A Look Into the Information Your Smartphone Leaks

Timothy A. Chadza§, Francisco J. Aparicio-Navarro*, Konstantinos G. Kyriakopoulos†, Jonathon A. Chambers*
§Electrical Engineering Department, University of Malawi-The Polytechnic, Blantyre, P/Bag 303, Malawi
*School of Electrical and Electronic Engineering, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK
†School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, Loughborough, LE11 3TU, UK
e-mails: tchadza@poly.ac.mw, {francisco.aparicio-navarro, jonathon.chambers}@ncl.ac.uk, k.kyriakopoulos@lboro.ac.uk

***Abstract*—Some smartphone applications (APPs) pose a risk to users' personal information. Events of APPs leaking information stored in smartphones illustrate the danger that they present. In this paper, we investigate the amount of personal information leaked during the installation and use of APPs when accessing the Internet. We have opted for the implementation of a Man-in-the-Middle proxy to intercept the network traffic generated by 20 popular free APPs installed on different smartphones of distinctive vendors. This work describes the technical considerations and requirements for the deployment of the monitoring WiFi network employed during the conducted experiments. The presented results show that numerous mobile and personal unique identifiers, along with personal information are leaked by several of the evaluated APPs, commonly during the installation process.**

Keywords—Information Leaking; Mallory Proxy; Man-in-the-Middle Attack; mitmproxy; Mobile APPs; Smartphone Security; WiFi Networks

I. INTRODUCTION

Smartphone applications (APPs) have been developed at a tremendous pace over recent years. In May 2016, market analysts indicated that there were more than 3 million APPs available on the market [1]. Many businesses and private users benefit from the services that they provide, and make daily use of these APPs. Nonetheless, some APPs also pose a risk to users' personal information, as they have access to an increasing amount of information about their users and their cyber activities [2]. Some studies have suggested that the main business model for some APP developers is based on the commercialisation of personal information by leaking it to third parties, such as advertising companies [3]. Incidents of APPs leaking personal information stored in the smartphones illustrate the danger that the use of these applications pose [4].

In this paper we focus on investigating the amount and type of personal information that benign APPs might be leaking, and the potential privacy-related threats that these applications present. In some cases, the developers are responsible for the insecurity of the APPs. Smartphone APPs commonly do not use secure communication protocols, as many APPs often use the Hypertext Transfer Protocol (HTTP) for communication [5]. Even when HTTP Secure (HTTPS) is used, if certificates are not properly validated, smartphone APPs can cause serious security and privacy issues. In fact, the authors of [6] identified a large number of APPs that contain Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols that either accept all certificates or all hostnames as trusted certificates. However, on other occasions, the human factor is responsible for the insecurity of the APPS, due to the access permission that is granted. When an APP is installed and launched for the first time, the user is usually prompted with an access permission request to personal information stored in the device. Most users do not understand the implications of granting permission to this information to the APPs, nor consider what mobile unique identifiers and personal unique identifiers might be shared due to this permission.

When an APP makes a system call to the smartphone Operating System (OS), it can seek access to a broad list of the user's personal information, such as the list of contacts, location, device name, unique identifiers (e.g. device ID), calendar, reminders, photos and videos, notes, accounts, call information, and WiFi connection information. Accessing this information is not necessarily a breach of a user's privacy, as long as the information is utilised locally to provide appropriate service [3]. However, when this information is transmitted remotely to third parties, then allowing access to this information becomes a privacy concern. Although most smartphones have provision for disabling tracking settings, this does not guarantee that some confidential information is not leaked by the APPs. Therefore, it is essential to consider carefully the security and trustworthiness of the smartphone APPs being installed and used.

In this paper, the main aim is to evaluate the amount of personal information leaked during the installation and use of APPs when accessing the Internet. We consider that an APP leaks personal information when such data are transmitted to a third party without the user's consent. We have opted for the implementation of a Man-in-the-Middle (MitM) proxy to intercept the network traffic generated by the smartphone APPs. To achieve this, it is necessary to design an active WiFi monitoring platform for the interception, decryption and analysis of a user's private information derived from popular APPs installed on different smartphones of distinctive vendors.

