EAI Endorsed Transactions
on Future Internet

Research Article EAI.EU

Reassessing Android Malware Analysis: From Apps to IoT System Modelling

Abraham Rodríguez-Mota[1,*], Ponciano Jorge Escamilla-Ambrosio[2], Jassim Happa[3], Eleazar Aguirre-Anaya[2]

[1]Instituto Politécnico Nacional, Escuela Superior de Ingeniería Mecánica y Eléctrica, Unidad Zacatenco, Av. IPN S/N C.P. 07738, Mexico City, Mexico
[2]Instituto Politécnico Nacional, Centro de Investigación en Computación, Mexico
[3]University of Oxford, Department of Computer Science, UK

Abstract

Applications based on the Internet of Things (IoT) are increasingly vulnerable to disruption from cyber attacks. Developers and researchers attempt to prevent the growth of such disruption models, mitigate and limit their impact. This requires the understanding and characterization of things and the technologies that empower the IoT. Futhermore, tools to evaluate, analyze and detect security threats in IoT devices are strongly required. This paper presents a web tool, named GARMDROID, aimed to help IoT software developers and integrators to evaluate IoT security threats based on the visualization of Android application hardware requests. This procedure is based on the static analysis of permissions requested by Android applications. A mapping from the malware analysis data obtained to a SysML block definition diagram is also briefly described. This mapping shows how data can be used to model IoT systems under different proposals such as Model Integrated Mechatronics (MIM) and UML4IoT.

Received on 23 November 2016; accepted on 21 September 2017; published on 15 January 2018

Keywords: Internet of Things, Android, Security Threats.

doi:10.4108/eai.12-1-2018.153565

1. Introduction

The Internet of Things (IoT) promises to extend "anywhere, anyhow, anytime" computing to "anything, anyone any service". Therefore, the IoT envisions a future where each person and thing has a locatable, addressable, and readable counterpart on the Internet. Such highly-distributed nature and use of fragile technologies, such as limited-function embedded devices in public areas, create weak links that malicious entities can exploit [1]. Consequently, a number of different factors may arise and lead to different types of security exposures, among them consistently defects, bugs and logical flaws are causes of commonly exploited software vulnerabilities [2]. Therefore, the challenge is to prevent the growth of such models or at least to mitigate and limit their impact.

Traditional IoT protection mechanisms, such as lightweight cryptography, secure protocols, and privacy assurance are not enough. In this sense, research must be oriented to analyze current security protocols and mechanisms, and decide whether such approaches are worth integrating into the IoT as is or if adaptation or entirely new designs will better accomplish security goals. Since attacks could involve various layers of the device infrastructure, they could include applications running on smartphones or tables, cloud services (firmware included), and network service stacks on WiFi modules (as well as the firmware and application layer on the host processor) [1].

In IoT mobile applications, new vulnerabilities continue to emerge as IoT becomes a more attractive target. In terms of the nature of mobile devices, their vulnerability surface share attributes with traditional client/server and Web applications. However the type of information that is trusted on mobile devices creates some unique attack vectors as well. For

*Corresponding author. Email: armesimez@gmail.ipn.mx

example, privacy violation weaknesses occurring on mobile devices can lead to the disclosure of location, sensitive images, and data entered from the keyboard or displayed on the screen and other personal information [2].

Taking into account that in recent years Android OS has become one of the principal sharers in the global mobile devices market [3], our research has focused on the analysis and detection of security threats in Android applications. This paper presents a subset of functionalities of an Android malware hybrid analysis and detection software system, called GARMDROID, currently under development. Although GARMDROID has a bigger aim, oriented to integrate static and dynamic malware analysis, since static analysis is usually the first approach to malware analysis, we focus this discussion on the capabilities of GARMDROID to provide quick feedback to developers producing a visualization of app's permissions and features requirements. As discussed later on, this visualization results very handy in the identification of potential threats or bad designed software. This system has been named GARMDROID as a result of the fusion of the words GARM and Android (in Norse mythology, Garm is a dog described as a blood stained watchdog that guards Hel's gate [4]).

2. Android Overview

An Android device can have a wide variety of sensors. Android's sensing capabilities are derived from the available hardware on Android devices and from creative use of it. A capability may use values directly from hardware that can measure physical quantities or it may use hardware that the user typically interacts with, such as the camera and microphone. A capability may even use a combination of hardware and server-based processing, such as speech recognition. Whatever the source, the resulting data can inform an application (app) about the device's state and the environment in which it resides [5].

In any app, acquiring sensor data requires similar code. Each kind of data requires different boilerplate. In many cases, is not trivial to initialize the API and acquire the data. Once an app can initialize and acquire sensor data, it needs to utilize the APIs to collect the data while the app is running. Data can be collected in different ways depending on how an app uses it. For example, location tracking is a common use of location sensors, in this case some apps need to persistently track location while an app performs other tasks. In the case of speech recognition, such app needs to have other components besides actually running the speech recognizer. An app also needs to allow the user to activate speech and mediate turn taking between when the user can speak and when the app is listening [5].

