Mobile App collusions and its cyber security implications

Abdullahi Arabo
Abdullahi.arabo@uwe.ac.uk
Centre for Complex Cooperative Systems (CCCS).
Department of Computer Science and Creative Technologies
Faculty of Environment and Technology
UWE, Frenchay, Bristol, BS16 1QY, UK

***Abstract*—the key focus in securing mobile software systems is substantiality intended in detecting and mitigating vulnerabilities in a single app or apps developed by the same individual. It fails to identify vulnerabilities that arise as a result of interaction or the colluding of multiple apps either from the same or different vendors. The current state-of-the-art also fails to contextualize this in reference to its impact on security and cyber security issues. This paper proposes a solution that makes use of both static and dynamic analysis, to detect and notify device users of such a vulnerability and equips the user with knowledge on inter-app interaction, the misuse of Personal Identifiable Information (PII) and the sharing of other sensitive information without their consent.**

Keywords—App security; mobile device; app collusion; cybersecurity; Personal Identifiable Information (PII)

I. INTRODUCTION (*HEADING 1*)

The focus of Cyber security is currently on computers and not smart devices. However, there is now a growing development of cyber-criminals who are using malicious apps, with collusion capabilities, to infect smartphones with the intent of accessing PII and collecting private data. The play store is full of malicious apps, which can gain access to information such as address books, GPS coordinates, passwords and pin numbers. This information can then be redirected across the net for malicious use. The apps can also forcefully navigate users' to phishing sites and have the ability to bypass the two-step authentication process used to access an ever-increasing number of online services such as banking or email.

Hence, criminals can monetise this information in a number of ways – by getting your phone to send messages to premium numbers, by remotely controlling an infected phone, by tricking you into revealing passwords and by using your stolen data. Although this can be achieved by apps working on their own, the latest cyber-threat to smartphones comes from apps working together or colluding. Colluding apps can be described as follows (i.e. Android):

- Two or more apps that share Personal Identifiable Information (PII) and or financial data without the user knowledge
- Two genuine apps from same developer would collude with each other to collect PII
- Malicious app could also collude with insecure app to collect PII and send to command and control (C&C)
- Collecting PII from one or more colluding app allows creation of user profiles
- User profile PII would be used by cybercriminals to target users to commit fraud
- Colluding apps are not detected by current mobile security technology
- Spear phishing SMS or email attack vector might also be more successful

An example of collusion consists of one app with permission to access personal data, passing this data to a second app that is allowed to transmit data over the network (see Figure 1). Hence, two or more apps can collectively collect and build a profile of private information that can be used by criminals, with malicious intent without the consent of the user. It is obvious that the user would not normally disclose such information or install apps requiring access to such information on their device. With these collective capabilities, a single app may have the ability to carry out an attack that it would not normally be able to execute alone. The aim of this paper is to investigate new techniques to detect colluding apps and to raise user awareness in an attempt to contain the threat before it becomes widespread.

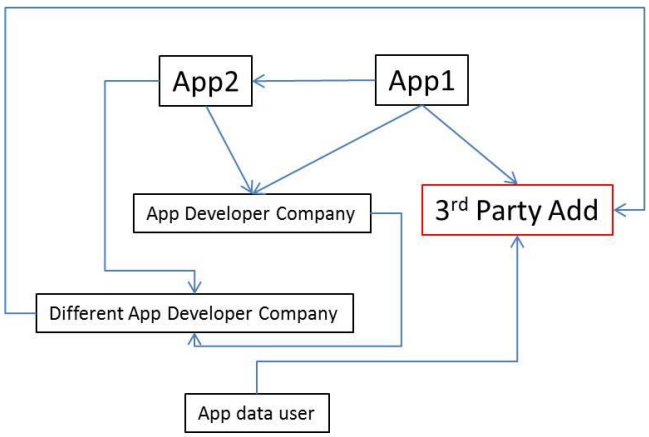

Figure 1 Third Party App Collusion

The problem of app collision can be exploited in two main ways: horizontally or vertically. In the first case, the same malware/app author allowing each application(s) to be responsible for the collection and delivery of the user's personal and sensitive information to a server, before finally exploiting this information for malicious use. In the second instance, one application tailors the use of another insecure application and makes use of exported intents or content providers, to collect information off the user, which would be later used to attack the victim. This in effect will enable an attacker to use another applications' sandbox with the intent of collecting information for another application. These threats are quite new and so far, nothing has been done within the research community, to understand and provide proof of concepts to these attacks or to understand how they work. In this paper we will try to examine these threats from both the application layer/stack and the SELinux kernel level to understand and produce an application/solution that will be able to inject malicious code and collude with other vulnerable apps in the play store. It is evident from our research that the play store adapts the usage of manual/static analysis of apps in their market.

