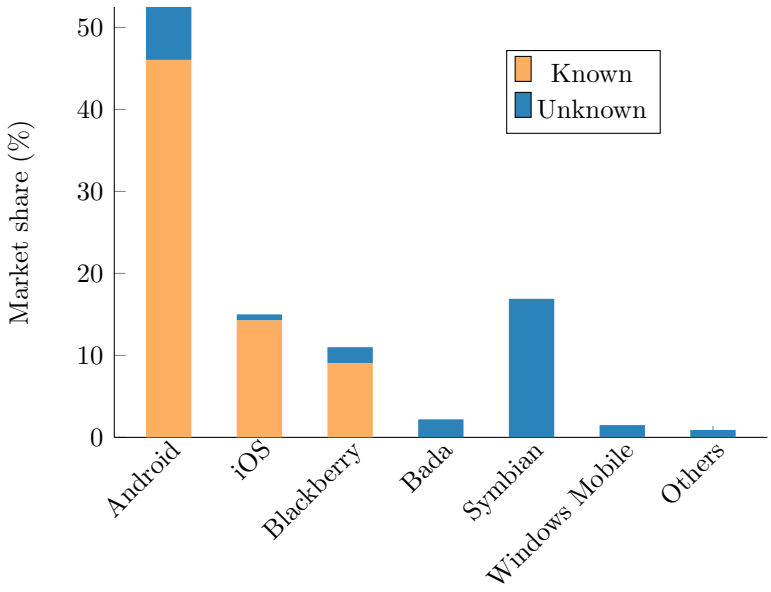

Figure 6: Market share of OS sending Probe Requests

4.2 Detect devices which connect to rogue Access Point

In the second part, several devices got connected to the rogue Access Point. The operating systems types and version which are seen to connect are listed in Table 3

Operating System
Android 2.1
Android 2.2
Android 2.3
Android 4.0
Android 4.1
Blackberry 6.0
Blackberry 7.0
iOS 5.0
iOS 6.0

Table 3: Operating systems sending Probe Requests

This data is also reflected on the market-share to give perspective of the

amount of affected devices. This can be seen in Figure 7.

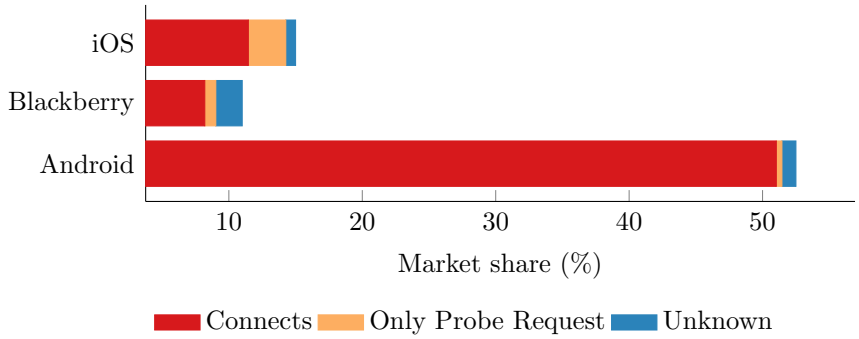

Figure 7: Market share of OS connecting to rogue AP

4.2.1 Automatically spoof SSIDs

When the Dynamic Access Point was put the work, there were a couple of observations. First it is seen that the boot-time of the device took some time, around 30 seconds. But when it was in operation, it responded quite well to the basic SSID. When the device was in spoofing mode, the reload time for the *hostapd* process to update the list of SSIDs was quite slow as it took about 10 seconds before the spoofed SSID was working. When the device was operating, it is observed that devices were able to automatically connect to the spoofed SSIDs. It was tested with a handful of devices, but all got connected.

5 Conclusion

From the results gathered from devices, it is clearly observed that a large part of the Android, iOS and Blackberry devices send out Probe Requests. Since these devices have a large market share, a lot of devices might be vulnerable. As it is proven that the automated spoofing process works and devices can easily be triggered into connecting to a rogue access point, it is considered an issue. This proves that there are devices that only use the SSID to connect to an access point. These SSIDs are stored within the device.

6 Discussion

The reliability of the numbers representing the market share of known devices which are sending out Probe Request can be further improved by performing more research on this topic. This research did not investigate every type of device so the numbers can be higher than found. Also, not all devices run the default operating systems. Custom roms can provide a different Wireless stack. It is also possible that some vendors change the wireless stack and thus research in this area is needed to give more details on the exact range of the problem.

