Policy-based Security Assessment of Mobile End-user Devices
An Alternative to Mobile Device Management Solutions for Android Smartphones

Thomas Zefferer and Peter Teufl
Institute for Applied Information Processing and Communications, Graz University of Technology,
Inffeldgasse 16a, 8010 Graz, Austria

Keywords: Smartphone, Security, Policy, Assessment, Android.

Abstract: For security-critical applications, the integrity and security of end-user devices is of particular importance. This especially applies to mobile applications that use smartphones to process security-critical data. Unfortunately, users often compromise the security of smartphones by disabling security features for convenience reasons or by unintentionally installing malware from untrusted application sources. Mobile device management (MDM) solutions overcome this problem by providing means to centrally manage and configure smartphones. However, MDM is mainly suitable for corporate environments but often cannot be applied in non-corporate fields of application such as m-banking or m-government. To address this problem, we propose an alternative approach to assure the security and integrity of smartphones. Our approach relies on a device assessor that evaluates the current state of a smartphone according to a security policy. Integration of this device assessor allows smartphone applications to condition the processing of security-critical data on the smartphone's compliance with a defined security policy. We have shown the practicability of the proposed approach by means of a concrete implementation for the Android platform. We have evaluated this implementation on different Android devices. Obtained results show that our approach constitutes an appropriate alternative for scenarios, in which MDM cannot be applied.

1 INTRODUCTION

Smartphones have emancipated from classical end-user devices and are nowadays frequently used to access services from various fields of application. This includes security-critical fields such as mobile banking (m-banking) or mobile government (m-government). In many cases, applications from these fields of application involve the processing of security-critical data on the user's smartphone.

The confidentiality of data being stored and processed on the user's smartphone heavily depends on the security of this device. The security of the device, in turn, mainly depends on the device configuration and on the smartphone's robustness against malware. Both aspects have recently turned out to be especially problematic on smartphones running the mobile operating system Google Android. Android provides users various ways to configure their smartphones. For instance, Android users are free to enable or disable access-protection or encryption mechanisms. Hence, the security of data being stored on smartphones heavily depends on settings chosen by the user.

At the same time, recent reports (Lookout Mobile Security, 2011) and malware indices (Hackmageddon, 2011) indicate that Android seems to be more prone to malware than other smartphone platforms. A recent incident that substantiates this observation has become known under the name Eurograbber (Schwartz, 2012). In 2012, Eurograbber stole $47.000.000 from private and corporate bank accounts by implementing a combined attack on Web browsers and Android smartphones to compromise SMS-based authentication schemes of European e-banking portals. Main reasons for the vulnerability of Android against malware such as Eurograbber are the availability of powerful system APIs for third-party applications, and Android's support for alternative application sources that do not enforce quality checks on distributed software. While these properties facilitate the development and easy distribution of powerful third-party applications, they also ease the spread of malware.

Weak device configurations and malware nowadays represent the main threats for security-critical

Zefferer T. and Teufl P..
Policy-based Security Assessment of Mobile End-user Devices - An Alternative to Mobile Device Management Solutions for Android Smartphones.
DOI: 10.5220/0004509903470354
In *Proceedings of the 10th International Conference on Security and Cryptography* (SECRYPT-2013), pages 347-354
ISBN: 978-989-8565-73-0

mobile applications. To dispel these threats, several vendors such as AirWatch[1] or Zenprise[2] have started to offer mobile device management (MDM) solutions for Android. These solutions allow for a central configuration of smartphones and can be used to enforce security policies on Android devices. Additionally, MDM solutions can be used to restrict the set of installed applications using black- or whitelists. Without doubt, MDM solutions have the potential to improve the overall security of smartphones. Unfortunately, MDM is mainly applicable in corporate environments, in which employers provide their employees with smartphones. In such scenarios, the employer has the right and the organizational feasibility to enforce policies on issued devices and to restrict their functionality. In scenarios such as m-banking or m-government, in which users typically use their private smartphones to access a service, MDM is usually not an option.

Due to the non-applicability of MDM solutions, non-corporate smartphone applications cannot make valid assumptions on the current state, configuration, and security of the mobile end-user device, on which these applications are executed. This is a major problem, as applications from security-critical fields might prefer to refuse execution of security-critical operations on insecure smartphones. As a solution to this problem, we propose a *device assessor* that assesses the current state and security of an Android device at runtime. By integrating the proposed device assessor, security-critical applications are able to condition the execution of security-critical operations on the smartphone's current state and configuration. In this paper, we discuss the general idea of using a device assessor to assure the security of processed data in application scenarios, in which MDM is not an option. We show the practicability of this approach by means of a concrete implementation for the Android platform and present first evaluation results.

