Poster: Full-fledged App Sandboxing for Stock Android

Michael Backes
CISPA, Saarland University & MPI-SWS
backes@cs.uni-saarland.de

Sven Bugiel, Christian Hammer,
Oliver Schranz, Philipp von Styp-Rekowsky
CISPA, Saarland University
{lastname}@cs.uni-saarland.de

***Abstract*—We present the first concept for full-fledged app sandboxing on stock Android. Our approach is based on application virtualization and process-based privilege separation to securely encapsulate untrusted apps in an isolated environment. In contrast to all related work on stock Android, we eliminate the necessity to modify the code of monitored apps, and thereby overcome existing legal concerns and deployment problems that rewriting-based approaches have been facing. We realize our concept as a regular Android app called BOXIFY that can be deployed without firmware modifications or root privileges. A systematic evaluation of BOXIFY demonstrates its capability to enforce established security policies without incurring a significant runtime performance overhead.**

I. MOTIVATION

Security research of the past five years has shown that the privacy of smartphone users—and in particular of Android OS users, due to Android's popularity and open-source mindset—is jeopardized by a number of different threats. Those include increasingly sophisticated malware and spyware [1], overly curious libraries [2], but also developer negligence and absence of fail-safe defaults in the Android SDK [3]. To remedy this situation, the development of new ways to protect the end-users' privacy has been an active topic of Android security research during the last years.

Status quo of deploying Android security extensions. From a deployment perspective, the proposed solutions followed two major directions: The majority of the solutions (e.g., [4], [5], [6], [7], [8]) extended the UID-centered security architecture of Android. In contrast, a number of solutions (e.g., [9], [10], [11]) promote inlined reference monitoring (IRM) as an alternative approach that integrates security policy enforcement directly into Android's application layer, i.e., the apps' code.

However, this dichotomy is unsatisfactory for end-users: While OS security extensions provide stronger security guarantees and are preferable in the long run, they require extensive modifications to the operating system and Android application framework. Since the proposed solutions are rarely adopted [12] by Google or the device vendors, users have to resort to customized aftermarket firmware if they wish to deploy new security extensions on their devices. However, installing a firmware forms a technological barrier for most users. In addition, fragmentation of the Android ecosystem and vendor customizations impede the provisioning of custom-built ROMs for all possible device configurations in the wild.

In contrast, solutions that rely on inlined reference monitoring avoid this deployment problem by moving the reference monitor to the application layer and allowing users to install security extensions in the form of apps. However, the currently available solutions provide only insufficient app sandboxing functionality [13] as the reference monitor and the untrusted application share the same process space. Hence, they lack the strong isolation that would ensure tamper-protection and non-bypassability of the reference monitor. Moreover, inlining reference monitors requires modification and hence re-signing of applications, which violates Android's signature-based same-origin model and puts these solutions into a legal gray area.

The sweet spot. The envisioned app sandboxing solution provides immediate strong privacy protection against rogue applications. It would combine the security guarantees of OS security extensions with the deployability of IRM solutions, while simultaneously avoiding their respective drawbacks. Effectively, such a solution would provide an OS-isolated reference monitor that can be deployed entirely as an app on stock Android without modifications to the firmware or code of the monitored applications.

II. OUR APPROACH

In this poster we present a novel concept for Android app sandboxing based on *app virtualization*, which provides tamper-protected reference monitoring without firmware alterations, root privileges or modifications of apps. The key idea of our approach is to encapsulate untrusted apps in a restricted execution environment within the context of another, trusted sandbox application. To establish a restricted execution environment, we leverage Android's *"isolated process"* feature, which allows apps to totally de-privilege selected components—a feature that has so far received little attention beyond the web browser. Code running within an isolated process has no platform permissions, no access to the Android middleware, nor the ability to make persistent changes to the file system. By loading untrusted apps into a de-privileged, isolated process, we shift the problem of sandboxing the untrusted apps from *revoking* their privileges to *granting* their I/O operations whenever the policy explicitly allows them. The I/O operations in question are syscalls (to access the file system, network sockets, bluetooth, and other low-level resources) and the Binder IPC kernel module (to access the application framework). We introduce a novel app virtualization environment that proxies all syscall and Binder channels of isolated apps (see Figure 1). By intercepting any interaction between the app and the system (i.e., kernel and app framework), our solution is able to enforce established and new

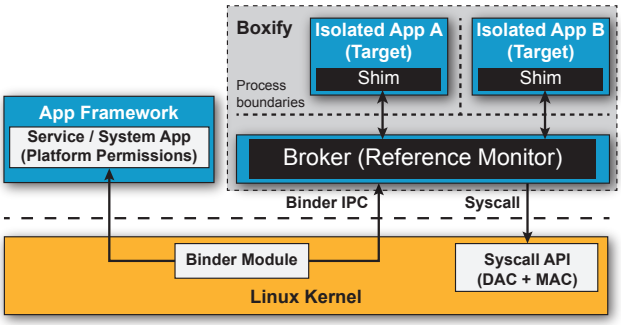

Fig. 1. High-level architecture of BOXIFY. Untrusted apps run in isolated processes (Target) within BOXIFY, their I/O operations (Binder IPC and Syscalls) are intercepted (Shim) and forwarded to the reference monitor (Broker) which runs in a seperate, privileged process. Policy-allowed operations are forwarded by the Broker to the underlying OS (Linux Kernel, App Framework) and their results are returned to the Target process.