The remainder of the paper is organised as follows. In Section II the most relevant related work is reviewed. The experimental methodology followed in this paper is explained in Section III. This includes the description of the network testbed and the APPs selection, as well as the monitoring system configuration. Section IV describes the experimental results. Finally, conclusions and future work are given in Section V.

II. RELATED WORK

Smartphone APPs have access to a broad list of a user's personal information. Therefore, it is essential to have a complete understanding of what information can be accessed by these APPs, and to assess the permission that they have to handle this information. Also, it is essential to evaluate the amount of information that is leaked by smartphone APPs to third parties.

Generally, Apple iOS devices do not require permission from the user. iOS gives limited access to many of the device's sensitive information. Only in certain cases are permission requests presented to the user. In contrast, during the installation of every Android APP, a list of all the permissions that the APP requires is presented to the user. The user has to decide whether the APP needs access to the requested information or not.

Other researchers have also focused on investigating the security of mobile APPs and the amount of information that they leak. In [7], the authors present a comparative analysis of the present state of mobile device security. The authors focus their study on Android and Apple iOS devices. The presented research analyses the smartphone security from different angles, such as the provenance, permissions, and encryption techniques of the APPs used by the two evaluated OSs. However, the authors do not practically evaluate the information that might be leaked to third parties by the smartphone APPs.

The authors of [4] describe a framework, which is an extension of the Android OS, that tracks in real-time how different APPs access and manipulate a user's personal information. The main objective of this work is to analyse the flow of privacy sensitive information through APPs, and to detect when personal data are leaked via untrustworthy APPs. The presented framework leveraged the Android's virtualised architecture to integrate four granularities of trace propagation.

In [8], the authors present a dynamic analysis platform that detects private information being leaked by APPs in Android and iOS devices. The platform makes analysis directly at the OS level of the devices. This work presents thorough comparison analysis of the data leaked by both smartphone OSs.

The authors of [2] present an automated tool to identify possible privacy breaches in iOS APPs, and analyse the threat they present to a user's personal information. The presented tool constructs control flow graphs to perform data flow analysis, which would allow the identification of flows that might leak personal information to third parties without a user's permission. This work focuses only on iOS APPs. Since no source code is available from the APPs, the tool that the authors present has to perform its analysis directly on the binaries.

In [6], the authors introduce a tool to detect potential vulnerabilities against MitM attacks posed by benign Android APPs that use SSL/TLS protocols. This tool implements a static code analysis of different aspects of the APPs, such as the validity of URLs found in APPs and examines inadequate SSL/TLS validation made by the APPs. Additionally, the authors conduct a real MitM attack against different APPs installed in a real phone, and audit the information leaked via potentially broken SSL communication channels.

III. Experimental Methodology

A. Testbed

One feasible method to evaluate if there is information leaked is by the interception of the generated network traffic communication by a smartphone using a MitM proxy. An experimental WiFi network testbed has been deployed to conduct our experiments. We have set up a MitM proxy using the software tools Mallory [9] and mitmproxy [10], which intercept and decrypt the communication in a transparent manner to the network users. The experimental WiFi, depicted in Fig. 1, includes one Access Point (AP) connected to the Internet through the University's network, one Laptop acting as the MitM monitoring machine running the MitM proxy and a rogue AP tool, and various smartphones acting as clients.

The MitM monitoring machine runs on Linux Ubuntu 16.04.1. It is connected wirelessly to the AP, and in turn provides access to the Internet to all associated smartphones through a rogue AP service. This laptop has a built-in wireless Network Interface Controller (NIC) that does not support packet injection. Therefore, an external ALFA Atheros wireless adapter with the Atheros 9271 chip was used during the experiments. The smartphones used as clients were one iPhone 4s running iOS 8, one iPhone 5 and one iPhone 5s running iOS 9, one Samsung Galaxy J5 running Android 4.1.2 OS, and one Samsung Galaxy S7262 running Android 5.1.1 OS.

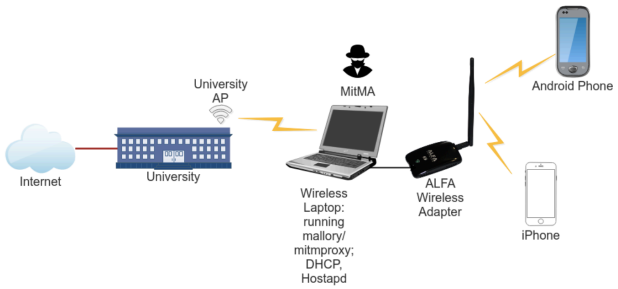

Fig. 1. Schematic design of the IEEE 802.11 network used for monitoring and interception of the data generated by the smartphones.