In this sense, a <uses-feature> element contained in an *AndroidManifest.xml* file, declares a single software feature that is used by an application. The purpose of declaring these elements is to inform any external entity of the set of hardware and software features on which an application depends. The element offers a required attribute that lets developers specify whether the application requires and cannot function without the declared feature, or whether it prefers to have the feature but can function without it. Because feature support can vary across Android devices, the declaration of these elements serves an important role in letting an application describe the device-variable features that it uses [6].

Declaring features is for informational purposes only. The Android system itself does not check for matching features support on the device before installing an application. However, other services (such as Google Play) or applications may check the declarations in the application as part of handling or interacting with the application. When a user searches or browses for applications using the Google Play application, the service compares the features needed by each application with the features available on the user's device. If all of an application's required features are present on the device, Google Play allows the user to see the application and potentially download it. If any required feature is not supported by the device, Google Play filters the application so that it is not visible to the user and not available for download [6].

An explicitly declared feature is one that an applications declares in a <*uses-feature*> element. The feature declaration can include an *android:required=["true" | "false"]* attribute (if the code is being compiled against function API level 5 or higher), which lets the developer specify whether the application absolutely requires the feature and cannot function properly without it, or whether the application prefers to use the feature if available, but it is designed to run without it. In general, if an application is designed to run on Android 1.6 and earlier versions, the *android:required* attribute is not available in the API and Google Play assumes that any and all feature declarations are required [6].

An implicit feature is one that an application requires in order to function properly, but which is not declared in the manifest file. Strictly speaking, every application should always declare all features that it uses or requires, so the absence of a declaration for a feature used by an application should be considered an error. However, as a safeguard for users and developers, Google Play looks for implicit features in each application and sets up filters for those features, just as it would do for an explicitly declared feature. Google Play attempts to discover an application's implied feature requirements by examining other

elements declared in the manifest file, specifically, *<uses-permission>* elements[6].

If an application requests hardware-related permissions, Google Play assumes that the application uses the underlying hardware features and therefore requires those features, even though there might be no corresponding features declarations. For such permissions, Google Play adds the underlying hardware features to the metadata that it stores for the application and sets up filters for them [6].

3. Android Threats

The way people experience and interact with devices is changing. More and more gadgets and devices are being added to the IoT ecosystem everyday. The interconnection between these gadgets and devices has the potential to create remarkable, new user experiences [7]. However, novel technology can lead to exposures, as the implications of new technologies can sometimes be difficult to guess and avenues of attack can be unexpected until observed in practice [2].

Mobile application vulnerabilities continue to evolve as Android devices become attractive targets. Mobile devices contain sensors and actuators of types not historically common in personal computers or servers, which collect and transmit private information about the user of the device. The list of sensors that can reveal sensitive information include cameras, microphones, accelerometers, gravity sensors, rotational vector sensors, gyroscopes, magnetometer, Global Positioning System (GPS) sensors, Near-Field Communication (NFC), light sensors, M7 tracking chips, barometers, thermometers, pedometers, heart-rate monitors, and fingerprint sensors [2].

Privacy-violation weaknesses occurring on mobile devices can lead to the disclosure of location, sensitive images, data entered from the keyboard or displayed on the screen and other personal information. While smartphones can be used for viewing, manipulating, and storing local data, these devices also allow users to interact with a world of interconnected resources from the convenience of their hands. Through communication protocols, both sensitive and benign data is shared between remote services in different devices [2]. In the context of Android, privacy violation weaknesses can be related to a set of security risks, Figure 1 presents 10 of the biggest Android security risks.

Additionally, it must also be considered that insecure deployment combines various configurations, settings, and states that result in unnecessary weaknesses. For mobile applications this may include not using technologies of content protection such as PlayReady DRM, not checking to determine if the application

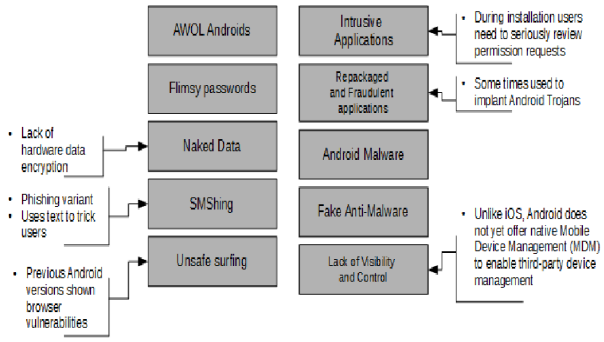

Figure 1. Android Security Risks, based on [8].

is running on a compromised device, or exhibiting properties that may indicate malicious intent [2].