2 ANDROID SECURITY

The security of Android has been a frequently discussed topic during the past years. Reports on smartphone malware frequently show that Android devices are more vulnerable to malware and related security threats than mobile devices from other smartphone platforms (Lookout Mobile Security, 2011). The topicality of security vulnerabilities on Android is also emphasized by a growing number of scientific publications dealing with different aspects of security on Android smartphones. Comprehensive analyses of Android's security model have for instance been provided in (Enck et al., 2009) and (Enck et al., 2011). Security mechanisms implemented on Android have been assessed in detail in (Shabtai et al., 2009). In-depth analyses of Android's integrated security systems and their capabilities to protect applications running on Android devices have also been provided in (Bing, 2012) and (Pacatilu, 2011).

From the results of published scientific work on Android security and from own experience gained during several projects dealing with design, implementation, and assessment of security-critical mobile applications, the following reasons for Android's security vulnerabilities can be derived.

- **Support for Alternative Application Sources.** Android users can install smartphone applications from arbitrary sources such as alternative application stores, memory cards, or Web resources. While applications offered through official application stores typically undergo a review process, this is often not the case for alternative application sources.
- **Optional Security Features.** Android provides a set of integrated security features that are suitable to protect data being stored and processed on smartphones. However, these security features are optional and often deactivated by default. In many cases, security features would be available but remain deactivated by the user for convenience reasons.
- **Inefficient Permission System.** During the installation of an application on an Android device, the user has to confirm a set of permissions that are requested by the application to be installed and that define its access rights and capabilities. However, users often do not understand or care for the implications of confirming arbitrary permissions requested by applications (Felt, 2012).
- **Rich API.** Compared to other smartphone platforms, Android offers application developers a much richer API[3] to access system functionality. While this allows for more powerful applications, it also increases the capabilities of malware. On Android, malware has – given the required permissions – access to resources of the smartphone (e.g. SMS functionality, file system, etc.) that would require root access to the operating system on other platforms.
- **Rich Runtime Capabilities.** Android offers a high degree of flexibility and a rich set of mecha-

[1] http://www.airwatch.com/solutions/android

[2] http://www.zenprise.com/solutions/MDM

[3] http://developer.android.com/reference/packages.html

nisms for inter-process and inter-application communication (Chin et al., 2011). Furthermore, Android supports – in contrast to other platforms – the execution of arbitrary background services. Again, these features improve the capabilities of third-party applications, but also allow for more powerful malware.

In summary, it can be stated that various properties of Android make this platform especially vulnerable to different security threats. Possible enhancements of Android's security system have been an emerging topic in scientific research for the past years. For instance, the use of SELinux on Android devices has been discussed in (Shabtai et al., 2010). Tang et al. proposed an extension to Android's security enforcement using a security distance model (Tang et al., 2011). Enhancing and replacing Android's encryption systems has recently also been a common topic of scientific work, which has for instance been discussed in detail in (Shurui et al., 2010), (Chen and Ku, 2009), and (Gasti and Chen, 2010). Although most of the proposed concepts and solutions seem promising at a first glance, few of them have actually made their way to current end-user devices.

From the various reasons for Android's vulnerability against security threats discussed in this section, two basic problems can be derived. First, Android's architecture obviously eases the development of powerful malware. Second, the user often turns out to be the weakest link in the line of defense and unintentionally compromises the security of the end-user device by disabling important security features, installing applications from untrusted sources, or confirming permissions that allow applications to compromise the smartphone's security. Android's basic architecture is usually beyond the sphere of influence and cannot be adapted or customized to prevent malware based attacks. Hence, solutions to improve the security of Android devices typically focus on the prevention of weak device configurations caused by the user.

