future internet

MDPI

Article

Push Attack: Binding Virtual and Real Identities Using Mobile Push Notifications

Pierpaolo Loreti, Lorenzo Bracciale * and Alberto Caponi

Electronic Engineering Department, University of Rome Tor Vergata, 00173 Rome, Italy; pierpaolo.loreti@uniroma2.it (P.L.); alberto.caponi@uniroma2.it (A.C.)
* Correspondence: lorenzo.bracciale@uniroma2.it; Tel.: +39-06-7259-7440

Received: 20 December 2017; Accepted: 29 January 2018; Published: 31 January 2018

Abstract: Popular mobile apps use push notifications extensively to offer an "always connected" experience to their users. Social networking apps use them as a real-time channel to notify users about new private messages or new social interactions (e.g., friendship request, tagging, etc.). Despite the cryptography used to protect these communication channels, the strict temporal binding between the actions that trigger the notifications and the reception of the notification messages in the mobile device may represent a privacy issue. In this work, we present the push notification attack designed to bind the physical owners of mobile devices with their virtual identities, even if pseudonyms are used. In an online attack, an active attacker triggers a push notification and captures the notification packets that transit in the network. In an offline attack, a passive attacker correlates the social network activity of a user with the received push notification. The push notification attack bypasses the standard ways of protecting user privacy based on the network layer by operating at the application level. It requires no additional software on the victim's mobile device.

Keywords: online social network; push notification; privacy

1. Introduction

Nearly two-thirds of American adults (64%) currently own a smartphone, and more than 91% of smartphone owners aged 18–29 use social networking at least once a week [1]. Facebook has 1.31 billion monthly mobile active users worldwide (44% of them only log in from a mobile device); Twitter has 252 million monthly mobile active users; Instagram 300 million; while the social dating app Tinder has around 50 million active users per month (source: http://www.statista.com/).

Besides the big players, a myriad of different applications usually reside on people's smartphones, allowing their users to play, interact with other users, watch news, etc. To use services (e.g., online social networking, OSN), users must log in to these applications with their accounts. These form their virtual identities.

A virtual identity cannot always be directly traced to the real identity of an OSN user. Pseudonyms represent a simple and easy-to-use way to provide good anonymity. Virtual identities (together with the information associated: political ideas, sexual preferences, etc.) are apparently separated from real identities. In fact, in online social networks, users voluntarily share personal information with the implicit assumption that it is very difficult to link partial information with a person's real identity [2]. This assumption is often false or questionable, as there are many attacks attempting to establish these links: structural re-identification attacks, inference attacks, information aggregation attacks and re-identification attacks [3]. Usually, these techniques use data mining to exploit the correlation among the different information published by one or more people to infer the publisher's real identity.

A binding between virtual and real identities can lead to several issues in countries that limit civil rights, where people get persecuted for their online activity, when virtual identities reveal very

personal or sensitive information or when the real person is a public figure. For this reason, the "real names" policy (the policy that obliges users to identify themselves with their real names, banning the use of pseudonyms) adopted by many online platforms over time brought about several complaints from civil liberties groups. Google apologized and reversed its position (https://plus.google.com/u/0/+googleplus/posts/V5XkYQYYJqy) (July 2014), allowing users to pick pseudonyms. Furthermore, Facebook recently promised to pay more heed to real-name complaints (October 2015) (https://www.eff.org/deeplinks/2015/11/facebooks-new-name-policy-changes-are-progress-not-perfection).

In this work, we present a new attack that expressly targets smartphone users to bind their virtual identities with their real ones. Unlike traditional website/desktop applications, smartphone applications have two unique key features:

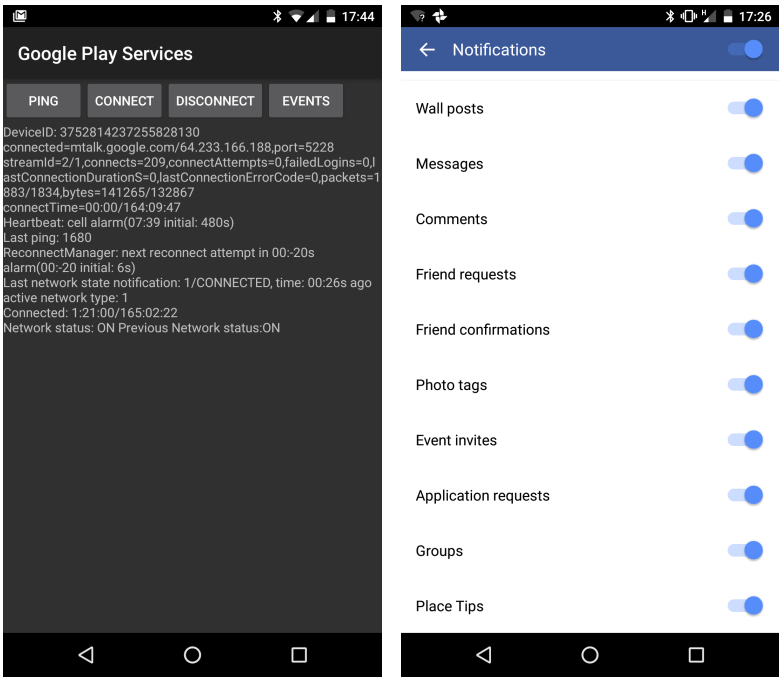
- smartphones follow users everywhere; so the real identity is, in a certain sense, known a priori.
- smartphone operating systems implement a push notification service that can be exploited to reveal the virtual identity associated with the device owner as described in this work.

These "push notifications" can be emitted asynchronously by app servers and reach mobile clients, even if the related application is closed or suspended and if the terminal is on standby. Push notifications typically (but not always) create new messages in the notification bar, to keep the user updated on what has happened since the last time the app was opened.

It is not uncommon for mobile applications to need asynchronous communication with a remote server.