3.1. Android Malware Analysis

Malware analysis is a process in which the malware is taken apart for studying its code structure, operation and functionality. It is conducted with specific objectives which include: to understand the vulnerability that was exploited, to study the severity of the attack and counteracting measures, to penetrate into the compromised data in order to investigate its origin and to obtain information about other compromised machines [9]. Detection techniques for Android malware use statically extracted data from the manifest file or from Android API function calls, as well as dynamically obtained information from network traffic and system call tracing [10]. Most of current systems used to detect malicious code are largely based on syntactic signatures and employ static analysis techniques. Static analysis techniques can be evaded by malware applications using techniques such as polymorphism and metamorphism, since syntactic signatures are ignorant of semantics of instructions [11].

4. GARMDROID

GARMDROID is based on the capabilities provided by the Android SDK tool set, specifically the Android Asset Packaging Tool (AAPT) which is contained as part of the *platform tools* set. In this implementation clients can upload malware samples and request analysis via a Web interface. Figure 2a presents a general representation of the Web system.

During analysis, once an android application file (.apk) has been uploaded by a user, GARMDROID uses a set of bash and python scripts to command AAPT to extract the contents of the app's AndroidManifest.xml file and to filter out the important strings. In this case, as shown in Figure 2b, the system's software stack includes Java at the bottom layer as it is required

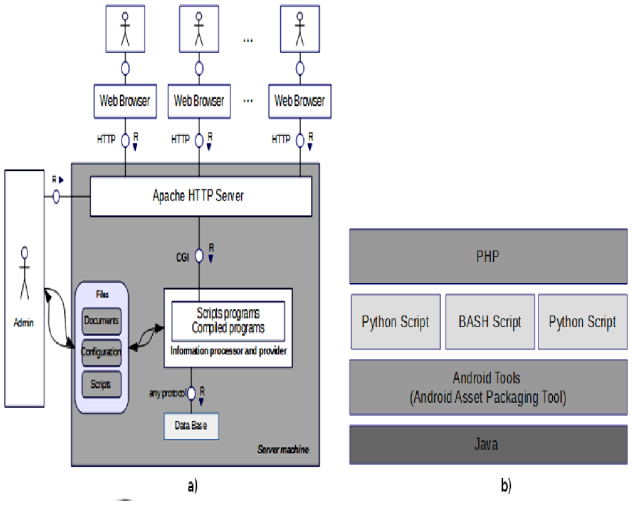

Figure 2. System representation, a) Web system and b) software stack.

to run the AAPT. Python and Bash programming is on top of the AAPT layer since a set of python and bash scripts are used to filter out permissions and feature-request strings from the AAPT output. Further processing, based on the characteristics of implicit features and explicit features declarations provided by Android, helps GARMDROID to deduce the set of requested hardware features related to the app's specific set of permissions requests. This association between permissions and requests with hardware features is performed also by a python script. Finally, PHP scripts are employed to obtain the web visual representation of the data via HTML and SVG elements. GARMDROID is available at www.garmdroid.org.

Figure 3 shows the main page of the system from where users can upload files and see the results after file processing. Once the application file is processed the tool displays the name, mime type, size and md5 hash value of the file. Additionally, permissions and features are identified and displayed. In the case of permissions, Figure 4a, it has been selected to visualize the requested permissions as a matrix of dots where permissions requested by the application under analysis are indicated as red dots. Features have been represented as icons in order to facilitate visualization: Audio, Bluetooth, Camera, Infrared, Location, Microphone, NFC, Sensors (Accelerometer, Barometer, Compass, Gyroscope, Light, Proximity, Step Counter, Step Detector), Screen, Telephony, Television, Touchscreen, USB and WiFi, see Figure 4b.

5. Results

In this section a set of results obtained after processing a group of Android applications using GARMDROID is presented. Our results take form of five different

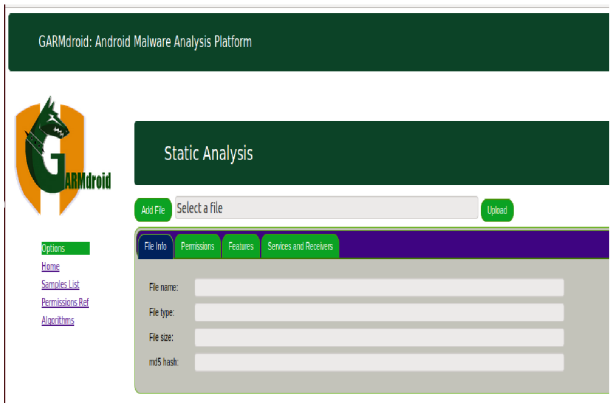

Figure 3. GARMDROID welcome page.