Since the devices just connect if the SSID matches, it may be a good idea to implement policies for remembering access points. For instance, a device should never remember unsecure networks. This way, spoofing is more complicated to perform. Other options are to, besides the SSID, keep record of the BSSID of the known access point. This method makes it harder because the rogue access point must also know the BSSID (also know as access point MAC address) and since this is not exposed with the Probe Request, it is hard to guess.

Further research can make clear what good policies are, so vendors can maintain these within their devices. Also the necessity of using the Probe Request can be further investigated as this problem is a real issue and should be used only if really necessary.

To even further increase the success rate of devices that connect to the DAP more smart techniques can be applied. SSIDs that are seen more often than others will be flagged with higher priority. Also successful connections can increase the affected SSID priority. Every SSID can have a frequencial attribute, meaning DAP would try to predict whether the SSID that has recorded is more likely to be Open, based on the previous mentioned flag.

7 References

References

[1] J. Bhardwaj, *What is your phone saying behind your back?*, Oct. 2012. [Online]. Available: `http://nakedsecurity.sophos.com/2012/10/02/what-is-your-phone-saying-behind-your-back/`.

[2] *Part 11: wireless lan medium access control (mac) and physical layer (phy) specifications*, IEEE Computer Society, Mar. 2012.

A Dynamic Access Point Source

Source can be found on the Github pages:
`https://github.com/oljasz/ssn_project2012`

B Market Share

The tables within this article are based on the following tables to compute the market-share for affected devices.

OS	Market share %
Android	52.5
Blackberry	11
iOS	15
Bada	2.2
Symbian	16.9
Windows Mobile	1.5
Others	0.9

Table 4: OS Market share for Q3 2012.
Source: http://www.gartner.com/it/page.jsp?id=2237315

Version	Distribution %
1.5	0.1
1.6	0.3
2.1	2.7
2.2	10.3
2.3 - 2.3.2	0.2
2.3.3 - 2.3.7	50.6
3.1	0.4
3.2	1.2
4.0.3 - 4.0.4	27.5
4.1	5.9
4.2	0.8

Table 5: Android World Distribution(3rd December 2012)
Source: http://developer.android.com/about/dashboards/index.html

Version	Distribution %
7.1	7.5
7.0	14.7
6.0	41.8
5.0	32.7
4.7	0.1
4.6	2.7
4.5	0.5

Table 6: BlackBerry OS distribution(July 2012)
Source: http://www.inteist.com/2012/07/blackberry-os-distribution-july-2012/
**Note: This is not an official information or official numbers. These is a sample from one of the apps that run on BlackBerry and have enough users so that it is statistically significant.

Version	Distribution %
6.0	74.3
5.1	18.5
5.0	2.1
4.3	3.7
4.2	1
4.1	0.2
4.0	0.2

Table 7: iOS distribution(November 2012)
Source: http://david-smith.org/iosversionstats/
**Note: This is not an official information or official numbers. These is a sample from one of the apps that run on iOS and have enough users so that it is statistically significant.