B. Shortlisting Mobile Applications

According to the market statistics published in May 2016, there were 3 million Android and iOS APPs available on the market and, by 2015, iOS APPs alone were downloaded more than 100 billion times [1]. Hence, it was essential to shortlist the number of APPs used in this work.

During May 2016, we used the rankings shown in iTunes and Google play in order to select the top 20 APPs on iOS and Android, respectively. It was initially observed that the ranking for the most relevant APPs was different for the two smartphone OSs. On the other hand, there are APPs exclusively to one OS, not available on the other OS. Hence, the adopted solution was to arbitrarily select 60 top APPs from both Android and iOS, and then identify those available for both OSs. The 20 smartphone APPs selected are shown in Table I.

TABLE II. List of Selected Mobile APPs on Android and iOS

Smartphone APPs			
Whatsapp	Pinterest	Skype	Slither.io
Facebook	Soundcloud	eBay	Snapchat
Tripadvisor	Microsoft Outlook	Stack	Messenger
Tinder	Amazon BuyVIP	Color Switch	Instagram
Uber	Twitter	Musical.ly	Spotify

C. Monitoring System Configuration

1) Selecting the Rogue Access Point Attacking Tool

One of the crucial steps during our experiments was to provide access to the Internet to the client smartphones, through the MitM monitoring machine, in a transparent manner to the clients. The smartphones would access the Internet through the ALFA Atheros wireless adapter and the communication had to be intercepted by the MitM proxy tool. The monitoring machine can create its own rogue AP by using several publicly available software, such as HostAPd [11] and Airbase-ng [12].

Airbase-ng and HostAPd are tools for turning a Linux wireless NIC into an AP. For the purposes of our work, they have been used for launching rogue AP services, i.e. lure the clients to associate with it, instead of connecting to the legitimate AP. The rogue AP masquerades as the legitimate AP, using the same MAC address. In case the wireless device of a client is already authenticated and associated with the legal AP, each of the tools can spoof the identity of the legal AP and send disassociation frames to the wireless device. After the client has been disassociated, rogue AP tools advertise themselves as the legal AP by sending beacon frames.

Throughout the course of our experiments using an Atheros based chipset wireless NIC, we found that the performance, particularly in terms of client-rogue AP connectivity, is better with the HostAPd tool. In addition, we have found that the HostAPd tool, in contrast to Airbase-ng, flags the MAC layer frame retransmissions appropriately. This is an advantageous feature because HostAPd achieves a performance that closely resembles the expected behaviour from a legitimate AP. After assessing both tools, it was concluded that that HostAPd was the most appropriate to be used in the experiments.

2) Man-in-the-Middle Proxy Selection and Configuration

Another essential step was to consider the selection of the MitM proxy. There exist numerous options that can be used to implement the MitM proxy, including Mallory, mitmproxy, Burp Suite, Ettercap, and Charles web debugging proxy. A brief description of well-known HTTP proxies written in Java and Python can be found in [13]. A list of considerations that help with the selection of the MitM proxy is provided in [14].

Mallory was initially chosen for our experiments among all the available proxies since it meets all the required parameters. Mallory allows the implementation of an extensible TCP/UDP MitM proxy that is designed to run as a communication gateway, and can listen to SSL/TLS encrypted network traffic from/to the smartphones. According to [9], there are three different installation setups for Mallory. These are WiFi hotspot, Point-to-Point Tunnelling Protocol (PPTP), and virtual machine. The Mallory installation setup WiFi hotspot allows the installation of a wireless card acting as an AP through which a victim can connect, and the rogue AP can forward the traffic through Mallory. Unfortunately, Mallory was discarded at a later stage because it was unable to unencrypt correctly the SSL/TLS intercepted network traffic in our bespoke setup. Therefore, several other proxies were evaluated and Mimtproxy, an alternative to Mallory was consequently adopted.