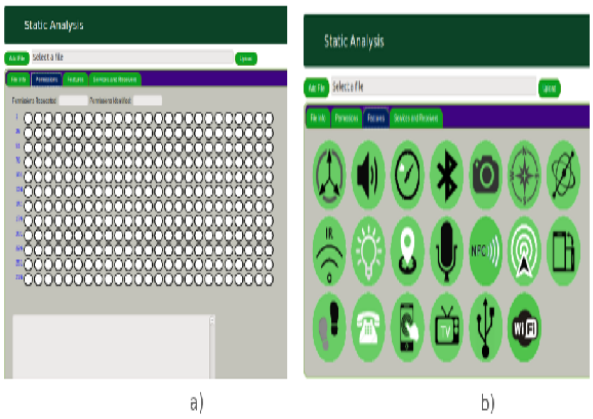

Figure 4. a) Permissions tab, additionally to the dot matrix representation, in which hovering over a circle provides the full permission name, a textbox element at the bottom of the tab also displays the identified permissions; b)Features tab, representing hardware features as icons which change its background color to red if they are requested by the file under analysis.

case scenarios (apps). In each case GARMDROID presents an inference of the set of hardware features requested by the app under analysis, plus the set of permissions requests. These cases serve a two-fold purpose: to demonstrate GARMDROID operation and direct the discussion towards observations which can lead to identify security threats in IoT-oriented Android applications. In brief, the five cases presented and conclusions drawn can be summarize as follow:

1. Hardware-Test app: granting high volume of permissions and access to hardware features may increase security risks.

2. Lighting app: inconsistency between app's functionality and hardware features requests must raise security concerns.

3. IR remote control apps: excessive hardware feature requests may imply security risks.

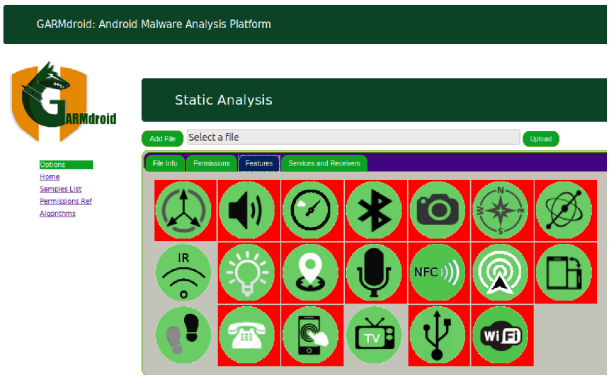

Figure 5. Features requests for a selected Hardware-Test application.

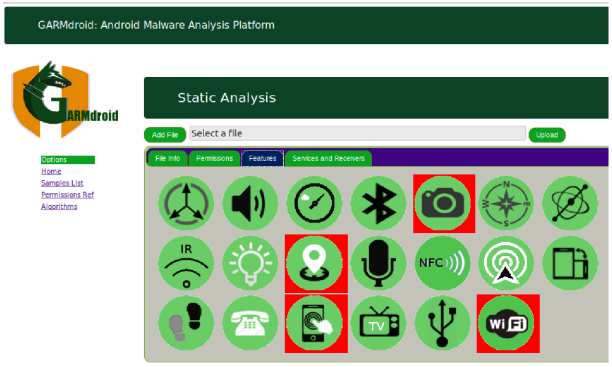

Figure 6. Features requested by a lighting app.

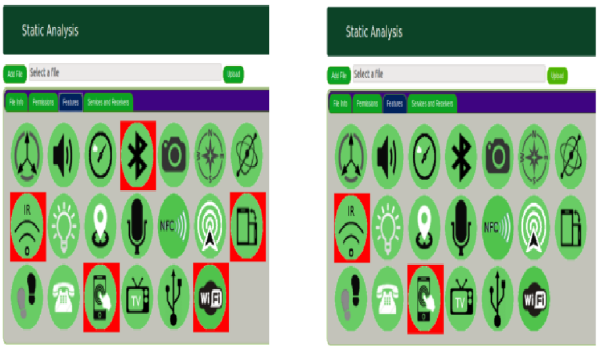

Figure 7. Comparison between features-requests by two different Remote-Control Infrared apps.

4. Gyroscope app: little or no hardware features requests may signify a security problem.

5. Hardware-Test app: problems inferring hardware features requests may imply security risk or app design problems.

In this description it has to be assumed that applications have been analyzed using VirusTotal [12], and in all cases where identified as benign, unless otherwise stated. Moreover, detailed information such as application name and hash values have been omitted on purpose to avoid misleading users from using such applications, since the provided results are only demonstrative and further analysis might be required to properly identify some of the applications as malware or bad software design samples.

Firstly, a Hardware-Test application was analyzed, see Figure 5. As it can be observed the analysis shows that this application requests access to Accelerometer, Audio, Barometer, Bluetooth, Camera, Compass, Gyroscope, Light, Location, Microphone, NFC, Proximity, Screen, Telephony, Touchscreen, USB and WiFi. In this case, results mainly demonstrate GARMDROID's capability to infer requested hardware features, but it is also interesting to observe that even though it is not identified as being malicious, it is easy to visualize that there is a high risk in allowing this kind of access to any application, due to the big number of hardware elements that are requested.

Secondly, Figure 6 shows the features requested by an allegedly lighting app. The results may raise suspicion since the application requests not only access to the camera (assuming that the lighting functionality is provided by using the camera flash functionality) but to Location and WiFi features as well.