The rest of this paper is structured as follows: in section 2, we cover related work, section 3 describes our proposed app collusion tool, and section 4 provides some discussion on the proposed architecture and research problem. The paper is concluded in section 5, with concluding remakes and future research direction.

II. RELATED WORK

In the realm of smartphone application security, there exists some work on the issues of application security and detecting malicious apps such as TaindDroid [1], RiskRanker [2], AppContext [3], XManDroid [4] and android malware collusions [5]. Based on our knowledge and research, there has been no work done in reference to exploiting and exploring the issues of mobile device app collusion as it is defined in section I. However, there are other related works in reference to the security and privacy of the usage of mobile apps, apps accessing personal information without users consent and malware collusions [5]. The use of mobile devices in healthcare is also more common these days such as in mobile-health. A typical example is having a health device connected to the home network, which is capable of transmitting data wirelessly to hospitals and other relevant parties. Most of the manufacturers of these devices do not work to ensure that these devices are secure, creating an opportunity for app developers to exploit any vulnerabilities present. If these devices are compromised not only will the information and privacy of the user of the device be compromised, but also the attacker will even possess the ability to change the settings of the devices, which could lead to lethal consequences. It has been shown that it is possible to hack into a pacemaker and read the details of the data stored in the device, such as names and medical information without having direct access to the devices (by just being in close proximity) [6].

Therefore, it is also possible to reconfigure the parameters of the device – which has the potential to incur a heart attack. This does not only apply to medical devices, but also any device that are used within a home network. Assuming your child is using their unsecure iPhone, which is connected to the network, a cybercriminal can take that opportunity to groom your child on social media.

According to Juniper Networks report [7], 76 percent of mobile users are relying on their mobile devices to access their most sensitive personal information, such as online banking or personal medical information. This trend is even more noticeable with those who also use their personal mobile devices for business purposes. Nearly nine in ten (89 percent) business users, report they use their mobile device to access sensitive work-related information.

Another more worrying scenario is when cybercriminals use the vast resources of the network to turn it to a botnet and launch a cyber-attack on national critical infrastructures. There are some Android based applications that, when downloaded from a third party (not Android market) are capable of accessing the root functionality of devices ("rooted") and turning them into botnet soldiers without the users' knowledge or explicit consent . People could easily and unwittingly download malware to their smart devices or fall prey to "man-in-the-middle" attacks whereby cyber-criminals pose as a legitimate body, intercept and harvest sensitive information, and then forward to the legitimate recipient so that no one is the wiser [8]. In 2011, there was a bunch of Android apps removed from the Android Market because they contained malware [8]. There were over 50 infected applications - these apps were copies of "legitimate" apps from legitimate publishers that were modified to include two root exploits and a rogue application downloader.

Juniper Networks Mobile Threat Centre (MTC) reported that in 2011 there was an unparalleled increase of mobile malware attacks, specifically a 155 percent increase from the previous year across all platforms [9]. It is also reported that Android Malware experienced dramatic increase of 3,325 percent in 2011. Notable in these findings is a significant number of malware samples obtained from third-party application stores or open market places, which do not enjoy the benefit, or protection from Google's newly announced Android Market scanning techniques. Previously, an Android developer could post an application to the official Android Market and have it available immediately, without inspection or vetting to block pirated or malicious applications. This dramatic malware increase is mainly due to the combination of Google Android's dominant market share (46.9 percent) and the lack of security control over the applications appearing in the various Android application markets. It is reported recently that play store which has more than 500,000 apps just passed 15 billion downloads [8].

Mobile devices including netbooks, iPad's, mobile phones and PDAs are the main devices used in ad hoc

networks. It is worth pointing out that the manufacturers of such devices have done little in terms of informing users if and when their information is used by third parties, or when other third party applications are downloaded and how they use users' personal information. This same issue has been pointed out in the work of Enck et al [10]. Their research was focused on the vulnerability of third party applications deployed on Android devices.