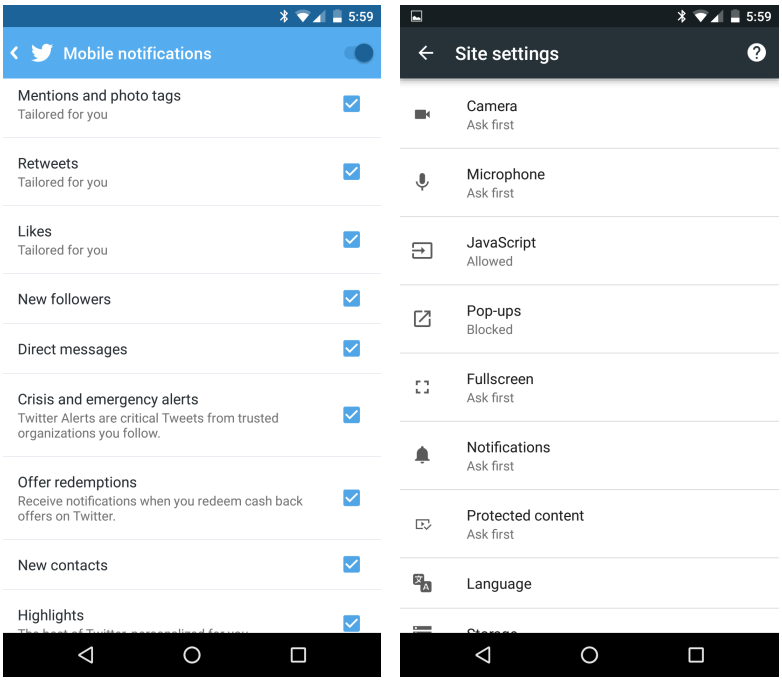
Push notifications are extensively used by almost all popular mobile apps and especially by social networking applications, for instance to notify the user about a new social interaction such as a new private message or a new friendship request. Furthermore, browsers are starting to support push notifications from websites, and an IETF working group is outlining Internet Drafts [4].

Typically, push notifications can be selectively disabled through app/OS settings. Most are enabled by default and are thus usually active (Figure 1).



(**a**) (**b**)

Figure 1. Cont.



(**c**) (**d**)

Figure 1. Push notifications: default behavior of popular mobile applications. (**a**) Google Play Service screen showing heartbeat and connection status; (**b**) default Facebook notification settings; (**c**) default Twitter notification settings; (**d**) push notification through Chrome mobile web browser.

If on the one hand, they are usually implemented as asynchronous ciphered messages exchanged through persistent TLS channels, on the other hand, push notifications can be actively triggered. For example, by requesting a social network user's friendship and by eavesdropping the network traffic, they can unveil the presence of a specific user in a specific location, even if the user is using a pseudonym.

1.1. Use Cases

We believe the attack described in this paper can be applied to many scenarios. We propose three reference use cases.

Marketing Campaigns

Customer profiling and tracking comprise an important asset for OSNs where business is mainly targeted to selling specific advertisements. Evaluating the conversion rate from the outcome of a social media campaign (involving virtual identities) into the number of real customers (involving real identities) represents a big issue for business intelligence. A technique that fills this gap will certainly be of interest.

An exemplary use case is determining how many Facebook contacts expressing their will to participate in an event (a concert, show or demonstration) really attend it. Another is identifying the Facebook profile names of the people in a supermarket. This clearly opens several ethical issues. It does though go in the same direction (even if maybe crossing the line) as location analytic services where WiFi-enabled smartphones can be used to indicate customer presence. Indeed, both commercial (e.g., Cisco Meraki CMX [5] or open source (e.g., [6]) software exists that identifies presence, time spent and repeat visits within the range of a WiFi access point by sniffing and analyzing smartphones' 802.11 probe requests.

Working Place

Employee productivity is an important driver for a business' continued viability. However, according to a work survey (http://www.forbes.com/sites/cherylsnappconner/2013/09/07/who-wastes-the-most-time-at-work/#179c48757b3a), 64% of employees visit non-work related websites daily. More than 61 % of workers spend at least one hour per day on these websites. Tumblr, Facebook and Twitter account for the greater share of this category.

Conversely, employee privacy is a fundamental right covered by employment laws in many countries all over the world. Employer surveillance versus employee privacy is indeed an open issue [7].

The privacy attack presented in this paper can reveal the virtual identity of an employee even if they use a pseudonym and a ciphered channel to connect to an OSN. In a workplace, the attack's feasibility increases thanks to the control on the employee's Internet access exercised by the boss, making eavesdropping easy.

Government Monitoring

All major OSNs offer tools for law enforcement and publish governments' information requests for transparency (e.g., Twitter Transparency Reports). At the same time, OSNs have been used to organize riots and protests against governments (2011 Arab Spring Riots) and even to recruit terrorists and to organize attacks (2015 terrorist attack in France and Belgium). This motivates the necessity for governments to reveal users' identities and track the real position of OSN users.

The attack presented in this work can be applied in such scenarios to track the owners of OSN virtual identities. Having control at the ISP level, and without any right to ask for this information from the OSN owner companies, governments can track down dissidents for licit or illicit purposes. The proposed attack can also be used when collaboration with the OSN is not feasible e.g., darknets or where data retention is not available/implemented by the OSN.

1.2. Privacy Implications

The proposed attack is collocated in the wide area of privacy threats on encrypted traffic, often referred to as side-channel information leaks [8]. This area and the related privacy implications have been extensively investigated for the desktop scenario (e.g., [9]). It is recently attracting increasing attention for the mobile scenario. Several works addressed privacy threats and implications because of specific technical and usage characteristics of mobile devices. These include the typical "one app at a time" use [10], the possible privacy leakage from the use of sensors [11], location privacy [12] or, more in general, privacy in mobile environments [13].

This paper, to the best of our knowledge, is the first to deal expressly with using mobile push notifications to bind real and virtual identities.

As already expressed by the authors of [14], the disclosure of virtual and real-world identities could raise severe privacy concerns. Linking them can cause even more damage to an individual's privacy.

Indeed, apparently shielded by pseudonyms and ciphered traffic data (as reported in [2]), people express themselves more freely on politics, religious or personal topics. Linking their virtual to their real identity could then be embarrassing or even harmful (in 2016, Saudi Arabia sentenced a Twitter user to 10 years in prison for publishing 600 tweets "which spread atheism").

Finally, the recent IETF Internet Draft [4] on push notifications also expresses concerns about privacy implications even if from the application's point of view.

1.3. Contributions

This work wants to provide an answer to the following set of questions:

- How do popular applications use push notifications and how can this be exploited to reveal information?

- To what extent is it possible to reveal the presence of a user profile using push notifications?
- What methodologies does the attack use?
- What mitigation techniques are available?

Several works deal with the passive inference of app usage (see Section 6); to the best of our knowledge, this is the first work examining the use of push notifications for an active attack.

This attack is more difficult to conduct than passive measurements, yet it provides a higher grade of precision. We can repeat a test on a given profile many times, lowering the probability of false positives.

This paper is organized as follows: Section 2 presents background information on how notifications and push notifications work; Section 3 presents the details of the proposed attack; Section 4 presents an implementation of the attack; Section 5 presents the results of a measurement campaign to better characterize push notifications in the real world. Finally, Section 6 reviews the present situation, and Section 7 draws conclusions.

2. Background: How Push Notifications Work

2.1. Built-In Push Notification Service

Both Android (https://developers.google.com/cloud-messaging/) and iOS (https://developer.apple.com/notifications/) operating systems provide a built-in service for push notifications. Vendors like Google and Apple offer application developers access to this service through a set of APIs that connect to an infrastructure in charge of delivering a message to mobile devices.