Thirdly, a couple of Infrared Remote Control apps were analyzed, see Figure 7. In this case we observed that there is a big difference between the set of features requested which may be a reason to promote a further analysis over the application requesting more than the IR feature (Bluetooth and WiFi).

The following case, see Figure 17, presents an application advertised as capable to provide gyroscope data. Interestingly, none permission was requested and only the touch screen request is made. At this stage there was no evidence to determine whether these characteristics are related to a security threat or a poor design, but provides a strong reason to think that further analysis is required.

As our final case, a Hardware-Test app is presented which requested features but not following the Android specification (Upper case text was used where the specification indicates lower case). This case was detected as a result of a further analysis of the app after observing that no features were indicated on the GAMDROID features tab. Although more information would be required to determine whether the application represents a threat or not, there is an indication of a bad software design. Figure 9 illustrates these results.

Finally, after analyzing four IoT oriented apps samples (home automation type) results were compared with those obtained from analyzing 369 FakeInstaller Android malware samples, see Figures 10. In this

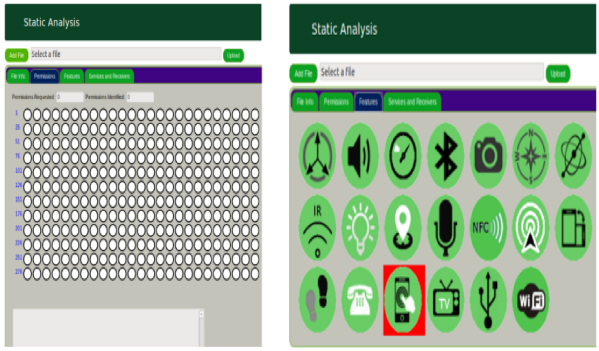

Figure 8. Gyroscope application which does not request any permission but request the touch screen feature only.

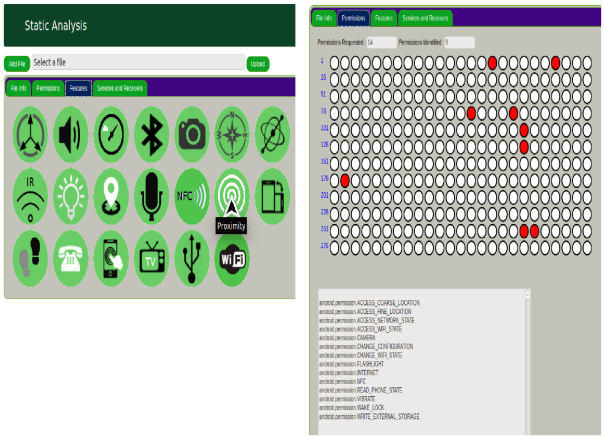

Figure 9. Hardware-Test app with anomalous feature-request declarations.

respect, although the selected IoT samples set is small, after comparing the results it can be observed that a request for telephony hardware is not a common feature for home automation apps. From the point of view of developers it can be assumed as a good indication that further analysis is required.

5.1. Physical IoT Objects Modelling from Malware Static Analysis Data

GARMDROID provides a mechanism to identify general characteristics of Android powered devices. On a first approach, it has been set an operational scheme where objects participating on an IoT system implementation can be characterized by reviewing static analysis data extracted using GARMDROID from a set of Android applications. This scheme is broadly illustrated in Figure 11.

Observing the proposed scheme, see Figure 11, there can be identified 6 major processes, numbered from 1 to 6 in the figure. At first, the analysis starts from an IoT Solution system implemented as a set of different

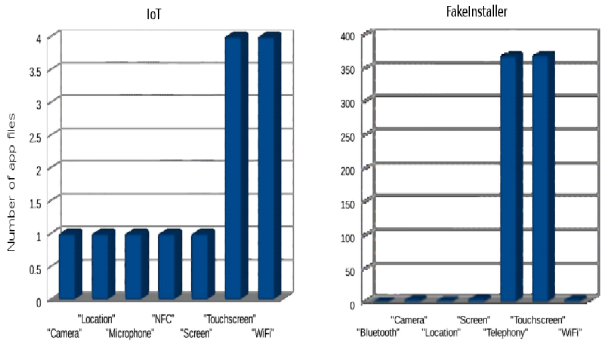

Figure 10. Features requested by IoT Android Samples (home automation type) and applications identified as FakeInstaller malware.