The research findings include half of the 30 applications studied shared location information and unique identifiers with advertisers. They also reveal that 15 of these applications sent out location information without informing users that the data was shared. Some of the applications gathered and dispatched location information even when they were not running normal operations for the users, and some of them also sent out information updates every 30 seconds. Seven of the studied applications shared unique identifiers known as IMEI (International Mobile Equipment Identity) numbers, and others shared users' personal phone numbers or serial card numbers.

Although an IMEI number is only used to identify a device and does not relate to a specific individual, it is still very useful information that, when compromised might raise some security concerns. However, other numbers such as the ESN (Electronic Serial Numbers) and MEID (Mobile Equipment Identifiers) can link an individual to a phone. Usually, an International Mobile Subscriber Identity (IMSI) number stored on a SIM card can identify the subscriber on a network [11].

While some of the applications ask permission to gather information from users before installation, none of them informs the users how the data will be used or who will share it. Hence, users have no control of such information after installing the applications–this implies that users blindly trust the applications. In some cases, applications have a legitimate reason for accessing users' sensitive and private data. However, it is of paramount importance that users have full control of such data and its usage. Thus, there is a need to ensure that the users' data will be used properly and they will be able to revoke the data usage.

Each time you install an app, you are asked to allow the app to access certain information from your device. Most of the time the permission listed is different from the once actually specified in the apps manifest files. This is a standard cost if you are not paying for free apps, but actually you are paying with your own data unknowingly. It is worth pointing out that although sometimes we know that an app can access a particular feature, we probably mentally write it off that is only done occasionally and the information is not misused.

It is not just Android devices that pass on user information to third parties, but also a far more controlled environment like Apple's IOS is also guilty , as reported by the study performed by Eric Smith [12]. There is a historical background for the study. In 1999, Intel announced a Pentium III processor that contained a unique serial number per processor. The main problem with this is that it could be used to track users' online behaviour, and some governments even went as far as asking for a ban on Pentium III processors. Intel removed this serial number shortly afterwards.

The study discussed above was carried out to "determine if the privacy fears surrounding the Pentium 3 have manifested themselves on the iPhone platform" [12]. Hence, the author studied 57 random popular applications from the App Store, and came out with two interesting conclusions:

- "We found that 68% of these applications were transmitting UDIDs to servers under the application vendor's control each time the application is launched. Furthermore, 18% of the applications tested encrypted their communications such that it was not clear what type of data was being shared". The study notes, "A scant 14% of the tested applications appear to be clean. We also confirmed that some applications are able to link the UDID to a real-world identity."
- "For example," the study continues, "Amazon's application communicates the logged-in user's real name in plain text, along with the UDID, permitting both Amazon.com and network eavesdroppers to easily match a phone's UDID with the name of the phone's owner. The CBS News application transmits both the UDID and the iPhone device's user-assigned name, which frequently contains the owner's real name."

The study [12] states that all these pose a real threat to iOS users. "Privacy and security advocates, personal iPhone owners, and corporate iPhone administrators should be concerned that it would be feasible–and technically, quite simple–for their browsing patterns, app usage, and physical locations collected and sold to unintended customers such as advertisers, spouses, divorce lawyers, debt collectors, or industrial spies". The study argues, "Since Apple has not provided a tool for end-users to delete application cookies or to block the visibility of the UDID to applications, iPhone owners are helpless in preventing their phones from leaking this information."

The above two examples have painted a gloomy picture of security worries in mobile applications and devices. Surely, as a matter of principle, devices should not discharge personal information or any information that can be linked to a user for everyone to see without a users' consent. On the other hand, allowing very fine-grained control over these matters will only serve to confuse most users. This confusion could have two outcomes. On the one hand, users could be prompted with a complicated privacy dialog and automatically cancel out of fear.

Or alternatively, by considering just how many applications use personal data, it could also lead to users becoming insensitive to such dialogs [13]. The insufficiency of transparency while exchanging people's identity and other information makes it hard or even impossible for users to participate in the protection process of their identities and personal information. In some applications or services that

provide such facilities, the service providers carry out most of the protection, rather than the users themselves.

