AppGuard – Enforcing User Requirements on Android Apps

Michael Backes[1,2], Sebastian Gerling[1], Christian Hammer[1], Matteo Maffei[1], and Philipp von Styp-Rekowsky[1]

[1] Saarland University, Saarbrücken, Germany
[2] Max Planck Institute for Software Systems (MPI-SWS)

Abstract. The success of Android phones makes them a prominent target for malicious software, in particular since the Android permission system turned out to be inadequate to protect the user against security and privacy threats. This work presents *AppGuard,* a powerful and flexible system for the enforcement of user-customizable security policies on untrusted Android applications. AppGuard does not require any changes to a smartphone's firmware or root access. Our system offers complete mediation of security-relevant methods based on callee-site inline reference monitoring. We demonstrate the general applicability of AppGuard by several case studies, e.g., removing permissions from overly curious apps as well as defending against several recent real-world attacks on Android phones. Our technique exhibits very little space and runtime overhead. AppGuard is publicly available, has been invited to the Samsung Apps market, and has had more than 500,000 downloads so far.

1 Introduction

Mobile devices nowadays store a plethora of sensitive information about us – both private and business-related. Usually, this information can be accessed in predefined locations, such as address books or photo folders, and is thus easily locatable by an attacker. Most of these locations, however, lack comprehensive access control and protection mechanisms. When users install a new app on Android, they have no choice but to grant an app all requested permissions at install time, and these permissions cannot be revoked later on. At the same time, these permissions are coarse-grained and their impact is hard to understand for the average user. In the past, several incidents have been reported where private information was deliberately leaked to external servers. Even widely used major apps like Twitter and WhatsApp used to clandestinely send the phone's whole address book to their servers to mine for possible contacts (for iOS, similar behavior was revealed, e.g., for the Facebook app).

In order to overcome this unsatisfactory situation, this paper presents AppGuard, a tool based on inline reference monitoring (IRM) [4,3] that allows the user to enforce fine-grained security and privacy policies on third-party apps. These policies enforced by AppGuard restrict the outreach of vulnerabilities both in third-party applications and the operating system. In short, the IRM algorithm proceeds in two steps. First, app binaries are rewritten to invoke at runtime a security monitor before each security-relevant program operation, usually before each function call to the Android system libraries. Second, the security monitor dynamically checks whether any of the currently enforced security policies allows the attempted operation, and then either grants the execution or executes alternative code (e.g., to return a mock value to prevent the app's termination due to an exception). Since IRM only affects the app binary and not the operating system, AppGuard allows for enforcing policies without rooting phones or changing the operating system.

AppGuard is deployed as a stand-alone app, has been installed on about 500,000 phones so far, and will be soon released to the Samsung Apps market after an explicit invitation from Samsung. The experimental evaluation and case studies discussed in this paper demonstrate the effectiveness of our approach: AppGuard exhibits very little overhead in terms of space and runtime, and it can be used to revoke permissions of excessively curious apps, to enforce complex policies, and to prevent several recent real-world attacks on Android phones.

Although several approaches for enforcing policies in Android based on IRM have recently been presented in the literature [7,2], AppGuard is the only IRM based security tool that has been deployed on a large scale and provides a fully automated on-the-phone instrumentation for third-party apps. In the remainder of this paper, we focus on the architecture and on usability and deployment aspects of the tool: for more details on the IRM algorithm and an extensive discussion of the related work, we refer to [1].

2 AppGuard

Architecture. AppGuard uses caller-site rewriting to inline the reference monitor into existing third-party apps. Fig. 1 provides an overview of its components.

Policies. AppGuard provides a set of built-in security and privacy policies. The tool, in particular, provides general purpose policies that aim at the revocation and restriction of critical Android permissions, such as the Internet-, Contacts- and SendSMS-permission. The Internet policy, for example, provides, besides a general on/off switch option, the possibility to specify a set of servers an app is allowed to connect to. The current version of AppGuard contains 24 different policies in total. Security policies are specified in an aspect-oriented programming style and include a detailed specification of all function calls that are to be controlled by the security monitor (cf. Section 3).