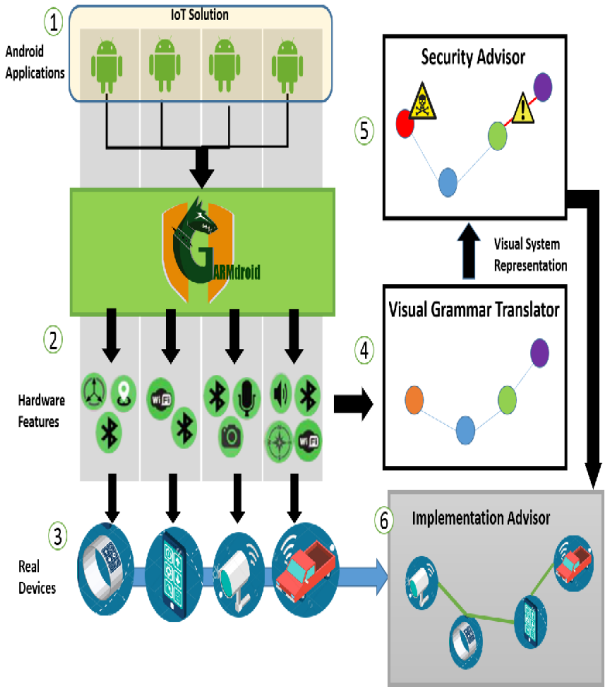

Figure 11. Conceptual representation of the system general architecture.

Android applications. These applications are entered to GARMDROID which infers the hardware features requested by the application from the static information contained in the application's AndroidManifest file. These features, together with information about the Android OS and Java version numbers, are used to create objects profiles that match the applications requirements, this step is labeled as number two in the figure. The objects profiles are used to identify real devices already known by the system (stored on a data base), this is step 3. During step number 4, the objects profile identified are handed in to the Visual Grammar Translator which task is to construct a model from

the given objects models. Once the model is obtained there is a further step, labeled as number 5, where the model is analyzed from the security point of view base on the hardware and services requirements previously identified. Finally, the output from the Security Advisor block is joined with the real devices profiles identified during step 3 and are feed into the Implementation Advisor which aims to provide a system specification considering the best security practices.

The heterogeneous nature of the IoT current development stage represents a big challenge for modelling and designing initiatives as there is little consensus between technology providers. In this sense, since the achievement of the proposed representation process described above, see Figure 11, requires a representation of the devices of the system, as a first approach, it has been decided to represent the devices, according to the features requested, using the System Modeling Language (SysML) Block Definition Diagram (BDD) notation [14]. SysML BDD has been selected taking into account the growing adoption of SysML for modelling mechatronic systems and its relationship with other IoT modelling initiatives, such as the Model Integrated Mechatronic (MIM) [15], 3+1 SysML View-Model [16] and UML4IoT [17] approaches.

SysML can be adopted for the system modelling process related to the Model Integrated Mechatronics (MIM) [16]. MIM is a new paradigm that applies domain-specific modeling languages for the concurrent engineering of mechanical, electronic and software components of mechatronic systems [19] [15]. This paradigm was proposed to address the need for an integrated development in mechatronic systems. MIM supports the model-driven development of complex mechatronic systems (MTSs) through the evolution of models that have as primary construct the mechatronic component (MTC). The MIMs architecture is shown in Figure 12.

The most common kind of SysML diagram is the Block Definition Diagram (BDD). It is possible to display various kinds of model elements and relationships on a BDD to express information about a system's structure. Blocks can represent any level of the system hierarchy including the top-level system, a subsystem, or logical or physical component of a system or environment, as well as software entities. Blocks are modular units of system descriptions. Each block defines a collection of features to describe a system or other element of interest. These may include both structural and behavioral features, such as properties and operations, to represent the state of the system and behavior that the system may exhibit [14].

Blocks provide a general-purpose capability to model systems as trees of modular components. SysML blocks can be used throughout all phases of system specifications and design, and can be applied to many

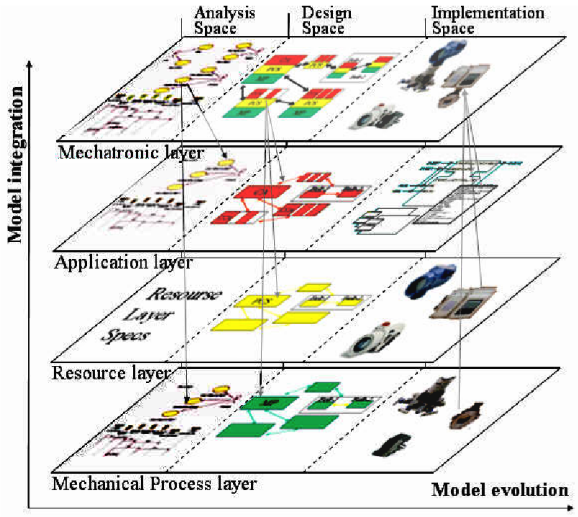

Figure 12. Model Integrated Mechatronics (MIM) Architecture [15].

different kinds of systems. SysML blocks are based on UML [13] classes as extended by UML composite structures. Some capabilities available for UML classes, such as more specialized forms of associations, have been excluded from SysML blocks to simplify the language. SysML Blocks also extend the capabilities of UML classes and connectors with reusable forms of constraints, multi-level nesting of connector ends, participant properties for composite association classes, and connector properties [14].

As a prove of concept, a set of BDD diagrams have been generated based on the results, described in the previous section, provided by GARMDROID. See Figures 13 to 17.