This issue is not limited to applications developed for mobile devices. As reported recently, Facebook had to take the action of banning developers that they caught selling user names and contact lists. In the report, it is stated that developers trade user details to data brokers who use the information to target advertising more precisely. However, Facebook did confirm that the sale of user identities did not give access to other personal information. Hence, no private data has been sold or compromised [14].

III. App Collusion Tool

To be able to dynamically detect app collusion scenarios, our tool is designed to inspect the manifest file of each APK, identify the permission request vs the actual permission displayed to the user during installation. We also dynamically check what API's are used within the apps and what other third party apps the original APK is communicating with, or sending PII information to. To contextualise this, the motivation of the research is illustrated in Figure 2.

This is where multiple apps that either belongs to the same developer or multiple developers, request different permissions, but share the different information obtained by each app, with others without the knowledge of consent of the end user. On the other hand another case is where a malicious app is able to dynamically acquire permissions granted by other apps and utilising these permissions without user's consent.

The proposed overall architecture is presented in Figure 3. This is divided into two main components: the front and back end.

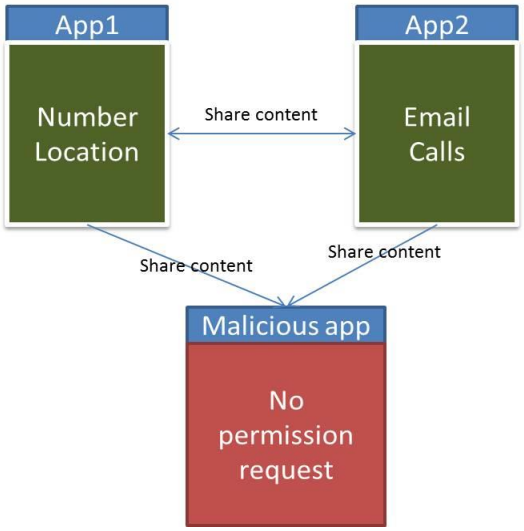

Figure 2 App collusion Scenario

The front end architecture comprises of two main components, a mobile App client and a desktop software client. Where both components of the front-end are resolute on providing dynamic analysis of APK's to identify possible app collusion vulnerabilities and warn the user against possible vulnerabilities.

The desktop version is aimed to provide more granular details and where possible, help to provide a structural view of all possibilities of PII misuse and channels of communication between apps, API's and third parties.

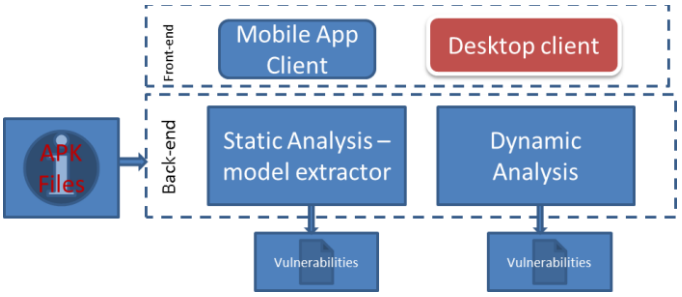

Figure 3 App Collision Architecture

The back end on the other hand also consists of two components: a static analysis model extractor and a dynamic analysis extractor. Both of these take in input in the form of an APK file(s) and outputs a set of identified vulnerabilities.

IV. Discussion

Static analysis model extractor: we have manually decompiled and analysed various apps from the play store, this is done by performing manual analysis of the manifest files and source code. All of the APK's analysed reveal the fact that it uses more permissions then it requests at installation. Also, it appears that some of the APK's contains 2 additional DEX files that are encrypted which require further investigation to establish the role and security implications of these files.

