Trust Evaluation in Mobile Devices: An Empirical Study

Richard Weiss
The Evergreen State College
Olympia, WA, USA
Email: weissr@evergreen.edu

Leon Reznik
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
Email: lr@cs.rit.edu

Yanyan Zhuang
Computer Science and Engineering
New York University
New York, NY, USA
Email: yyzh@nyu.edu

Andrew Hoffman
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
Email: arh4555@rit.edu

Darrell Pollard
Department of Computer Science
Rochester Institute of Technology
Rochester, NY, USA
Email: dap2525@rit.edu

Albert Rafetseder
Computer Science and Engineering
New York University
New York, NY, USA
Email: albert.rafetseder@univie.ac.at

Tao Li
Computer Science and Engineering
New York University
New York, NY, USA
Email: litao.alex@gmail.com

Justin Cappos
Computer Science and Engineering
New York University
New York, NY, USA
Email: jcappos@nyu.edu

***Abstract*—Mobile devices today, such as smartphones and tablets, have become both more complex and diverse. This paper presents a framework to evaluate the trustworthiness of the individual components in a mobile system, as well as the entire system. The major components are applications, devices and networks of devices. Given this diversity and multiple levels of a mobile system, we develop a hierarchical trust evaluation methodology, which enables the combination of trust metrics and allows us to verify the trust metric for each component based on the trust metrics for others.**

The paper first demonstrates this idea for individual applications and Android-based smartphones. The methodology involves two stages: initial trust evaluation and trust verification. In the first stage, an expert rule system is used to produce trust metrics at the lowest level of the hierarchy. In the second stage, the trust metrics are verified by comparing data from components and a trust evaluation is produced for the combined system. This paper presents the results of two empirical studies, in which this methodology is applied and tested. The first study involves monitoring resource utilization and evaluating trust based on resource consumption patterns. We measured battery voltage, CPU utilization and network communication for individual apps and detected anomalous behavior that could be indicative of malicious code. The second study involves verification of the trust evaluation by comparing the data from two different devices: the GPS location from an Android smartphone in an automobile and the data from an on-board diagnostics (OBD) sensor of the same vehicle.

This work was partially supported by NSF grants 1141341, 0834243, 1223588, 1205415, 1241568, and an NYU-AIG (American International Group) grant. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

I. Introduction

Android-based mobile devices and smartphones are becoming increasingly popular. The number of mobile phones sold has surpassed the number of laptops, reaching 1.3 billion in 2014 [1]. Google is reported to have more than a billion active users of Android-based devices [2]. As their popularity increases, so does their value as a target for malware. This is particularly true for low cost smartphones sold in developing countries. According to [3] some vendors there intentionally create conditions facilitating various security violations on these devices. There are many possible risks associated with using compromised devices. Nowadays due to universal interconnectivity and interdependence between devices and networks, the possible compromise of a mobile device will affect not only applications on it and its users, but all other networked computers and communication infrastructure. With the development of the mobile communication platforms that share different devices resources, applications and data, vulnerabilities and security threat will become more wide-spread.

For mobile devices to communicate with each other and download apps securely, it is desirable to compute trust metrics among them. Trust can be modeled at multiple levels, e.g. at the application level, on a device (hardware and software), or among a network of devices. Ultimately, our goal is to integrate these metrics into a single conceptual framework so that we can reason about complex systems at a higher level, and we can also use them to verify trust for individual components. The trust evaluation could be applied to optimize data collection and communication schemes in order to satisfy multiple criteria such as data quality, overall system performance and/or resource consumption, subject to the constraints based on security and privacy requirements. Also, the user of a device may benefit from the trust evaluation as it might provide useful information about areas in need of improvement. The trust evaluation could also be combined with other techniques for non-signature based intrusion detection.

The more sophisticated mobile devices become, the more complex the threat model is, and the more opportunities there are for vulnerabilities to appear. Trust evaluation should be sensitive to the detection of viruses and other malicious agents in a system. However, finding viruses and other malware using software signatures is less likely to work. Signature based intrusion detection systems have to be complemented with a system-wide approach that involves assessing trust for the different components by detecting anomalies in sensor-originated data.

The concepts of trust and trust evaluation have been discussed by many others [4], [5], [3]; however, it seems that the problem of quantification is largely unsolved, especially with respect to complex systems. Yet, the nature of our work points to ways in which this could be used to produce secure or trustworthy systems. This paper presents the development of the novel hierarchical model that enables the evaluation of trust for a network of mobile devices. Trust evaluation depends on numerous factors. The hierarchical or umbrella structure allows for the inclusion of various trust evaluation systems used to assess diverse trusted components as well as their integration in order to produce a cumulative trust score. Also, it allows for extending the framework by inclusion of new trust metrics and facilitates both self-evaluation for a particular device as well as the collaborative evaluation of diverse devices and applications. The paper describes the version developed for Android-based devices.