The approach to prevent users from making bad decisions (e.g. disabling of security features, installation of malware) in order to improve the security of end-user devices is actually not a new idea. Especially in corporate environments, mobile device management (MDM) solutions are increasingly used to centrally control and manage end-user devices. MDM solutions are typically based on a server component (MDM server) that is used to define security policies and to transmit these policies to smartphones. On the smartphone, an MDM client enforces obtained policies by appropriately configuring the mobile device. MDM works well in corporate environments, in which companies supply their employees with smartphones and hence have the legal and organizational power to manage these devices and to restrict their functionality according to the company's security policies. In non-corporate environments, MDM is usually not an option. For instance, a bank has neither the legal nor the organizational power to limit or reconfigure a customer's smartphone before allowing it to access an m-banking application. For the same reasons, MDM can even be problematic in corporate environments, when employees are allowed to use their own private smartphones to access company data. This trend has recently become known under the term bring-your-own-device (BYOD) and is increasingly gaining relevance (Woods, 2013).

For scenarios, in which MDM solution cannot be deployed due to legal and organizational reasons, alternative approaches to assure the security of end-user devices need to be followed. We propose the use of a device assessor that can be easily integrated into arbitrary third-party applications. In contrast to MDM solutions, the device assessor does not reconfigure the user's device. Instead, it assesses if a given security policy is fulfilled by the smartphone, on which an application is about to be executed. Based on the results of this assessment, applications that integrate the device assessor can decide whether or not to execute security-critical operations on the given device. Details of this approach and the chosen architectural design of the proposed device assessor are presented and discussed in the next section.

3 ARCHITECTURE

In this section we present the architecture of the proposed device assessor. Ease of integration, sustainability, and flexibility have been defined as basic design requirements for the proposed device assessor. Although these requirements appear to be rather generic, they are indeed of special relevance for this particular solution. Ease of integration is important as mobile applications are typically subject to fast development processes and short release cycles. Sustainability is another important requirement as it can be expected that the relevance of smartphones will continue to increase in future. Finally, also flexibility is of relevance as different applications potentially have different requirements regarding the security of the user's smartphone.

Considering theses design requirements, the architecture shown in Figure 1 has been chosen for the proposed device assessor. Figure 1 illustrates core components of the device assessor, as well as external components that interact with the device assessor through well-defined interfaces.

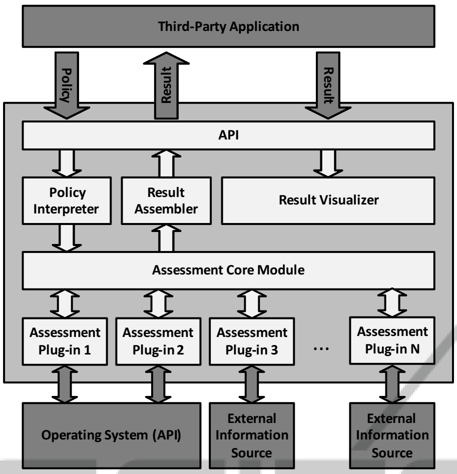


Figure 1: Architecture of the device assessor.

As shown in Figure 1, the architectural design defines an *application programming interface (API)*, through which the device assessor's functionality can be accessed by third-party applications. Reliance on an API assures an easy integration of the device assessor's functionality into arbitrary third-party applications.

As shown in Figure 1, the provided API can be used by third-party applications for three purposes. First, applications can use the API to define security policies to be assessed on the given device. Second, the API provides applications with the results of conducted assessments. Third, applications can optionally send obtained results back to the device assessor, in order to display them to the user. For this purpose, the device assessor provides a *result visualizer*. The result visualizer allows applications to easily inform users about the results of device-assessment processes and releases applications from implementing own result interpretation and visualization methods.

The chosen architecture follows a plug-in based approach. *Assessment plug-ins* are used to assess the given device. Each assessment plug-in supports the assessment of certain security aspects. To assess these aspects, plug-ins either access public APIs provided by the Android operating system, or retrieve additional information from external information sources. For instance, information on the current status of the file-system encryption can be retrieved directly from the Android operating system. In contrast, external information sources might be necessary when applying black- or whitelist based assessment processes. In general, the plug-in based approach assures sustainability by guaranteeing that new security aspects and innovative assessment methods can easily be added to the existing solution at a later stage.

Finally, also flexibility has been defined as design requirement for the proposed device assessor. Hence, the chosen architectural design shown in Figure 1 defines the integration of a *policy interpreter* that is able to handle arbitrary security policies. This way, each third-party application that integrates the device assessor can define its own security policy for the given device. The policy interpreter extracts relevant security aspects from the obtained policy and forwards them to the device assessor's *assessment core module*. This module acts as controller and delegates the assessment of relevant security aspects to appropriate assessment plug-ins. Results of conducted assessment processes are collected by the assessment core module and forwarded to the *result assembler*. The result assembler combines obtained results and returns them to the calling third-party application.