C User Agent Strings

```
"Android-Wifi/0.1"
"Android/1.0 (GT-S5830 GINGERBREAD)"
"Android/1.0 (GT-S6102 GINGERBREAD)"
"Android/1.0 (marvel ASN0020121106)"
"Android/1.0 (picasso_m IML74K)"
"Apache-HttpClient/UNAVAILABLE (java 1.4)"
"BlackBerry8520/5.0.0.592 Profile/MIDP-2.1
    Configuration/CLDC-1.1 VendorID/129"
"BlackBerry9360/7.1.0 Profile/MIDP-2.1 Configuration/
    CLDC-1.1 VendorID/603"
"BlackBerry9360/7.1.0.336 Profile/MIDP-2.1
    Configuration/CLDC-1.1 VendorID/603"
"BlackBerry9700/6.0.0.668 Profile/MIDP-2.1
    Configuration/CLDC-1.1 VendorID/134"
"boingo client"
"Box Sync/3.3.33;Windows 7/6.1.7601;AMD64/64bit"
"BTWebClient/3100(26671)"
"CaptiveNetworkSupport-166 tmobile_wispr1"
"CaptiveNetworkSupport-183 tmobile_wispr1"
"CaptiveNetworkSupport-183 wispr"
"CaptiveNetworkSupport-183.5 tmobile_wispr1"
"CaptiveNetworkSupport-183.5 wispr"
"CaptiveNetworkSupport-209.31 tmobile_wispr1"
"CaptiveNetworkSupport-209.31 wispr"
"ccmhttp"
"com.google.android.youtube/4.1.47(Linux; U; Android
    4.0.3; nl_NL; GT-P3110 Build/IML74K) gzip"
"Dalvik/1.4.0 (Linux; U; Android 2.3.6; GT-S6102 Build/
    GINGERBREAD)"
"Dalvik/1.6.0 (Linux; U; Android 4.0.3; A510 Build/
    IML74K)"
"Dalvik/1.6.0 (Linux; U; Android 4.0.3; GT-I9100 Build/
    IML74K)"
"Dalvik/1.6.0 (Linux; U; Android 4.0.3; GT-P3110 Build/
    IML74K)"
```

```
"Dalvik/1.6.0 (Linux; U; Android 4.0.4; GT-I9100 Build/
    IMM76D)"
"Dalvik/1.6.0 (Linux; U; Android 4.0.4; MZ601 Build/I
    .7.1-45)"
"Dalvik/1.6.0 (Linux; U; Android 4.0.4; Wildfire S
    A510e Build/ASN0020121106)"
"Dalvik/1.6.0 (Linux; U; Android 4.1.1; GT-I9300 Build/
    JRO03C)"
"Dalvik/1.6.0 (Linux; U; Android 4.1.2; Wildfire S
    Build/benjamingwynn20121029)"
"Dalvik/1.6.0 (Linux; U; Android 4.2; Nexus 7 Build/
    JOP40C)"
"GeoServices/84 CFNetwork/609 Darwin/13.0.0"
"Google Update/1.3.21.123;winhttp;cup"
"GoogleAnalytics/1.4.2 (Linux; U; Android 2.3.3; en-us;
    GT-I9001 Build/GINGERBREAD)"
"GoogleAnalytics/1.4.2 (Linux; U; Android 4.0.3; nl-nl;
    GT-P3110 Build/IML74K)"
"GoogleAnalytics/2.0 (Linux; U; Android 4.1.1; en-gb;
    GT-I9300 Build/JRO03C)"
"Microsoft NCSI"
"Microsoft Windows Network Diagnostics"
"Microsoft-CryptoAPI/6.1"
"MICROSOFT_DEVICE_METADATA_RETRIEVAL_CLIENT"
"Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services
    Client Protocol 2.0.50727.5466)"
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1;
    Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
    3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0;
    .NET4.0C; AskTbBLPV5/5.13.1.18107; BRI/2; MANM)"
"Mozilla/4.0 (compatible; MSIE 8.0; Win32; Trident/4.0)
    "
"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1;
    WOW64; Trident/4.0; GTB7.4; SLCC2; .NET CLR
    2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729;
    Media Center PC 6.0; BRI/2; .NET4.0C; .NET4.0E)"
"Mozilla/4.0 (compatible; Win32)"
```

```
"Mozilla/5.0 (Android; Mobile; rv:17.0) Gecko/17.0
   Firefox/17.0"
"Mozilla/5.0 (BlackBerry; U; BlackBerry 9700; en)
   AppleWebKit/534.8+ (KHTML, like Gecko) Version
   /6.0.0.668 Mobile Safari/534.8+"
"Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1;
   WOW64; Trident/5.0)"
"Mozilla/5.0 (iPad; CPU OS 5_1_1 like Mac OS X)
   AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9B206"
"Mozilla/5.0 (iPad; CPU OS 6_0_1 like Mac OS X)
   AppleWebKit/536.26 (KHTML, like Gecko) Mobile/10A523
   "
"Mozilla/5.0 (iPhone; CPU iPhone OS 5_0_1 like Mac OS X
   ) AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9
   A405"
"Mozilla/5.0 (iPhone; CPU iPhone OS 5_0_1 like Mac OS X
   ) AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9
   A406"
"Mozilla/5.0 (iPhone; CPU iPhone OS 5_1 like Mac OS X)
   AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9B176"
"Mozilla/5.0 (iPhone; CPU iPhone OS 5_1_1 like Mac OS X
   ) AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9
   B206"
"Mozilla/5.0 (iPhone; CPU iPhone OS 6_0 like Mac OS X)
   AppleWebKit/536.26 (KHTML, like Gecko) Mobile/10A403
   "
"Mozilla/5.0 (iPhone; CPU iPhone OS 6_0_1 like Mac OS X
   ) AppleWebKit/536.26 (KHTML, like Gecko) Mobile/10
   A523"
"Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_0 like Mac OS
   X; en-us) AppleWebKit/532.9 (KHTML, like Gecko)
   Version/4.0.5 Mobile/8A293 Safari/6531.22.7"
"Mozilla/5.0 (iPod; CPU iPhone OS 5_1_1 like Mac OS X)
   AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9B206"
"Mozilla/5.0 (Linux; Android 4.2; Nexus 7 Build/JOP40C)
    AppleWebKit/535.19 (KHTML, like Gecko) Chrome
   /18.0.1025.166 Safari/535.19"
```