In a typical usage pattern, push notifications start with a third party app server that contacts the vendor server (e.g., gateway.push.apple.com) using a push notification service API. It asks to deliver a given structured message to a certain registered user of an application, identified by a device ID. The push notification server contacts the mobile device passing the push notification to the operating system when the device is online. When offline, the OS stores the notification for a limited period of time, eventually delivering it to the device, when it becomes available. When a push notification reaches the device, it can be typically (but not mandatorily) shown on the notification bar and optionally trigger application code (this behavior strictly depends on the limits of the mobile operating system).

The benefits of implementing push notification at the system level are two-fold: (i) provide an easy way for application developers to have almost synchronous communication between server and application clients; (ii) optimize power consumption by avoiding several persistent connections (one per app), keeping only one shared connection for all apps, multiplexing, en-queuing and possibly stacking all messages directed to the same mobile device.

Although this general schema is valid, the specific implementation of the push notification service varies by the vendor. In what follows, we briefly describe the case of Android and iOS.

Google Cloud Messaging (GCM) (https://developers.google.com/cloud-messaging/) was built in 2012 by Google to provide application short messages (up to 4 kB) from a server to Android devices. Even if the source code of the implementation has not been released (being part of the Google Play Store application), some details were publicly and officially released.

Android push notifications work through one persistent TLS connection opened on port 5228 connecting a device with the GCM servers. This connection is monitored and kept open with periodic heartbeats (e.g., every 240 s) and re-established every time the phone connects to a network. The status of this connection is easily accessible by typing *#*#426#*#* on the phone (Figure 1a).

Apple Push Notification (https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html) (APN) works similarly. Each notification is limited to 2 kB. Unlike GCM, APN makes it mandatory for software developers to let their server software notify their user when applications are closed, since there is no way of creating persistent sockets. APN though sets no limit to the number of notifications. APN

comes from a ciphered channel (TLS peer-to-peer authentication) on ports 5223 or 443. The source IP addresses are easily identifiable as belonging to Apple's AS (17.0.0.0/8).

2.2. Custom Notification Channels

System notifications also present drawbacks. Besides limits on message size, notification messages can be dropped or delayed. Therefore, some app developers implement their own persistent channel to connect their apps directly with their own servers. See WhatsApp and Facebook on Android.

These channels are usually opened using the system-level push notification service and kept open using keep-alive messages until the user disconnects from the network. In these cases, updates arrive directly from the third party app server to the app on the end user's device, without passing via the vendor infrastructure. These channels transport proprietary messages processed by dedicated code in the apps. They can optionally result (as with the Facebook app) in a new notification shown in the notification bar of the mobile operating system. Conversely, iOS developers must use the Apple built-in notification channel since they cannot build a persistent socket that remains active when the application is closed or suspended.

2.3. Notifications Bar and Erasable Notifications

Applications decide whether to show notifications in the bar. Developers program this choice. For our purposes, the interesting part is that some notifications can hide previous notifications, thus concealing the attack. This behavior is demanded of the application developer. This is a useful feature adopted for instance by Facebook app developers when a friend request is issued and then revoked. In this case, revoking the friendship request causes the previous friendship notification to disappear from the bar. These "erasable notifications" are perfect to perform an attack without alerting the victim. Facebook friend requests are currently erasable on Android, but no longer on iOS.

3. Attack Description

3.1. Adversary Model

The attacker's primary goal is to infer users' personal activities by revealing an exact match between their online (virtual) and real identities. The following reports the attacker's capabilities.

Packet Sniffing

The attacker is modeled as a semi-passive adversary that can silently sniff packets directed to or originated by the victim. To achieve this, the attacker accesses the target user(s)' network A semi-passive rival is analogous to the 'honest-but-curious' or 'semi-honest' adversary model used for security and cryptographic protocols [15]. This attack can be applied to most wireless environments with public access, e.g., WiFi hotspots in neighborhoods, coffee shops, public events, etc. Replication on cellular networks is discussed in Section 3.4. We assume that the interesting traffic is encrypted and that the attacker cannot decrypt transport level packets (e.g., TLS). However, the attack can access lower layer protocol information such as IP source and destination addresses, TCP ports and MAC addresses.

Push Triggering

Besides passive network monitoring, the adversary can trigger push notification messages to the target's personal device using interfaces and actions that are legitimately available on the platform chosen for the attack. For example, a semi-passive adversary can trigger push notification messages by requesting a friendship through an OSN (e.g., Facebook, Twitter, Google+, etc.) or contacting the victim using a messaging platform (e.g., WhatsApp, Viber, WeChat, etc.). The victim must have the related application installed on their mobile device.

Social Data Retrieval

Depending on the type of attack (online, offline), the attacker needs to gather the information on the virtual identities of those users whom they want to bind with their real identities. For example, this information can be the victim's Facebook profile or activity log on the social network.

3.2. On-Line Attack

The proposed attack consists of actively triggering a push notification message to the victim's personal device with the aim of capturing the related data packet sent by the OSN platform to the targeted device. The temporal correlation between the triggering and the sniffed data marks the presence of the victim on the observed network.

The attack scenario in Figure 2 involves the following parties:

- Adversary and victim: The attacker and the targeted virtual identity should be on the same network, for example served by the same WiFi access point or connected to the same Ethernet network.
- Online platform: This usually is the OSN where the victim is registered and represented by the so-called virtual identity. The platform gives the attacker the functionality to trigger push notifications on the user's devices.
- Push notification infrastructure: This is the framework needed to deliver the push notification. It is either platform-specific (e.g., Facebook notification system for Android devices) in the sense that the platform manages push notifications through its own infrastructure or OS-specific when the OS provider manages the notification infrastructure (mandatory for Apple iOS and a common choice for Google Android).

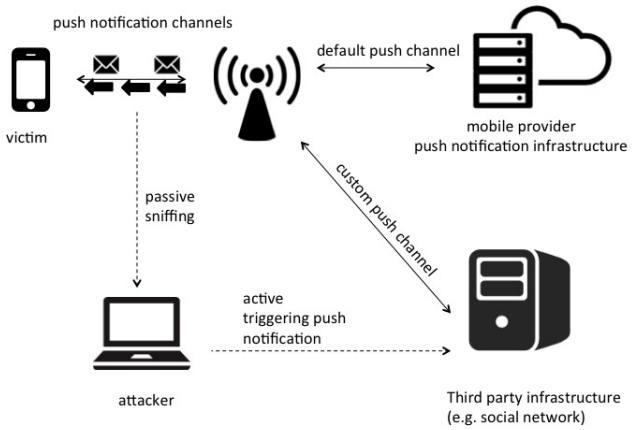


Figure 2. Attack scenario.