The rest of this paper is organized as follows. Section II describes the framework design principles and an overall architecture. Also, this section briefly describes a few of the trust evaluation metrics developed for individual apps and smartphones that are included in the current implementation. Details about a few of the trust evaluation metrics are provided in Section III–V. In particular, Section III discusses metrics for individual apps based on measuring resource utilization such as battery voltage, CPU and network bandwidth usage, which can be done on the smartphone. Section IV describes metrics based on multiple sensors that impact the level of privacy supported, and how privacy-enhancing tools can interact with trust evaluation. Section V discusses the possible ways in which trust evaluation can be verified and adjusted, based on multiple sources of data. We show that data collected simultaneously from a smartphone and on-board diagnostics (OBD) sensor of an automobile can be used in trust evaluation.

II. Framework Umbrella Architecture and Design

A hierarchical trust analysis can look at the entire system and combine measurements from multiple sources, making it more powerful than measuring a single component or layer. The components include smartphone and other mobile devices, which themselves are composed of applications, OS, and hardware. The networks of mobile devices form another layer in the hierarchy. The hierarchical trust analysis includes both interactions among components at the same level, as well as interactions across adjacent levels. We could use the term *umbrella* because the structure is not a strict containment hierarchy. For example, an application could be installed on multiple devices and thus not contained by only one. This section discusses a comprehensive mechanism that provides a scalable and extendable methodology of trust evaluation and analysis, which has been implemented on Android-based mobile devices.

Trust evaluation of a mobile smartphone device is a complex subject, which depends on multiple characteristics, e.g. sensor accuracy, the rate at which messages can be encrypted, and/or the probability of a system's breakdown over a given period of time. Its evaluation should integrate various metrics ranging from the accuracy and reliability of the data sources to the security of the procedures and tools used. The major research challenge of the framework design is integrating the numerous metrics needed to characterize a device's trustworthiness while working with limited resources and processing power.

We address this challenge by hierarchically structuring the composition of trust metrics as well as by designing a specialized calculus to evaluate the overall trust metric. Therefore, the major innovative aspect of our framework design is the integration of a wide variety of indicators and their evaluation procedures. The framework procedures output the overall trust evaluation indicators and additionally calculate the individual metrics characterizing system features, which could then be used to produce recommendations for improvement. The trust evaluation will facilitate decision-making, improve performance and increase accountability through the collection, analysis, and reporting of relevant performance-related data. This design facilitates the framework extension through the inclusion of other metrics, as well as the ease of modification and improvement, as shown in Figure 1. The current implementation of this framework includes the following measures and functionality:

1) **The analysis of the installed applications through the application-specific meta-data provided by the Play store.** Applications represent the largest security and privacy risk to a device and user's data. The data provided by the Play store leverages the experiences of millions of users and holds all data associated with the distribution of an application including its associated documentation. The Play store also provides meta-information about applications that gives useful characteristic data about an application. These data can be used to assess an individual application's risk. Rules can be generated to classify each application into an impact class based on this meta-data. The combined trust classes of all applications installed on a device would be used to create a security risk rating for the device.
2) **The usage of security tools embedded into the operating system and proper preventative security practices.** Android provides users with many different tools which increase the security and privacy of their devices in addition to updates that patch exposed vulnerabilities. When properly used, these

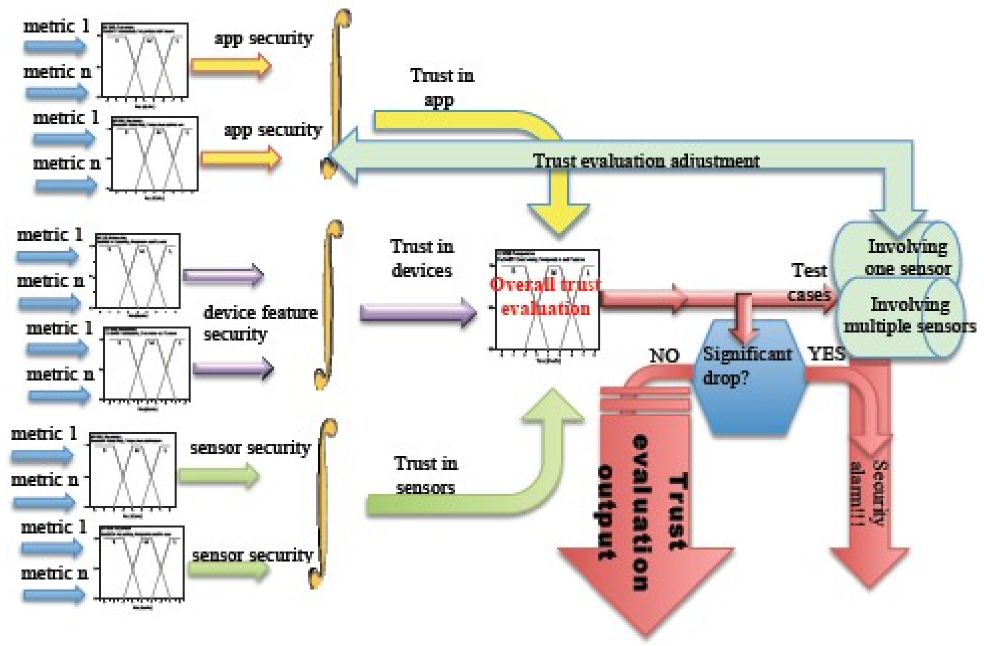

Fig. 1. Framework operation and architecture.

tools improve the security of the devices. They are intended to gather a comprehensive overview of the software running on a mobile device by analyzing the operating system version and user settings. First, the operating system is checked to confirm that it is running the most recent version available. Second, the personal security settings on the device are examined to determine if the user is utilizing the appropriate tools to secure the device. These operating system verification checks are combined to generate a score, which is the output of the framework operation.