"Mozilla/5.0 (Linux; U; Android 2.3.3; en-us; GT-I9001 Build/GINGERBREAD) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"
"Mozilla/5.0 (Linux; U; Android 2.3.3; es-es; GT-S5830 Build/GINGERBREAD) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1 (Mobile; afma-sdk-a-v4.3.1)"
"Mozilla/5.0 (Linux; U; Android 2.3.3; es-es; SAMSUNG GT-S5830/S5830BUKPE Build/GINGERBREAD) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"
"Mozilla/5.0 (Linux; U; Android 2.3.6; ro-ro; GT-S6102 Build/GINGERBREAD) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"
"Mozilla/5.0 (Linux; U; Android 4.0.3; en-us; A510 Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Safari/534.30"
"Mozilla/5.0 (Linux; U; Android 4.0.3; nl-nl; GT-P3110 Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Safari/534.30"
"Mozilla/5.0 (Linux; U; Android 4.0.3; nl-nl; GT-P3110 Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Safari/534.30 (Mobile; afma-sdk-a-v4.1.0)"
"Mozilla/5.0 (Linux; U; Android 4.0.3; nl-nl; GT-P3110 Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Safari/534.30 (Mobile; afma-sdk-a-v4.1.1)"
"Mozilla/5.0 (Linux; U; Android 4.0.3; nl-nl; GT-P3110 Build/IML74K) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Safari/534.30 (Mobile; afma-sdk-a-v6.0.1)"
"Mozilla/5.0 (Linux; U; Android 4.0.3; nl_NL; GT-P3110 BUILD/IML74K) AppleWebKit/528.5+ (KHTML, like Gecko) Version/3.1.2 Mobile Safari/525.20.1"
"Mozilla/5.0 (Linux; U; Android 4.1.1; en-gb; GT-I9300 Build/JRO03C) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30"

```
"Mozilla/5.0 (Linux; U; Android 4.1.1; en-gb; GT-I9300
    Build/JRO03C) AppleWebKit/534.30 (KHTML, like Gecko)
     Version/4.0 Mobile Safari/534.30 GSA/2.1.12.516912"
"Mozilla/5.0 (Linux; U; Android MrBang WWE 2.1 release
    5.1; pl-pl; HTC Hero Build/ERE27) AppleWebKit/530.17
     (KHTML, like Gecko) Version/4.0 Mobile Safari
    /530.17"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8)
    AppleWebKit/534.57.2 (KHTML, like Gecko) Version
    /5.1.7 Safari/534.57.2"
"Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.11 (KHTML
    , like Gecko) Chrome/23.0.1271.95 Safari/537.11"
"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.11
     (KHTML, like Gecko) Chrome/23.0.1271.95 Safari
    /537.11"
"SeaPort/3.1"
"Skype WISPr"
"Syncer/5.00 (ais-1426;p)"
"tmobile_wispr1"
"Windows-RSS-Platform/2.0 (MSIE 9.0; Windows NT 6.1)"
"YahooMobile/1.0 (mail; 1.4.2.11841); (Linux; U;
    Android 2.3.3; GT-S5830 Build/GINGERBREAD_MR1);"
"YahooMobileMail/1.0 (Android Mail; 1.4.6) (GT-S6102;
    samsung;GT-S6102;2.3.6/GINGERBREAD)"
"YahooMobileMessenger/1.0 (Android Mail; 1.4.2) (GT-
    S5830; samsung; GT-S5830; 2.3.3/GINGERBREAD)"
"YahooMobileMessenger/1.0 (Android Messenger; 1.8.1) (
    picasso_m; Acer; A510; 4.0.3/IML74K)"
```