Further investigation was conducted to establish the links between the companies; it has been found that Company A and app developer B have made a joint operation framework agreement for games in Oct 2014. Some of the websites linked to these apps are selling all sorts of security/clean-up software. Our investigation also found that there is a common issue of Company A's internet security installing itself without permission. Another app developed by app developer B uses multiple permissions that are not stated when downloading it, including full control over system settings, access to PII and access to internet. Some of the identified manifest issues are presented in Listing 1. What we think this does is allow broadcasts of type install referrer to be sent to third party website. It also uses third party advertising websites but this could just be related with the in-app adds.

```
Manifest issues:
  <receiver
        android:name="com.appsflyer.MultipleInstallBroadcastReceiver"
        android:exported="true">
        <intent-filter>
          <action
            android:name="com.android.vending.INSTALL_REFERRER"/>
        </intent-filter>
    </receiver>
```

Listing 1

After installing an app called network connections, it tracks all the processes that are connecting to the internet and shows the connection details. When the app's analysed in this research are launched, most of them made multiple connections to amazon cloud services in the US, and Facebook in Ireland and the Netherlands. Possible reason: storing or retrieving something from the cloud, PII, Facebook profile data? The app uses the same ijinshan library as the other app's from the same developer, which we believe is the parent company/site of both. After a bit of reading, it seems that the reason the apps that use the third party site are connecting to amazon cloud and Facebook is to collect PII, and give the app developer anonymous usage statistics. They claim that they don't pass on PII, just process it and pass on anonymous data. They store the PII on the cloud, and send the stats to developers Facebook account. We have no means to test those claims. The more permissions the app acquired, the more accurate stats they can give, so that might explain the reason some apps ask for so many permissions as possible, because we think that up until this year some apps from some countries weren't allowed to charge money for apps in the play store, so they would rely heavily on stats for marketing purposes.

While looking through some of the app source files, we were alarmed to find that significant effort had been placed into programming for Google wallet and Facebook integration. This is of importance since this app does not let you leave from the home screen.

So for that much effort to be placed into programming style sheets, for this level of integration, backs up our assumption that this app is designed to leak user information for explicit purposes.

During an examination of the android manifest file (for one of the apps), a backdoor for the app developers was discovered (Figure 4).

```
−<receiver android:name="com.depop.common.receivers.SecretCodeReceive
  −<intent-filter>
      <action android:name="android.provider.Telephony.SECRET_CODE"/>
      <data android:scheme="android_secret_code" android:host="33767"/>
    </intent-filter>
```

Figure 4 Manifest file backdoor

It is not clear how disastrous this backdoor could be. However, given that the app already has access to several high value permissions; it could be a possible security risk.

Dynamic Analysis: the second aspect of back-end system in to design and implement simple software (app) that will be able to dynamically analyse app APK's and display result to users of the app. The outputs will be a list/set of identified vulnerabilities. To achieve this we proposed an application which uses *content* provider of other applications to access data. However, since the content providers are pre-defined, we can only manage to access data of the listed applications. Moreover, an interesting aspect of this is the fact that we are able to identify that some of the data listed within the application, is not displayed to the user for permission before or during app installation. Another aspect that we have explored is to monitor broadcast intents however, it has been discovered that only simple data can be transferred.

A sample of the implementation code, which is able to read all installed apps within a device and display the requested permissions in the actual manifest files for the apps, is shown in Figure 5.

```
PackageInfo packageInfo = null;
try {
    packageInfo =
packageManager.getPackageInfo(applicationInfo.packageNa
me, PackageManager.GET_PERMISSIONS);
} catch (PackageManager.NameNotFoundException e) {
    e.printStackTrace();
}

String[] requestedPermissions =
packageInfo.requestedPermissions;

final int n = (requestedPermissions != null) ?
requestedPermissions.length : 1;

//final TextView[] permissionText = new TextView[n];

appName.setText(applicationInfo.loadLabel(packageManage
r));

if(requestedPermissions != null) {
    for(int i = 0; i < n; i++) {
        final TextView row = new TextView(context);

row.setText(requestedPermissions[i]/*applicationInfo.pa
ckageName*/);
        linear.addView(row);
    }
} else {
    packageName.setText("No Permissions");
```

Figure 5 Code Sample

Some examples of the output from three sample installed files are also shown in Figure 6.