3) **The evaluation of trust based on the level of privacy provided.** The greater the level of privacy provided by a device, the more trustworthy the device is. An anomalous value of a device's internal sensors can sometimes indicate the existence of a privacy or security problem that would otherwise be missed by the other metrics. Most mobile devices now come equipped with a variety of sophisticated sensors which are capable of measurements of their surrounding environment. As the data from these sensors are used in more security critical applications, the importance that these data remain accurate and legitimate should not be underestimated. For example, data from the GPS sensor can be verified to be trustworthy and assigned a trust rating. The combination of ratings from all sensors would produce the device's sensor trust score.

The result of applying these metrics is to assign a trust level to an application based on usage patterns (this is a part of the overall trust evaluation hierarchy). The classifications are:

1) Low trust: These applications are considered to have a low trust evaluation and a high probability of its negative impact on the overall device security;
2) Moderate trust: These applications are evaluated to have less negative impact on the overall device security;
3) High trust: These applications are considered to have a high trust evaluation or a low probability of negative impact on the overall device security.

Unlike other available tools, our framework has an umbrella structure that allows for integration of diverse trust evaluation mechanisms by means of a rule-based classification system. This open architecture can also be extended to include self-learning capabilities to allow for its optimization towards a particular device and a criteria set. Each of these procedures given above generates a rating, which is then integrated into the umbrella framework. This framework takes into account the varied landscape of mobile devices and is designed to be flexible and easily adaptable to a changing security environment. Based on this design, the contribution of each of the procedures can be adjusted depending on the target.

We evaluate overall trust based on a separate analysis of each application. The first step lists all of the applications installed on the device. After that the manifest file for each application installed is analyzed in order to fetch the application name, package name, required features, version, required permission, path info, date on which the application was installed and the target SDK version. This information is used to evaluate trust according to the classification. However, this is refined using application category. This information could be retrieved from the Google Play store. The Google Play store holds all data associated with the distribution of an application including its APK file and associated documentation. Following features can be retrieved from the meta-data:

- Number of installs: Total number of installs across the apps life;
- Number of reviews: Total number of reviews from unique users;
- Score: User rating of 1.0 to 5.0;
- Developer: Name of the developer;
- Permissions: Which resources can be accessed by the app.

The first three fields can be used to find an applications popularity that, when matched with a history of values, shows user trend information. Although it may not be possible to determine if an application has a security risk based on this information, data from a large number of users could be very reliable, and they can be used in rules in combination with other data [4].

Here are some examples of rules that are used:

- If number of downloads were low and had low ratings, the application was classified as low trust.
- If the number of downloads were low and the application score was good, the application was classified as moderate trust.
- If the application had lower recent score in comparison against the previous scores, it was classified as moderate trust, stating that there was something wrong with the latest patch released by the developer.
- If the application was from a unknown publisher with low score and low number of downloads, it was classified as low trust.
- If the application was from an unknown publisher with high number of downloads and high score, it was classified as moderate trust.

III. Trust Evaluation Metrics Based on Technological Parameters

Anomalous misbehavior of a smartphone may be due to malicious exploit of an application, the operating system, or the hardware. In order to detect these anomalies, we first need to establish baseline measurements for normal behavior. In this paper, we complement the standard debugging and anomaly detection techniques and focus on metrics that could be obtained without inspection of the apps source or binary code due to the very sophisticated obfuscation techniques that will continue to be developed. We have enhanced some existing debugging techniques with additional parameter measurements to inspect app behavior. The three measurements that we chose are battery voltage, CPU usage, and network usage, which are easy to collect. The following subsections describe how these three measurements can be linked to specific applications and can be used to monitor their behavior and, therefore, adjust their trust evaluations.

A. Battery voltage

The battery voltage is a proxy metric for the amount of remaining energy in the battery. Any kind of activity on the device will result in a change of battery voltage, but the resolution of readings (both in time and in voltage) only allows for coarse, averaged measurements under general, non-lab conditions. If the circumstances can be controlled tightly, then approaches like *Eprof* [6] can be used to estimate energy consumption of individual activities and assign credit to likely originators.

TABLE I. A comparison of battery use with anomalous adware compared with normal. The ad was 15 seconds.