The full attack comprises the following steps:

1 Fill a list of possible profile candidates: the attacker targets victims related to a specific location or habits associated with a physical location providing the network access
2 Trigger push notifications to all candidates: the attacker starts to trigger notifications to victims' devices by soliciting online platform events with a pool of bots. This produces push notifications directed to victims' devices that the attacker tries to capture where the targeted user could be connected.
3 Inspect the data over the air, searching for a well-known data pattern: the attacker collects packets related to push notifications from known platforms (e.g., Facebook, Twitter, WhatsApp, etc.) to detect specific size and time patterns that clearly identify the targeted profile.

Victims often cannot detect the attack since attackers can use specific events triggered by the online platform that erase the previous action. An example is requesting the victim's friendship as a first step and later erasing the notification (hopefully before the OS displays it) on the victim's device by removing the friendship request.

Finally, knowing the victim's device's MAC address enables user position estimation by passive triangulation or even by active searching. One can repetitively send ICMP echo requests or ARP requests to the device's MAC address, read the RSSI of the incoming packets and then progressively approach the target.

3.3. Offline Attack

The offline attack has two steps: first, the attacker collects the raw networking data of the monitored location and then correlates this data trace with information crawled from the social network. For example, in Facebook, it is possible to collect timing information on events such as published posts or messages in groups. With this information, the attacker can search the data trace for evidence of the traffic accountable for the push notifications generated by those events.

In this approach, the attacker follows the steps described below:

1 The attacker collects all the packets in a given location related to push notifications from all known OSN platforms (e.g., Facebook, Twitter, WhatsApp, etc.) without actively interacting with victims' profiles.
2 The attacker collects data about the timing of the OSN's events that produced a push notification towards the users.
3 The attacker mixes the data acquired in Steps 1 and 2 to find evidence of the presence of a targeted user in the considered location.

Although the offline attack is conceptually feasible and brings about the discovery of information on the targets, we practically focus the present work on the more viable (and easier to test) online approach. The offline attack is suitable when the attacker has a log trace of the connections performed by several users on different hotspots. An example is having access to logs collected for data retention or security purposes.

3.4. Feasibility of the Attack

Two key aspects impact feasibility: (i) how to fill the list of possible profile candidates; and (ii) how to choose the location for the attack. Both are closely related to the specific context examined for the attack. The former aspect can be (i-a) a "profile check" if the attacker wants to test the presence of a single profile identity (e.g., a given pseudonym) in a given location or (i-b) a "profile search" if the attacker wants to detect if some people belonging to a group are in the surroundings.

In both cases, the WiFi's limited range could hinder the attack. A similar methodology could be used to extend the attack in the case of a wider wireless connection such as LTE (as partially investigated in [16]). In some scenarios, there is no need to eavesdrop the WiFi packets: when the attacker has control of the network backbone serving the WiFi hotspots.

On top of that, offline and online scenarios present specific difficulties. The online attack forces all profile checks in a limited temporal window due to the victim's presence in a given location. Once found, the attacker easily pinpoints the victim using active searching techniques as described above. Conversely, an offline attack requires knowledge of the users' relations (social graph, interaction lists), which sometimes are concealed by the OSN provider according to user privacy settings.

This attack's strong points are that: (i) it assumes no particular setting on the victim's mobile device, (ii) it requires no modification of the standard OSN apps, (iii) it requires no particular equipment (a normal laptop with a WiFi card suffices) and (iv) in some cases (see Section 2), the victim might not even detect it.

As for the differences between the two platforms, we experienced greater technical difficulties in defining the pattern matching on iOS for the reason reported in Section 2. Anyway, Android currently holds 4/5 of the smartphone market.

One can extend the attack to cellular networks using the Paging Message of LTE triggered by incoming network connections. The work in [16,17] adopted this mechanism to track user location. Further investigation on the temporal correlation of LTE paging packets and push notification triggering needs to be carried out to prove the feasibility of the proposed approach on LTE.

3.5. Defense Mechanism

Our technique primarily exploits OSNs' possibility of exploring users' habits by triggering a push notification on the user's device, observing it and exploiting it to infer the victim's physical presence. This exploit allows the attacker to bind a user's virtual identity to a real identity, resulting in a breach of personal privacy. It is thus crucial to provide initial advice to prevent this information linking. Solutions like traffic padding and/or morphing [18,19] can avoid these attacks, but are usually not feasible since it introduces a high amount of network overhead. This can cause several problems especially for not-so-powerful mobile devices: (i) waste of computational resources, (ii) waste of energy and (iii) waste of network traffic that could become a problem for users with limited bandwidth/traffic. Users need more specific solutions to effectively prevent an analysis of push notifications to their mobile devices. Two features allow the enemy to attack:

1. Observing the time between the triggering and the effective reception of the packet on the device. It is highly probable that a push notification packet is received on the user's device shortly after the triggering.
2. Analyzing the size of received packets. The attacker can distinguish between packets related to different push notifications and thus infer the related triggering functionality (e.g., friendship request or comment liked).

A feasible mitigation could be to develop scheduling algorithms focused on avoiding these information leaks. This scheduling should introduce a random delay on message delivery or collect a certain number of push notifications to deliver before triggering the user device with bulk messages. Obviously, this is feasible when the transmission of push notifications is not time-sensitive (i.e., it is not necessary to deliver the friendship request push notification to the user instantly, but it is important in some cases to receive a chat message). The problem of inferring information by observing packet size could be mitigated by hiding information related to the actual size of data in the packet. This could be achieved by producing randomly-sized padded packets for each push notification. This would prevent the attacker from understanding the functionality that triggered the push notification message, i.e., the attacker cannot distinguish if the received message corresponds to a friendship request or a chat message. In addition to the above, in cases in which the push notification packet is delivered by different servers according to the application (e.g., Android GCM), the attacker has one more features to observe and exploit to infer information. Indeed, knowing the packet's source releases information about the specific platform that sent the message and can help the attacker filter out other push notification messages. This could be mitigated by multiplexing push notification traffic like APN already does; only one server delivers all push notifications in the same ciphered data channel.

4. Implementation

We tested the attack using the Facebook Android app, as it is the OSN application with most users. The FB app has the following set of notifications enabled by default: post on wall, private messages, comments, friendship requests, friendship request confirmations, tags, events, application requests, groups. We used the friendship request because it generates a push notification even if the sender and receiver are not friends. In the Android version, it is "erasable" as specified in Section 2; a subsequent

friendship request cancellation (revoke) removes any trace in the notification bar, making the attack difficult to spot by the user.

As previously stated, the Android Facebook app does not use the built-in notification system, but sets up, maintains and tears down its own data channel through a persistent Android background service. MQTT [20] messages originated by or directed to Facebook servers transit on these connections. We can easily separate notification data from regular Facebook browsing using reverse DNS lookups, as the former is addressed to servers whose associated name contains the string 'mqtt'. Figure 3 shows the keep alive messages that transit on channel, sent every 60 s to keep the network path up, in an easily recognizable pattern.