Rewriter. Policies constitute the working basis for the rewriting component to inline the specified checks in front of function calls. The rewriter takes an existing application package (`.apk` file), extracts the `classes.dex` file, and disassembles it. After analyzing the converted assembly code, the rewriter merges the security checks specified by the policy into the existing application code. Finally, it reassembles the `classes.dex` file and repackages the apk file. Our implementation handles both reflective JAVA calls and virtual methods.

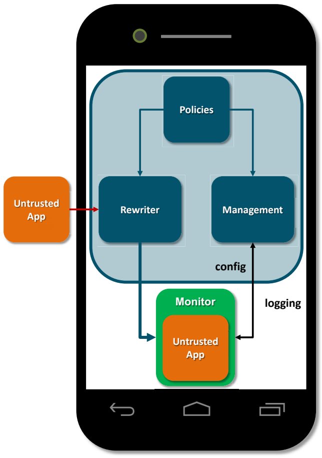
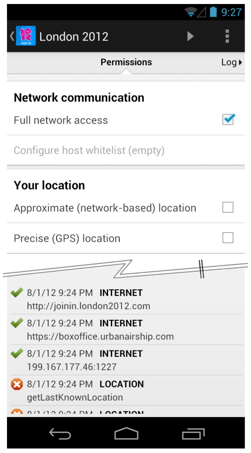


Fig. 1. Architecture of AppGuard



Network communication
Full network access ☑
Configure host whitelist (empty)
Your location
Approximate (network-based) location ☐
Precise (GPS) location ☐

```
✓ 8/1/12 9:24 PM  INTERNET
  http://joinin.london2012.com
✓ 8/1/12 9:24 PM  INTERNET
  https://boxoffice.urbanairship.com
✓ 8/1/12 9:24 PM  INTERNET
  199.167.177.46:1227
✗ 8/1/12 9:24 PM  LOCATION
  getLastKnownLocation
```

Fig. 2. Permission revocation policies (upper part) and the event log (lower part)

Management. Our management component offers the possibility to select a set of predefined policies and to switch single policies on/off on the fly.

Monitor. The monitor is responsible for the actual enforcement of security policies. It tracks the state of the program execution and decides based on the policy configuration whether a security-relevant operation is allowed or not.

Deployment. A major design decision for AppGuard was its development as a standalone app. This is a crucial requirement for a broad deployment on existing smartphones, since the average user of smartphones is not able or willing to "root" the smartphone or to modify the operating system. As Android enforces app isolation by running every app in its own dedicated sandbox, there is no direct possibility to modify the code of other apps, which, however, is required for inline reference monitoring. We solve this problem by leveraging the fact that Android stores app packages in a world-readable location of the filesystem. Thereby, AppGuard can read the `.apk` packages of installed apps and start the rewriting process. In order to install the modified (secured) app, the user is asked to uninstall the original app and to confirm the installation of the secured app instead. This is due to the fact that, for security reasons, Android does not allow apps to silently uninstall other apps.

Since our rewriting process modifies the original app package, the package signature becomes invalid. Therefore, we have to re-sign the secured application with a new key (usually one key per app developer) such that the original app behavior is preserved. For example, Android makes it possible for apps signed with the same key to access each other' s data: the signing mechanism implemented in AppGuard preserves this behavior.

Usability. AppGuard is designed for ease-of-use and does not require any specific security knowledge. In the following, we briefly outline the typical workflow experienced by the user. Whenever a new app is installed on the phone, AppGuard prompts the user to secure the new app. Clicking the notification takes the user to the initial screen for the app instrumentation, which explains the 3-step rewriting process: (*i*) scanning and rewriting of the target app, (*ii*) uninstallation of the original app, and (*iii*) installation of the modified app. Once the modified app is installed, AppGuard allows the user to grant or revoke individual permissions and configure predefined security policies. For example, the user can specify which hosts the app is allowed to connect to. Furthermore, AppGuard keeps a log of all security-relevant operations performed by an app, providing insights into the app behavior and enabling the user to make informed decisions about the policy configuration. Finally, AppGuard includes an on-the-phone manual and gives an overview of installed and secured apps.