6. Conclusions

Despite the fact that openness has been an important factor in Android fast positioning into the mobile market, it is clear that it implies certain security challenges. In the case of the Internet of Things (IoT) the growing adoption of devices and solutions that incorporate Android has brought those challenges into the realm of the IoT. Therefore, in order to guarantee high security levels IoT developers need to get more involved in the analysis and detection of security threats.

Since IoT development requires a vast and detailed knowledge of diverse technological aspects it is always difficult to count with personnel experienced in those many areas. Consequently, the use and development of new tools and analysis techniques that facilitate or simplify in some extent security analysis are becoming important research and development areas. This paper presented a proof-of-concept that demonstrates visual

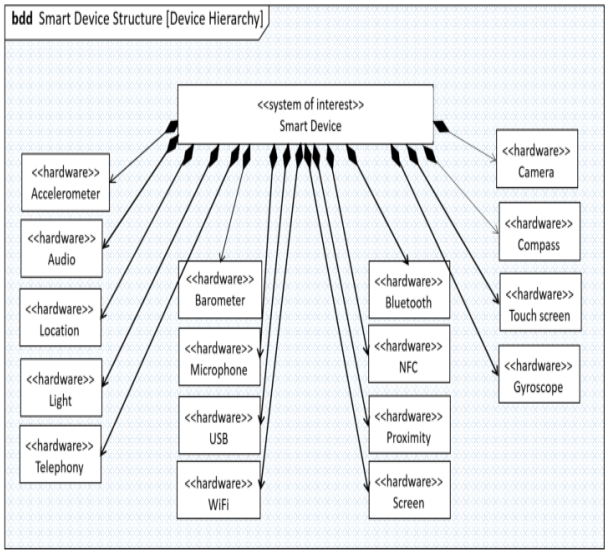

Figure 13. SysML Block Definition Diagram for the Hardware-Test app.

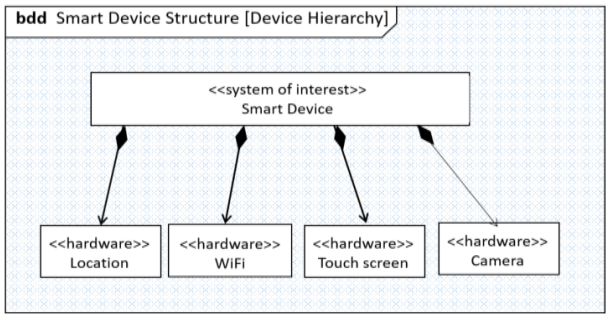

Figure 14. SysML Block Definition Diagram for the lighting sample app.

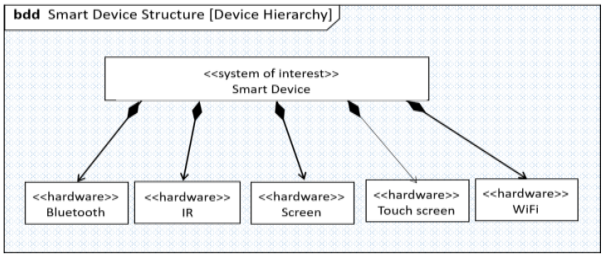

Figure 15. SysML Block Definition Diagram for an IR remote control sample app.

representations of some application's static features that could help developers to direct security analysis.

Although only a part of the system under development is described in this paper, it is considered that the features provided currently represent a useful asset for software development in the IoT area when compared with other options currently in the marked. As an example, the identification of "suscpicious" hardware

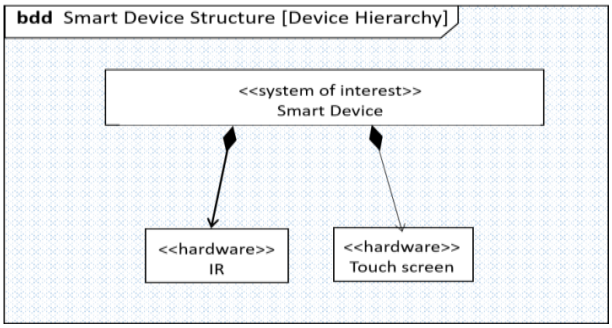

Figure 16. SysML Block Definition Diagram for a second IR remote control sample app.

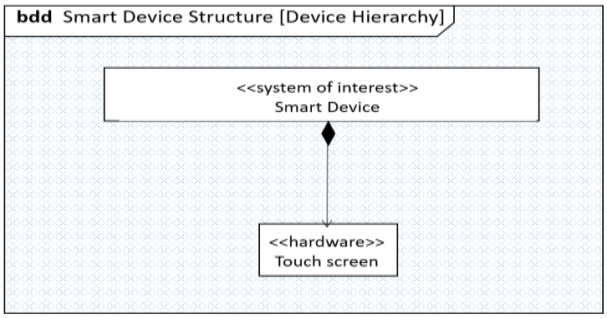

Figure 17. SysML Block Definition Diagram for the gyroscope sample app.

features requests discussed in this paper only required from a user a quick review of the visual information, whether a similar analysis using raw analysis data, e.g. from VirusTotal, would require more effort reading all permissions identified and selecting those that could let to infer the hardware features. It must be considered that this task can be performed easily for few samples but it becomes error prone as the number of permissions per app and apps under analysis increases.