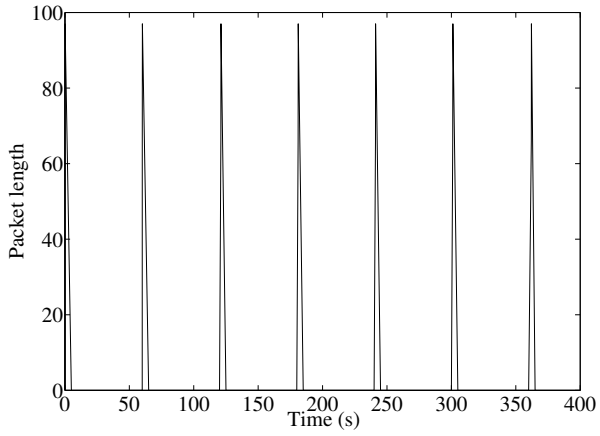


Figure 3. When idle, only heartbeat traffic is visible. This is one packet from client to server and one in the opposite direction, with the respective answers.

When the friendship request arrives, an IP packet of 196 bytes is pushed from the server to the mobile device, followed by a second bigger packet of 615 bytes and a small return packet (85 bytes). Then, a new connection is set up with a different IP address, and a relatively large amount of data is downloaded for a total of 5333 bytes exchanged at the IP level. We suppose these data contain the image and other friendship request meta-data displayed in the notification bar. On the former socket, a small encrypted packet of 195 bytes is sent for the friendship cancellation. This behavior is shown in Figure 4, where we can see a high peak at $t = 153$ corresponding to the friendship request, followed by a smaller peak at $t = 161$, which matches the friendship cancellation request.

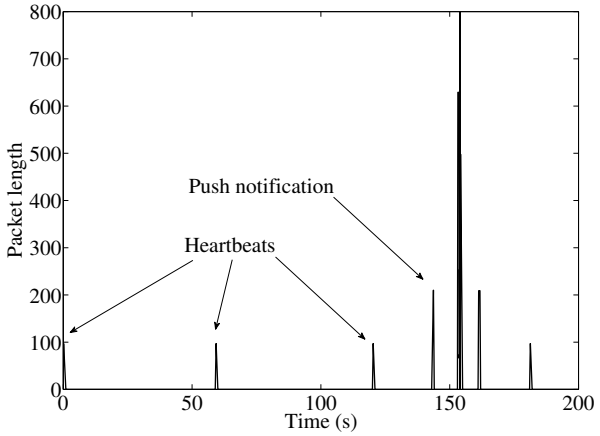


Figure 4. Friendship request and subsequent friendship revoke: received packet lengths over time.

With reference to the steps described in Section 3, we implement the push notification attack in the following way.

Step 1: list possible profile candidates:

To perform the attack, we must leverage prior knowledge of where to search for the list of potential profiles to verify. We consider three different cases: Facebook groups, Facebook participants to an event and Facebook profiles that like a given page. Retrieving the list of users in the first two cases is straightforward as we can use the Facebook graph API to fetch all the members of a group (https://developers.facebook.com/docs/graph-api/reference/v2.4/group/members) or attendees to an event (ttps://developers.facebook.com/docs/graph-api/reference/event/attending/). There are no direct APIs for Facebook pages, probably to protect privacy and prevent spam. Then, we use the Facebook social plugin (https://developers.facebook.com/docs/plugins/page-plugin) designed to embed a widget in a web page showing a random list of people that liked a page. This list gives us a valuable hint about the profiles to check since it includes all users that like a pub, soccer team, political party, campus, conference, course, etc. Using a simple program, we poll this plugin several times to populate the list of possible profile candidates to test.

Step 2: trigger push notifications to all candidates:

Facebook does not provide an API to make friendship requests. By design, Facebook prevents making friendship requests through their Graph API to avoid abuses. Therefore, we use a web browser automation library called Selenium (http://www.seleniumhq.org/). With this library, we build a pool of bots that log in to the social network with different accounts and process the list of candidates by sending and canceling a friendship request to each one, as shown in Figure 5. Facebook limits the number of friendship requests to a few dozen (this quota depends on inner mechanisms: empirically, the number of requests toward a target depends on the relationship between requester and target, e.g., if they share friends or are members of the same group), but increasing the number of fake profiles and inserting a delay, we can easily cope with this limitation. We repeat that we choose to use friendship requests for the reason described above. However, using private messages lets us forge packet length, facilitating pattern matching, and sending messages to a group will trigger a notification to all its members, easing the discovery process.

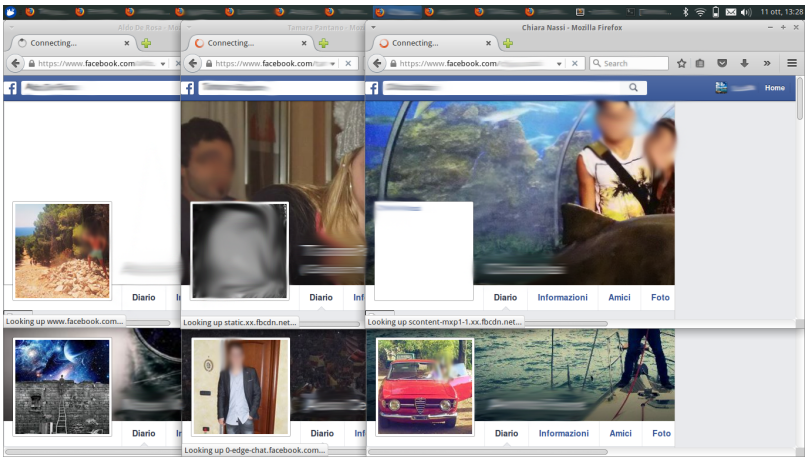


Figure 5. Pool of bots triggering push notifications to a set of profiles.

Step 3: Inspect the data over the air, searching for a well-known data pattern:

We put the WiFi interface in monitor mode and sniff the packets over the air. When we observe an IP packet size of 210 bytes (header + payload) coming from port 443 of a Facebook server whose associated name contains 'mqtt', we mark the user as potentially revealed. A revoke request is 294 bytes

long. In this case, the message transits on the Facebook custom notification channel. Empirically, we see that the size of the packet could vary by a few bytes depending on the length of the Facebook ID. If more statistical guarantees of false positives/negatives are needed, one can repeat the experiment or send a pre-determined number of bytes in a private message to a selected subset of users. To cope with WPA encrypted networks with known passwords (as is the case in many public venues), we pre-process the data with a decrypting program (dot11decrypt) that implements monitor mode on 802.11 networks with WPA/WPA2. The decoded traffic is encapsulated using Ethernet frames and written to a tap device.