Time	**15s**	**60s**
% of total battery use with ads (avg)	81.4%	53.0%
% of total battery use with ads (max)	86.0%	61.5%

In the following experiment, the battery voltage was measured both with and without video ads. The battery voltages were used to deduce the current being drawn from the battery, and the data are reported in units of current. The baseline is the battery consumption for a music app. This was recorded for 15 seconds and 60 seconds, respectively. Then, the measurement was repeated with both video ads and the music app. The video ads were only running for the first 15 seconds. The smartphone consumes only 31.5 mA with no apps running, and 56.5 mA with the music app running. However, when playing a video ad, the consumption is 169.5 mA. In the experiment, the video ad lasted 15 seconds. Over the first 15 seconds, the average battery use attributed to the video ads was 81% of the total battery use. This is very significant and would be difficult to disguise. The data is shown in Table I.

B. CPU usage

Similar to battery voltage, monitoring the overall CPU usage (which is an operation requiring no special privileges on Android) can be used to get a coarse-grained overview of resource utilization. However, the existing debugging and tracing interfaces permit finer-grained views as well. Assuming an app is not designed to evade or complicate debugging deliberately, the *Dalvik Debug Monitoring Service* (DDMS) provides insight at the system call level[1].

Figure 2 shows a typical session in Traceview, an execution log viewer of the Android platform. The main thread is doing almost all of the work and making system calls to request resources from the operating system. Figure 3 shows what happens when a video ad starts executing. There is a dramatic change in activity and the main app thread is no longer the primary, uniform consumer of CPU cycles. The app under scrutiny comprises several threads with different temporal activity patterns. Some threads bear names indicative of their performed functions. Colored bars in the threads' activity timelines further detail system-call-level interactions with app and system libraries, with the height of sub-bars proportional to the frequency of specific calls. Other log views (not shown) list all of an app's threads, show the call stack for each, and display cumulated and individual CPU time consumption, and relative usage.

[1]In computing, a system call is how a program requests a service from an operating system's kernel. This may include hardware-related services (for example, accessing a hard disk drive), creation and execution of new processes, and communication with integral kernel services such as process scheduling [7].

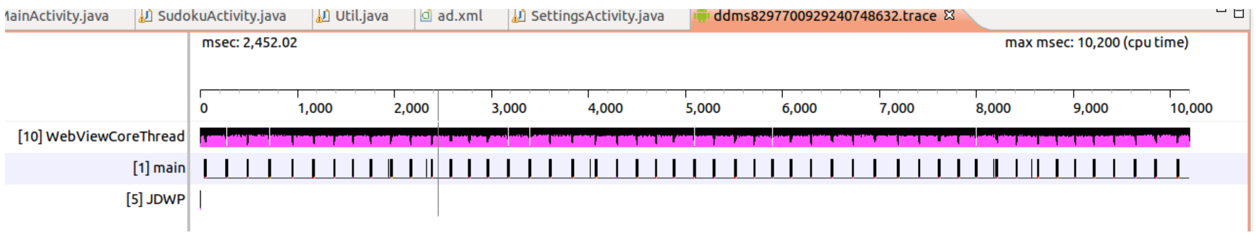

Fig. 2. The Traceview log shows that the WebViewCoreThread is busy, but main thread's workload is very light. main is responsible for the user interface, in general. WebView is responsible for rendering ads using the webkit library.

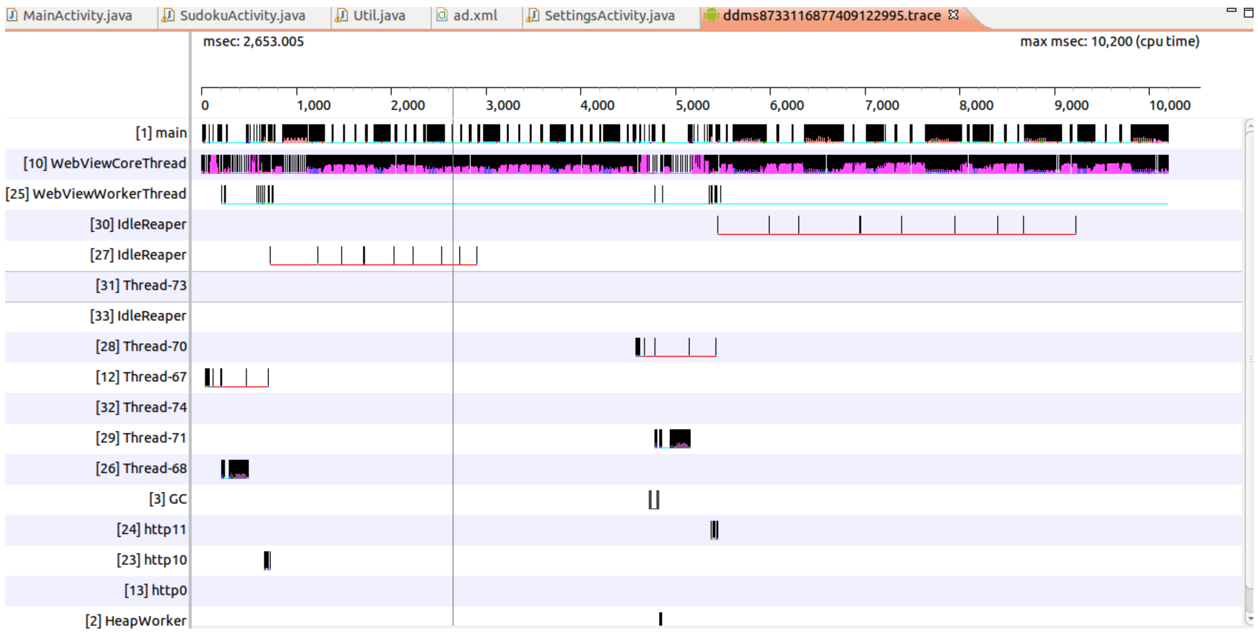

Fig. 3. A trace of the same App with anomalous Ads. Notice that main is doing much more work and there are other threads that are getting significant CPU time.