The architecture shown in Figure 1 basically meets all predefined design requirements. Although being mainly tailored to the Android platform, the proposed architecture can basically be used to develop device assessors for arbitrary smartphone platforms. However, in consideration of the various security vulnerabilities of Android that have been discussed in detail in Section 2, we have decided to verify the practicability of the architectural design by means of a concrete implementation for the Google Android platform. Details of this implementation are provided in the next section.

4 IMPLEMENTATION

To assure ease of integration, the developed device assessor has been implemented as Android library project[4]. Implementation details of the developed library project and its integration into third-party applications are presented in this section by means of a concrete example. For this example, we assume that an Android application wants to condition its execution on the satisfaction of the following requirements:

- **R1.** To assure the confidentiality of data stored by the application, the smartphone's file system must already be encrypted or currently being encrypted.
- **R2.** To preclude the existence of SMS intercepting malware on the smartphone, alternative application sources must be disabled and no static SMS broadcast receivers must be registered.
- **R3.** To preclude the existence of manipulated keyboards that log user input, alternative keyboards must not be installed on the smartphone.

[4] http://developer.android.com/tools/projects/index.html

Note that these specific requirements have been defined for demonstration purposes only and do not necessarily make sense in real-world scenarios. In the following subsections we will discuss how these exemplary requirements can be converted into an appropriate security policy, how the implemented device assessor assesses the given smartphone according to this policy, and how obtained results of the assessment process are displayed by the device assessor.

4.1 Policy Definition

The use of security policies guarantees that each application that integrates the developed device assessor can define and assess its own critical security aspects. The use of security policies and related policy description languages (PDL) is a common approach that is frequently followed when security requirements need to be defined in a structured way (Lv and Yan, 2006) (Hashimoto et al., 2009). It is thus reasonable that our solution relies on a policy based approach as well. However, we have refrained from using established PDLs for the sake of simplicity. Instead of using a powerful but complex PDL, our implementation relies on a simple proprietary security-policy format. However, due to the modular design of our implementation, alternative PDLs can easily be integrated into our implementation at a later stage if required.

So far, our implementation supports rather simple policies. The basic building blocks of these policies are so called *security properties*. Security properties and the two logic operators AND and OR can be used to assemble arbitrary security policies. The device assessor's current implementation supports assessment of 22 security properties including the ones listed below.

- **SecProp1.** A password containing numeric characters is required to access the smartphone.
- **SecProp2.** A password containing alphabetic characters is required to access the smartphone.
- **SecProp7.** Encryption has been activated and the device's file system is encrypted.
- **SecProp8.** Encryption is currently being activated.
- **SecProp9.** Encryption is available on the device but has not yet been activated.
- **SecProp10.** Encryption is not available on the device.
- **SecProp12.** No alternative keyboards are installed on the device.
- **SecProp13.** The device's default keyboard is currently in use.
- **SecProp14.** An alternative keyboard is currently in use.
- **SecProp16.** No suspicious application has been registered to receive incoming SMS messages.
- **SecProp17.** There is evidence that the device has been rooted.
- **SecProp22.** Alternative application sources are disabled on the device.

Note that some properties have been omitted due to space limitations. A complete list of all supported security properties is available in the project documentation[5]. Using these security properties, arbitrary security policies can be defined. Considering our concrete example, the security properties *SecProp7*, *SecProp8*, *SecProp12*, *SecProp16*, and *SecProp22* can be used to map the rather informal definition of the requirements *R1*, *R2*, and *R3* to an appropriate security policy.

Using the identified relevant security properties and appropriate logic operators, the overall security policy *SecPol* can be assembled according to Equations (1), (2), and (3).

$$SubPolA = OR(SecProp7, SecProp8) \quad (1)$$

$$SubPolB = AND(SecProp16, SecProp22) \quad (2)$$

$$SecPol = AND(SubPolA, SubPolB, SecProp12) \quad (3)$$

As shown in Figure 2, security policies can be graphically represented by hierarchical trees. The tree root represents the complete security policy. Tree nodes represent sub-policies that in turn consist of two or more child nodes (security properties or sub-policies). Leaf nodes (i.e. nodes with no child nodes) represent security properties.