Current Release. AppGuard is implemented as a stand-alone app for Android and is purely written in Java. Our tool was first released in July 2012 and its current code base consists of approx. 6500 lines of code. It builds upon the *dexlib*[1] library, which is used for manipulating App binaries (`dex` files). A prototype of AppGuard was originally implemented by researchers at Saarland University [1]. The currently deployed version is based on that work, and it has been built and is maintained by a spin-off company called Backes SRT. The app has been evaluated on a number of real world applications from Google's official app market Google Play (cf. Section 3 for details on some of our case studies).

AppGuard is available for free and supports all Android versions starting from Android 3.0. The application binary can be downloaded from several websites[2]. It has achieved within a few months a large user basis, especially in Europe. Since its first release, it received significant attention in the German media (e.g., a report within ARD Tagesschau, a news transmission of the first German TV channel). The current version has been downloaded more than 500,000 times and the downloads are increasing steadily. Recently, we have been invited to put AppGuard into Samsung's Apps market where Samsung maintains selected apps especially for their own smartphones.

Upcoming Release. Besides the previously described functionalities, the upcoming release of AppGuard implements a wider range of policies (e.g., for redirecting HTTP connections to HTTPS and for preventing `Runtime.Exec()` calls, which are commonly used in recent Android malware). Further, the new release takes care of updating secured apps as the previously described re-signing procedure renders the updates within Google Play impossible.

Limitations. AppGuard monitors both direct Java calls and calls from native code to Java methods. However, it does not monitor function calls inside of native libraries. In case native code is present, it informs the user and asks whether native code should be executed. According to Zhou et al. [8], only less than 5% of all apps include native libraries.

[1] Part of the *smali* disassembler for Android by Ben Gruver [6]

[2] `http://www.chip.de/downloads/SRT-AppGuard-Android-App_56552141.html`
`http://www.heise.de/download/srt-appguard-1187469.html`

Table 1. Inliner evaluation: sizes of apk file, classes.dex, inlined classes.dex, diff. of dex file, # of total and changed instructions, inlining time on the phone

App (Version)	**Size** [Kb]				**Instructions**		**Time** [sec]
	Apk	**Dex**	**Inl**	**Diff**	**Total**	**Chg**	**Phone**
Angry Birds (2.0.2)	15018	994	1038	+44	79311	100	43.4
Endomondo (7.0.2)	3263	1635	1680	+45	134452	88	23.0
Facebook (1.8.3)	4013	2695	2744	+48	224285	218	47.3
PPXIU (1.0) (infected w/ YZHCSMS)	856	793	839	+46	114427	120	19.9
Super Guitar Solo (1.0.1) (infected w/ DroidDream)	1617	120	161	+41	8641	18	4.5
Twitter (3.0.1)	2218	764	813	+48	105594	107	16.7
Wetter.com (1.3.1)	4296	958	1000	+43	89655	36	15.7

3 Performance Evaluation and Case Studies

This section presents the results of the performance evaluation that we conducted on a Google Galaxy Nexus smartphone (1.2 GHz CPU, 1GB RAM) with Android 4.0.4. and discusses some of the case studies conducted with AppGuard.

Table 1 provides statistics on inlining a representative set of apps with AppGuard. We observed a negligible increase in filesize and reasonable inlining times. Besides, we evaluated the runtime overhead introduced by AppGuard through micro-benchmarks (cf. Table 2). The overhead varies depending on the measured function call, but overall we did not recognize any noticeable slowdown. Performance critical apps like games and video players are usually not negatively affected by this slowdown since most time critical computations are performed in native libraries where no security critical information is involved. Our studies indicate that monitored API functions are not frequently called (e.g. in loops).