C. Network usage

Android provides a number of built-in features that allow the observation of a device's network conditions for any app granted the `ACCESS_NETWORK_STATE` permission. This permission allows applications to access information about networks. The `ConnectivityManager` class, an Android class that answers queries about the state of network connectivity and network connectivity changes, lets an app discover the current connectivity status and type (WiFi, 3G, Bluetooth, Ethernet) [8]. For cellular access such as LTE or 3G, the Android `TelephonyManager` class provides access to information about the telephony services on the device and thus makes further details available. The stateful nature of cellular data connectivity is reflected by various indicators of data activity, thereby allowing any app to detect when other apps transfer data over the cellular interface. The Application Resource Optimizer project (ARO [9]) and [10] provide further insights into what is essentially radio resource control (RRC-based type of diagnostics).

If the device is *rooted* (i.e., system-level administrator privileges are granted to the user launching an app), standard packet tracing tools such as `tcpdump` can be used to record the exact data transferred across network interfaces. However, the precondition is not met on almost any commercial stock firmware.

When packet-level tracing on the device is infeasible, the network connection of the device might be tapped instead. A natural place for this would be a WiFi router acting as the device's gateway. Neither on-device nor on-path packet tracing allow decryption of HTTPS and other encrypted network traffic. However, at least for HTTPS implementations using the system libraries, deliberate man-in-the-middle (MITM) attacks on traffic may be attempted by adding a self-provided certificate to the system's certificate storage, and redirecting outgoing HTTPS traffic to a local proxy server using that certificate.

IV. Trust Evaluation Metrics Based on The Supported Privacy Level

Privacy is part of trust evaluation. The greater the level of privacy that an application or device can provide, the greater its trust evaluation would be. A privacy enhancing system such as BlurSense [11] can improve the trustworthiness of an Android device by filtering or restricting the sensor data that an application can access. In addition, one can apply trust metrics to automate the application of BlurSense itself. For example, if an app has a low trust evaluation, then BlurSense can restrict the sensor data that the app has access to.

A. *Motivation*

Although the sensing capabilities of smartphones enhance the convenience of user interfaces and application usefulness, they also raise serious privacy concerns [5]. For instance, through accessing sensor data, malicious applications could retrieve sensitive information about the mobile phone users, such as location, passwords, and credit card numbers [12], [13], [14], [15]. They even might be able to send sensitive information to remote attackers [16], [17]. There has been alarming news about privacy breaches of personal data on smart devices: 25% of Android apps in Google Play can access user's personal data [18]; an iOS app can auto-post false piracy accusations on users' Twitter accounts [19]; apps could steal sensitive information like passwords using the smartphone's motion sensors to determine key taps [12]; and a huge botnet was discovered that was collecting sensor data on more than a million end user smartphones [20]. The Federal Trade Commission (FTC) even recommended that mobile platforms should provide in-time disclosures to users of accessing sensitive content on smart devices [21].

B. *Applying Trust Metrics to BlurSense*

Trust metrics can also be used by privacy enhancing tools such as BlurSense to limit access to sensor data. The current access control to the smartphone resources, such as sensor data, is static and coarse-grained. Take the Android platform as an example, the access permissions are either granted or denied completely during the installation of applications based on a request XML manifest file. As a result, applications may ask for more permissions than are actually required for operation. Having been granted the requested permissions, applications have access to those resources permanently. A trust metric could be used during installation of an app to decide whether or not to deny access specified in its manifest file. A better approach is to restrict access to sensor data in a dynamic and fine-grained framework. Some systems that have been proposed to address this issue require modifications to the Android platform [22], [23]. This increases the cost of maintenance, is less flexible and cannot be used in legacy systems. In addition, the user would need to trust that the new operating system is not malicious and is not more vulnerable than the standard one.