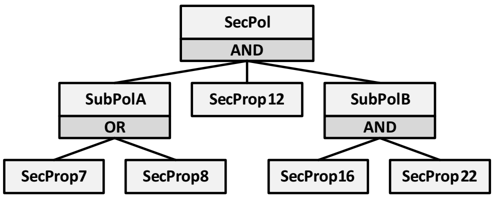


Figure 2: Tree representation of the security policy defined in Equations (1), (2), and (3).

The implemented device assessor's API provides a *PolicyFactory* class that allows third-party applications to define security properties and to assemble arbitrary security policies from different security properties using logic AND and OR operators.

[5] http://demo.egiz.gv.at/

4.2 Device Assessment

According to the architectural design shown in Figure 1, the given Android device is assessed by the assessment core module and a set of assessment plug-ins. While the assessment core module basically provides a common API-based interface to the implemented assessment capabilities, the assessment of different security properties is implemented by the assessment plug-ins. Each plug-in is able to assess a certain set of security properties. So far, seven plug-ins have been implemented that are able to assess the 22 security properties that are currently supported by our implementation.

- **Access-Protection Plug-in.** This plug-in checks the configured access-protection method of the assessed Android device. Required information is retrieved directly from the Android system.
- **Encryption Plug-in.** This plug-in checks the current state of the assessed device's encryption system.
- **Keyboard Plug-in.** This plug-in gives security-critical applications the opportunity to check whether alternative keyboards are installed and used on the assessed Android device. A whitelist containing default keyboards of all common vendors is used to accomplish this task.
- **SMS Broadcast-Receiver Plug-in.** This plug-in detects if there are suspicious applications installed on the assessed device, which have registered themselves to receive incoming SMS messages. These applications are identified by analyzing the Android manifest files of installed applications.
- **Root-Detection Plug-in.** The terms *rooting* and *jailbreaking* subsume methods to gain root access to a smartphone's operating system. Therefore, a root-detection plug-in has been developed that allows security-critical applications to check whether the underlying device has been rooted.

  In contrast to other plug-ins, the root-detection plug-in is not able to provide definite results. Although the plug-in implements various different methods to determine if the assessed device has been rooted, all implemented methods can in theory be circumvented by attackers (or malware) with unlimited root access to the device.
- **Malware-detection Plug-in.** To give applications the opportunity to refuse execution on devices infected with malware, this plug-in implements a simple malware-detection method. This detection method uses an externally provided blacklist in order to separate benign applications from malicious ones.
- **Application-Store Plug-in.** This plug-in verifies whether the assessed device has been configured to allow for the installation of third-party applications from alternative application sources. The required information for this assessment is obtained from Android's public API.

4.3 Result Assembly

Different security properties supported by the implemented plug-ins can be arbitrarily combined to specific security policies using logic AND and OR operators. Assessment results determined by the different plug-ins are collected by the device assessor's assessment core module and forwarded to the result assembler. The result assembler combines obtained results according to the logic operations defined by the security policy. Reconsidering the exemplary security policy shown in Figure 2, the final result is combined from assessments conducted by the encryption plug-in, application-store plug-in, SMS broadcast-receiver plug-in, and keyboard plug-in.

The final result can again be represented by a tree structure that exactly matches the structure of the security policy. Figure 3 shows an exemplary result for the policy shown in Figure 2. In Figure 3, each leaf node represents the assessment result of the corresponding security property. Security properties can either be fulfilled (OK) or not be fulfilled (NOK). Results of sub-policies (i.e. tree nodes) are determined by applying the sub-policy's logic operator (AND or OR) to the results of the tree node's child nodes. This way, each tree node evaluates either to OK or to NOK. The final result of the entire assessment process is represented by the result tree's root node. In the example shown in Figure 3, the given security policy could obviously not be assessed successfully, as one sub-policy is not satisfied by the assessed device.

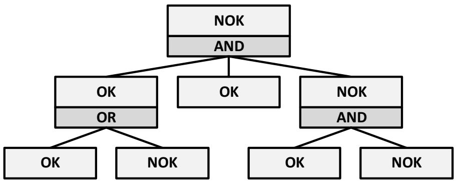


Figure 3: Sample assessment result.