With iOS devices, the attack can be conducted in a similar way except that notifications no longer come from custom channels, but from the APN. This makes recognition harder as we cannot filter out the packets coming from Facebook IPs. However, we can filter the Apple subnet to separate push notifications from the rest of the traffic. What we have is aggregated traffic of all push notifications addressed to the device as shown in Figure 6. Empirically, friendship requests correspond to packets 590 bytes long.

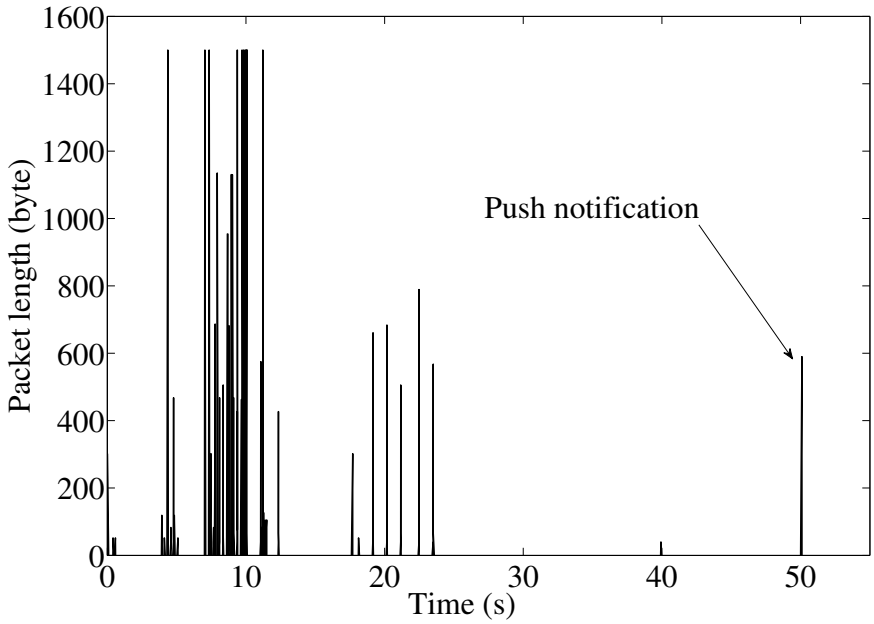


Figure 6. iPhone trace: the push notification is the last spike. The inability to separate Facebook push notifications from other app notifications makes recognition harder than with Android.

The code used in the attack is freely available online (https://github.com/netgroup/push_attack/).

5. Measurements

Being a side channel attack, we are interested in the following information:

- How strict is the temporal binding between the notification triggering and the packet's arrival on the network?
- What is the user identification false positive ratio?

To answer the first question, we repetitively trigger Facebook friendship request notifications and measure the time between the triggering and notification delivery to the mobile device, to estimate the application level round trip time (RTT). Figure 7 reports the empirical distribution of the RTT for a sample of 50 friendship requests. In particular, we recognize the first packet of the push notification that, in turn, triggers other connections responsible for fetching accessory data needed to properly

display the notification (e.g., fetching the requester's profile image). Therefore, even if the notification is viewed on the mobile device after several seconds, the packet of interest can be captured after, on average 1.08 s. Despite it being far bigger than the end-to-end network delay (measured at 12 ms), the application RTT is fast enough to enable quite a precise correlation between the triggering of the mobile notification and the packet's detection on the network.

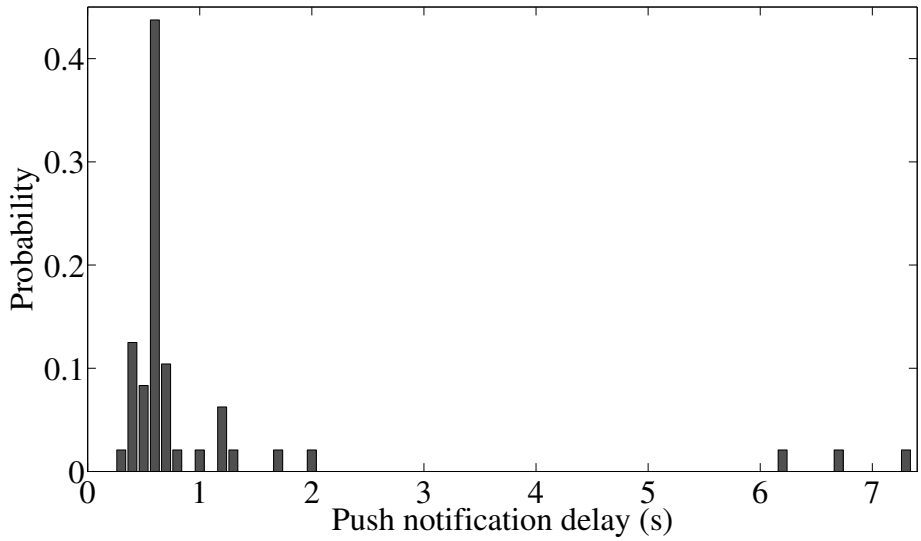


Figure 7. Empirical PDF of round trip time of Facebook friendship request push notification.

As for the second question, we point out that while the attack is performed, victims can also receive push notifications originated by regular application behavior (e.g., a regular friendship request). The attacker can mistake these push notifications for the ones they originated for the attack. In this case, the attacker may erroneously associate a user's identity with the wrong virtual identity, leading to a false positive detection.

Because the attacker knows the length of the push notification that they are searching for, the false positive ratio depends in turn on the distribution of the lengths of push notifications received by users. For instance, a push notification of 196 bytes can be erroneously attributed to a friendship request event.

To estimate the false positive rate and to tune the parameters of the attack, we conduct a measurement campaign in several different environments: a bar, a concert hall, a popular fast-food restaurant, a square, a library. We choose environments where most users connect to the free WiFi with their mobile phones, and we analyze 2 GB of mobile traffic. We extrapolate from the captured data the distribution of the lengths of the push notification messages from Facebook servers to clients. As we can see from Figure 8, the high spike at 66–97 bytes derives from the keep-alive mechanism. The other lengths are quite evenly distributed from 100 to 430 bytes. This fact, on the one hand, suggests the possibility of forging custom length notifications (e.g., private messages) to better cope with a possible false positive detection. On the other hand, the relatively low number of notifications in 2 GB of sniffed traffic demonstrates that application-specific push notifications are quite rare events. This ensures a high degree of precision even in the case of fixed size notifications such as friendship requests.