Similarly to Mallory, mitmproxy has various modes of operation. Regular is the default mode where mitmproxy needs to be assigned in the client's proxy configuration settings. In cases where we do not have control over the client, mitmproxy should work in transparent mode. In this case, the mitmproxy can be configured as the client's next hop node through the Dynamic Host Configuration Protocol (DHCP) settings. Finally, there is a reverse mode of operation, which is only used to proxy traffic from a server to a client.

The mitmproxy Certificate Authority (CA) should install a certificate in the mobile devices, otherwise, the client can refuse the SSL/TLS handshake and cancel the communication. For each SSL/TLS encrypted destination server, the mitmproxy CA will generate a dummy certificate on-the-fly to impersonate the visited website. The CA is created the first time mitmproxy is initiated [10]. It is worth noting that some APPs employ HTTP Public Key Pinning (HPKP) to prevent possible MitM attacks. Warning messages will be triggered if the client receives an untrusted certificate that it is not configured to accept. Hence, to circumvent this problem, social engineering methods are usually employed to lure clients to accept unauthorised certificates.

The documentation in [10] provides two possible options for installing the CA certificate in the mobile devices: quick setup or manual setup. In the former, there is provision for a built-in certificate installation application which can be accessed, using what is called *magic domain mitm.it*, through a web browser. A three-step example of the quick installation of the mitmproxy CA is displayed in Fig. 2. The client should select the icon that corresponds to its smartphone. The selection of the appropriate icon leads to the installation of the CA. On the other hand, the manual setup is used when the quick setup is not available. mitmproxy has provided a list of pointers to manual certificate installation documentation for some common platforms in [10].

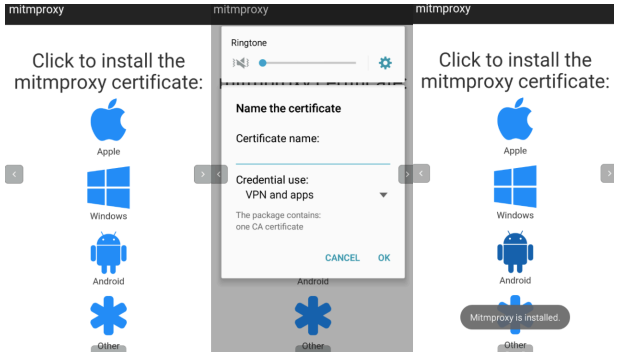

Fig. 2. Overview of the three-step mitmproxy CA installation process. a) The client selects the appropriate platform icon; b) The mitmproxy CA is installed; c) The smartphone confirms the correct CA installation.

There is still a need for running the DHCP server and then assigning firewall rules using the iptables tool. There was also the need for port redirection and port masquerading while running mitmproxy. By default, mitmproxy listens on TCP port 8080, and, in order to permit interception of HTTP and HTTPS, ports 80 and 443 had to be forwarded to port 8080. The enabling of Network Address Translation (NAT) is also required. Both the NAT functionality and the port forwarding were done using the iptables tool. Further details about all the commands and steps undertaken to install, configure and customise the system can be found in [15]. After completing the configuration, a smartphone connected to the rogue AP could access the Internet. The Ubuntu variant for assigning DHCP is the isc-dhcp-sever. This is a necessary step to assign IP addresses to clients connecting to the rogue AP tool HostAPd.

IV. Evaluation Analysis and Results

This section describes the findings from running the two MitM proxies. It is worth noting that Mallory proxy was unable to successfully decrypt the communication traffic. Hence, most of the results provided in this section are from mitmproxy.

The intercepted network traffic was stored in an SQLite database when Mallory was utilised, whereas it was stored as binary files when the MitM proxy used was mitmproxy. In the case of the Mallory proxy, the tool DB Browser for SQLite [16] was chosen to extract and analyse the information due to the data

manipulation flexibility that this tool provides. In the case of mitmproxy, the MitM proxy itself provides a mechanism to read binary files and this method was used when applicable.

Mitmproxy was able to intercept many personal and private parameters. We have focused our analysis on the unique identifiers International Mobile Equipment Identity (IMEI), International Mobile Subscriber Identity (IMSI), Unique Device Identifier (UDID), Universally Unique Identifier (UUID), Mobile Country Code (MCC), Mobile Network Code (MNC), MAC address of the smartphones, address book contacts, email, usernames, passwords, and location.