The entire result tree is finally returned to the calling Android application through the device assessor's API. By evaluating the result of the tree's root node, the calling application can easily determine, whether its security policy is fulfilled by the device or not. Additionally, the application can parse the obtained result tree in order to identify problematic security properties of the assessed device.

4.4 Result Visualization

In most cases, security-critical applications that integrate the device assessor might want to inform the user about identified problematic security properties. Applications can do so by parsing the obtained result tree and by implementing own appropriate visualization methods. Alternatively, applications can also use the *result visualizer* provided by the device assessor. The result visualizer can be called via the device assessor's API and takes a result tree as input. The main task of the result visualizer is to appropriately display this result tree and related information to the user.

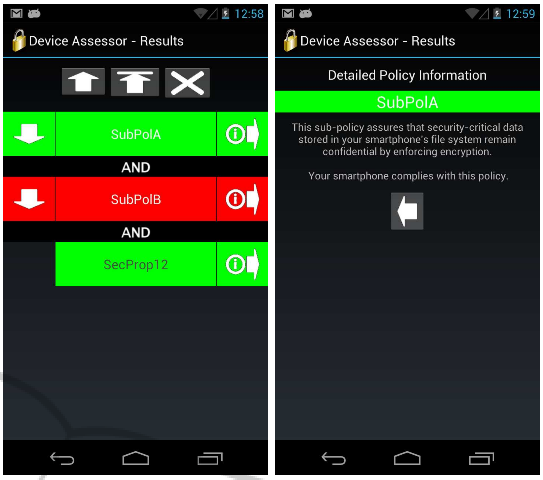
Since security policies – and hence also obtained result trees – can be of arbitrary length and complexity, the result visualizer follows a modular approach to display assessment results. Only one hierarchical level of the result tree is displayed at once. The result visualizer's GUI allows users to navigate between different hierarchical levels.

Figure 4 shows the GUI of the result visualizer. The left screenshot shows the second hierarchical level of the given result tree. The three horizontal bars represent the two sub-policies and the security property of this level. The bar's color indicates whether the property or sub-policy is fulfilled on the particular device or not. Using the info button at the right end of each bar, the user can access additional information about the respective sub-policy (or security property) and its assessment result. This is shown in the right screenshot of Figure 4 for the sub-policy called SubPolA. The displayed text (as well as the name of the property or sub-policy) can be dynamically defined by the calling application during the definition of the security policy.

Users can close this view and return to the result-tree visualization using the leftward-arrow button that can be found below the displayed policy information. Furthermore, the other displayed arrow buttons can be used to navigate through different hierarchical levels of the result tree. This way, the result visualizer helps users to explore identified security weaknesses of their smartphones.

5 EVALUATION

In order to evaluate our implementation, we have assessed the implemented device assessor's applicability and reliability in practice. This has been achieved by evaluating our implementation on both virtual and real devices covering smartphones manufactured by different hardware vendors and featured with different versions of Android. Furthermore, we have made our implementation publicly available[6] and have invited application developers from both the public and the corporate sector to use our solution. Obtained feedback is collected and used to continuously improve our implementation.


12:58
Device Assessor - Results
SubPolA
AND
SubPolB
AND
SecProp12
12:59
Device Assessor - Results
Detailed Policy Information
SubPolA

This sub-policy assures that security-critical data stored in your smartphone's file system remain confidential by enforcing encryption.

Your smartphone complies with this policy.

Figure 4: Left screenshot: Assessment results of security policy's sub-policies and security properties (child nodes of root node). Right screenshot: Detailed information on first sub-policy.

First evaluation results show that our solution basically works reliably on non-rooted Android devices. On rooted devices, correct results are likely but cannot be guaranteed, as the security of the developed solution itself cannot be taken for granted if an attacker has gained unlimited root access to the operating system.

The conducted evaluation has also shown that the proposed solution is basically applicable on all current Android versions. It might however be necessary to provide version-specific instances of the developed device assessor, in order to consider differences in the Android API between different Android versions. Hence, not all features of the proposed solution are available for all Android versions.