privacy-protecting policies. Additionally, it is carefully crafted to be transparent to the encapsulated app in order to keep the app agnostic about the sandbox and retain compatibility to the regular Android execution environment. By executing the untrusted code as a de-privileged process with a UID that differs from the sandbox app's UID, the kernel securely and automatically isolates at process-level the reference monitor implemented by the sandbox app from the untrusted processes. Technically, we build on techniques that were found successful in related work (e.g., libc hooking [10]) while introducing new techniques such as Binder IPC redirection through ServiceManager hooking. We realize our concept as a regular app called BOXIFY that can be deployed on stock Android. To the best of our knowledge, BOXIFY is the first solution to introduce *application virtualization* to stock Android.

With BOXIFY, untrusted applications are not executed by the Android system itself, but run completely encapsulated within the runtime environment that BOXIFY provides. Thereby, BOXIFY allows the instantiation of a wide range of security models from the literature on Android OS security extensions (e.g., [4], [8], [7]) purely at the application layer. BOXIFY is capable of monitoring multiple (untrusted) apps at the same time. By creating a number of isolated processes, multiple apps can run in parallel yet securely isolated in a single instance of BOXIFY. Further, BOXIFY fully controls all inter-component communication between the sandboxed apps and is thus able to not only separate different apps from one another but also to allow controlled collaboration between them. Moreover, BOXIFY has the ability to execute apps that are not regularly installed on the phone: Since BOXIFY executes other apps by dynamically loading their code into one of its own processes and handles all the interaction between the sandboxed application and the OS, there is no need to register the untrusted app with the Android system. Hence, applications can be installed into, updated, or removed from BOXIFY without having Android system permissions.

III. CONCLUSION

We presented the first application virtualization solution for the stock Android OS which combines the strong security guarantees of OS security extensions with the deployability of application layer only solutions. BOXIFY is deployable as a regular app on stock Android (no firmware modification and no root privileges required) and avoids the need to modify sandboxed apps. Our current BOXIFY prototype supports devices with Android version 4.1 through 4.4 (i.e., four out of five devices in the Android ecosystem) and we are currently finalizing support for the Android Runtime of Android v5.0 and later. Furthermore, we will make the BOXIFY source code freely available.

BOXIFY offers all the security advantages of traditional sandboxing techniques and is thus of independent interest for future Android security research. As future work, we are currently investigating different application domains of BOXIFY, such as application-layer only taint-tracking for sandboxed apps, programmable security APIs, or BOXIFY-based malware analysis tools.

REFERENCES

[1] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. 33rd IEEE Symposium on Security and Privacy (Oakland'12)*. IEEE Computer Society, 2012.

[2] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A Study of Android Application Security," in *Proc. 20th USENIX Security Symposium (SEC'11)*. USENIX Association, 2011.

[3] S. Fahl, M. Harbach, T. Muders, M. Smith, L. Baumgärtner, and B. Freisleben, "Why Eve and Mallory love Android: An analysis of Android SSL (in) security," in *Proc. 19th ACM Conference on Computer and Communication Security (CCS'12)*. ACM, 2012.

[4] Y. Zhou, X. Zhang, X. Jiang, and V. Freeh, "Taming information-stealing smartphone applications (on Android)," in *Proc. 4th International Conference on Trust and Trustworthy Computing (TRUST'11)*. Springer, 2011.

[5] W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone application certification," in *Proc. 16th ACM Conference on Computer and Communication Security (CCS'09)*. ACM, 2009.

[6] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry, "Towards Taming Privilege-Escalation Attacks on Android," in *Proc. 19th Annual Network and Distributed System Security Symposium (NDSS'12)*. The Internet Society, 2012.

[7] X. Wangy, K. Sun, and Y. W. J. Jing, "DeepDroid: Dynamically Enforcing Enterprise Policy on Android Devices," in *Proc. 22nd Annual Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society, 2015.

[8] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, "Practical and lightweight domain isolation on Android," in *Proc. 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'11)*. ACM, 2011.

[9] J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein, "Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications," in *Proc. 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'12)*. ACM, 2012.

[10] R. Xu, H. Saïdi, and R. Anderson, "Aurasium – Practical Policy Enforcement for Android Applications," in *Proc. 21st USENIX Security Symposium (SEC'12)*. USENIX Association, 2012.

[11] B. Davis, B. Sanders, A. Khodaverdian, and H. Chen, "I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android Applications," in *Proc. Mobile Security Technologies 2012 (MoST'12)*. IEEE Computer Society, 2012.

[12] S. Smalley and R. Craig, "Security Enhanced (SE) Android: Bringing Flexible MAC to Android," in *Proc. 20th Annual Network and Distributed System Security Symposium (NDSS'13)*. The Internet Society, 2013.

[13] H. Hao, V. Singh, and W. Du, "On the Effectiveness of API-level Access Control Using Bytecode Rewriting in Android," in *Proc. 8th ACM Symposium on Information, Computer and Communication Security (ASIACCS'13)*. ACM, 2013.