A. *Analysis of Intercepted Data by Mallory Proxy*

The clients could access the Internet through the rogue AP, and Mallory proxy was able to capture the communication streams, as well as the source and destination IP addresses. Upon running different APPs, the data intercepted were saved and extracted using DB browser for SQLite. In total, five tables were created: *connections*, *dgram*, *flows*, *fuzztcp* and *fuzzudp*. All the tables were critically analysed to identify whether a user's personal information is actually leaked by the evaluated APPs.

We found that, apart from a number of GET and POST requests, the rest of the intercepted information was still encrypted despite the use of Mallory. Despite multiple technical configuration changes, Mallory was unable to decrypt the communication. Both Airbase-ng and HostAPd were tested to determine if the use of an alternative rogue AP would produce different results, but this was not the case as similar behaviour was observed. The lack of readable decrypted information proved that the use of Mallory proxy was ineffective, and this is the reason why we decided to discard the use of Mallory. As part of the experiments, it was also observed that when a connection was established to a client, the connection was extremely slow and the communication link was disconnected in most cases.

B. *Analysis and Results of Intercepted Data by mitmproxy*

During the experiments conducted with mitmproxy, the APPs were downloaded and installed in the smartphones while connected to the rogue AP in the monitoring machine running the MitM proxy. The reason for doing this was to observe if any personal information was leaked during this required process towards the use of the APP. Then, we accessed and clicked on all the available links within the APP for at least 10 minutes. During the installation process or when the APPs were launched for the first time, a list of all the resources that the APP required permission to access was presented to the user. However, it is not a simple process to verify what specific information each APP actually accesses. It is expected that this information is utilised by the APPs locally to the device to provide appropriate service [3]. Additionally, there were some cases in which the APP did not request access permission.

Focusing first on the results extracted from the smartphones running Android OS, mitmproxy shows that several unique identifiers were leaked by different APPs. Mitmproxy GUI presents three tabs that can be accessed and different pieces of personal information are presented in each of the tabs. These are *Request*, *Response*, and *Details*. For instance, Fig. 3 shows the mitmproxy output when the APP *Twitter* was installed and launched. In this example, the UDID of the smartphone was leaked, along with other device ID parameters, such as the IP address. An assessment on the APP *Whatsapp* also shows that the MAC address, UUID, UDID and IMEI were all leaked during an update. This is shown in Fig. 4. However, after completing the update installation process, not all information observed was leaked. This implies that there are APPs that leak information mainly during the installation process. Furthermore, the APP *Whatsapp* requested access permission to email addresses, contact names, phone numbers, and images. Another example is the assessment of the APP *Spotify*. Mitmproxy was able to intercept, among others, the IMEI and IMSI from the overall information leaked by this application.

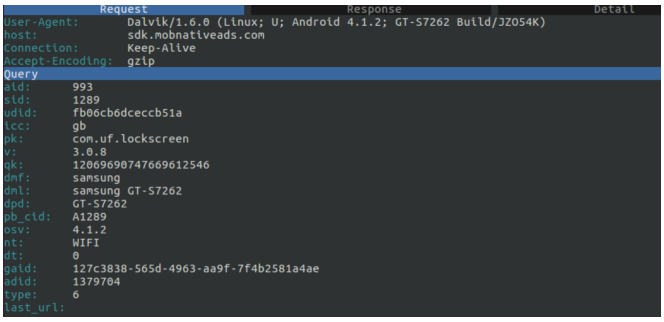

Fig. 3. Mitmproxy GUI: Overview of the leaked information intercepted when the APP *Twitter* was installed and launched in an Android OS device.

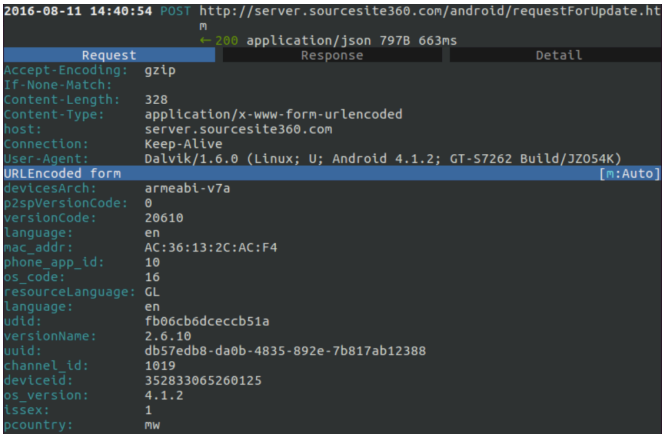

Fig. 4. Mitmproxy GUI: Overview of the leaked information intercepted when the APP *Whatsapp* was installed and launched in an Android OS device.