The BlurSense project deals with this problem of fine-grained control of sensor data by requiring that all untrusted apps be installed through an app that filters sensor data through a reference monitor [11], as shown in Figure 4. The policy for filtering can be set by the user based on a trust metric. The reference monitor can exercise fine-grained control to degrade the accuracy of the sensor data, render it completely meaningless, or pass it through unchanged. In the case of geolocation, GPS measurements can be set to the center of the nearest large city or random noise can be added. Another approach would be to allow the user to decide whether to install the app or not, based on the trust evaluation. The reference monitor approach poses less risk than others; the worst that can happen is that the app will have the same access to the sensor data that is permitted by the manifest file. In the best case, BlurSense would limit the access to sensor data based on a combination of trust metrics for the app. BlurSense is currently able to filter data on battery level, CPU

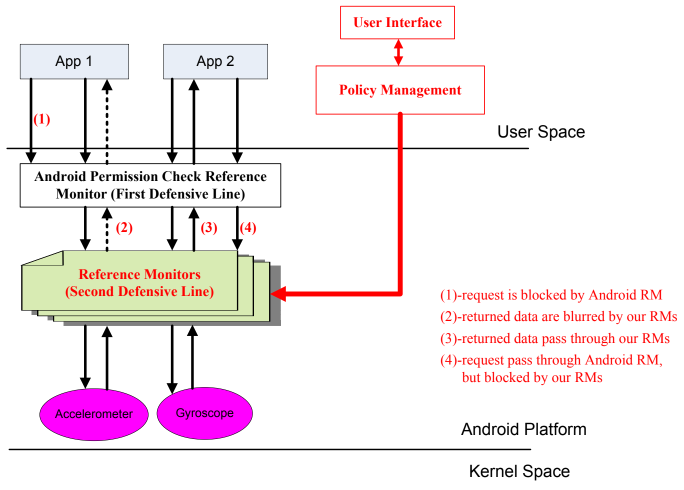
User Interface
Policy Management
App 1
App 2
(1)
User Space
Android Permission Check Reference Monitor (First Defensive Line)
(2)
(3)
(4)
Reference Monitors (Second Defensive Line)
(1)-request is blocked by Android RM
(2)-returned data are blurred by our RMs
(3)-returned data pass through our RMs
(4)-request pass through Android RM, but blocked by our RMs
Accelerometer
Gyroscope
Android Platform
Kernel Space

Fig. 4. System architecture based on reference monitors.

usage, geolocation (latitude, longitude, altitude, accuracy, and speed if available), and network related measurements such as mobile network type and operator, nearby WiFi access point and Bluetooth devices.

V. Trust Evaluation Testing and Verification

Another form of trust evaluation is verification based on the use of sensors on multiple devices [24]. For example, a smartphone may communicate directly with a smart watch or on-board diagnostics (OBD) sensor on an automobile. These other devices have sensors with the same functionality as some of the sensors on the smartphone. If there is a discrepancy in the sensor measurements, the trust evaluation of the device can be reduced unless it can be determined that the trust metric for the external sensor data is low. We introduce a case where smartphones communicate with an in-vehicle OBD sensor to get geolocation information, such as speed, engine RPM, fuel consumption, etc.

A. *Vehicle data collection*

Vehicular data collection consists of a process on a mobile device that directly communicates with in-vehicle sensors and collects sensor data [25]. Data can be encrypted and transferred to a centralized server for permanent data storage. To deploy such a process on a mobile device, a device owner first installs an app [26] on their device[2]. Since we are interested in vehicular data, the target group of device owners are also the vehicle owners. An owner simply inserts a WiFi OBD [27] sensor into the car's OBD port (located under the steering wheel), and connect a smartphone or tablet to the sensor, which also runs as a WiFi access point [28]. Note that OBD systems are available in most cars and light trucks on the road today [29]. Trust evaluation could be used to determine if the phone should trust the car, or vice versa, as well as whether the user should trust the combined system.

The vehicular data is uploaded to a remote server by the app which runs in the background of the mobile device and

[2]Currently, Android smartphones and tablets are supported.

TABLE II. Speed comparison: GPS speed comparing to OBD speed.

	GPS speed	Vehicle speed	Speed difference
Med	53.0 kph	54.9 kph	2.15 kph
STD	12.79 kph	13.61 kph	4.57 kph

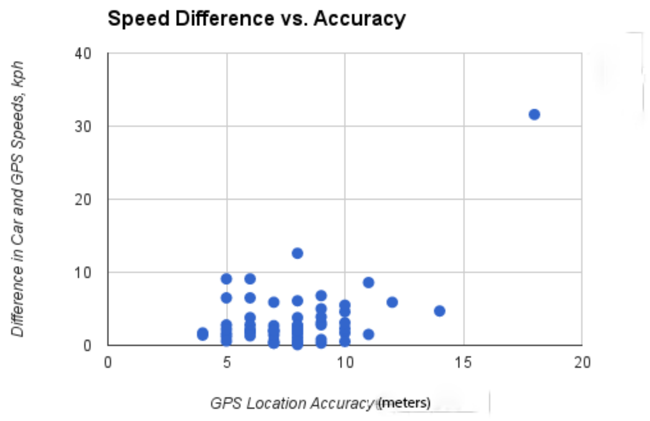

Fig. 5. Speed difference and accuracy.

communicates with an in-vehicle OBD sensor. The device owner need not worry about having to interact with the app or about being interrupted while using the smartphone. Note that all code runs in a secure sandbox in our app [26], which limits the amount of storage, network, memory, battery, and CPU resources used by that code [30]. To protect the end user's device from malicious attackers, the sandboxed code is securely isolated from other programs on the same device. Any bugs in the code will be contained in the sandbox, and will not affect the rest of the user device [30].