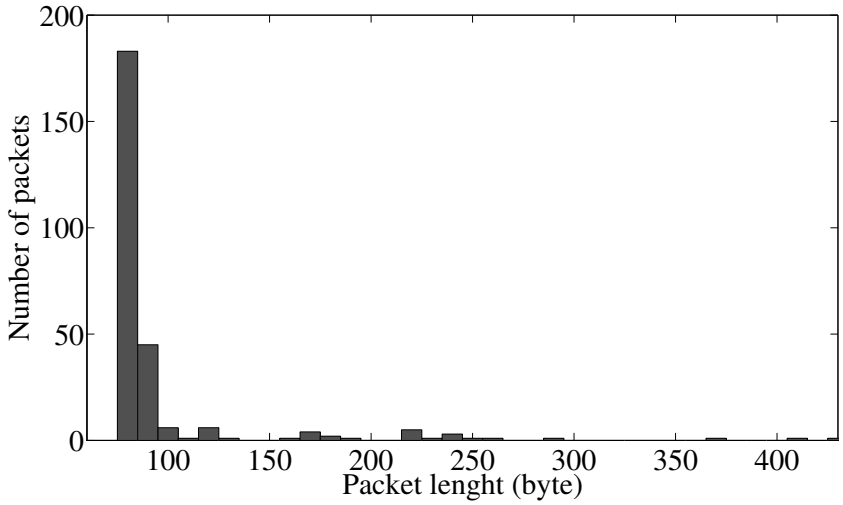


Figure 8. Histogram of the length of Facebook push notifications.

6. Related Work

Mobile OS Vulnerability

Recent works have pointed out how the mobile operative systems present several privacy vulnerabilities that can be exploited by malicious software to retrieve personal data. In some cases, the data are retrieved actively. This is the case of [21], where the authors present a phishing attack based on an installed Trojan application that shows fake notifications to the user.

Sometimes, the apps can just overuse their authorizations to leak personal user data. This problem occurs in open OS, such as Android, where there is no control of the real app behavior. To prevent this abuse, in [22], the authors propose a method for the detection of the personal information leakage. Moreover, since Android Version 6, the user is requested to authorize app permissions at the execution time and not only at the installation time.

As described in Section 3, our approach only needs to trigger push notifications and sniff the network traffic to find a correlation. This is different from other approaches such as [21], as we do not require any control of the victim's smartphone or any special applications installed on their device.

Side Channel Attacks

In addition to basic OS vulnerabilities, the mobile terminals are naturally exposed to attacks based on the wireless traffic analysis that can be performed sniffing the broadcast channel or operating man-in-the-middle attacks. Information encryption is the main technique adopted to protect user privacy in cellular and wireless networks. For example, both Facebook and Twitter communicate with their respective backends using HTTPS and MQTT SSL. However, many works (such as [23–27]) have demonstrated that privacy can be violated by simply analyzing the encrypted traffic, i.e., operating what is often called side-channel information leaks. Our work is in the same field of analyzing ciphered data.

Encrypted traffic patterns can be easily used to detect user activities such as used apps, visited web pages, identification, etc. These attacks operate irrespective of the deployed encryption means and allow one to extract, from the statistical analysis of the generated packet sizes and of their inter-arrival times, valuable confidential information such as the employed applications [28], the application layer protocols [29], the physical devices used [30] or the web page accessed [31]. The side channel attack is more effective in smartphones. For example, in [11], built-in sensors have been utilized to extract application usage patterns. In [9], the authors show that laptop users' activities can be inferred by

MAC addresses using machine learning methods. Moreover, tools for the privacy attacks are freely available: for example, [6] is a distributed tracking and profiling framework that allows one to perform tracking and profiling of mobile users through the use of WiFi. Unlike other side channel attacks on encrypted traffic, the proposed attack is based on active triggering. Consequently, the statistical analysis is devoted solely to finding a well-known pattern in the eavesdropped data. Most other works focus on classifying traffic to infer user data.

Attacks to Anonymity

Both OS vulnerabilities and side channel attacks can be used to acquire personal information of users such their real identities. An example of this attack is reported in [14], where the authors present a technique to link social network profiles with whitepages.com user data. They showed that in some cases, it is possible to correlate the location of the published posts or tweets with the user home location. We differ from [14] as we do not exploit any correlation between public data available on different social network platforms. Furthermore, simple actions such as clicking the Facebook "Like Button" can be used to accurately predict highly sensitive personal data. Some protocols are able to keep un-linkable the identity of their authors with attributes she/he wants to reveal (see [32]). Attacks to privacy in cellular networks have been reported for both GSM [33] and LTE [16]. These attacks are devised to understand the user location exploiting the characteristics of cellular wireless protocols. In particular, [16] shows how it is possible to use a standard LTE terminal to receive the Smart Paging messages generated by the incoming connections (such as VoLTE (Voice over LTE) or application-level messages). They present a methodology similar to the one presented in this paper in one of the proposed attack showing that push notification (together with notifications of Voice over LTE) triggers page notifications that can be sniffed and correlated with the globally unique temporary identifier (GUTI). This analysis complements our work since it is based on active triggering and sniffing on mobile networks to disclose user information. Yet, it has a different goal (location privacy), uses different techniques (smart paging sniffing) and also a different network scenario (cellular network).

7. Conclusions

In this work, we investigated to what extent it is possible to use application-level real-time push notifications to bind users' virtual and real identities and the related privacy concerns. With this aim, we analyzed the behavior and the usage of push notification systems within the most popular mobile social networks and related mobile applications. Most Android applications are easier to attack because developers can use custom/proprietary push notification infrastructures. In these cases, attackers can discover the push notification's source infrastructure. Conversely, iOS applications make use of the built-in OS notification service that multiplexes the notifications of all the applications, making traffic recognition harder. To demonstrate the feasibility of binding by exploiting push notifications, we implemented an example of the attack on the Facebook mobile application. We evaluated the false positive rate through a real-world measurement campaign highlighting the key aspects that characterize the occurrence of push notifications with respect to normal traffic. The push notification attack bypasses the standard technological protections of user privacy, since it works at the application level without the need for additional software on the victim's device. It can be exploited to connect the real to the related virtual identity, causing serious privacy concerns. Future work will focus on assessing the performance of the offline attack using an extended dataset to measure the impact of the attack and on considering different operating systems.