In order to verify that our mitmproxy implementation was correctly configured, we used multiple mobile web applications (e.g. access *Facebook* through a web browser rather than a dedicated APP) to securely authenticate with the web server. Mitmproxy was successful in intercepting the username and password in these situations. These results indicate that all keystrokes entered in the mobile web applications were being captured and decrypted by mitmproxy.

Table II presents an overall description of all the information leaked that we have identified after analysing all the chosen APPs. In summary, the device model, OS version, codename, IP address, device ID and country name were leaked by all the APPs. The results also indicate that 5 out of the 20 APPs leaked the IMEI and IMSI. Only 3 APPs leaked information regarding the location. Furthermore, almost all the APPs leaked the MNC and MCC. The MAC address was leaked only by the APPs *Whatsapp* and *Pinterest*. Also, neither *Uber* or *Stack* leaked any email address, username and password. Only 6 out of the 20

examined APPs leaked email addresses, whereas the rest of the APPs leaked email addresses, username and password.

It is also worth noting that some of the APPs could not be installed in Android 4.1.2 due to incompatibility problems. In such cases, the APPs were installed only on the available Android 5.1.1. Furthermore, despite the two Android phones accepting the mitmproxy certificate, there were cases in which a warning message of client certificate error was shown. However, the user could ignore these messages and the connection would, therefore, be accepted. Finally, mostly in the case of Android 5.1.1, a warning message was shown indicating that that the communication might be compromised. These messages were ignored for experimental purposes.

Once we finished with the analysis of the phones running Android OS, we focused our experiments on the iOS devices. In contrast to the previous experiments, it was not possible to install any APP in any iPhone while connected to the rogue AP. This applies to all the iPhones that were tested. Hence, we decided to install the APPs using a legitimate AP and then, once the APPs were correctly installed in the iPhones, proceed with the analysis using the rogue AP. Although this approach would let us to proceed with the experiments, the results obtained from the devices running Android could not be used for comparison, since we are missing the possible information being leaked during the installation process. Therefore, a set of comparison experiments was conducted in both Android OS and iOS during the usage process of the APPs, not considering the installation process. During this new analysis, ten APPs were evaluated. The results for these experiments are presented in Table III.

As we can see from the results, none of the APPs leaked the MAC address, contacts, IMEI and IMSI. These results indicate that multiple APPs leak information solely during installation process. This is clear specially in the case of the IMEI and IMSI. Also, information regarding the location is leaked by almost the same number of APPs both in Android and iOS (i.e. 3 and 2 out of 10, respectively). Furthermore, there is an evident difference between the amount of information leaked by the APPs in Android and iOS, when observing the rest of the evaluated unique identifiers and personal information. For instance, the logging details were leaked by 8 APPs on Android while none of the APPs on iOS leaked this information. Similarly, the MNC and MCC, and the UDID and UUID were leaked by 7 and 5 APPs, respectively, in Android. In contrast, this information was only leaked by 1 APP in iOS. Additionally, there were two