At the remote server, the vehicular sensor data collected from multiple end user devices is stored in a non-relational database. The collected data is stored in JSON format. The collected data set can be visualized on Google Maps to identify fuel efficient routes, routes with higher traffic activity. Furthermore, the data set could also be used to detect reckless or illegal driving behaviors, in which case trust metrics would be significant.

B. Geolocation data analysis

Using the aforementioned measurement, we collected speed data from both GPS on the smartphone inside a vehicle, and the speed via the OBD sensor. The data from these two sources can corroborate to verify normal or abnormal geolocation data. Table II shows the median and standard deviation of the speed measured by GPS and OBD sensor, over 58 data samples collected. The overall statistics do not show any anomaly in speed. However, when plotting individual data samples, we can see a data outlier in Figure 5. Excluding this point, the rest of the data samples roughly follow a Gaussian distribution. This outlier seems to be due to the initialization of the GPS sensor. Initially, the GPS parameters are under-constrained, leading to a large geolocation error that is then rapidly reduced as more measurements are made.

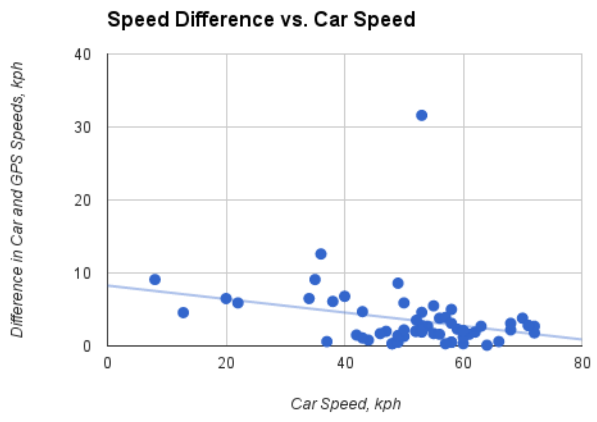

Fig. 6. The difference between the OBD sensor speed and GPS speed decreases slightly as vehicle speed increases.

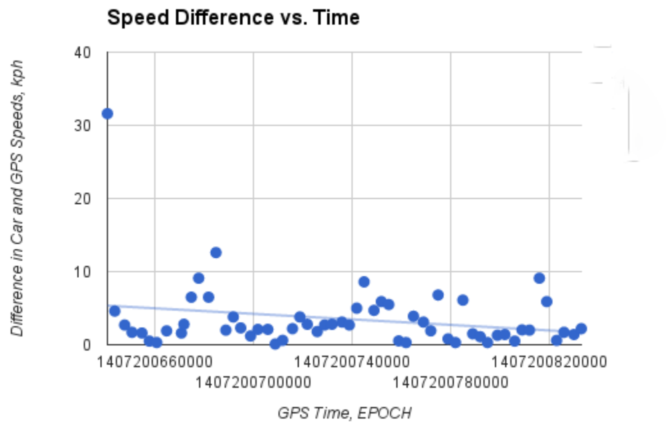

Fig. 7. After the initialization of the GPS sensor, the difference between the OBD sensor speed and GPS speed remains between 0 and 12 kph.

To further investigate the speed data, we plot the differences between vehicle speed and GPS speed, and compare the differences against the varying vehicle speed. As shown in Figure 6, the speed differences vary linearly with the vehicle speed (other than the outlier), i.e., the higher the vehicle speed, the smaller difference between vehicle speed and GPS speed. A minimum mean square error (MMSE) linear fit is shown in the figure.

We also investigate the vehicle and GPS speed differences over time, as shown in Figure 7. From the figure, we can conclude that excluding the outlier, the differences between vehicle speed and GPS speed fluctuate around a constant value over time.

VI. Conclusion

With the growing number of mobile devices owned by ordinary citizens all around the world and an increase in network connectivity and data provided, the problem of evaluating trust becomes more complex. The risks to the quality and security of data and devices become greater and more important to investigate and evaluate. The umbrella trust evaluation framework that we have developed includes procedures and tools to evaluate trust in mobile devices and applications, in particular in Android-based smartphones. The hierarchical structure of the framework allows for incorporating multiple diverse factors affecting trust evaluation as well as facilitating its extension. The current framework version includes evaluation procedures based on the analysis of the following factors: the applications installed and executed, the devices operating system settings and configuration, the level of privacy supported by the devices and applications, the patterns of the battery drain and CPU and network bandwidth usage, and the corroboration of sensor data by external sensors.

This empirical study demonstrates a significant difference between the change of voltage in the cases of an execution of normal applications and the same applications with embedded advertisements. Additional observations produced distinctive patterns of the CPU and network use. The framework includes not only the trust metrics but also the rules for their combination into a trust evaluation and the methods of its verification. Trust verification and adjustment could be performed through comparison of data received from different data sources on the same device as well as by comparing the data originated from various devices. A significant discrepancy between data originated from various sources should result in decreasing trust assessment for the corresponding data sources and devices. In the paper, we have described a comparative study of the speed measurements obtained from a vehicle on-board diagnostics (OBD) sensor and the speed calculated with the GPS location sensor measurements. This illustrates how the framework could be employed not only for trust evaluation but also for detecting various anomalies, which might reflect malicious attacks against the mobile devices.