In terms of the results presented in this work, it can be concluded that visualization of features requested by an Android app may provide a simple and quick overview of the app's real intentions. This, combined with the knowledge of the permissions requested by the application, provides a good reference for developers that are faced with the decision of whether or not to reuse code, install a new application, grant permissions, define features requests, among other tasks. Additionally, a higher abstract level representation, in this case SysML Block Diagrams, based on the hardware features requested by software applications appears to be a viable way towards producing expressive and reliable systems representations which can be further extended to include other aspects such as security and management details. Further analysis and development is planed in this research in order to integrate these results with others from more elaborated techniques, such as

machine learning, in order to provide a more detailed and holistic analysis. Some work in this direction is in progress at our research institution.

6.1. Acknowledgements

This material is based on work supported by the Mexican National Council of Science and Technology (CONACYT) under grant 216747. Also the authors acknowledge support from IPN under grant SIP-20161697.

References

[1] Roman, R.,Najera, P., Lopez, J. (2011) *Securing the Internet of Things* (IEEE Computer), vol. 44, no. 9, 51–58.

[2] Childs, D.,Gilliland, A., Gorenc, B., Goudey, H., Gunn, A.,Hoole, A.,Lancaster, J., Muthurajan, S., Wook Oh, J., Tsipenyuk O'Neil, Y., Park,J., Petrovsky, O., Sechman, J., Shah, N., Sotack, T., Svajcer, V. (2015) *The HPE Cyber Risk Report 2015* (HP).

[3] Gartner (2013) *Gartner Says Worldwide Smartphone Sales Recorded Slowest Growth Rate Since 2013*,http://www.gartner.com/newsroom/id/3115517, 6 1 2016.

[4] Wikipedia *Garmr*. [Online]. Available:https://en.wikipedia.org/wiki/Garmr, 15 11 2015.

[5] Milette, G., Stroud, A. (2012) *Professional, Android Sensor Programming*. (John Wiley & Sons, Inc.)

[6] Android developers *uses-features*. [Online]. Available: http://developer.android.com/intl/es/guide/topics/manifest/uses-feature-element.html, 10 12 2015.

[7] Embarcadero *Internet of Things Solutions*. [Online]. Available: https://www.embarcadero.com/solutions/internet-of-things, 2 1 2016.

[8] Phifer, L. *Top 10 Android Security Risks*. [Online]. Available: http://www.esecurityplanet.com/views/article.php/3928646/Top-10-Android-Security-Risks.htm, 14 05 2015.

[9] Kendall, K. *Practical Malware Analysis*. [Online]. Available: https://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Paper/bh-dc-07-Kendall_McMillan-WP.pdf, 07 05 2015.

[10] Afonso, V., de Amorim, M., GrÃĭgio, A. R. A., Junquera, G. & de Geus, P (2015) "Identifying Android malware using dynamically obtained featuresÂťÂť (Springer-Verlag) *Journal of Computer Virology and Hacking Techniques 2015*, vol. 11, pp.9–17.

[11] Moser, A., Kruegel, C. & Kirda, E. (2007) "Limits of Static Analysis for Malware DetectionÂťÂť *Computer Security Applications Conference 2007*,ACSAC 2007, pp. 421–430.

[12] VirusTotal. [Online]. Available: https://www.virustotal.com/es-mx/, 05 12 2015.

[13] OMG (2015) OMG Unified Modeling Language (OMG UML). Version 2.5. March 2015. [Online]. Available:http://www.omg.org/spec/UML/2.5/.

[14] OMG (2015) OMG System Modeling Language (OMG SysML). Version 1.4. September 2015. [Online]. Available: http://www.omg.org/spec/SysML/1.4/PDF/.

[15] Thramboulidis, K. (2004) *Model Integrated Mechatronics: An Architecture for the Model Driven Development of Mechatronic Systems*, 2nd IEEE International Conference on Mechatronics, pp. 497-502, Istanbul, Turkey.

[16] Thramboulidis, K. (2010) *The 3+1 SysML View-Model in Model Integrated Mechatronics* Journal of Software Engineering and Applications, Vol. 3 No. 2, pp. 109-118.

[17] Thramboulidis, K.,Christoulakis, F. (2016) *UML4IoT-A UML-based approach to exploit IoT in cyber-physical manufacturing systems*. Computers in Industry. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016636151630094X, 10 05 2016.

[18] Developer Android (2015) *Top 10 Android Security Risks*. [Online]. Available: http://www.esecurityplanet.com/views/article.php/3928646/Top-10-Android-Security-Risks.htm, 14 05 2015.

[19] Thramboulidis, K. (2005) "Model Integrated Mechatronics Towards a new Paradigm in the Development of Manufacturing Systems". *IEEE Transactions on Industrial Informatics*, vol. 1, No.1.