TABLE II. PERSONAL INFORMATION LEAKED DURING THE INSTALLATION AND USE OF APPS RUNNING IN ANDROID OS

APP	**Confidential Parameters of Interest**						
	IMEI / IMSI	***Location***	***Email / Username / Password***	***Contacts***	***MAC Address***	***UDID / UUID***	***MCC / MNC***
Whatsapp	Yes	No	Email only	Yes	Yes	Yes	Yes
Facebook	No	No	All	No	No	Yes	Yes
Tripadvisor	Yes	No	All	No	No	Yes	Yes
Tinder	No	No	All	No	No	Yes	Yes
Uber	No	No	No	No	No	Yes	Yes
Pinterest	Yes	No	All	No	Yes	Yes	Yes
Soundcloud	No	No	All	Yes	No	Yes	Yes
Microsoft Outlook	No	No	All	No	No	Yes	Yes
Amazon BuyVIP	No	No	Email only	No	No	Yes	Yes
Twitter	No	No	Email only	Yes	No	Yes	Yes
Skype	No	No	Email only	Yes	No	Yes	Yes
eBay	No	Yes	All	No	No	Yes	Yes
Stack	No	Yes	No	No	No	Yes	Yes
Color Switch	No	No	Email only	No	No	Yes	Yes
Musical.ly	No	No	All	No	No	Yes	Yes
Slither.io	Yes	No	All	No	No	Yes	Yes
Snapchat	No	No	All	No	No	Yes	Yes
Messenger	No	No	Email only	No	No	No	Yes
Instagram	No	Yes	All	Yes	No	Yes	No
Spotify	Yes	No	All	No	No	Yes	Yes

TABLE III. PERSONAL INFORMATION LEAKED DURING THE USE OF APPS RUNNING IN ANDROID OS AND APPLE IOS

APP	**Confidential Parameters of Interest**													
	IMEI / IMSI		***Location***		***Email / Username / Password***		***Contacts***		***MAC Address***		***UDID / UUID***		***MCC / MNC***	
	Android	**iOS**	**Android**	**iOS**	**Android**	**iOS**	**Android**	**iOS**	**Android**	**iOS**	**Android**	**iOS**	**Android**	**iOS**
Accuweather	No	No	Yes	Yes	No	No	No	No	No	No	No	No	Yes	Yes
Slither.io	No	No	No	No	Yes	No	No	No	No	No	No	Yes	No	No
Musical.ly	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No
eBay	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No
Messenger	No	No	No	No	No	No	No	No	No	No	Yes	No	Yes	No
Spotify	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No
Pinterest	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No
ISS Tracker	No	No	Yes	Yes	Yes	No	No	No	No	No	No	No	Yes	No
Tripadvisor	No	No	Yes	No	Yes	No	No	No	No	No	Yes	No	Yes	No
Soundcloud	No	No	No	No	Yes	No	No	No	No	No	Yes	No	Yes	No

instances in which information relating to the location, and the UDID and UUID was not leaked during the initial experiments shown in Table II, but was leaked during the second set of experiments. Overall, these results evidence that, although the APPs in iOS leak less information than in Android OS, several pieces of personal information may still be leaked in iOS APPs.

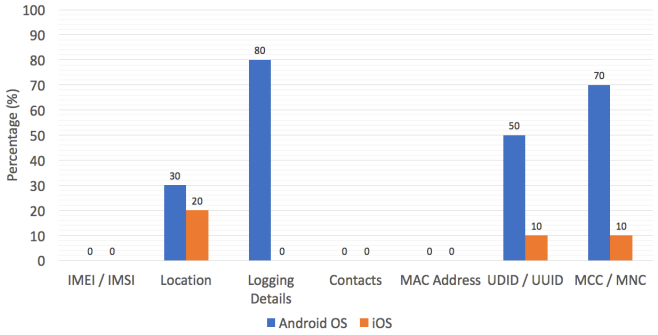

Fig. 5. Android OS and iOS comparison; Percentage of evaluated APPs that leak unique identifiers and personal information during the usage process.

V. Conclusions and Future Work

In this paper we have evaluated the amount of personal information and unique identifiers that benign smartphone APPs leak to third parties during the installation and use of the APPs, when accessing the Internet. We have implemented a MitM proxy setup to intercept the network traffic generated by the APPs. An active WiFi monitoring platform has been designed for the interception, decryption and analysis of a user's personal information derived from popular APPs installed on different smartphones of two distinctive vendors.

The experimental methodology implemented to conduct the experiments has involved the configuration and evaluation of multiple software tools. During the assessment of the two rogue AP tools used to provide access to the Internet through the MitM monitoring machine, it was concluded that HostAPd was the performing better in our experiments. We found that HostAPd, in contrast to Airbase-ng, flags the MAC layer frame retransmissions appropriately. This is an advantage because HostAPd achieves a performance that closely resembles the expected behaviour from a legitimate AP. Similarly, we have evaluated two different MitM proxies, Mallory and mitmproxy. Mallory was initially chosen for our experiments, but it was discarded at a later stage as it was unable to unencrypt correctly the SSL/TLS intercepted network traffic. Hence, Mimtproxy was consequently adopted as an alternative to Mallory.