References

[1] "Smartphone OS Market Share, Q4 2014." http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Accessed: June 29, 2015.

[2] "Google: We Have 1 Billion Monthly Active Android Users." http://www.businessinsider.com/google-we-have-1-billion-monthly-active-android-users-2014-6. Accessed: June 29, 2015.

[3] M. Zheng, M. Sun, and J. Lui, "Droidray: a security evaluation system for customized android firmwares," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp. 471–482, ACM, 2014.

[4] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu, "Riskmon: Continuous and automated risk assessment of mobile applications," in *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, pp. 99–110, ACM, 2014.

[5] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer, "Google android: A comprehensive security assessment," *Security & Privacy, IEEE*, vol. 8, no. 2, pp. 35–44, 2010.

[6] A. Pathak, Y. C. Hu, and M. Zhang, "Fine grained energy accounting on smartphones with eprof," *EuroSys' 12*, 2012.

[7] "System call." https://en.wikipedia.org/wiki/System_call. Accessed: June 29, 2015.

[8] X. Che, X. Ju, and H. Zhang, "The case for addressing the limiting impact of interference on wireless scheduling," in *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, pp. 196–205, IEEE, 2011.

[9] "Application Resource Optimizer (ARO)." https://github.com/attdevsupport/ARO. Accessed: June 29, 2015.

[10] F. Ricciato, A. Coluccia, and D. A., "A review of dos attack models for 3g cellular networks from a system-design perspective," *Computer Communications*, vol. 33, no. 5, pp. 551 – 558, 2010.

[11] J. Cappos, L. Wang, R. Weiss, Y. Yang, and Y. Zhuang, "Blursense: Dynamic fine-grained access control for smartphone privacy," in *Sensors Applications Symposium (SAS), 2014 IEEE*, pp. 329–332, IEEE, 2014.

[12] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pp. 113–124, ACM, 2012.

[13] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: your finger taps have fingerprints," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 323–336, ACM, 2012.

[14] N. Xu, F. Zhang, Y. Luo, W. Jia, D. Xuan, and J. Teng, "Stealthy video capturer: a new video-based spyware in 3g smartphones," in *Proceedings of the second ACM conference on Wireless network security*, pp. 69–78, ACM, 2009.

[15] L. Cai and H. Chen, "Touchlogger: inferring keystrokes on touch screen from smartphone motion," in *Proceedings of the 6th USENIX conference on Hot topics in security*, pp. 9–9, USENIX Association, 2011.

[16] R. Schlegel, K. Zhang, X.-y. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones.," in *NDSS*, vol. 11, pp. 17–33, 2011.

[17] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp) iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 551–562, ACM, 2011.

[18] "More Than 25% of Android Apps Know Too Much About You." http://yro.slashdot.org/story/12/11/02/1316238/more-than-25-of-android-apps-know-too-much-about-you. Accessed: June 29, 2015.

[19] "App Auto-Tweets False Piracy Accusations." http://yro.slashdot.org/story/12/11/13/2249203/app-auto-tweets-false-piracy-accusations. Accessed: June 29, 2015.

[20] "China mobile users warned about large botnet threat." http://www.bbc.co.uk/news/technology-21026667. Accessed: June 29, 2015.

[21] "US Wants Apple, Google, and Microsoft To Get a Grip On Mobile Privacy." http://yro.slashdot.org/story/13/02/02/2124204/us-wants-apple-google-and-microsoft-to-get-a-grip-on-mobile-privacy. Accessed: June 29, 2015.

[22] M. Conti, V. Nguyen, and B. Crispo, "Crepe: Context-related policy enforcement for android," *Information Security*, pp. 331–345, 2011.

[23] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 639–652, ACM, 2011.

[24] X. Ju, H. Zhang, and D. Sakamuri, "Neteye: a user-centered wireless sensor network testbed for high-fidelity, robust experimentation," *International Journal of Communication Systems*, vol. 25, no. 9, pp. 1213–1229, 2012.

[25] "Sensibility Testbed." https://sensibilitytestbed.com/. Accessed: June 29, 2015.

[26] "Sensibility Testbed, Google Play Store." https://play.google.com/store/apps/details?id=com.sensibility_testbed. Accessed: June 29, 2015.

[27] "On-board diagnostics." http://en.wikipedia.org/wiki/On-board_diagnostics. Accessed: June 29, 2015.

[28] M. Reininger, S. Miller, Y. Zhuang, and J. Cappos, "A first look at vehicle data collection via smartphone sensors," in *Sensors Applications Symposium (SAS), 2015 IEEE*, IEEE, 2015.

[29] "Does My Car Have OBD-II?." http://www.obdii.com/connector.html. Accessed: June 29, 2015.

[30] J. Cappos, A. Dadgar, J. Rasley, J. Samuel, I. Beschastnikh, C. Barsan, A. Krishnamurthy, and T. Anderson, "Retaining sandbox containment despite bugs in privileged memory-safe code," in *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, (New York, NY, USA), pp. 212–223, ACM, 2010.