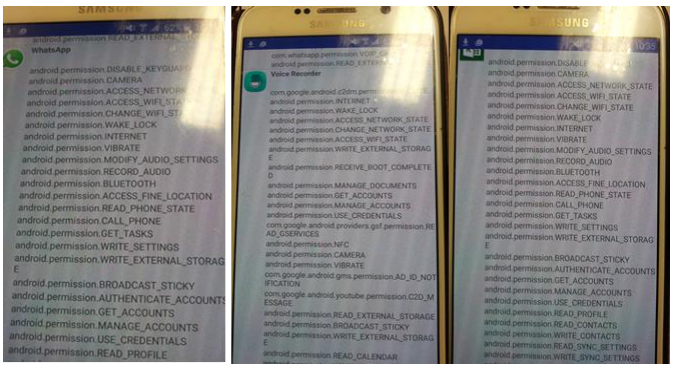

Figure 6 Sample Results

With the recent update/introduction of android M to provide more granular app permission, which most people believe to minimise applications right, and help individuals to revoke and manage when android apps access user data. However, it is worth pointing out that most people are not on android M yet and also if some of these permissions are disabled or revoked it restricts app functionality and sometimes even just crash the app and render it useless. Hence, this is not a solution to app collusion or its vulnerabilities but only a quick fix that will lead to more vulnerabilities and misuse.

V. CONCLUSION

The paper presents the problem of app collusion and its cyber security implications. We have been able to identify that the presented permissions during app installations are not always the same as those listed in the actual app manifest files. We have presented our proposed architecture and early result of the prototype of one of the front end development of the tool. From the result provided is it clear that some inter-application is a serious vulnerability and even though we acknowledge the fact that we allow apps to access certain features within out device. We mentally write it off as this might only occurs as and when it is needed, the app is only accessing the information that we agree during installation and that this information is not been misused. However, this is not always the case. Future work includes an extension of this architecture to apps within iOS and Windows devices forms part of future work for this research project.

Acknowledgement

This forms part of the research funded by VC Early Career award. The authors want to thank Sam Oliver for his initial review of manuscript, Milos Matovic and Vaibhav Mathur for contribution toward proof of concept.

VI. REFERENCE

1. Enck, W., et al., *TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones.* ACM Transactions on Computer Systems (TOCS) 2010. **32**(2): p. Article 5.
2. Grace, M., et al., *RiskRanker: scalable and accurate zero-day Android malware detection.* in Proc. of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys). ACM, 2012: p. 281-294.
3. Yang, W., et al., *Appcontext: Differentiating malicious and benign mobile app behaviors using context.* in Proc. of the International Conference on Software Engineering (ICSE), 2015.
4. Bugiel, S., et al., *Towards Taming Privilege-Escalation Attacks on Android.* In Proc. of the 19th Annual Network and Distributed System Security Symposium, 2012.
5. Elish, K.O., D.D. Yao, and B.G. Ryder, *On the Need of Precise Inter-App ICC Classification for Detecting Android Malware Collusions.* Proceedings of IEEE Mobile Security Technologies (MoST), in conjunction with the IEEE Symposium on Security and Privacy. , 2015.
6. Blumberg, J., *Cybersecurity, Health Care, and Mobile Devices*, in *Dartmouth Now*. 2011.
7. Juniper (2012) *Trusted Mobility Index*. **Volume**,
8. Arabo, A. and B. Pranggono, *Mobile Malware and Smart Device Security: Trends, Challenges and Solutions.* 19th International Conference on Control Systems and Computer Science (CSCS), 2013 2013: p. 526 - 531.
9. Juniper (2012) *Juniper Networks 2011 Mobile Threats Report*. **Volume**,
10. Enck, W., et al. *TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones*. in *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10)*. 2010. Vancouver, BC, Canada.
11. Arabo, A., Q. Shi, and M. Merabti, *Data Mishandling and Profile Building in Ubiquitous Environments*, in *IEEE International Conference on Privacy, Security, Risk and Trust*. 2010, IEEE Cpmputer Society: Minneapolis, Minnesota, USA. p. 1056-1063.
12. Smith, E., *iPhone Applications & Privacy Issues: An Analysis of Application Transmission of iPhone Unique Device Identifiers (UDIDs)*, in http://www.pskl.us/wp/wp-content/uploads/2010/09/iPhone-Applications-Privacy-Issues.pdf *(online)*. 2010.
13. Holwerda, T., *Studies Show Android, iOS Transmit Private Data to 3rd Parties*, in http://www.osnews.com/story/23865/Studies_Show_Android_iOS_Transmit_Private_Data_to_3rd_Parties *(online)*. 07/10/2010.
14. BBCNews, *Facebook uncovers user data sales*, in *BBC News Technology* http://www.bbc.co.uk/news/technology-11665120 *[accessed 06/11/2010]*. 2010.