Author Contributions: Lorenzo Bracciale provided the original idea, the description of the attack, and the data analysis; Alberto Caponi performed the security assessment of the attack; Pierpaolo Loreti provides the technological support, the feasibility study and the proof of concept.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. *The Smartphone Difference*; Technical Report; Pew Research Center: Washington, DC, USA, 2015.
2. Guerses, S.; Diaz, C. Two Tales of Privacy in Online Social Networks. *IEEE Secur. Priv.* **2013**, *11*, 29–37.
3. Yang, Y.; Lutes, J.; Li, F.; Luo, B.; Liu, P. Stalking Online: On User Privacy in Social Networks. In Proceedings of the Second ACM Conference on Data and Application Security and Privacy, San Antonio, TX, USA, 7–9 February 2012; pp. 37–48.
4. IETF. Webpush Working Group. Available online: https://tools.ietf.org/wg/webpush/ (accessed on 30 January 2018).
5. *Meraki Whitepaper CMX*; Technical Report; Cisco: San Jose, CA, USA, 2015.
6. Wilkinson, G.; Cuthbert, D. Snoopy: A Distributed Tracking and Profiling Framework. Available online: https://www.sensepost.com/blog/2012/snoopy-a-distributed-tracking-and-profiling-framework/ (accessed on 25 February 2014).
7. Stanton, J.M.; Stam, K.R. The Visible Employee: Using Workplace Monitoring and Surveillance to Protect Information Assets–without Compromising Employee Privacy Or Trust; Information Today, Inc.: Medford, NJ, USA, 2006.
8. Chen, S.; Wang, R.; Wang, X.; Zhang, K. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, 16–19 May 2010; pp. 191–206.
9. Zhang, F.; He, W.; Liu, X.; Bridges, P.G. Inferring Users' Online Activities Through Traffic Analysis. In Proceedings of the Fourth ACM Conference on Wireless Network Security, Hamburg, Germany, 14–17 June 2011; pp. 59–70.
10. Wang, Q.; Yahyavi, A.; Kemme, B.; He, W. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS), Florence, Italy, 28–30 September 2015; pp. 433–441.
11. Do, T.M.T.; Blom, J.; Gatica-Perez, D. Smartphone Usage in the Wild: A Large-scale Analysis of Applications and Context. In Proceedings of the 13th International Conference on Multimodal Interfaces, Alicante, Spain, 14–18 November 2011; pp. 353–360.
12. Di Luzio, A.; Mei, A.; Stefa, J. Mind Your Probes: De-Anonymization of Large Crowds Through Smartphone WiFi Probe Requests. In Proceedings of the 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, USA, 10–14 April 2016.
13. Arunkumar, S.; Srivatsa, M.; Rajarajan, M. A Review Paper on Preserving Privacy in Mobile Environments. *J. Netw. Comput. Appl.* **2015**, *53*, 74–90.
14. Alsarkal, Y.; Zhang, N.; Zhou, Y. Linking virtual and real-world identities. In Proceedings of the 2015 IEEE International Conference onIntelligence and Security Informatics (ISI), Baltimore, MD, USA, 27–29 May 2015; pp. 49–54.
15. Goldreich, O. *Foundations of Cryptography: Volume 2, Basic Applications*; Cambridge University Press: Cambridge, UK, 2004.
16. Shaik, A.; Borgaonkar, R.; Asokan, N.; Niemi, V.; Seifert, J. Practical attacks against privacy and availability in 4G/LTE mobile communication systems. arXiv **2015**, arXiv:1510.07563.
17. Stöber, T.; Frank, M.; Schmitt, J.; Martinovic, I. Who Do You Sync You Are?: Smartphone Fingerprinting via Application Behaviour. In Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks, Budapest, Hungary, 17–19 April 2013; pp. 7–12.
18. Zhang, F.; He, W.; Chen, Y.; Li, Z.; Wang, X.; Chen, S.; Liu, X. Thwarting Wi-Fi Side-Channel Analysis through Traffic Demultiplexing. *Wirel. Commun. IEEE Trans.* **2014**, *13*, 86–98.
19. Wright, C.V.; Coull, S.E.; Monrose, F. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 8–11 February 2009.
20. OASIS. MQTT Specifications. Available online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html (accessed on 30 January 2018).
21. Xu, Z.; Zhu, S. Abusing Notification Services on Smartphones for Phishing and Spamming. In Proceedings of the 6th USENIX Conference on Offensive Technologies, Bellevue, WA, USA, 6–7 August 2012; p. 1.

22. Choi, J.; Sung, W.; Choi, C.; Kim, P. Personal information leakage detection method using the inference-based access control model on the Android platform. *Pervasive Mob. Comput.* **2015**, *24*, 138–149.
23. Herrmann, D.; Wendolsky, R.; Federrath, H. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier. In Proceedings of the 2009 ACM Workshop on Cloud Computing Security, Chicago, IL, USA, 13 November 2009; pp. 31–42.
24. Jiang, T.; Wang, H.J.; Hu, Y.C. Preserving Location Privacy in Wireless Lans. In Proceedings of the 5th International Conference on Mobile Systems, Applications and Services, San Juan, Puerto Rico, 11–13 June 2007; pp. 246–257.
25. Michalevsky, Y.; Nakibly, G.; Schulman, A.; Boneh, D. PowerSpy: Location Tracking using Mobile Device Power Analysis. In Proceedings of the 24th USENIX Security Symposium, Washington, DC, USA, 12–14 August 2015.
26. Sun, Q.; Simon, D.; Wang, Y.M.; Russell, W.; Padmanabhan, V.; Qiu, L. Statistical identification of encrypted Web browsing traffic. In Proceedings of the Security and Privacy, Berkeley, CA, USA, 12–15 May 2002; pp. 19–30.
27. Conti, M.; Mancini, L.V.; Spolaor, R.; Verde, N.V. Can't you hear me knocking: Identification of user actions on Android apps via traffic analysis. *arXiv* **2014**, arXiv:1407.7844.
28. Bernaille, L.; Teixeira, R.; Salamatian, K. Early Application Identification. In Proceedings of the 2006 ACM CoNEXT Conference, Lisboa, Portugal, 4–7 December 2006.
29. Crotti, M.; Gringoli, F.; Pelosato, P.; Salgarelli, L. A statistical approach to IP-level classification of network traffic. In Proceedings of the IEEE International Conference on Communications, Istanbul, Turkey, 11–15 June 2006; Volume 1, pp. 170–176.
30. Kohno, T.; Broido, A.; Claffy, K. Remote physical device fingerprinting. *IEEE Trans. Dependable Secur. Comput.* **2005**, *2*, 211–225.
31. Bissias, G.D.; Liberatore, M.; Jensen, D.; Levine, B.N. Privacy Vulnerabilities in Encrypted HTTP Streams. In Proceedings of the 5th International Conference on Privacy Enhancing Technologies, Cavtat, Croatia, 30 May–1 June 2005; pp. 1–11.
32. Buccafurri, F.; Fotia, L.; Lax, G.; Saraswat, V. Analysis-preserving protection of user privacy against information leakage of social-network Likes. *Inf. Sci.* **2016**, 328, 340–358.
33. Kune, D.F.; Koelndorfer, J.; Hopper, N.; Kim, Y. Location leaks on the GSM Air Interface. In Proceedings of the NDSS Symposium, San Diego, CA, USA, 5–8 February 2012.