6 CONCLUSIONS

In this paper, we have proposed a device assessor for smartphones. This device assessor can be integrated into arbitrary third-party applications in order to verify if the smartphone, on which the application is executed, satisfies a certain security policy. This way, the

[6] http://demo.egiz.gv.at

proposed device assessor allows third-party applications to refrain from executing security-critical operations on potentially insecure smartphones. We have shown the practicability of this approach by means of a concrete implementation for the Android platform. Furthermore, we have evaluated the reliability and applicability of our implementation on a set of virtual and real Android smartphones. Obtained results of this evaluation show that the proposed device assessor is able to reliably assess a wide range of security properties on smartphones.

Although the developed device assessor has already been made publicly available and is ready for productive operation, several further developments are considered as future work. This includes a further improvement of the implmented assessment methods and the evaluation of alternative policy-description languages that could potentially replace the currently used policy format. We are also evaluating potentialities to port the existing Android based solution also to other smartphone platforms.

Even though there is still room for improvement, the current implementation presented in this paper already demonstrates the general practicability of the proposed approach and its capability to assure the security of mobile end-user devices in scenarios such as m-banking or m-government, in which MDM is not an option.

REFERENCES

Bing, H. (2012). Analysis and Research of System Security Based on Android. In *2012 Fifth International Conference on Intelligent Computation Technology and Automation*, pages 581–584. IEEE.

Chen, Y. and Ku, W.-S. (2009). Self-Encryption Scheme for Data Security in Mobile Devices. In *2009 6th IEEE Consumer Communications and Networking Conference*, number 607, pages 1–5. IEEE.

Chin, E., Felt, A. P., Greenwood, K., and Wagner, D. (2011). Analyzing Inter-Application Communication in Android. In *Components*, MobiSys '11, pages 239–252. ACM Press.

Enck, W., Octeau, D., Mcdaniel, P., and Chaudhuri, S. (2011). A Study of Android Application Security. In *USENIX Security*, number August in SEC'11, pages 935–936. USENIX Association.

Enck, W., Ongtang, M., and McDaniel, P. (2009). Understanding Android Security. In *IEEE Security Privacy Magazine*, volume 7, pages 50–57. IEEE.

Felt, A. P. (2012). Android Permissions : User Attention, Comprehension, and Behavior. In *Science And Technology*, pages 1–16.

Gasti, P. and Chen, Y. C. Y. (2010). Breaking and Fixing the Self Encryption Scheme for Data Security in Mobile Devices. In *Parallel Distributed and NetworkBased Processing PDP 2010 18th Euromicro International Conference on*, pages 624–630. IEEE.

Hackmageddon (2011). One Year Of Android Malware (Full List). http://hackmageddon.com/2011/08/11/one-year-of-android-malware-full-list/.

Hashimoto, M., Kim, M., Tsuji, H., and Tanaka, H. (2009). Policy Description Language for Dynamic Access Control Models. In *2009 Eighth IEEE International Conference on Dependable Autonomic and Secure Computing*.

Lookout Mobile Security (2011). 2011 Mobile Threat Report. https://www.lookout.com/resources/reports/mobile-threat-report.

Lv, T. and Yan, P. (2006). A Web Security Solution based on XML Technology. In *2006 International Conference on Communication Technology*, pages 1–4. Ieee.

Pacatilu, P. (2011). Android Applications Security. In *Informatica Economica*, volume 15, pages 163–171. Informatica Economica.

Schwartz, M. J. (2012). Zeus Botnet Eurograbber Steals $47 Million. http://www.informationweek.com/security/attacks/zeus-botnet-eurograbber-steals-47-millio/240143837.

Shabtai, A., Fledel, Y., and Elovici, Y. (2010). Securing Android-Powered Mobile Devices Using SELinux. In *IEEE Security Privacy Magazine*, volume 8, pages 36–44. IEEE Computer Society.

Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., and Dolev, S. (2009). Google Android: A State-of-the-Art Review of Security Mechanisms. In *Neural Networks*, volume 126, page 42.

Shurui, L., Jie, L., Ru, Z., and Cong, W. (2010). A Modified AES Algorithm for the Platform of Smartphone. In *Computational Aspects of Social Networks CASoN 2010 International Conference on*, pages 749–752. IEEE.

Tang, W., Jin, G., He, J., and Jiang, X. (2011). Extending Android Security Enforcement with a Security Distance Model. In *2011 International Conference on Internet Technology and Applications*, pages 1–4. IEEE.

Woods, S. (2013). Bring Your Own Device (BYOD) Increasingly Important to Small Business Budgets. http://technorati.com/business/small-business/article/bring-your-own-device-byod-increasingly/.