We evaluated AppGuard in several case studies based on real world applications and successfully enforced different classes of security and privacy policies. Let us consider as an example the *Twitter* app that used to upload the user's address book to the Twitter servers without user consent: revoking the Contacts permission preserves all major functionalities (the find friends function is, of course, limited), but it prevents the privacy leak. Following the same approach AppGuard can also successfully curb the impact of malware. The YZHCSMS malware, for example, sends SMS to premium numbers, which can be prevented by revoking the SendSMS permission. By means of the *Wetter.com* app, we demonstrate the enforcement of a more fine-grained policy by only allowing connections to the

Table 2. Runtime comparison with micro-benchmarks for function calls in unmodified apps and inlined apps with policies disabled and enabled. The runtime overhead is presented for the inlined app with disabled policies.

Function Call	**Original App**	**Inlined App**		**Overhead**
		Pol. disabled	**Pol. enabled**	
`Socket-><init>()`	0.2879 ms	0.3022 ms	0.0248 ms	5.0%
`ContentResolver->query()`	10.484 ms	11.138 ms	0.1 ms	6.2%
`Camera->open()`	150.8 ms	152.36 ms	0.6 ms	1.0%

wetter.com servers, which are used to retrieve the current weather forecast. By blocking all other network connections, the app no longer displays in-app advertisements. Furthermore, AppGuard is able to mitigate weaknesses both in third party apps and in the OS itself. The *Endomondo Sports Tracker*, for example, leaks the authentication token via unsecured HTTP connections. AppGuard can successfully prevent this leakage by enforcing the usage of HTTPS (if supported). An example of an OS vulnerability in Android is the lack of access control mechanisms for the Android photo storage, which is demonstrated by the proof-of-concept exploit implemented in the *(Evil)Tea Timer* app [5]. AppGuard successfully fixes this vulnerability by allowing the user to control the access to her private photos. Finally, many malware authors try to use binary root exploits to gain elevated privileges (e.g. DroidDream). By monitoring API-calls like `Runtime.exec()`, AppGuard can also prevent this type of attacks.

4 Conclusion

This work presents *AppGuard,* a powerful and flexible system for the enforcement of user-defined security policies on untrusted Android applications. AppGuard is based on IRM and does not require any changes to a smartphone's firmware or root access. We demonstrated the feasibility of our approach through an experimental evaluation and several case studies. We take the size of the current user basis of AppGuard as an indication that it tackles a pressing need on Android.

Acknowledgement. This work was supported by the initiative for excellence, the Emmy Noether program of the German research foundation (DFG) and the German Federal Ministry of Education and Research (BMBF) within the Center for IT-Security, Privacy and Accountability (CISPA) at Saarland University.

References

1. Backes, M., Gerling, S., Hammer, C., Maffei, M., von Styp-Rekowsky, P.: Appguard - real-time policy enforcement for third-party applications. Tech. Rep. A/02/2012, Saarland University, Computer Science (July 2012)
2. Davis, B., Sanders, B., Khodaverdian, A., Chen, H.: I-ARM-Droid: A rewriting framework for in-app reference monitors for android applications. In: Mobile Security Technologies 2012, MoST 2012 (2012)
3. Erlingsson, Ú.: The Inlined Reference Monitor Approach to Security Policy Enforcement. Ph.D. thesis, Cornell University (January 2004)
4. Erlingsson, Ú., Schneider, F.B.: IRM enforcement of java stack inspection. In: Proc. 2002 IEEE Symposium on Security and Privacy (Oakland 2002), pp. 246–255 (2002)
5. Gootee, R.: Evil tea timer (2012), `https://github.com/ralphleon/EvilTeaTimer`
6. Gruver, B.: Smali: A assembler/disassembler for android's dex format
7. Jeon, J., Micinski, K.K., Vaughan, J., Fogel, A., Reddy, N., Foster, J., Millstein, T.: Dr. Android and Mr. Hide: Fine-grained permissions in android applications. In: ACM CCS Works. on Sec. & Privacy in Smartphones and Mobile Devices (2012)
8. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In: Proc. NDSS 2012 (February 2012)