The results presented in this work have shown that numerous unique identifiers and private information are leaked by multiple of the evaluated APPs. One significant finding is that most of the information is leaked solely during the installation process. Hence, it is recommended that APPs are installed only on trustworthy networks to reduce the risk of personal information being compromised. Additionally, iOS APPs were identified to leak much less information than in the Android OS.

The technical advancement that we have conducted on setting up the active monitoring platform has allowed us to be able to implement MitM attacks in a WiFi network. As for future work, we wish to use this platform to enhance our understanding of MitM attacks and thereby assess the possibility of extending and complementing this type of attack with injection capabilities, and to develop a detection mechanism that would accurately identify the presence of these attacks.

Acknowledgment

All the conducted experiments and produced results in this paper were implemented by Timothy A. Chadza at the Wolfson School of Mechanical, Electrical and Manufacturing Engineering, Loughborough University, UK, as part of his M.Sc. dissertation [15]. All the smartphones were acquired for research purposes only, and the smartphones' owner gave explicit permission to conduct the experiments. None of the APPs mentioned in this paper were hacked or modified as part of this work. The personal information described in this paper was collected from the wireless communication channel set up in our testbed. Successive APP updates released after the date this research was conducted may have partially or completely eliminated the issues described throughout this paper.

References

[1] A. Dogtiev, "Top 10 most popular iOS APPs of all time," Available: http://www.businessofapps.com/top-10-popular-ios-apps-time/ (Access Date: 29 Nov, 2016).

[2] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS: Detecting privacy leaks in iOS applications," in *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2011, pp. 177-183.

[3] J. P. Achara, F. Baudot, C. Castelluccia, G. Delcroix, and V. Roca, "Mobilitics: Analyzing privacy leaks in smartphones," in *ERCIM Newsletter*, 2013, pp. 30-31.

[4] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no.2, 2014, pp. 1-15.

[5] A. K. Jain, and D. Shanbhag, "Addressing security and privacy risks in mobile Applications," in *IT Professional*, vol. 14, no. 5, 2012, pp.28-33.

[6] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why Eve and Mallory love Android: An analysis of Android SSL (in)security," in *Proc. of the ACM Conference on Computer and Communications Security* (*CCS*), 2012, pp. 50-61.

[7] I. Mohamed, and D. Patel, "Android vs iOS security: A comparative study," in *Proc. of the 12th International Conference on Information Technology-New Generations (ITNG)*, 2015, pp. 725-730.

[8] J. P. Achara, V. Roca, C. Castelluccia, and A. Francillon, "MobileAppScrutinator: A simple yet efficient dynamic analysis approach for detecting privacy leaks across mobile OSs," in *Proc. of the 32nd Annual Computer Security Applications Conference (ACSAC)*, 2016, pp. 1-14.

[9] Atlassian Bitbucket, "Mallory Wiki home page," Available: https://bitbucket.org/IntrepidusGroup/mallory/wiki/Home (Access Date: 29 Nov, 2016).

[10] A. Cortesi, M. Hils, and T. Kriechbaumer "mitmproxy Project," Available: https://mitmproxy.org/index.html (Access Date: 29 Nov, 2016).

[11] J. Malinen, "Hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIOUS Authenticator," Available: http://w1.fi/hostapd/ (Access Date: 29 Nov, 2016).

[12] Aircrack-ng, "Airbase-ng description" Available: http://www.aircrack-ng.org/doku.php?id=airbase-ng (Access Date: 29 Nov, 2016).

[13] A. Kennedy, "A database of open-source HTTP proxies," Available: http://proxies.xhaus.com/ (Access Date: 29 Nov, 2016).

[14] J. Allen, and R. Umadas, "Network stream debugging with Mallory," in *Intrepidus Group technical report*, 2010, pp. 1-18.

[15] T. A. Chadza, "How much private information is your phone leaking?," M.Sc. Thesis, Loughborough University, August 2016.

[16] R. Hipp, "DB browser for SQLite" Available: http://sqlitebrowser.org (Access Date: 1 Dec, 2016).